Tema 1: Sistemas Basados en Reglas

1.1 Componentes y Estrategias de Inferencia

Grado en Tecnologías Interactivas

Lidia Contreras Ochando

VRAIN
Valencian Research Institute for Artificial Intelligence

DSIIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Introducción

Sistemas Basados en Reglas (SBR)

Son sistemas expertos que utilizan una representación formal de conocimiento y un mecanismo de inferencia para razonar.

Buscan capturar la experiencia humana para resolver problemas y tomar decisiones.

Introducción

Sistemas Basados en Reglas (SBR)

Son sistemas expertos que utilizan una representación formal de conocimiento y un mecanismo de inferencia para razonar.

Buscan capturar la experiencia humana para resolver problemas y tomar decisiones.

Hechos y Reglas

Se basan en una combinación de hechos y reglas para tomar decisiones y producir soluciones.

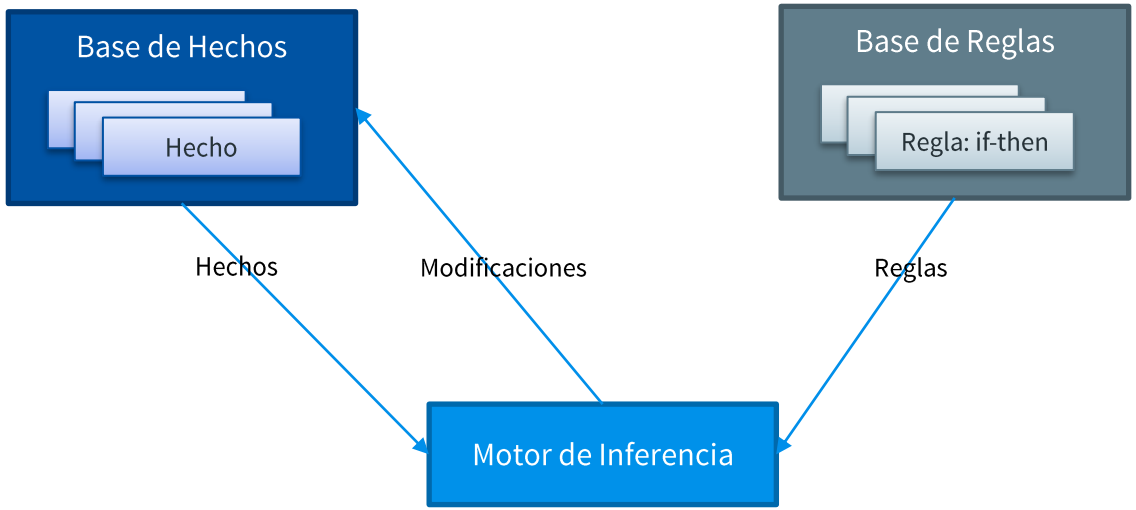
Un hecho es una afirmación verdadera que se conoce acerca del mundo:

"El sol brilla en el cielo”

Una regla es una estructura lógica que describe la relación entre hechos y cómo se pueden utilizar para llegar a una conclusión:

“**Si** el sol brilla en el cielo, **entonces** es un día soleado”

Componentes de un SBR



Base de hechos

Contiene la información actual del problema:

- Datos iniciales
- Datos que se obtienen al aplicar las reglas
- Metas a alcanzar

Hechos:

- Conocimiento declarativo (describen estados o circunstancias)
  - “El paciente tiene fiebre”
- El sistema los utiliza para razonar y tomar decisiones

Base de reglas

Contiene una colección de reglas que describen la relación entre los hechos y las conclusiones que se pueden derivar de ellos

- Las reglas se utilizan para guiar la inferencia y la toma de decisiones del sistema
- No se ejecutan en orden, si no que se seleccionan según distintos factores

Reglas:

- Conocimiento declarativo (describen estados o circunstancias)
- Se componen de dos partes: antecedente → consecuente
  - El **antecedente** (condición) describe una situación en la Base de Hechos
  - El **consecuente** (acción) describe una conclusión a la que se llega en función del antecedente:
    - Añadir un hecho a la BH
    - Eliminar un hecho de la BH
    - Consultas al usuario
    - Acciones externas (print, read, etc).

Base de reglas

Contiene una colección de reglas que describen la relación entre los hechos y las conclusiones que se pueden derivar de ellos

- Las reglas se utilizan para guiar la inferencia y la toma de decisiones del sistema
- No se ejecutan en orden, si no que se seleccionan según distintos factores

Reglas:

- Conocimiento declarativo (describen estados o circunstancias)
- Se componen de dos partes: antecedente → consecuente
  - El **antecedente** (condición) describe una situación en la Base de Hechos
  - El **consecuente** (acción) describe una conclusión a la que se llega en función del antecedente:
    - Añadir un hecho a la BH
    - Eliminar un hecho de la BH
    - Consultas al usuario
    - Acciones externas (print, read, etc).

(antecedente) Si el paciente tiene fiebre y dolor de cabeza
(consecuente) Entonces el paciente puede tener una infección

Base de reglas

Contiene una colección de reglas que describen la relación entre los hechos y las conclusiones que se pueden derivar de ellos

- Las reglas se utilizan para guiar la inferencia y la toma de decisiones del sistema
- No se ejecutan en orden, si no que se seleccionan según distintos factores

Reglas:

- Conocimiento declarativo (describen estados o circunstancias)
- Se componen de dos partes: antecedente → consecuente
  - El **antecedente** (condición) describe una situación en la Base de Hechos
  - El **consecuente** (acción) describe una conclusión a la que se llega en función del antecedente:
    - Añadir un hecho a la BH
    - Eliminar un hecho de la BH
    - Consultas al usuario
    - Acciones externas (print, read, etc).

(antecedente) Si el paciente tiene fiebre y dolor de cabeza
(consecuente) Entonces el paciente puede tener una infección

Para que una regla se dispare, **TODAS** las clausulas del antecedente se deben cumplir

Motor de inferencia

El motor de inferencia es el encargado de realizar el proceso de razonamiento para llegar a una conclusión a partir de las reglas y hechos que se han definido.

Este proceso se realiza en varias fases:

- Fase de matching: Comprueba qué reglas pueden dispararse con qué hechos
- Fase de selección: Selecciona la regla a aplicar dependiendo de la estrategia que se use
- Ejecuta la regla
- Actualiza la base de hechos

Hay diferentes tipos de inferencia, como el encadenamiento hacia adelante y hacia atrás.

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto.

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(hecho4) Juan tiene formación universitaria

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(hecho4) Juan tiene formación universitaria

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(hecho4) Juan tiene formación universitaria

(hecho5) Juan puede votar

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(hecho4) Juan tiene formación universitaria

(hecho5) Juan puede votar

(hecho6) Juan puede ser profesor

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Ejemplo

(hecho1) Juan tiene 40 años

(hecho2) Juan es ingeniero

(hecho3) Juan es adulto

(hecho4) Juan tiene formación universitaria

(hecho5) Juan puede votar

(hecho6) Juan puede ser profesor

(hecho7) Juan es capaz de diseñar sistemas

(regla1) Si Juan tiene más de 20 años, entonces Juan es adulto

(regla2) Si Juan es ingeniero, entonces Juan tiene formación universitaria

(regla3) Si Juan es adulto, entonces Juan puede votar

(regla4) Si Juan es ingeniero y es adulto, Juan puede ser profesor

(regla5) Si Juan tiene formación universitaria, entonces Juan es capaz de diseñar sistemas

- **ESTRATEGIA**: Ante un conflicto, seleccionamos la regla de menor número
- Una vez aplicada una regla a un hecho, no se vuelve a aplicar después

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}		

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1
Nuevo Hecho → H4: La temperatura es cálida			
2	{H1, H2, H3, H4}		

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1
Nuevo Hecho → H4: La temperatura es cálida			
2	{H1, H2, H3, H4}	{R2}	

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1
Nuevo Hecho → H4: La temperatura es cálida			
2	{H1, H2, H3, H4}	{R2}	R2

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1
Nuevo Hecho → H4: La temperatura es cálida			
2	{H1, H2, H3, H4}	{R2}	R2
Nuevo Hecho → H5: Es verano			
3	{H1, H2, H3, H4, H5}		

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1
Nuevo Hecho → H4: La temperatura es cálida			
2	{H1, H2, H3, H4}	{R2}	R2
Nuevo Hecho → H5: Es verano			
3	{H1, H2, H3, H4, H5}	{R3}	

Inferencia: Encadenamiento hacia delante

Se parte de los hechos conocidos y, a través de la aplicación de las reglas, se llega a la conclusión de nuevos hechos.

