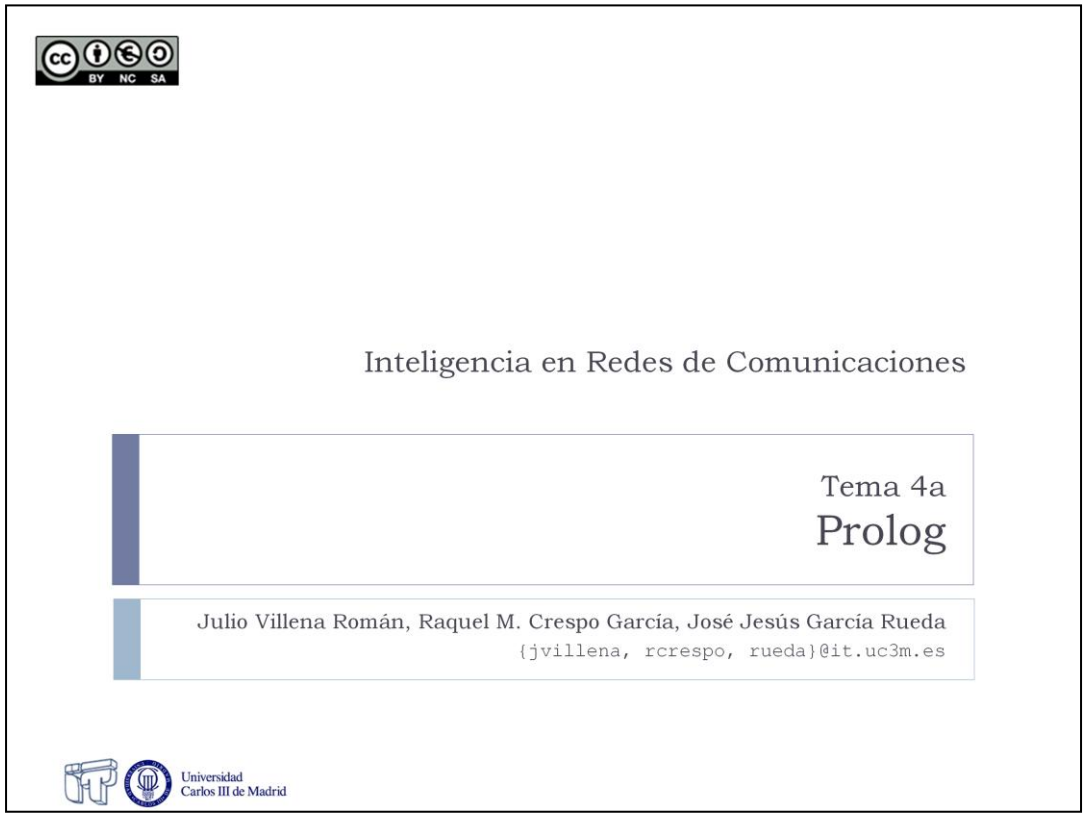
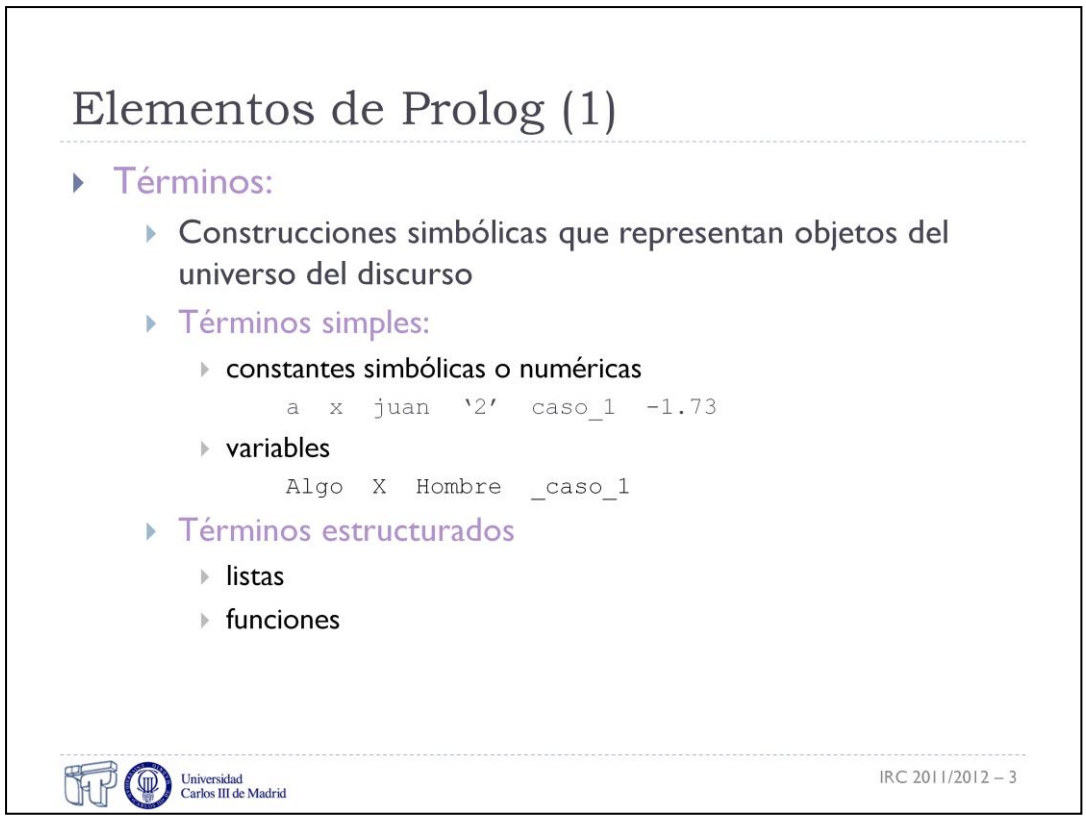


