Términos

Comparando términos ==

- Prolog contiene un predicado importante para comparar términos.
- Este es el predicado de identidad ==
- El predicado de identidad == no crea instancias de variables, es decir, se comporta de manera diferente a =

Comparando términos ==

- ?- a==a.
  - true
- ?- a==b.
  - false
- ?- a==‘a’.
  - true
- ?- a==X.
  - false

Comparando variables

- Dos variables diferentes no instanciadas no son términos idénticos
- Las variables instanciadas con un término T son idénticas a T.
- ?- X==X.
  - true
- ?- Y==X.
  - false
- ?- a=U, a==U.
  - U = a

Comparando términos \==

- El predicado \== se define para que tenga éxito en aquellos casos donde == falla.
- En otras palabras, tiene éxito cuando dos términos no son idénticos, y falla cuando lo son.
- ?- a \== a.
  - false
- ?- a \== b.
  - true
- ?- a \== 'a’.
  - false
- ?- a \== X.
  - true

Términos aritméticos

- +, -, <, >, etc. son functores y expresiones como 2+3 son en realidad términos complejos ordinarios.
- El término 2+3 es idéntico al término +(2,3)
- ?- 2+3 == +(2,3).
  - true
- ?- -(2,3) == 2-3.
  - true
- ?- (4<2) == <(4,2).
  - true

Resumen de predicados de comparación

- = Predicado de unificación
- =\= Negación del predicado aritmético de igualdad
- =:= Predicado de igualdad aritmética
- \== Negación del predicado de identidad
- == Predicado de identidad
- \= Negación del predicado de unificación

Listas como términos

- Usando el | constructor, hay muchas maneras de escribir la misma list.
- ?- [a,b,c,d] == [a|[b,c,d]].
  - true
- ?- [a,b,c,d] == [a,b,c|[d]].
  - true
- ?- [a,b,c,d] == [a,b,c,d|[]].
  - true
- ?- [a,b,c,d] == [a,b|[c,d]].
  - true

Listas de Prolog internamente

- Internamente, las listas se construyen a partir de dos términos especiales:
  - – [] (que representa la lista vacía)
  - – '.' (un funtor de aridad 2 utilizado para construir listas no vacías
- Estos dos términos también se denominan constructores de lista.
- Una definición recursiva muestra cómo construyen lista.
- La lista vacía es el término []. Tiene longitud 0.
- Una lista no vacía es cualquier término de la forma .(term,list), donde term es cualquier termino, y list es una lista de Prolog. Si la lista tienen un largo n, entonces .(term,list) tiene un largo n+1.