Con estos hechos y estas reglas, ¿a qué conclusiones puedo llegar?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Conjunto Conflicto	Resolución
1	{H1, H2, H3}	{R1, R2}	R1
Nuevo Hecho → H4: La temperatura es cálida			
2	{H1, H2, H3, H4}	{R2}	R2
Nuevo Hecho → H5: Es verano			
3	{H1, H2, H3, H4, H5}	{R3}	R3
Nuevo Hecho → H6: La gente va a la playa			
4	{H1,H2,H3, H4, H5, H6}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}	{R1, R8, R9}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}	{R1, R8, R9}	R1
5	{ b, c, x, d, e, f}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}	{R1, R8, R9}	R1
5	{ b, c, x, d, e, f}	{R3, R8, R9}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}	{R1, R8, R9}	R1
5	{ b, c, x, d, e, f}	{R3, R8, R9}	R3
6	{ b, c, x, d, e, f, a}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}	{R1, R8, R9}	R1
5	{ b, c, x, d, e, f}	{R3, R8, R9}	R3
6	{ b, c, x, d, e, f, a}	{R6, R8, R9}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia delante:

Ciclo	BH	Conjunto Conflicto	Resolución
1	{b, c}	{R4, R7}	R4
2	{b, c, x}	{R7, R8, R9}	R7
3	{b, c, x, d}	{R5, R8, R9}	R5
4	{b, c, x, d, e}	{R1, R8, R9}	R1
5	{ b, c, x, d, e, f}	{R3, R8, R9}	R3
6	{ b, c, x, d, e, f, a}	{R6, R8, R9}	R6
7	{ b, c, x, d, e, f, a, h}		

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa			

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa	{Hoy es domingo, hace sol, es verano}	{R3}	

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa	{~~Hoy es domingo~~, hace sol, es verano}	{R3}	

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa	{~~Hoy es domingo~~, ~~hace sol~~, es verano}	{R3}	

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa	{~~Hoy es domingo~~, ~~hace sol~~, es verano}	{R3}	
	{H1,H2,H3}	La gente va a la playa	{Hoy es 1 de Agosto}	{R2}	

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa	{~~Hoy es domingo~~, ~~hace sol~~, es verano}	{R3}	
	{H1,H2,H3}	La gente va a la playa	{~~Hoy es 1 de Agosto~~}	{R2}	R2
Nuevo Hecho → H4: Es verano					
2	{H1, H2, H3, H4}	La gente va a la playa	{~~Hoy es domingo~~, ~~hace sol~~, es verano}	{R3}	

Inferencia: Encadenamiento hacia atrás

Se parte de un objetivo o consecuencia deseada y se retrocede hasta llegar a la base de hechos.

Para llegar a este objetivo, ¿qué hechos se deben cumplir?

BH:

H1: Hoy es Domingo

H2: Hace sol

H3: Hoy es 1 de Agosto

Meta:

La gente va a la playa

BR:

R1: Si hoy es domingo y hace sol → La temperatura es cálida

R2: Si hoy es 1 de Agosto → Es verano

R3: Si hoy es Domingo y hace sol y es verano → La gente va a la playa

Ciclo	BH	Meta	Nueva Meta	CC	Resolución
1	{H1,H2,H3}	La gente va a la playa	{~~Hoy es domingo~~, ~~hace sol~~, ~~es verano~~}	{R3}	
	{H1,H2,H3}	La gente va a la playa	{~~Hoy es 1 de Agosto~~}	{R2}	R2
Nuevo Hecho → H4: Es verano					
2	{H1, H2, H3, H4}	La gente va a la playa	{~~Hoy es domingo, hace sol, es verano~~}	{R3}	R3
Nuevo Hecho → H5: La gente va a la playa					
3	{H1, H2, H3, H4, H5}	La gente va a la playa			

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h			

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
	{b, c}	h	{**a**, x}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{R2, R3, R8}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{**d**, g}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{**d**, g}	{R7, R9}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**~~, g}	{R7, R9}	R7
4	{b, c, d}	h	{~~d~~, **g**}		

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**~~, ~~g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~d~~, ~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**, g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{c, f}		

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**, g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, **f**}		

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**, g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, **f**}	{R1}	

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**, g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, **f**}	{R1}	
6	{b, c, d}	h	{b, d, e}		

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**, g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, **f**}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, **e**}		

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**, g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, **f**}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, **e**}	{R5}	

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**~~, ~~g~~}	{R7, R9}	R7
4	{b, c, d}	h	{~~**g**~~}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, **f**}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, ~~**e**~~}	{R5}	R5
7	{b, c, d, e}	h	{**f**}	{R1}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{**a**, x}	{**R2**, R3, R8}	
3	{b, c}	h	{~~**d**~~, g}	{R7, R9}	R7
4	{b, c, d}	h	{**g**}		
	{b, c, d}	h	{**a**, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, ~~**f**~~}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, ~~**e**~~}	{R5}	R5
7	{b, c, d, e}	h	{~~**f**~~}	{R1}	R1
8	{b, c, d, e, f}	h	{**a**, x}	{~~R2~~, **R3**, R8}	

Ejemplo

BH: **Meta:**

{b, c} h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{~~a~~, x}	{**R2**, R3, R8}	
3	{b, c}	h	~~{d, g}~~	{R7, R9}	R7
4	{b, c, d}	h	{~~g~~}		
	{b, c, d}	h	{~~a~~, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, ~~f~~}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, ~~e~~}	{R5}	R5
7	{b, c, d, e}	h	{~~f~~}	{R1}	R1
8	{b, c, d, e, f}	h	{~~a~~, x}	{~~R2~~, R3, R8}	R3
9	{b, c, d, e, f, a}	h	{~~a~~, **x**}		

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{~~a~~, x}	{**R2**, R3, R8}	
3	{b, c}	h	~~{d, g}~~	{R7, R9}	R7
4	{b, c, d}	h	{~~g~~}		
	{b, c, d}	h	{~~a~~, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, ~~f~~}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, ~~e~~}	{R5}	R5
7	{b, c, d, e}	h	{~~f~~}	{R1}	R1
8	{b, c, d, e, f}	h	{~~a~~, x}	{~~R2~~, R3, R8}	R3
9	{b, c, d, e, f, a}	h	{~~a~~, **x**}	{R4}	

Ejemplo

BH: {b, c}

Meta: h

BR:

R1: b & d & e → f

R2: d & g → a

R3: c & f → a

R4: b → x

R5: d → e

R6: a & x → h

R7: c → d

R8: x & c → a

R9: x & b → d

Encaminamiento hacia atrás:

Ciclo	BH	Meta	Nuevas Metas	CC	Resolución
1	{b, c}	h		{R6}	
2	{b, c}	h	{~~a~~, ~~x~~}	{**R2**, R3, R8}	
3	{b, c}	h	~~{d, g}~~	{R7, R9}	R7
4	{b, c, d}	h	{~~g~~}		
	{b, c, d}	h	{~~a~~, x}	{~~R2~~, **R3**, R8}	
5	{b, c, d}	h	{~~c~~, ~~f~~}	{R1}	
6	{b, c, d}	h	{~~b~~, ~~d~~, ~~e~~}	{R5}	R5
7	{b, c, d, e}	h	{~~f~~}	{R1}	R1
8	{b, c, d, e, f}	h	{~~a~~, ~~x~~}	{~~R2~~, R3, R8}	R3
9	{b, c, d, e, f, a}	h	{~~a~~, ~~x~~}	{R4}	R4
10	{b, c, d, e, f, a, x}	h		{R6}	R6
11	{b, c, d, e, f, a, x, h}	h			

Tema 1: Sistemas Basados en Reglas

1.2 Sintaxis (CLIPS) y Pattern-Matching

Grado en Tecnologías Interactivas

Lidia Contreras Ochando

VRAIN
Valencian Research Institute for Artificial Intelligence

DSIIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Sintaxis (CLIPS)

CLIPS (C Language Integrated Production System) es un lenguaje de programación que se utiliza para construir sistemas expertos basados en reglas.

Algunos comandos para la aplicación:

- (clear) Elimina todas las reglas y hechos de la memoria. Equivalente a apagar y reiniciar CLIPS
- (reset) Elimina la información de hechos de la memoria (pero no las reglas) y restablece la agenda
- (run) Comienza a ejecutar un programa CLIPS

Sintaxis (CLIPS)

CLIPS (C Language Integrated Production System) es un lenguaje de programación que se utiliza para construir sistemas expertos basados en reglas.