En este Tema 4a se van a estudiar los fundamentos de Prolog, el lenguaje de programación lógica más extendido.

Prolog

- La lógica proporciona:
  - un lenguaje base para la representación del conocimiento
  - modelos para el razonamiento
- Prolog es:
  - una sintaxis para la construcción de sistemas expertos
  - una herramienta basada en el lenguaje de la lógica, con muchas versiones:
    - Prolog-10 (1975-79), Prolog II (1981) …
    - SICStus, GNU Prolog, SWI Prolog, Ciao Prolog

Prolog es un lenguaje de programación para representar conocimiento y realizar razonamientos sobre él basado en el lenguaje de la lógica de predicados de primer orden.



La sintaxis de Prolog se compone de diferentes elementos. Los más básicos son los “términos”, que representan objetos del universo del discurso, y pueden ser términos simples (constantes simbólicas o numéricas, que empiezan por LETRA MINÚSCULA, o variables, que son las que empiezan por LETRA MAYÚSCULA), y términos estructurados (listas, funciones, etc.).

Elementos de Prolog (2)

- Predicados:
  - Sintaxis: nombre (termino1, termino2…)
  - Semántica: representan propiedades de objetos representados por los términos, o relaciones entre ellos
  - "Verdadero" si se cumple la relación

```
rojo(X)   padre(juan,X)   not contiene(Maquina,pieza)
```

- Literales:
  - <predicado> o not <predicado>
- Hechos:
  - Predicados sin variables

```
rojo(rosa)   padre(juan,luis)   contiene(pc1,cpu-i586)
```

Otros elementos de Prolog son los predicados, que como se trata de un lenguaje basado en la lógica de predicados de primer orden, representan propiepades de objetos o relaciones entre objetos. Los predicados sin variables son los hechos.

Elementos de Prolog (3)

- Consultas:
  - Son respuestas del procesador de Prolog
  - Pueden valer "SI"/"NO" o el valor de la variable

```
?- padre(juan,luis).
YES
?- padre(X,luis).
X = juan
```


Las consultas son preguntas que se hacen al procesador de Prolog. Devuelven como valor SÍ/NO (verdadero/falso) o el valor concreto de la variable que hace cierta la consulta.

Elementos de Prolog (4)

- Reglas:
  - <predicado> :- <literal1>, <literal2> ...

```
hijo(X,Y) :- padre(Y,X), hombre(X).
hijo(X,Y) :- madre(Y,X), hombre(X).
hija(X,Y) :- padre(Y,X), mujer(X).
hija(X,Y) :- madre(Y,X), mujer(X).
abuelo(X,Y) :- padre(X,Z), padre(Z,Y), hombre(X).
abuela(X,Y) …

contiguo(X,Y) :- sobre(X,base),
                 sobre(Y,base),
                 not entre(X,Y,Z),
                 not(X=Y).
```

Las reglas en Prolog son de la forma “A :- B, C, D…” que significa “si B y C y D… entonces A”. La coma es el operador AND.