Listas de Prolog internamente

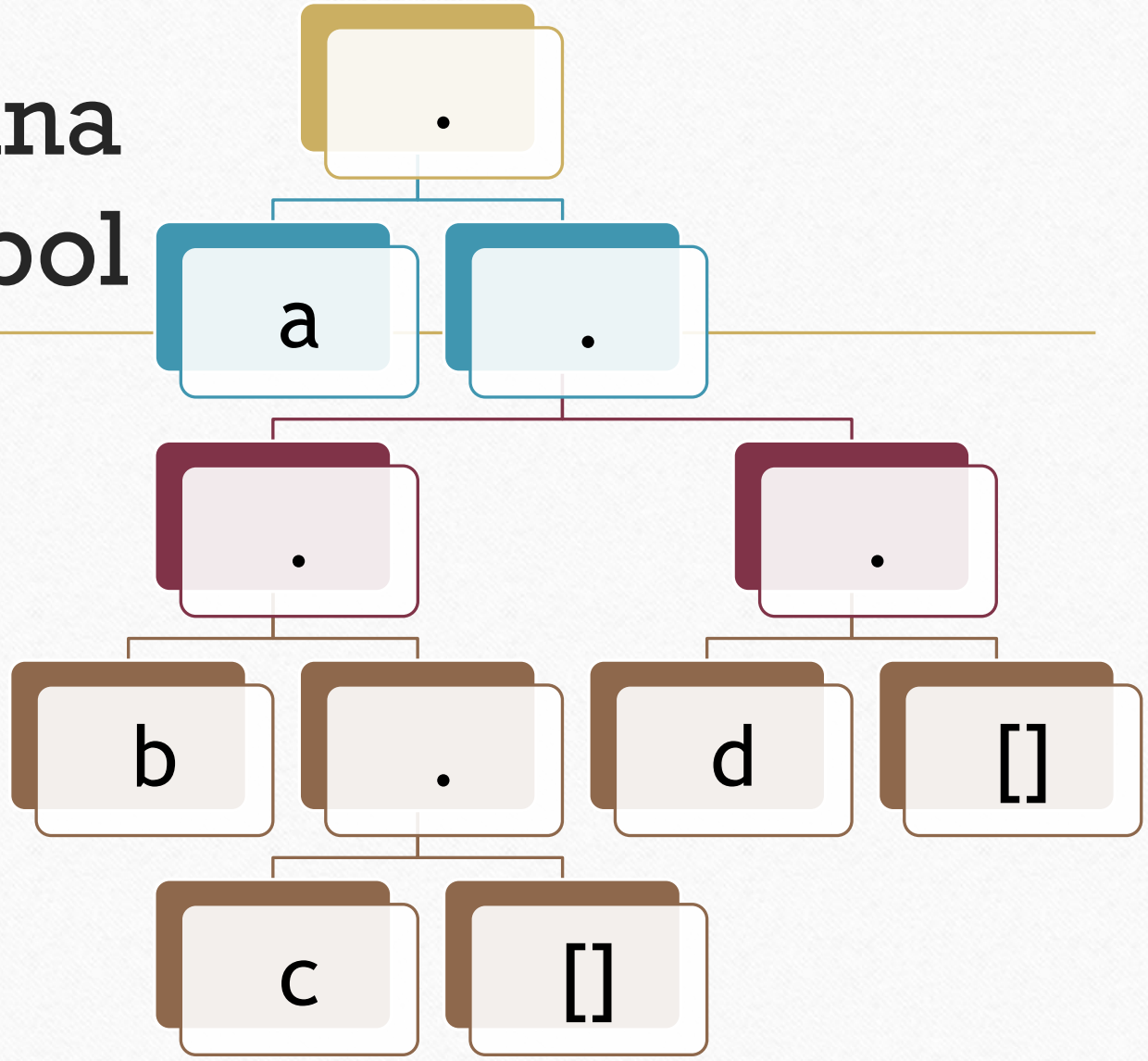
- ?- .(a,[]) == [a].
  - true
- ?- .(f(d,e),[]) == [f(d,e)].
  - true
- ?- .(a,.(b,[])) == [a,b].
  - true
- ?- .(a,.(b,.(f(d,e),[]))) == [a,b,f(d,e)].
  - true

Representación interna de las listas

- Otras notaciones similares a `|`:
- Representa una lista en dos partes
  - Su primer elemento, la cabeza
  - el resto de la lista, la cola
- El truco es leer estos términos como árboles
  - Los nodos internos están etiquetados con `.`
  - Todos los nodos tienen dos nodos secundarios
    - Subárbol debajo del hijo izquierdo es la cabeza
    - Subárbol debajo del hijo derecho es la cola

Ejemplo de una lista como árbol

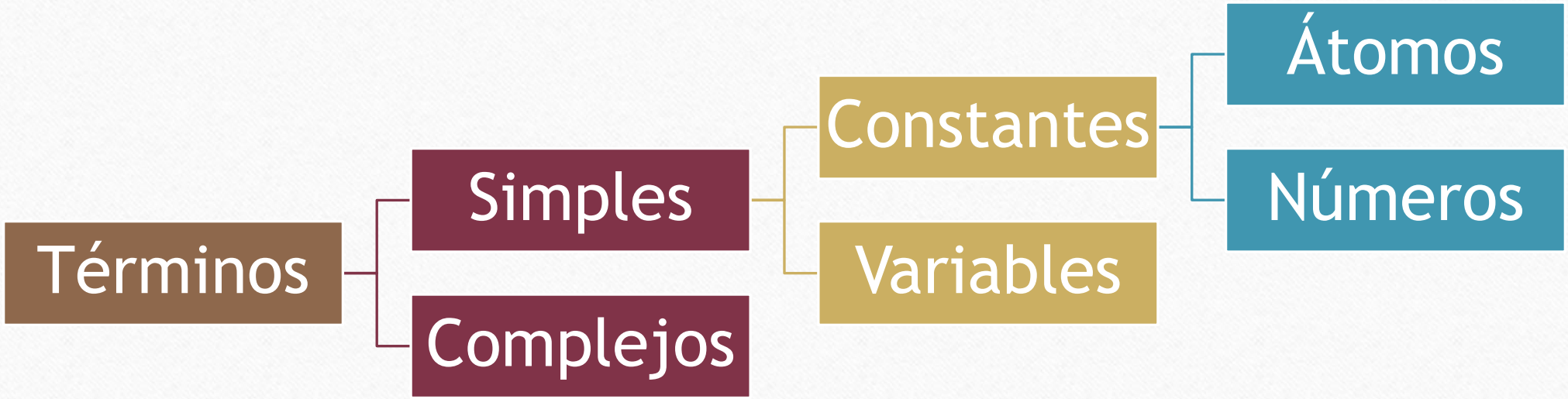
- Ejemplo [a,[b,c],d]