Algunos comandos para la aplicación:

- (clear) Elimina todas las reglas y hechos de la memoria. Equivalente a apagar y reiniciar CLIPS
- (reset) Elimina la información de hechos de la memoria (pero no las reglas) y restablece la agenda
- (run) Comienza a ejecutar un programa CLIPS

Todos los comandos, hechos y reglas se expresan entre paréntesis

CLIPS: Hechos

Los hechos se describen con elementos y valores ordenados y separados por espacios en blanco.
El primer elemento de un hecho establece una ‘relación’ con el resto de elementos

Se crean usando el comando `assert`.

```
(assert (color verde) (color azul)
```

CLIPS: Hechos

Los hechos se describen con elementos y valores ordenados y separados por espacios en blanco.
El primer elemento de un hecho establece una ‘relación’ con el resto de elementos

Se crean usando el comando `assert`.

```
(assert (color verde) (color azul)
```

Valores

CLIPS: Hechos

Los hechos se describen con elementos y valores ordenados y separados por espacios en blanco.
El primer elemento de un hecho establece una ‘relación’ con el resto de elementos

Se crean usando el comando `assert`.

```
(assert (color verde) (color azul)
```

Valores

El orden de los valores es importante:

```
(Pepe 22514954 38 abogado)
(Pepe abogado 22514954 38)
```

Son hechos diferentes

CLIPS: Hechos

Los hechos se describen con elementos y valores ordenados y separados por espacios en blanco.
El primer elemento de un hecho establece una ‘relación’ con el resto de elementos

Se crean usando el comando `assert`.

```
(assert (color verde) (color azul)
```

Valores

El orden de los valores es importante:

```
(Pepe 22514954 38 abogado)
(Pepe abogado 22514954 38)
```

Son hechos diferentes

El comando (`facts`) enumerará todos los hechos actuales.

```
CLIPS>(facts)

f-0 (color verde)

For a total of 1 fact.
```

CLIPS: Definir hechos

Los hechos pueden definirse de diferentes maneras. La manera que se escoja definirá cómo debemos crear las reglas.

1	2	4
7	8	6
3		5

CLIPS: Definir hechos

Los hechos pueden definirse de diferentes maneras. La manera que se escoja definirá cómo debemos crear las reglas.

1	2	4
7	8	6
3		5

```
BH = { (puzzle 1 1 1)
       (puzzle 1 2 2)
       (puzzle 1 3 4)
       (puzzle 2 1 7)
       (puzzle 2 2 8)
      …
}
```

CLIPS: Definir hechos

Los hechos pueden definirse de diferentes maneras. La manera que se escoja definirá cómo debemos crear las reglas.

1	2	4
7	8	6
3		5

```
BH = { (puzzle 1 1 1)
       (puzzle 1 2 2)
       (puzzle 1 3 4)
       (puzzle 2 1 7)
       (puzzle 2 2 8)
      …
}
```

```
BH = { (puzzle 1 2 4 7 8 6 3 0 5) }
```

CLIPS: Hechos

Los hechos también se pueden eliminar de la base de datos de hechos usando el comando `retract`.

```
CLIPS>(assert (color verde))
<Fact-0>
CLIPS>(assert (color rojo))
<Fact-1>
CLIPS>(retract 0)
CLIPS>(facts)
f-1 (color rojo)
For a total of 1 fact.
```

CLIPS: Hechos

Los hechos también se pueden eliminar de la base de datos de hechos usando el comando `retract`.

```
CLIPS>(assert (color verde))
<Fact-0>
CLIPS>(assert (color rojo))
<Fact-1>
CLIPS>(retract 0)
CLIPS>(facts)
f-1 (color rojo)
For a total of 1 fact.
```

Para eliminar un hecho, se debe especificar un número (el índice del hecho).

CLIPS: Hechos

Los hechos también se pueden eliminar de la base de datos de hechos usando el comando `retract`.

```
CLIPS>(assert (color verde))
<Fact-0>
CLIPS>(assert (color rojo))
<Fact-1>
CLIPS>(retract 0)
CLIPS>(facts)
f-1 (color rojo)
For a total of 1 fact.
```

Para eliminar un hecho, se debe especificar un número (el índice del hecho).

Los índices de hechos no se reutilizan.

CLIPS: Hechos persistentes

`deffacts` es una forma de especificar hechos que se recrean cada vez que se ejecuta un (`reset`).

```
(deffacts hechos-iniciales (animal perro) (animal pato) (animal gato))
```

Afirmará tres hechos en la base de datos cada vez que se reinicie el sistema.

CLIPS: Reglas

En general, una regla se expresa en la forma 'SI algo es cierto ENTONCES acción'.

```
(defrule perro
      (animal perro)
 =>
      (assert (sonido guau))
      (printout t "Es un perro" crlf))
)
```

CLIPS: Reglas

En general, una regla se expresa en la forma 'SI algo es cierto ENTONCES acción'.

```
(defrule perro
      (animal perro)
 =>
      (assert (sonido guau))
      (printout t "Es un perro" crlf))
)
```

Un nombre único

CLIPS: Reglas

En general, una regla se expresa en la forma 'SI algo es cierto ENTONCES acción'.

```
(defrule perro
    (animal perro)
 =>
    (assert (sonido guau))
    (printout t "Es un perro" crlf))
)
```

- `(defrule perro` → Un nombre único
- `(animal perro)` → Parte ‘SI’ de la regla

CLIPS: Reglas

En general, una regla se expresa en la forma 'SI algo es cierto ENTONCES acción'.

```
(defrule perro
    (animal perro)
 =>
    (assert (sonido guau))
    (printout t "Es un perro" crlf))
)
```

- `(defrule perro` → Un nombre único
- `(animal perro)` → Parte ‘SI’ de la regla
- `=> … (printout t "Es un perro" crlf))` → Parte ‘ENTONCES’ de la regla

CLIPS: Reglas

En general, una regla se expresa en la forma 'SI algo es cierto ENTONCES acción'.

```
(defrule perro
    (animal perro)
 =>
    (assert (sonido guau))
    (printout t "Es un perro" crlf))
)
```

Un nombre único

Parte ‘SI’ de la regla

Parte ‘ENTONCES’ de la regla

Imprime texto por pantalla:

t → Pantalla (terminal)

crlf→ Salto de línea

CLIPS: Reglas

Los elementos condicionales del antecedente de una regla son de dos tipos:

- **Patrón (pattern):** Se utiliza para determinar los hechos que satisfacen la condición especificada en el patrón. Los patrones de una regla unifican con los hechos
  - Se satisface si existe al menos un hecho en la BH que hace matching con el patrón

- **Test:** expresión booleana que se evalúa a TRUE o FALSE
  - Se satisface si se evalúa a TRUE

TODOS los patrones y tests del antecedente de la regla se deben satisfacer para que la regla sea aplicable

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Mediante el uso de **variables**, podemos comprobar qué reglas se pueden disparar con los hechos existentes en la BH.

```
BH:{(animal perro) (animal pato) (animal gato))}

(defrule animal

      (animal ?nombre)

=>

      (printout t ?nombre " encontrado" crlf)

)
```

Las variables monovaluadas coinciden exactamente con un valor y se puede utilizar en otras partes de la regla.

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Mediante el uso de **variables**, podemos comprobar qué reglas se pueden disparar con los hechos existentes en la BH.

```
BH:{(animal perro) (animal pato) (animal gato))}

(defrule animal

      (animal ?nombre)

=>

      (printout t ?nombre " encontrado" crlf)

)
```

Las variables monovaluadas coinciden exactamente con un valor y se puede utilizar en otras partes de la regla.

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Mediante el uso de **variables**, podemos comprobar qué reglas se pueden disparar con los hechos existentes en la BH.

```
BH:{(animal perro) (animal pato) (animal gato))}

(defrule animal
      (animal ?nombre)
=>
      (printout t ?nombre " encontrado" crlf)
)
```

Las variables monovaluadas coinciden exactamente con un valor y se puede utilizar en otras partes de la regla.

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Mediante el uso de **variables**, podemos comprobar qué reglas se pueden disparar con los hechos existentes en la BH.

```
BH:{(animal perro) (animal pato) (animal gato))}

(defrule animal
      (animal ?nombre)
=>
      (printout t ?nombre " encontrado" crlf)
)
```

```
CLIPS>(run)
    perro encontrado
    pato encontrado
    gato encontrado
    CLIPS>
```

Las variables monovaluadas coinciden exactamente con un valor y se puede utilizar en otras partes de la regla.

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

También se puede usar el **comodín** sin nombre (animal **?**), pero en este caso la palabra que vaya después de nombre no quedará guardada en ninguna variable y por tanto no podremos usarla en ninguna otra parte de la regla.

```
BH = {(animal perro) (animal pato) (animal gato)}

(defrule animal

      (animal ?)

=>

      (printout t  "Se ha encontrado un animal" crlf)

)
```

Los comodines monovaluados coinciden exactamente con un valor.

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

También se puede usar el **comodín** sin nombre (animal **?**), pero en este caso la palabra que vaya después de nombre no quedará guardada en ninguna variable y por tanto no podremos usarla en ninguna otra parte de la regla.

```
BH = {(animal perro) (animal pato) (animal gato)}

(defrule animal

      (animal ?)

=>

      (printout t  "Se ha encontrado un animal" crlf)

)
```

Los comodines monovaluados coinciden exactamente con un valor.