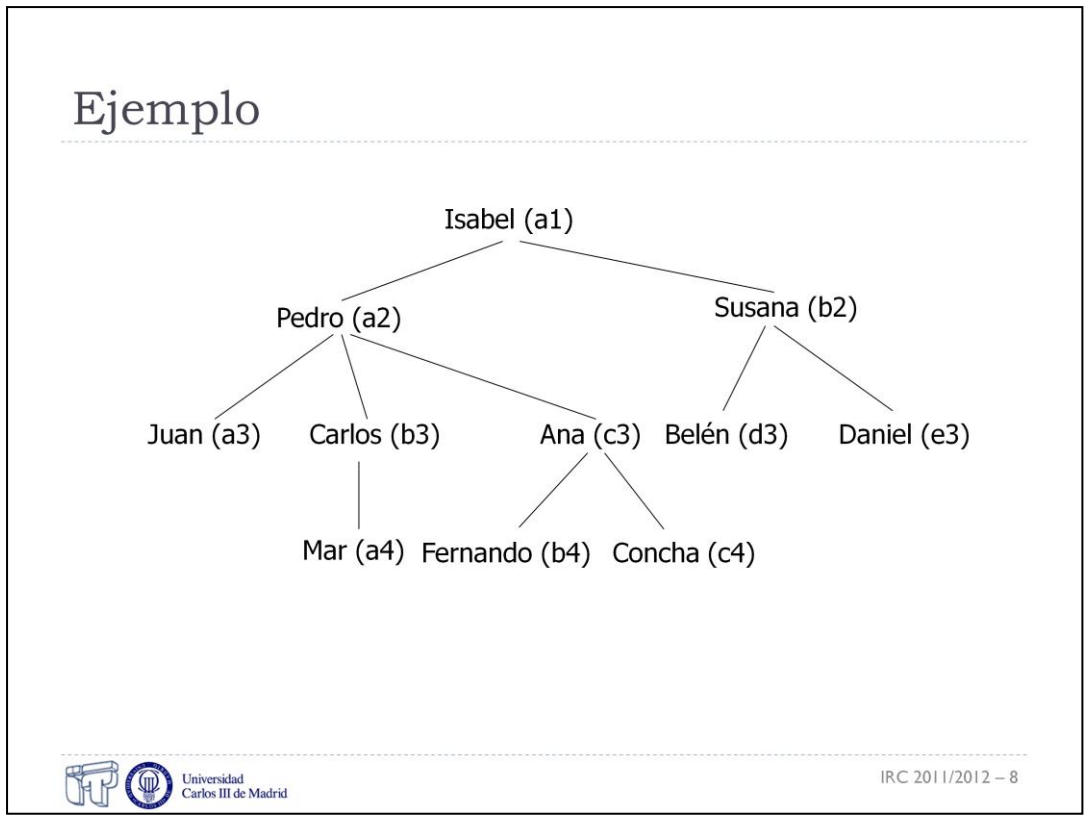
El operador OR se realiza escribiendo varias veces la regla: “A:- B”, “A:- C” que significa “si B o C entonces A”.

Elementos de Prolog (5)

- Programas:
  {hechos} +
  {reglas} +
  consulta

- {hechos} y {reglas} son cláusulas que forman la base de conocimientos
- Prolog es un lenguaje declarativo

Los programas en Prolog se componen de hechos más reglas más la consulta que inicia la ejecución del razonamiento.



Los ejemplos de código de las siguientes diapositivas estarán basados en esta red semántica, que representa la jerarquía de jefe-empleado en una empresa ficticia. Por ejemplo, Isabel ("a1" para simplificar) es jefa de Pedro ("a2") y Susana ("b2"), etc.

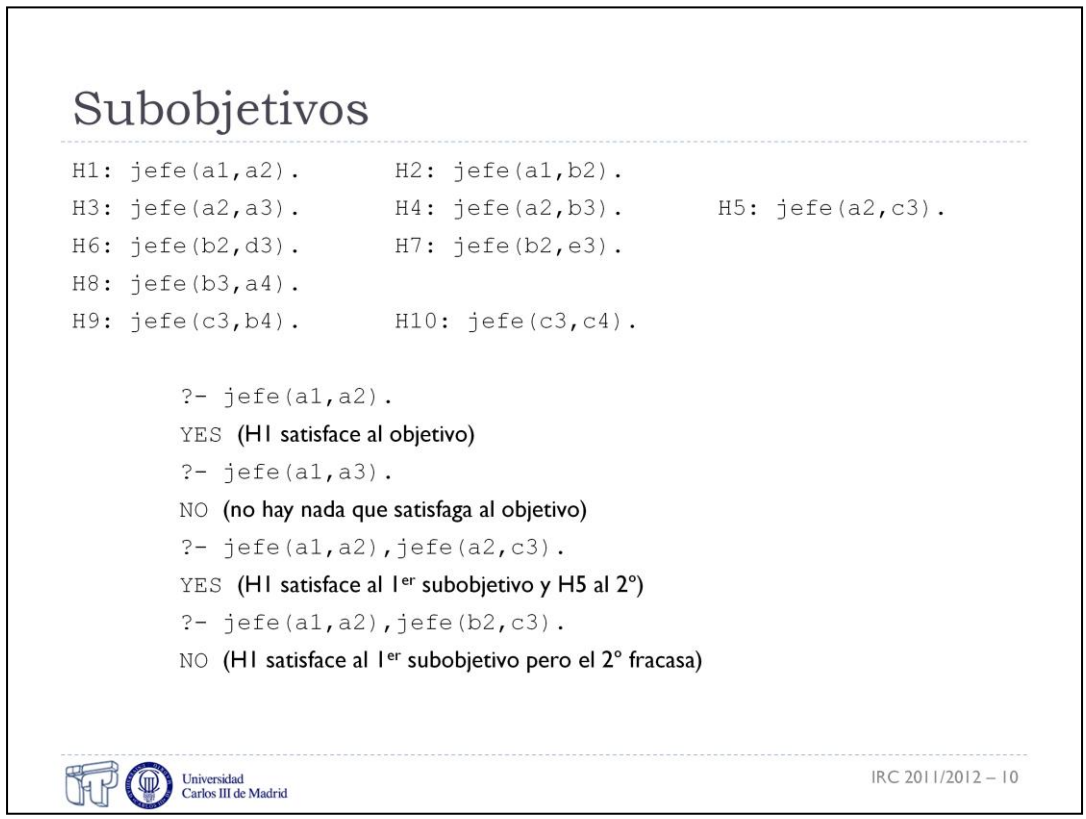
Consultas de confirmación/con variables

```
jefe(a1,a2).      jefe(a1,b2).
jefe(a2,a3).      jefe(a2,b3).      jefe(a2,c3).
jefe(b2,d3).      jefe(b2,e3).
jefe(b3,a4).
jefe(c3,b4).      jefe(c3,c4).
```

```
?- jefe(a1,a2).
YES
?- jefe(a1,a3).
NO
?- jefe(a1,a2), jefe(a2,c3).
YES
?- jefe(a1,a2), jefe(b2,b3).
NO
?- jefe(a1,X).
X=a2
X=b2
?- jefe(a1,X), jefe(X,c3).
X=a2
?- jefe(a1,X), jefe(X,Y), jefe(Y,Z).
X=a2;    Y=b3;    Z=a4
X=a2;    Y=c3;    Z=b4
X=a2;    Y=c3;    Z=c4
```