Examinando los términos

- Ahora veremos los predicados incorporados que nos permiten examinar los términos de Prolog más de cerca.
  - Predicados que determinan el tipo de términos
  - Predicados que nos dicen algo sobre la estructura interna de los términos

Tipo de términos



Chequeando los términos

- atom
- integer
- float
- number
- atomic
- var
- nonvar

- El argumento es un atomo?
- … un entero?
- … un numero de punto flotante?
- … un entero o flotante
- … una constante?
- … una variable no instanciada?
- … una variable instanciada u otro termino que no que no es una variable no instanciada.

Chequeando…

- ?- atom(a).
  - true
- ?- atom(7).
  - false
- ?- atom(X).
  - false
- ?- X=a, atom(X).
  - X = a
  - true
- ?- atom(X), X=a.
  - false

Chequeando…

- ?- atomic(mia).
  - true
- ?- atomic(5).
  - true
- ?- atomic(loves(vincent,mia)).
  - true
- ?- var(mia).
  - false
- ?- var(X).
  - true
- ?- X=5, var(X).
  - false

Chequeando…

- ?- nonvar(X).
  - false
- ?- nonvar(mia).
  - false
- ?- nonvar(23).
  - dalse

La estructura de los términos

- Dado un término complejo de estructura desconocida, ¿qué tipo de información podríamos querer extraer de él?
- Obviamente:
  - El functor
  - La aridad
  - El argumento
- Prolog proporciona predicados integrados para producir esta información.
- El predicado functor da el functor y la aridad de un predicado complejo.

Predicado Functor

- El predicado functor da el funtor y la aridad de un predicado complejo
- ?- functor(amigos(pablo,paola),A,C).
  - A = amigos
  - C = 2
- ?- functor([pablo,paola,vero],A,C).
  - A = '[|]'
  - C = 2
- ¿Qué sucede cuando usamos functor con constantes?
- ?- functor(pablo,A,C).
  - A = pablo
  - C = 0

Construir términos con Functor

- También puede utilizar functor rara construir términos:
- ?- functor(Terminos,amigos,2).
  - Terminos = amigos(_,_)

Arg

- Prolog también nos proporciona el predicado arg.
- Este predicado nos habla de los argumentos de términos complejos.
- Se necesitan tres argumentos:
  - Un número N
  - Un término complejo T
  - El enésimo argumento de T
- ?- arg(2,likes(lou,andy),A).
  - A = andy

String

- Las cadenas se representan en Prolog mediante una lista de códigos de caracteres
- Prolog ofrece comillas dobles para una notación fácil de las cadenas.
- ?- S = “Pao“.
- ?- S = [86,105,99,107,121]

Propiedades de los Operadores

- Operadores de infijo
  - Functor escrito entre sus argumentos
  - Ej: + - = == , ; . -->
- Operadores de prefijo
  - Functor escrito antes de su argumento
  - Ej: - (para representar números negativos)
- Operadores de postfijo
  - Functor escrito después de su argumento
  - Ej: ++ en el lenguaje de programación C

Precedencia

- Cada operador tiene una cierta precedencia para resolver ambigüedad en las expresiones.
- Por ejemplo, ¿Qué significa 2+3*3?
- ¿2+(3*3), o (2+3)*3?
- Porque la precedencia de + es mayor que el de *, Prolog elige + ser El funtor principal de 2+3*3

Asociatividad

- Prolog usa asociatividad para desambiguar operadores con el mismo valor de precedencia.
- Ejemplo: 2+3+4
- ¿Esto significa (2+3)+4 o 2+(3+4)?
  - Asociativo de izquierda
  - Asociativo de derecha
- Los operadores también pueden definirse como no asociativos, en cuyo caso se ve obligado a usar corchetes en casos ambiguos.
-

Definiendo operadores

- Prolog le permite definir sus propios operadores.
- Las definiciones de operador se ven así:
- Precedencia: :- op(Precedence, Type, Name).
  - número entre 0 y 1200
  - Tipo: el tipo de operador