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Los comodines y variables pueden usarse para encontrar el mismo patrón en diferentes hechos.

```
(defrule mamifero
      (animal ?nombre)
      (not (pone-huevos ?nombre))
 =>
      (assert (mamifero ?nombre))
      (printout t ?nombre " es un mamifero" crlf)
)
```

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Los comodines y variables pueden usarse para encontrar el mismo patrón en diferentes hechos.

```
(defrule mamifero
      (animal ?nombre)
      (not (pone-huevos ?nombre))
 =>
      (assert (mamifero ?nombre))
      (printout t ?nombre " es un mamifero" crlf)
)
```

Niega el hecho → Busca que NO exista un hecho pone-huevos con esa variable ?nombre

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Los comodines y variables pueden usarse para encontrar el mismo patrón en diferentes hechos.

```
(defrule mamifero
      (animal ?nombre)
      (not (pone-huevos ?nombre))
 =>
      (assert (mamifero ?nombre))
      (printout t ?nombre " es un mamifero" crlf)
)
```

Niega el hecho → Busca que NO exista un hecho pone-huevos con esa variable ?nombre

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

En una misma comprobación se puede usar más de un comodín/variable.

```
(defrule cachorro
      (mamifero ?nombre)
      (cachorro-de ?nombre ?hijo)
  =>
      (assert (mamifero ?hijo))
      (printout t ?hijo " es un mamifero" crlf)
)
```

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

En una misma comprobación se puede usar más de un comodín/variable.

```
(defrule cachorro
      (mamifero ?nombre)
      (cachorro-de ?nombre ?hijo)
  =>
      (assert (mamifero ?hijo))
      (printout t ?hijo " es un mamifero" crlf)
)
```

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

En una misma comprobación se puede usar más de un comodín/variable.

```
(defrule cachorro
      (mamifero ?nombre)
      (cachorro-de ?nombre ?hijo)
  =>
      (assert (mamifero ?hijo))
      (printout t ?hijo " es un mamifero" crlf)
)
```

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Para borrar un hecho es necesario conocer su índice.

Este índice podemos vincularlo a una variable en las reglas:

```
(defrule borrar-mamifero
      ?f <- (mamifero ?)
 =>
      (retract ?f)
)
```

CLIPS: Pattern-Matching (Variables y Comodines Monovaluados)

Para borrar un hecho es necesario conocer su índice.

Este índice podemos vincularlo a una variable en las reglas:

```
(defrule borrar-mamifero
     ?f <- (mamifero ?)
 =>
      (retract ?f)
)
```

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-1: (block A colour red)
             f-2: (block B colour blue)
             f-3: (available paint green)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-1: (block A colour red)
             f-2: (block B colour blue)
             f-3: (available paint green)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-1: (block A colour red)
             f-2: (block B colour blue)
             f-3: (available paint green)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-1: (block A colour red)
             f-2: (block B colour blue)
             f-3: (available paint green)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

Cuando la regla se dispara, se aplican los comandos del consecuente.

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-1: (block A colour red)
             f-2: (block B colour blue)
             f-3: (available paint green)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

Cuando la regla se dispara, se aplican los comandos del consecuente.

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-1: (block A colour red)
             f-2: (block B colour blue)
             f-3: (available paint green)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

Cuando la regla se dispara, se aplican los comandos del consecuente.

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-2: (block B colour blue)
             f-4: (available paint red)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

Cuando la regla se dispara, se aplican los comandos del consecuente.

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-2: (block B colour blue)
             f-4: (available paint red)
             f-5: (block A colour green)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

Cuando la regla se dispara, se aplican los comandos del consecuente.

CLIPS: Pattern-Matching (Ejemplo)

```
BH= {        f-2: (block B colour blue)
             f-4: (available paint red)
             f-5: (block A colour green)
    }

(defrule paint-block
        ?fa <- (block A colour red)
        ?fb <- (available paint green)
    =>
        (retract ?fa ?fb)
        (assert (block A colour green))
        (printout  t “Bloque pintado de verde “ crlf))
```

?fa y ?fb son variables que almacenan los índices de los hechos que hacen matching con los patrones.

(?fa=f-1, ?fb=f-3)

Cuando la regla se dispara, se aplican los comandos del consecuente.

CLIPS: Pattern-Matching (Variables y Comodines Multivaluados)

Las variables y comodines multivaluados coinciden con 0 o más símbolos:

```
(beatles john_lennon paul_mccartney george_harrison ringo_starr)
```

Si queremos buscar “paul_mccartney”, pero no sabemos en qué posición está, podemos usar:

```
(beatles $? paul_mccartney $?)
```

Las variables y comodines multivaluados coinciden con un espacio vacío (cero elementos) o con cualquier otro número de elementos.

CLIPS: Pattern-Matching (Variables y Comodines)

Si quisiéramos imprimir de uno en uno todos los miembros de la banda, podríamos hacerlo usando los dos tipos de variables (monovaluadas y multivaluadas):

```
(beatles john_lennon paul_mccartney george_harrison ringo_starr)
(defrule miembros-banda
      (beatles $?miembros1 ?miembro $?miembros2)
=>
      (printout t ?miembro “ es miembro de la banda”)
)
```

Coincidencias posibles:

#Match	$?miembros1	?miembro	$?miembros2

CLIPS: Pattern-Matching (Variables y Comodines)

Si quisiéramos imprimir de uno en uno todos los miembros de la banda, podríamos hacerlo usando los dos tipos de variables (monovaluadas y multivaluadas):

```
(beatles john_lennon paul_mccartney george_harrison ringo_starr)
(defrule miembros-banda
      (beatles $?miembros1 ?miembro $?miembros2)
=>
      (printout t ?miembro " es miembro de la banda")
)
```

Coincidencias posibles:

#Match	$?miembros1	?miembro	$?miembros2
1	john_lennon paul_mccartney george_harrison	ringo_starr	()
2	john_lennon paul_mccartney	george_harrison	ringo_starr
3	john_lennon	paul_mccartney	george_harrison ringo_starr
4	()	john_lennon	paul_mccartney george_harrison ringo_starr

CLIPS: Pattern-Matching (Ejemplo)

```
(list a a b a c b b a c d c a b c)
```

```
(list $?x c $?y)
```

Coincidencias posibles:

#Match	$?x	$?y

CLIPS: Pattern-Matching (Ejemplo)

```
(list a a b a c b b a c d c a b c)

(list $?x c $?y)
```

Coincidencias posibles:

#Match	$?x	$?y
1	(a a b a c b b a c d c a b)	()
2	(a a b a c b b a c d)	(a b c)
3	(a a b a c b b a)	(d c a b c)
4	(a a b a)	(b b a c d c a b c)

CLIPS: Operadores

- and: Ambos hechos deben ser ciertos.
- or: Uno de los dos hechos debe ser cierto.
- Not: No existe ese hecho.

- + Suma
- - Resta
- / División
- * Multiplicación
- Mod Módulo

- > Mayor
- < Menor
- <= Menor o igual
- >= Mayor o igual
- <> Diferente
- = Igual

Utilizamos test para evaluar el resultado.

CLIPS: Operadores

- and: Ambos hechos deben ser ciertos.
- or: Uno de los dos hechos debe ser cierto.
- Not: No existe ese hecho.

```
(defrule paraguas
      (test (or (clima lloviendo)
                (clima nevando)))
      =>
      (assert (paraguas usar))
)
```

- + Suma
- - Resta
- / División
- * Multiplicación
- Mod Módulo

- > Mayor
- < Menor
- <= Menor o igual
- >= Mayor o igual
- <> Diferente
- = Igual

Utilizamos test para evaluar el resultado.

CLIPS: Operadores

- and: Ambos hechos deben ser ciertos.
- or: Uno de los dos hechos debe ser cierto.
- Not: No existe ese hecho.

```
(defrule paraguas
        (test (or (clima lloviendo)
                  (clima nevando)))
        =>
        (assert (paraguas usar))
)
```

- + Suma
- - Resta
- / División
- * Multiplicación
- Mod Módulo

- > Mayor
- < Menor
- <= Menor o igual
- >= Mayor o igual
- <> Diferente
- = Igual

El operador se coloca antes de los argumentos (notación de prefijo).

Utilizamos test para evaluar el resultado.

CLIPS: Operadores

- and: Ambos hechos deben ser ciertos.
- or: Uno de los dos hechos debe ser cierto.
- Not: No existe ese hecho.

+	Suma
-	Resta
/	División
*	Multiplicación
Mod	Módulo

>	Mayor
<	Menor
<=	Menor o igual
>=	Mayor o igual
<>	Diferente
=	Igual

```
(defrule paraguas
        (test (or (clima lloviendo)
                  (clima nevando)))
        =>
        (assert (paraguas usar))
)
```

El operador se coloca antes de los argumentos (notación de prefijo).

Utilizamos test para evaluar el resultado.

CLIPS: Operadores

Los operadores se pueden anidar para hacer una comprobación mayor.

```
(defrule check-varicela
      (paciente ?p)
      (edad ?p ?e)
      (test (< ?e 10))
      (test(> ?e 2))
      (síntoma ?p fiebre)
=>
      (assert (enfermedad ?p varicela))
)
```

```
(defrule check-varicela
      (paciente ?p)
      (edad ?p ?e)
      (test (and (< ?e 10)(> ?e 2)))
      (síntoma ?p fiebre)
=>
      (assert (enfermedad ?p varicela))
)
```

Unificación de Reglas (Ejemplo 1)

Base de Reglas

```
(defrule R1
    (list $? ?x $?)
    (test (evenp ?x))
=>
    (assert (number even ?x))
    (printout t ?x " is an even number of the list " crlf))


(defrule R2
    (list $? ?x $?)
    (test (oddp ?x))
=>
    (assert (number odd ?x))
    (printout t ?x " is an odd number of the list " crlf))
```

```
(deffacts data
    (list 26 2 17 18))
```

→

Base de Hechos

f-1: (list 26 2 17 18)

Instancias de reglas

Rule	# Match	Facts

¿Cuántas instancias de reglas se pueden generar?