IRC 2011/2012 – 9

Lista de hechos que codifican el conocimiento almacenado en la red semántica. Ejemplo de varias consultas de confirmación (sí/no) y consultas con variables (¿quién es el jefe de quién?).

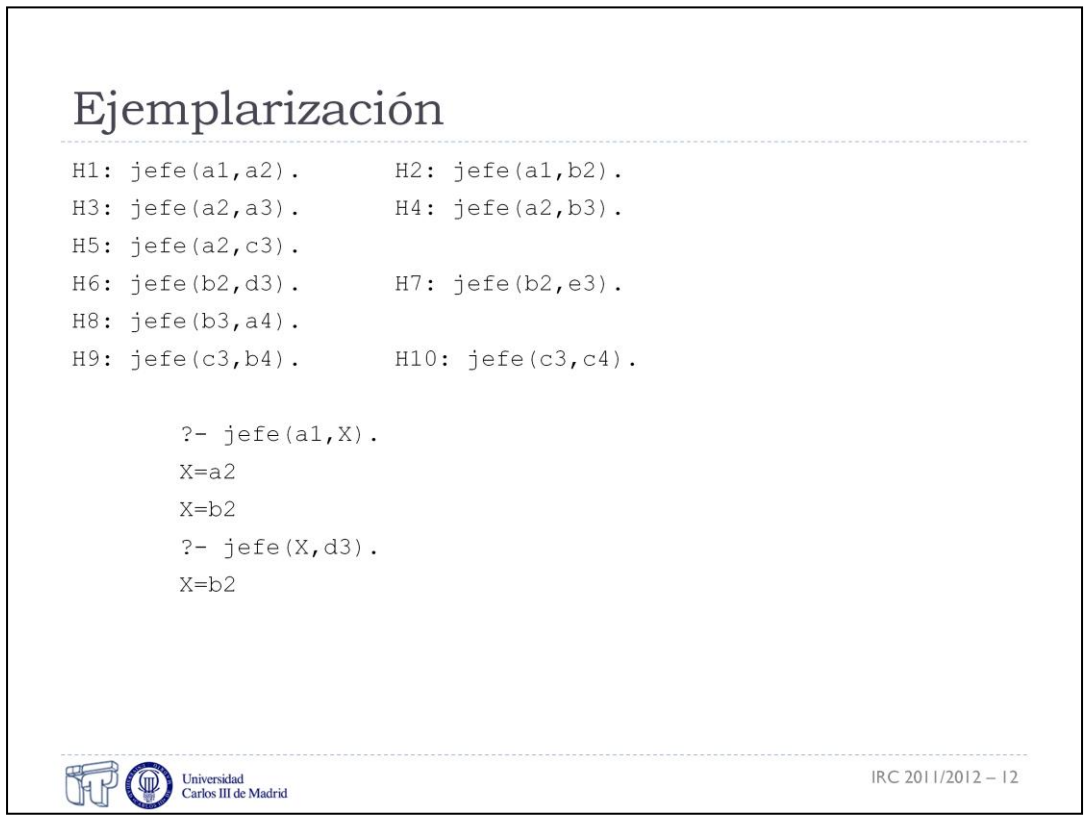


Para llevar a cabo los razonamientos, Prolog va comprobando la lista de subobjetivos, de izquierda a derecha, hasta llegar al final de la regla o devolver “false”.