Unificación de Reglas (Ejemplo 1)

Base de Reglas

```
(defrule R1
    (list $? ?x $?)
    (test (evenp ?x))
=>
    (assert (number even ?x))
    (printout t ?x " is an even number of the list " crlf))


(defrule R2
    (list $? ?x $?)
    (test (oddp ?x))
=>
    (assert (number odd ?x))
    (printout t ?x " is an odd number of the list " crlf))
```

```
(deffacts data
    (list 26 2 17 18))
```

→

Base de Hechos

f-1: (list 26 2 17 18)

Instancias de reglas

Rule	# Match	Facts
R1	1 2 3	f-1: {?x=18} f-1: {?x=2} f-1: {?x=26}
R2	1	f-1: {?x=17}

Se pueden generar cuatro instancias de reglas:

- 3 instancias de la regla R1
- 1 instancia de la regla R2

Unificación de Reglas (Ejemplo 2)

Base de Reglas

```
(defrule R1
    (number 2)
=>
 ...

(defrule R2
    (number ?num)
=>
 ...

(defrule R3
    (number ?n1)
    (number ?n2)
=>
 ...

(defrule R4
    (list $? ?n1 $? ?n2 $?)
    (test (= (mod ?n1 2) 0))
    (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

→

Base de Hechos

```
f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)
```

Rule	# Match	Facts
R1		

Unificación de Reglas (Ejemplo 2)

Base de Reglas

```
(defrule R1
    (number 2)
=>
 ...

(defrule R2
    (number ?num)
=>
 ...

(defrule R3
    (number ?n1)
    (number ?n2)
=>
 ...

(defrule R4
    (list $? ?n1 $? ?n2 $?)
    (test (= (mod ?n1 2) 0))
    (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

→

Base de Hechos

f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)

Rule	# Match	Facts
R1	1	f-1

Unificación de Reglas (Ejemplo 2)

Base de Reglas

```
(defrule R1
    (number 2)
=>
 ...

(defrule R2
    (number ?num)
=>
 ...

(defrule R3
    (number ?n1)
    (number ?n2)
=>
 ...

(defrule R4
    (list $? ?n1 $? ?n2 $?)
    (test (= (mod ?n1 2) 0))
    (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

→

Base de Hechos

f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)

Rule	# Match	Facts
R1	1	f-1
R2	1 2 3	f-1: {?num=2} f-2: {?num=5} f-3: {?num=17}

Unificación de Reglas (Ejemplo 2)

Base de Reglas

```
(defrule R1
    (number 2)
=>
 ...

(defrule R2
    (number ?num)
=>
 ...

(defrule R3
    (number ?n1)
    (number ?n2)
=>
 ...

(defrule R4
    (list $? ?n1 $? ?n2 $?)
    (test (= (mod ?n1 2) 0))
    (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

→

Base de Hechos

f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)

Rule	# Match	Facts
R1	1	f-1
R2	1 2 3	f-1: {?num=2} f-2: {?num=5} f-3: {?num=17}
R3	1 2 3 4 5 6 7 8 9	f-1, f-1: {?n1=2, ?n2=2} f-1, f-2: {?n1=2, ?n2=5} f-1, f-3: {?n1=2, ?n2=17} f-2, f-1: {?n1=5, ?n2=2} f-2, f-2: {?n1=5, ?n2=5} f-2, f-3: {?n1=5, ?n2=17} f-3, f-1: {?n1=17, ?n2=2} f-3, f-2: {?n1=17, ?n2=5} f-3, f-3: {?n1=17, ?n2=17}

Unificación de Reglas (Ejemplo 2)

Base de Reglas

```
(defrule R1
    (number 2)
=>
 ...

(defrule R2
    (number ?num)
=>
 ...

(defrule R3
    (number ?n1)
    (number ?n2)
=>
 ...

(defrule R4
    (list $? ?n1 $? ?n2 $?)
    (test (= (mod ?n1 2) 0))
    (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

→

Base de Hechos

```
f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)
```

Rule	# Match	Facts
R1	1	f-1
R2	1 2 3	f-1: {?num=2} f-2: {?num=5} f-3: {?num=17}
R3	1 2 3 4 5 6 7 8 9	f-1, f-1: {?n1=2, ?n2=2} f-1, f-2: {?n1=2, ?n2=5} f-1, f-3: {?n1=2, ?n2=17} f-2, f-1: {?n1=5, ?n2=2} f-2, f-2: {?n1=5, ?n2=5} f-2, f-3: {?n1=5, ?n2=17} f-3, f-1: {?n1=17, ?n2=2} f-3, f-2: {?n1=17, ?n2=5} f-3, f-3: {?n1=17, ?n2=17}
R4	1 2	f-4: {?n1=2, ?n2=17} f-4: {?n1=26,?n2=17}

Unificación de Reglas (Ejemplo 2)

Base de Reglas

```
(defrule R1
    (number 2)
=>
 ...

(defrule R2
    (number ?num)
=>
 ...

(defrule R3
    (number ?n1)
    (number ?n2)
=>
 ...

(defrule R4
    (list $? ?n1 $? ?n2 $?)
    (test (= (mod ?n1 2) 0))
    (test (<> (mod ?n2 2) 0))
=>
 ...

(defrule R5
    (number ?n1)
    (list $?list)
    (test (member ?n1 $?list))
=>
 ...
```

```
(deffacts data
    (number 2)
    (number 5)
    (number 17)
    (list 26 2 17 18))
```

→

Base de Hechos

```
f-1: (number 2)
f-2: (number 5)
f-3: (number 17)
f-4: (list 26 2 17 18)
```

Rule	# Match	Facts
R1	1	f-1
R2	1 2 3	f-1: {?num=2} f-2: {?num=5} f-3: {?num=17}
R3	1 2 3 4 5 6 7 8 9	f-1, f-1: {?n1=2, ?n2=2} f-1, f-2: {?n1=2, ?n2=5} f-1, f-3: {?n1=2, ?n2=17} f-2, f-1: {?n1=5, ?n2=2} f-2, f-2: {?n1=5, ?n2=5} f-2, f-3: {?n1=5, ?n2=17} f-3, f-1: {?n1=17, ?n2=2} f-3, f-2: {?n1=17, ?n2=5} f-3, f-3: {?n1=17, ?n2=17}
R4	1 2	f-4: {?n1=2, ?n2=17} f-4: {?n1=26,?n2=17}
R5	1 2	f-1,f-4: {?n1=2, $?list=(26 2 17 18)} f-3,f-4: {?n1=17, $?list=(26 2 17 18)}

Portfolio

Ejercicio 4 del boletín de ejercicios

Subir a espacio compartido

Nombre del fichero:
Tema1_ejercicio4

Tema 1: Sistemas Basados en Reglas

1.3 Estrategias y Resolución de Conflictos

Grado en Tecnologías Interactivas

Lidia Contreras Ochando

VRAIN
Valencian Research Institute for Artificial Intelligence

DSIIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Refracción

- Una regla solo se puede instanciar una vez con los mismos hechos (hechos con los mismos índices) y mismas instanciaciones de variables.

- CLIPS aplica **refracción**: no permite que una regla se dispare dos veces con los mismos datos, evitando así bucles infinitos.

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7)}
```

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7) }
```

Matching

R1: f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7) }
```

Matching

R1: ~~f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)}
```

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7) }
```

Matching

R1: ~~f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}~~

Selección **Ejecución** →

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)}
```

Matching

R1: f-2 = {$?x=(), ?y=12, ?z=24, $?w=(5 7)}

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7) }
```

Matching

R1: ~~f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)}
```

Matching

R1: ~~f-2 = {$?x=(), ?y=12, ?z=24, $?w=(5 7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)
      f-3: (lista 24 12 5 7)}
```

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7) }
```

Matching

R1: ~~f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)}
```

Matching

R1: ~~f-2 = {$?x=(), ?y=12, ?z=24, $?w=(5 7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)
      f-3: (lista 24 12 5 7)}
```

Matching

R1: f-3 = {$?x=(), ?y=24, ?z=12, $?w=(5 7)}

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

```
BH= { f-1: (lista 12 5 24 7) }
```

Matching

R1: ~~f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)}
```

Matching

R1: ~~f-2 = {$?x=(), ?y=12, ?z=24, $?w=(5 7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)
      f-3: (lista 24 12 5 7)}
```

Matching

R1: ~~f-3 = {$?x=(), ?y=24, ?z=12, $?w=(5 7)}~~ → **Selección** **Ejecución**