Conceptos

- Unificación: asignación provisional de valores (constante o variables) a variables para emparejar dos predicados iguales
- Ejemplarización: unificación de un predicado cuyos argumentos son constantes (sustitución de variables por constantes)
- Retroceso: mecanismo de vuelta atrás cuando falla la unificación de un predicado (para unificar subobjetivos)

Conceptos básicos del modelo de razonamiento de Prolog: unificación, ejemplarización y retroceso. Para comprobar objetivos, Prolog va unificando variables ejemplarizando hechos, y si algún subjetivo es falso, ejecuta el mecanismo de retroceso para continuar buscando otro subobjetivo (si existe).

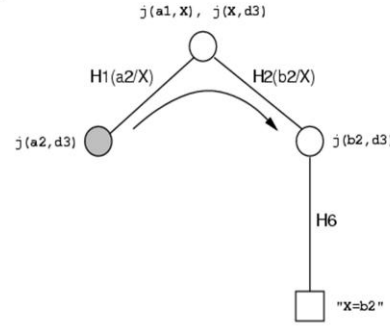


Ejemplo de ejemplarización: dar valores a variables para que hagan cierto el objetivo.

Retroceso

```
H1: jefe(a1,a2).      H2: jefe(a1,b2).
H3: jefe(a2,a3).      H4: jefe(a2,b3).
H5: jefe(a2,c3).
H6: jefe(b2,d3).      H7: jefe(b2,e3).
H8: jefe(b3,a4).
H9: jefe(c3,b4).      H10: jefe(c3,c4).
```

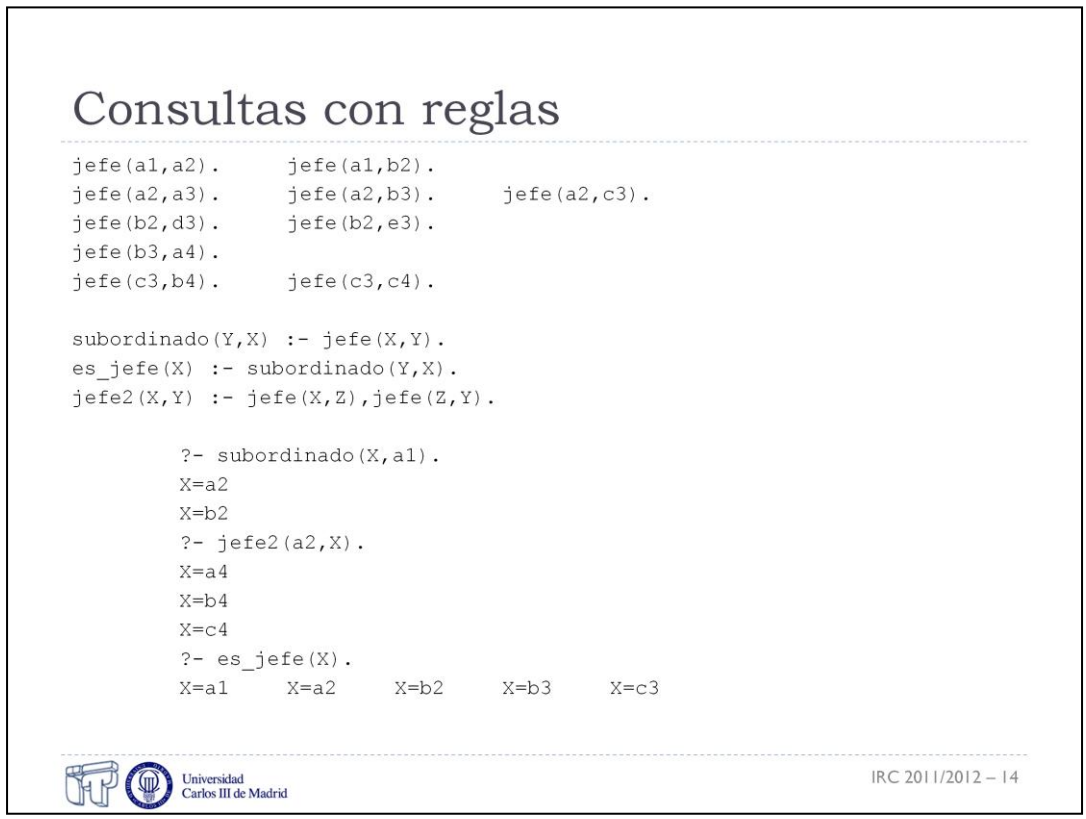
```
?- jefe(a1,X),jefe(X,d3).
X=b2
```



Universidad Carlos III de Madrid

IRC 2011/2012 – 13

Ilustración del mecanismo de retroceso. Después de unificar X como a2, llega a un punto en el que no es posible encontrar ningún hecho que haga cierto el razonamiento, con lo que vuelve atrás (mecanismo de retroceso) deshaciendo la unificación y probando otros posibles valores, en este caso, X=b2.