```
BH= { f-1: (lista 12 5 24 7)
      f-2: (lista 12 24 5 7)
      f-3: (lista 24 12 5 7)}
```

Refracción

```
(defrule R1
        (lista $?x ?y ?z $?w)
        (test (evenp ?z))
    =>
        (assert (lista $?x ?z ?y $?w)))
```

BH= { f-1: (lista 12 5 24 7) }

Matching

R1: ~~f-1 = {$?x=(12), ?y=5, ?z=24, $?w=(7)}~~ → **Selección** **Ejecución**

BH= { f-1: (lista 12 5 24 7)
f-2: (lista 12 24 5 7)}

Matching

R1: ~~f-2 = {$?x=(), ?y=12, ?z=24, $?w=(5 7)}~~ → **Selección** **Ejecución**

BH= { f-1: (lista 12 5 24 7)
f-2: (lista 12 24 5 7)
f-3: (lista 24 12 5 7)}

Matching

R1: ~~f-3 = {$?x=(), ?y=24, ?z=12, $?w=(5 7)}~~ → **Selección** **Ejecución**

BH= { f-1: (lista 12 5 24 7)
f-2: (lista 12 24 5 7)
f-3: (lista 24 12 5 7)}

CLIPS no vuelve a insertar el hecho (lista 12 24 5 7)

Resolución de conflictos

Cuando hay más de una instancia en la Agenda, la estrategia de resolución de conflictos decide cuál activar.

- **Estrategia prioridades explícitas:** Valor numérico “salience” en las reglas. Cuanto mayor sea el valor, más alta es la prioridad de la regla. Si no se especifica salience en la regla, su prioridad es 0.
- **Estrategia profundidad:** Las nuevas activaciones se manejan como una cola de prioridades. Los empates se resuelven mediante una estrategia LIFO (más prioridad para las instancias más recientes).
- **Estrategia anchura:** Las nuevas activaciones se manejan como una cola de prioridades. Los empates se resuelven mediante una estrategia FIFO (más prioridad para las instancias más antiguas).
- **Estrategia aleatoria:** A cada activación se le asigna un número aleatorio entre aquellas que tienen el mismo salience.
- **Estrategia de la regla más específica:** Se selecciona la regla que contiene más restricciones. Cada comparación con una constante o con una variable ligada, añade un punto de especificidad. Las llamadas a una función dentro de un test también.

Resolución de conflictos: Estrategia de prioridades explícitas

Ejemplo: Queremos seleccionar 5 personas para jugar al baloncesto, preferiblemente con una altura mayor de 2m; sino, escoger gente por debajo de los 2m.

```
(defrule over-2m
   (declare (salience 30))
   ?f1 <-  (height ?per ?tall)
   (test (>= ?tall 2))
   ?f2 <- (count ?numper)
=>
   (retract ?f1 ?f2)
   (assert (count (+ ?numper 1))))
```

```
(defrule below-2m
   (declare (salience 10))
   ?f1<-  (height ?per ?tall)
   ?f2 <-  (count ?numper)
=>
   (retract ?f1 ?f2)
   (assert (count (+ ?numper 1))))
```

```
(defrule final-1
   (declare  (salience 100))
   (count 5)
=>
   (halt)
   (printout t “We already have 5 people to play basket “ crlf))
```

```
(defrule final-2
=>
   (halt)
   (printout t “We were not able to find 5 people to play basket “, crlf))
```

```
(deffacts data
   (height John 2.02) (height Peter 1.92) (height Terry 1.86)
   (height Lucas 2.01)(height Nick 2.05)(height Joshua 1.94)
   (count 0))
```

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1} R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1} ← R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}
f2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}
f2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1~~: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}~~
f2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}
f3: (list 2 3 7 5)	R1: f-3 {$?x=(2 3),?y=7, ?z=5, $?w=(), ?f=3}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1~~: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}~~
f2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2} ←
f3: (list 2 3 7 5)	R1: f-3 {$?x=(2 3),?y=7, ?z=5, $?w=(), ?f=3}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1~~: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1~~}
f2: (list 3 2 5 7)	R1~~: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2~~}
f3: (list 2 3 7 5)	R1: f-3 {$?x=(2 3),?y=7, ?z=5, $?w=(), ?f=3}
f4: (list 2 3 5 7)	

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1~~: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}~~
f2: (list 3 2 5 7)	R1~~: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}~~
f3: (list 2 3 7 5)	R1: f-3 {$?x=(2 3),?y=7, ?z=5, $?w=(), ?f=3} ←
f4: (list 2 3 5 7)	

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1~~: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}~~ R1~~: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1~~}
f2: (list 3 2 5 7)	R1~~: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2~~}
f3: (list 2 3 7 5)	~~R1: f-3 {$?x=(2 3),?y=7, ?z=5, $?w=(), ?f=~~ 3}
f4: (list 2 3 5 7)	

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (retract ?f)
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (retract ?f)
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1: (list 3 2 7 5)	R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1} R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (retract ?f)
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1~~: (list 3 2 7 5)~~	~~R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=~~1} ~~R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=~~1}
f2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (retract ?f)
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1~~: (list 3 2 7 5)~~	~~R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=~~1} ~~R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=~~1}
f2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2} ←

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales.

```
BH={(list 3 2 7 5)}
```

```
(defrule R1
   ?f <- (list $?x ?y ?z $?w)
   (test (< ?z ?y))
=>
   (retract ?f)
   (assert (list $?x ?z ?y $?w)))
```

BH	Conjunto conflicto / Agenda
f1~~: (list 3 2 7 5)~~	~~R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=~~1} ~~R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=~~1}
f2~~: (list 3 2 5 7)~~	R~~1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=~~2}
f3: (list 2 3 5 7)	

Portfolio

Ejercicio 5 del boletín de ejercicios

Subir a espacio compartido

Nombre del fichero:
Tema1_ejercicio5

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l=1} ←

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t "List is " $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l=1}~~
f-3: (list c a b d e) f-4: (change-list yes)	move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l=3} ← print: f-4,f-1 {$?list=(a b c d e), ?ch=4} print: f-4,f-3 {$?list=(c a b d e), ?ch=4}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}

(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	mo~~ve: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l~~=1}
f-3: (list c a b d e) f-4: (change-list yes)	mo~~ve: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l~~=3} print: f-4,f-1 {$?list=(a b c d e), ?ch=4} print: f-4,f-3 {$?list=(c a b d e), ?ch=4}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}

(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l~~=1}
f-3: (list c a b d e) f-4: (change-list yes)	~~move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l~~=3} print: f-4,f-1 {$?list=(a b c d e), ?ch=4} print: f-4,f-3 {$?list=(c a b d e), ?ch=4}

Resolución de conflictos: Estrategia en Anchura

Ejemplo: Asumiendo que la estrategia de la Agenda es ANCHURA, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	mo~~ve: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l~~=1}
f-3: (list c a b d e) ~~f-4: (change-list yes)~~	mo~~ve: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l~~=3} pri~~nt: f-4,f-1 {$?list=(a b c d e), ?ch=4}~~ pri~~nt: f-4,f-3 {$?list=(c a b d e), ?ch=4}~~

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l=1}

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t "List is " $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2, ?l=1} ←

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2,~~ ?l=1}
f-3: (list c a b d e) f-4: (change-list yes)	move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l=3} print: f-4,f-1 {$?list=(a b c d e), ?ch=4} print: f-4,f-3 {$?list=(c a b d e), ?ch=4}

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}

(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2,~~ ?l=1}
f-3: (list c a b d e) f-4: (change-list yes)	move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l=3} print: f-4,f-1 {$?list=(a b c d e), ?ch=4} print: f-4,f-3 {$?list=(c a b d e), ?ch=4} ←

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2,~~ ?l=1}
f-3: (list c a b d e) f-~~4: (change-list yes~~)	move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l=3} ~~print: f-4,f-1 {$?list=(a b c d e), ?ch=4}~~ ~~print: f-4,f-3 {$?list=(c a b d e), ?ch=4}~~

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2,~~ ?l=1}
f-3: (list c a b d e) f-~~4: (change-list yes)~~)	move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2, ?l=3} ← ~~print: f-4,f-1 {$?list=(a b c d e), ?ch=4}~~ ~~print: f-4,f-3 {$?list=(c a b d e), ?ch=4}~~

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2,~~ ?l=1}
f-3: (list c a b d e) f~~-4: (change-list yes)~~)	~~move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2~~, ?l=3} ~~print: f-4,f-1 {$?list=(a b c d e), ?ch=4}~~ ~~print: f-4,f-3 {$?list=(c a b d e), ?ch=4}~~
f-5: (change-list yes)	print: f-5,f-1 {$?list=(a b c d e), ?ch=5} print: f-5,f-3 {$?list=(c a b d e), ?ch=5}

Resolución de conflictos: Estrategia en Profundidad

Ejemplo: Asumiendo que la estrategia de la Agenda es PROFUNDIDAD, realiza una traza del siguiente SBR y muestra el contenido final de la BH.