Ejemplo de reglas en la base de conocimiento: definición de subordinado de alguien, y jefe de nivel 2 (jefe del jefe).

Consultas con reglas recursivas

```
jefe(a1,a2).     jefe(a1,b2).
jefe(a2,a3).     jefe(a2,b3).     jefe(a2,c3).
jefe(b2,d3).     jefe(b2,e3).
jefe(b3,a4).
jefe(c3,b4).     jefe(c3,c4).

superior(X,Y) :- jefe(X,Y).
superior(X,Y) :- jefe(X,Z),superior(Z,Y).

        ?- subordinado(X,a4).
        X=b3
        X=a2
        X=a1
        ?- superior(a2,X).
        X=a3
        X=b3
        X=c3     X=a4     X=b4     X=c4
```

IRC 2011/2012 – 15

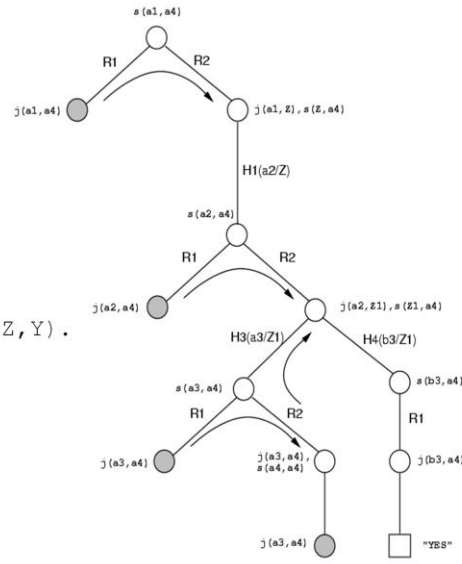
Ejemplo de reglas definidas de forma recursiva, para representar relaciones complejas: "X es superior de Y si X es jefe directo de Y o bien existe un Z tal que X es jefe de Z y Z es superior de Y".

Retroceso y recursión

```
H1: jefe(a1,a2).  H2: jefe(a1,b2).
H3: jefe(a2,a3).  H4: jefe(a2,b3).
H5: jefe(a2,c3).
H6: jefe(b2,d3).  H7: jefe(b2,e3).
H8: jefe(b3,a4).
H9: jefe(c3,b4).  H10: jefe(c3,c4).

superior(X,Y) :- jefe(X,Y).
superior(X,Y) :- jefe(X,Z),superior(Z,Y).

        ?- superior(a1,a4).
        YES
```



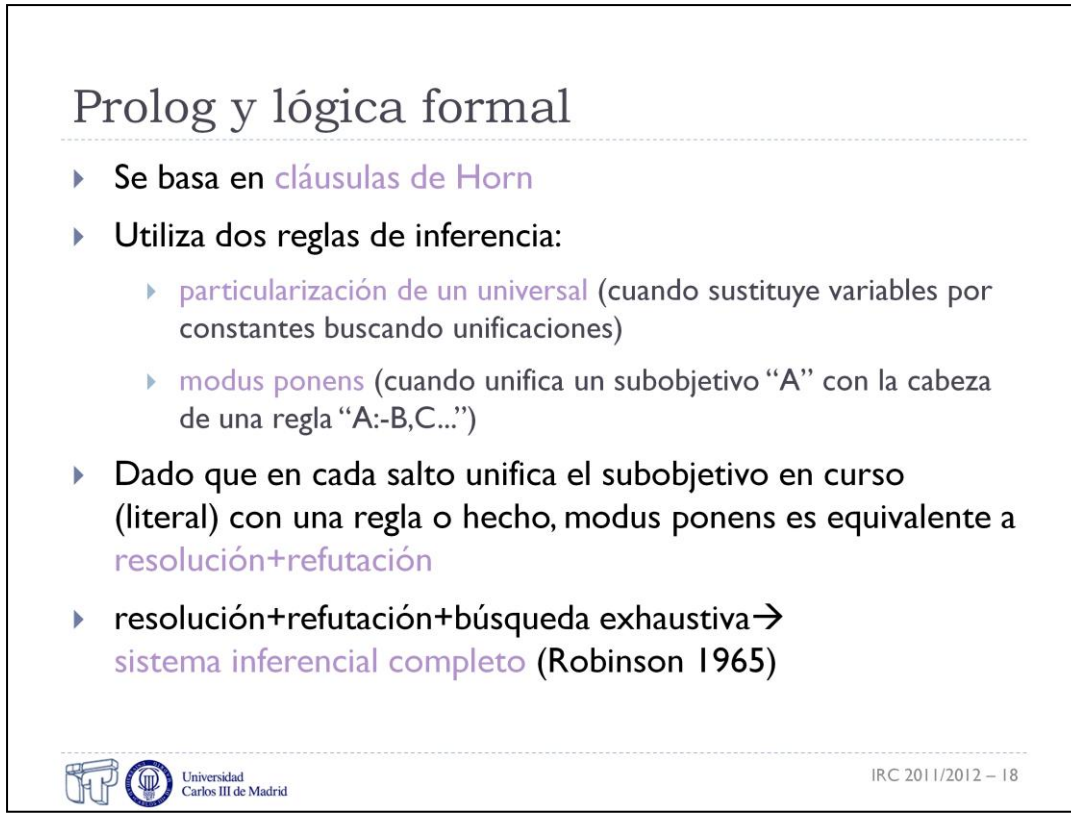
El mecanismo de retroceso se aplica de la misma forma, aunque lógicamente sea más complejo de seguir cuando hay reglas recursivas, comprobando además de cada subobjetivo (con AND), cada regla alternativa (con OR).