```
BH={(list a b c d e) (front c)}
```

```
(defrule move
   ?m<- (front ?who)
   ?l <- (list $?front ?who $?rear)

=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
   ?ch <- (change-list yes)
   (list $?list)
=>
   (retract ?ch)
   (printout t “List is “ $?list crlf))
```

BH	Conjunto conflicto / Agenda
f-1: (list a b c d e) f-2: (front c)	~~move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e), ?m=2,~~ ?l=1}
f-3: (list c a b d e) f~~-4: (change-list yes~~)	~~move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e), ?m=2~~, ?l=3} ~~print: f-4,f-1 {$?list=(a b c d e), ?ch=4}~~ ~~print: f-4,f-3 {$?list=(c a b d e), ?ch=4}~~
f~~-5: (change-list yes~~)	~~print: f-5,f-1 {$?list=(a b c d e), ?ch=5}~~ ~~print: f-5,f-3 {$?list=(c a b d e), ?ch=5}~~

Tema 1: Sistemas Basados en Reglas

1.4 Diseño de problemas mediante SBR

Grado en Tecnologías Interactivas

Lidia Contreras Ochando

VRAIN
Valencian Research Institute for Artificial Intelligence

DSIIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Diseño de problemas mediante SBR

- Nos permite definir las reglas y hechos necesarios para solucionar un problema determinado.
- Podemos construir sistemas capaces de tomar decisiones y realizar acciones en función de las condiciones que se presenten en cada momento.
- ¿Qué debemos tener en cuenta?
  1. Comprensión clara del problema que se quiere resolver antes de empezar a diseñar las reglas y hechos necesarios.
  2. Definir claramente el objetivo del problema, ya que esto permitirá enfocar adecuadamente el proceso de diseño.
  3. Comprensión profunda de los datos disponibles, ya que esto permitirá identificar patrones y relaciones relevantes para la solución del problema.
  4. Considerar las posibles limitaciones y restricciones que puedan existir en la solución del problema.

Diseño de problemas mediante SBR: Ejemplo

- Queremos diseñar un sistema basado en reglas para detectar fraudes en transacciones bancarias
  - La identificación del problema implica definir claramente el objetivo: detectar transacciones sospechosas de fraude.
  - Datos disponibles: el historial de transacciones, los patrones de gasto del cliente y los datos personales del cliente.
  - Con esta información, podemos definir las reglas y hechos necesarios para identificar transacciones sospechosas y tomar medidas para prevenir el fraude.

Diseño de problemas mediante SBR: Diseñando los hechos

- Existen dos tipos de hechos:
  - Hechos estáticos: información que no cambia durante el proceso.
    - Ejemplo: Información sobre los clientes (nombre, dirección, edad, historial de crédito, etc), información sobre el tipo de tarjeta de crédito utilizada (Visa, Mastercard, etc.).
  - Hechos dinámicos: información que cambia a lo largo de la ejecución del sistema.
    - Ejemplo: Información sobre las transacciones en tiempo real, como la cantidad de dinero gastado en una transacción, la ubicación geográfica de la transacción, etc.

- Definir los hechos de una manera que facilite la inferencia:
  - Notación clara y coherente
  - Debe representarse solo la información relevante para llegar al objetivo.
  - Organizarlos de una manera que refleje su relación con el problema.
    - Por ejemplo, los hechos pueden ser organizados en categorías, como "información del cliente“ o "información de la transacción“.

Diseño de problemas mediante SBR: Diseñando las reglas

- Las reglas deben ser claras, precisas y no contradictorias
  - Deben poder representar cada posible cambio de estado en el sistema.

- No se debe utilizar **retract** en el diseño de las reglas
  - Dificulta la traza de todos los estados y la capacidad de volver atrás si un camino no proporciona la solución.

- Las reglas deben estar diseñadas de tal manera que se pueda comprobar si se ha alcanzado el objetivo final
- No debería haber reglas que no se puedan utilizar o sean redundantes.

Diseño de problemas mediante SBR: ¿retract?

W		B	B	W

estado inicial

B	B		W	W

estado final

Diseño de problemas mediante SBR: ¿retract?

W		B	B	W

estado inicial

B	B		W	W

estado final

Posibles movimientos:

1. Mover vacío (Empty) 1 a la izquierda
2. Mover vacío (Empty) 1 a la derecha
3. Mover vacío (Empty) 2 a la izquierda
4. Mover vacío (Empty) 2 a la derecha

Diseño de problemas mediante SBR: ¿retract?

W		B	B	W

estado inicial

B	B		W	W

estado final

1. Utilizamos varios hechos para representar los estados del problema.
2. Las reglas tienen un comando retract en la BR

Base de Hechos