Estrategia de resolución y búsqueda

- Descomposición del problema (objetivo) en subproblemas:
  - el procesador de objetivos genera subproblemas
    - explora cada subobjetivo (izq→der) llamando al procesador de reglas; si falla, retrocede
  - el procesador de reglas genera caminos alternativos
    - explora las cláusulas (arr→aba) buscando unificaciones
- No tiene más remedio que ser una búsqueda en profundidad y exhaustiva
  - → ¡puede generar bucles infinitos!
- Es una búsqueda sin información del dominio

IRC 2011/2012 – 17

Como se ilustra por los anteriores ejemplos, para llevar a cabo los razonamientos, Prolog descompone el problema en subobjetivos, siguiendo una estrategia de búsqueda en profundidad y exhaustiva.

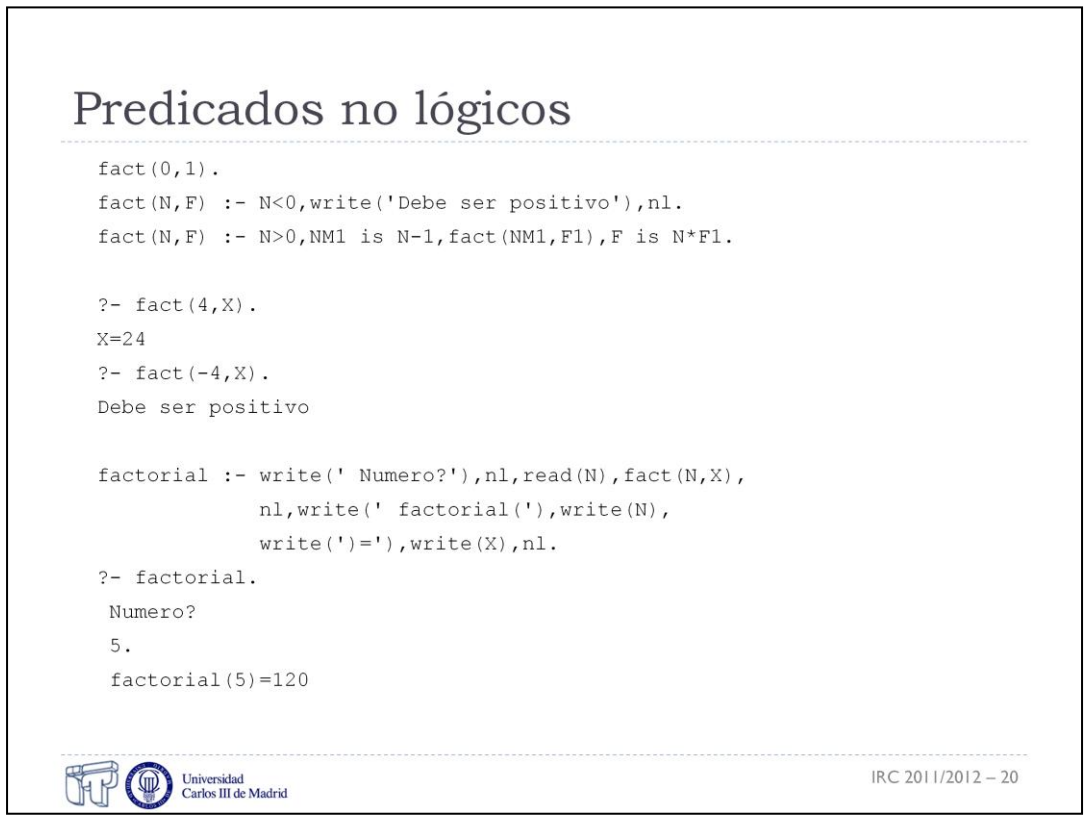


Formalmente, Prolog se basa en las llamadas cláusulas de Horn, y principalmente aplica continuamente dos reglas de inferencia: particularización de un universal (al dar valores a variables) y "modus ponens", que es equivalente a la combinación de las reglas de resolución más refutación, con lo que se podría demostrar teóricamente (Robinson 1965) que es un sistema inferencial completo.

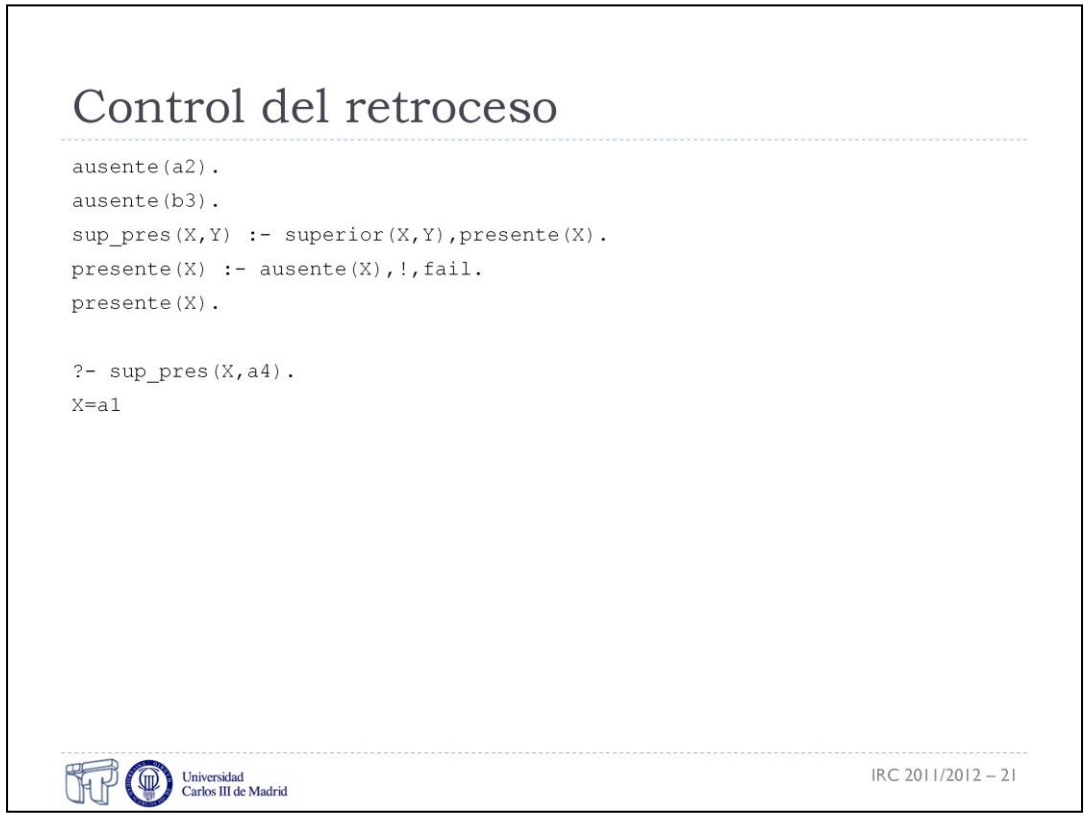
Prolog procedimental

- Es necesario o conveniente introducir en el lenguaje nuevas construcciones que aumenten la expresividad
- Predicados "no lógicos"
  - write(X), read(X)
  - get(X), put(<ascii>), nl
  - consult(<fichero>)
- Control del retroceso
  - **corte** (!): siempre VERDADERO e inhibe el retroceso
  - **fallo** (fail): siempre FALSO

Además de las construcciones puramente lógicas, es útil completar el lenguaje con cláusulas procedimentales, como leer de teclado, escribir en la pantalla, leer de un fichero, etc. Además existen dos operadores de control del retroceso: el operador de corte (representado como !), que una vez superado inhibe el retroceso, y el operador de fallo (fail) que siempre es falso y fuerza el retroceso.



Ejemplo de predicados no lógicos, representando el cálculo del factorial de un número.



Ejemplo de control del retroceso: el operador de corte (!) siempre es verdadero e inhibe el retroceso, y el operador de fallo es siempre falso y fuerza el retroceso. Así, las reglas representan el razonamiento “X está presente si no está ausente”.

Sintaxis de Prolog (reducida)

```
<programa> ::= <cláusula>{<cláusula>}

<cláusula> ::= <hecho>|<regla>|<consulta>

<hecho> ::= <cabeza>”.”

<regla> ::= <cabeza> “:=“ <cuerpo>”.”

<consulta> ::= <cuerpo>”.”

<cabeza> ::= <predicado>

<cuerpo> ::= <literal>{“,”<literal>}

<literal> ::= <predicado> | “not” <predicado>

<predicado> ::= <nombre>”(“<término>{“,”<término>}”)”

<término> ::= <constante>|<variable>|<lista>|<función>
```


Resumen de la sintaxis de Prolog (simplificada).