```
f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
```

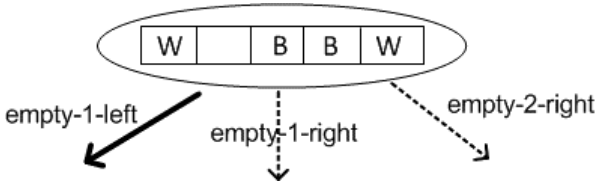
Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
     (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```

Diseño de problemas mediante SBR: ¿retract?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
     (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



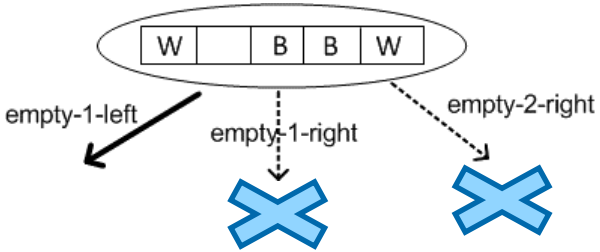
Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

Diseño de problemas mediante SBR: ¿retract?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
     (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

el hecho f-2: (cell 2 E) se borra y como consecuencia se eliminan las instancias de reglas de la Agenda

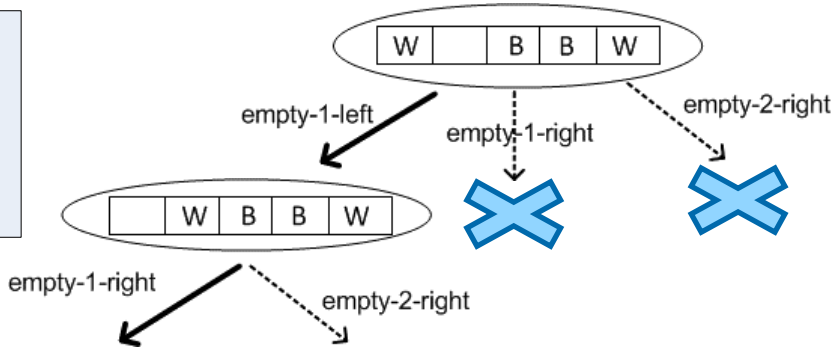
Base de Hechos

f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-6: (cell 1 E)
f-7: (cell 2 W)

Diseño de problemas mediante SBR: ¿retract?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
     (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

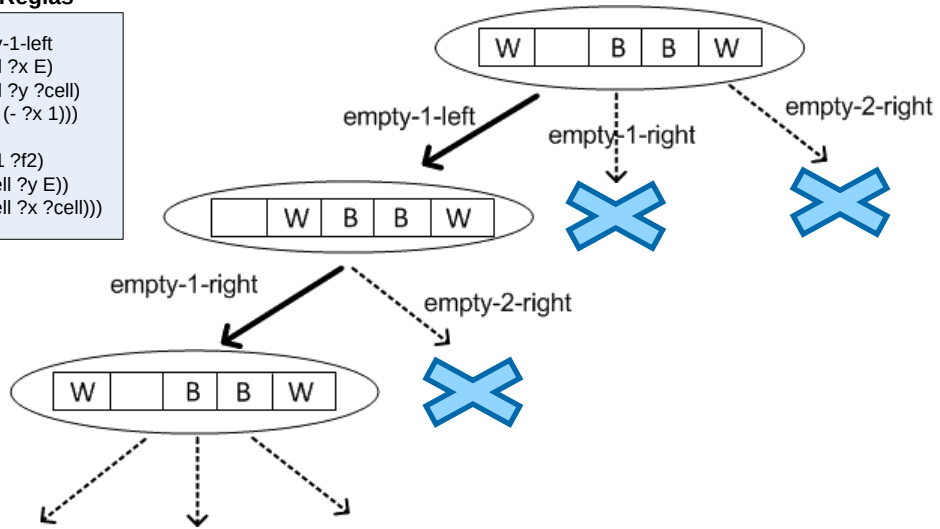
Base de Hechos

f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-6: (cell 1 E)
f-7: (cell 2 W)

Diseño de problemas mediante SBR: ¿retract?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
     (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

Base de Hechos

f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-6: (cell 1 E)
f-7: (cell 2 W)

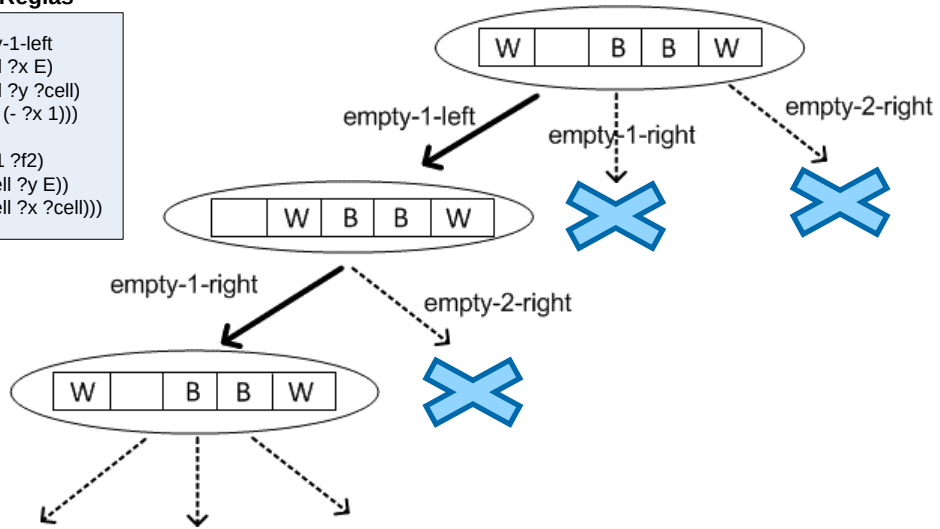
Base de Hechos

f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-8: (cell 2 E)
f-9: (cell 1 W)

Diseño de problemas mediante SBR: ¿retract?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
     (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

Base de Hechos

f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-6: (cell 1 E)
f-7: (cell 2 W)

Base de Hechos

f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-8: (cell 2 E)
f-9: (cell 1 W)

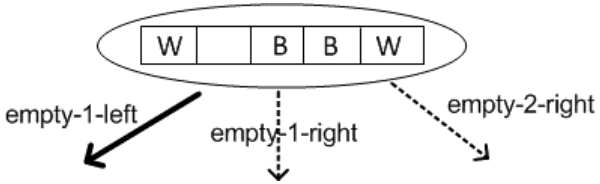
OBJETIVO: generar el árbol de búsqueda del problema hasta encontrar un nodo que coincida con el estado final del problema ➔ las reglas no deben eliminar hechos porque impiden la generación de estados del problema ➔ NO UTILIZAR COMANDO **retract**

Diseño de problemas mediante SBR: ¿Estados en diferentes hechos?

Eliminamos el **retract** de las reglas:

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
    (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



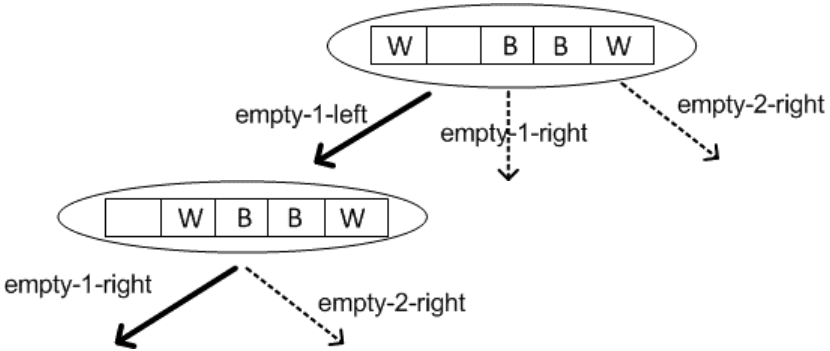
Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

Diseño de problemas mediante SBR: ¿Estados en diferentes hechos?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
    (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

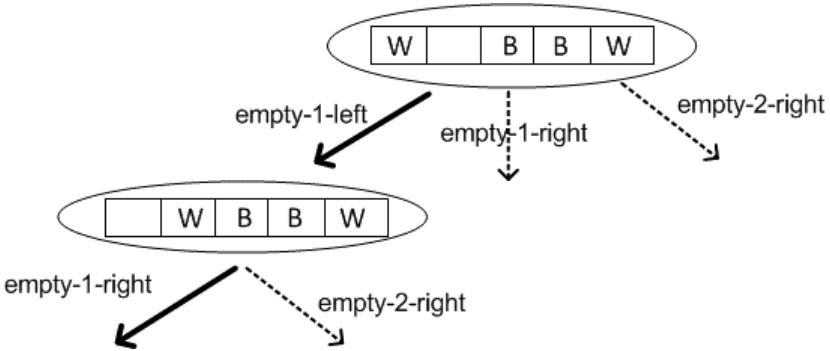
Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-6: (cell 1 E)
f-7: (cell 2 W)

Diseño de problemas mediante SBR: ¿Estados en diferentes hechos?

Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (cell ?x E)
     ?f2 <- (cell ?y ?cell)
     (test (= ?y (- ?x 1)))
  =>
    (retract ?f1 ?f2)
     (assert (cell ?y E))
     (assert (cell ?x ?cell)))
```



Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)

Base de Hechos

f-1: (cell 1 W)
f-2: (cell 2 E)
f-3: (cell 3 B)
f-4: (cell 4 B)
f-5: (cell 5 W)
f-6: (cell 1 E)
f-7: (cell 2 W)

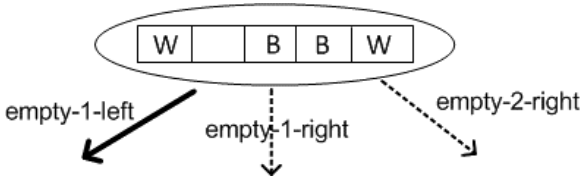
Estado inconsistente

La BH mantiene todos los hechos que insertan las reglas pero no es posible identificar los hechos que representan un estado u otro.

Solución: representar los estados del problema con un único hecho

Diseño de problemas mediante SBR: Resumen

Representación del puzzle en un solo hecho y sin retract en las reglas.



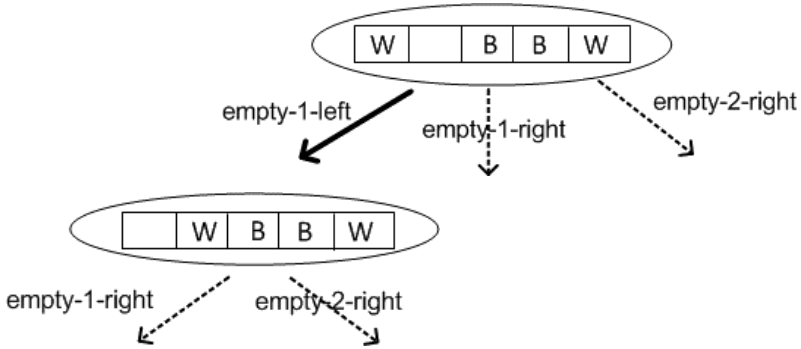
Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (puzzle $?x ?y E $?z)
  =>
    (assert (puzzle $?x E ?y $?z)))
```

Base de Hechos

f-1: (puzzle W E B B W)

Diseño de problemas mediante SBR: Resumen



Base de Reglas

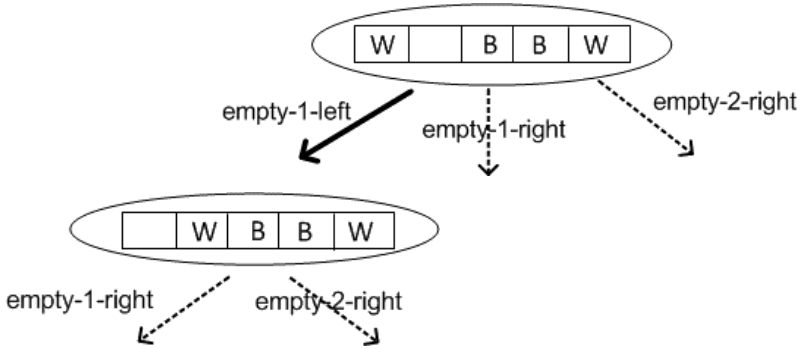
```
(defrule emtpy-1-left
    ?f1 <-  (puzzle $?x ?y E $?z)
  =>
    (assert (puzzle $?x E ?y $?z)))
```

Base de Hechos

f-1: (puzzle W E B B W)

f-2: (puzzle E W B B W)

Diseño de problemas mediante SBR: Resumen



Base de Reglas

```
(defrule emtpy-1-left
    ?f1 <-  (puzzle $?x ?y E $?z)
   =>
     (assert (puzzle $?x E ?y $?z)))
```

Base de Hechos

f-1: (puzzle W E B B W)

f-2: (puzzle E W B B W)

Conclusión:

- Usar un único hecho para la información dinámica.
- Se pueden utilizar tantos hechos como se desee para representar la información estática
- No usar retract para mantener todos los estados por los que hemos pasado
- Una estrategía en anchura debería encontrar la solución (si existe).

Portfolio

Ejercicio 11 del boletín de ejercicios

Subir a espacio compartido

Nombre del fichero:
Tema1_ejercicio11