Tipo de operadores

- yfx: asociativo izquierdo, infijo
- yfx: asociativo izquierdo, infijo
- xfy: asociativo derecha, infijo
- xfx no asociativo, infijo
- fx no asociativo, prefijo
- fy associativo derecha, prefijo
- xf no asociativo, postfijo
- yf asociativo izquierdo, postfijo
-

Tipo de operadores

1200	*xfx*	-->, :-
1200	*fx*	:-, ?-
1150	*fx*	dynamic, discontiguous, initialization, module_transparent, multifile, thread_local, volatile
1100	*xfy*	;, |
1050	*xfy*	->, op*->
1000	*xfy*	,
954	*xfy*	\
900	*fy*	\+
900	*fx*	~
700	*xfx*	<, =, =.., =@=, =:=, =<, ==, =\=, >, >=, @<, @=<, @>, @>=, \=, \==, is
600	*xfy*	:
500	*yfx*	+, -, /\, \/, xor
500	*fx*	+, -, ?, \
400	*yfx*	*, /, //, rdiv, <<, >>, mod, rem
200	*xfx*	**
200	*xfy*	^

Ejercicios

- ¿Cuál de las siguientes consultas se realiza correctamente y cuál falla?
- ?- 12 is 2*6.
- ?- 14 =\= 2*6.
- ?- 14 = 2*7.
- ?- 14 == 2*7.
- ?- 14 \== 2*7.
- ?- 14 =:= 2*7.
- ?- [1,2,3|[d,e]] == [1,2,3,d,e].
- ?- 2+3 == 3+2.
- ?- 2+3 =:= 3+2.
- ?- 7-2 =\= 9-2.
- ?- p == 'p'.
- ?- p =\= 'p'.
- ?- vincent == VAR.
- ?- vincent=VAR, VAR==vincent.

Ejercicios

- ¿Cuál de las siguientes consultas se realiza correctamente y cuál falla?
- ?- 12 is 2*6.
- ?- 14 =\= 2*6.
- ?- 14 = 2*7.
- ?- 14 == 2*7.
- ?- 14 \== 2*7.
- ?- 14 =:= 2*7.
- ?- [1,2,3|[d,e]] == [1,2,3,d,e].
- ?- 2+3 == 3+2.
- ?- 2+3 =:= 3+2.
- ?- 7-2 =\= 9-2.
- ?- p == 'p'.
- ?- p =\= 'p'.
- ?- vincent == VAR.
- ?- vincent=VAR, VAR==vincent.

Mas ejercicios

- ¿Cómo responde Prolog a las siguientes consultas?
  - ?- (a,.(b,.(c,[]))) = [a,b,c].
  - ?- (a,.(b,.(c,[]))) = [a,b|[c]].
  - ?- (.(a,[]),.(.(b,[]),.(.(c,[]),[]))) = X.
  - ?- (a,.(b,.(.(c,[]),[]))) = [a,b|[c]].

Mas ejercicios

- ¿Cómo responde Prolog a las siguientes consultas?
  - ?- (a,.(b,.(c,[]))) = [a,b,c].
  - ?- (a,.(b,.(c,[]))) = [a,b|[c]].
  - ?- (.(a,[]),.(.(b,[]),.(.(c,[]),[]))) = X.
  - ?- (a,.(b,.(.(c,[]),[]))) = [a,b|[c]].

Mas ejercicios

- Escriba un predicado de dos lugares termtype (Term, Type) que tome un término y devuelva los tipos de ese término (átomo, número, constante, variable, etc.). Los tipos deben devolverse en el orden de su generalidad. El predicado debe comportarse de la siguiente manera.

- ?- termtype(Vincent,variable).
  - true
- ?- termtype(mia,X).
  - X = atom ;
  - X = constant ;
  - X = term ;
- ?- termtype(linda(flor),X).
  - X = compount ;
  - X = term ;

Mas ejercicios

- Escriba un programa Prolog que defina el predicado groundterm(Term) que pruebe si Term es o no un término basico. Los términos básicos son términos que no contienen variables. Estos son ejemplos de cómo debe comportarse el predicado:
- ?- groundterm(X).
  - false
- ?- groundterm(french(bic_mac,le_bic_mac)).
  - true
- ?- groundterm(french(whopper,X)).
  - false

- Supongamos que tenemos las siguientes definiciones de operador.
  - :- op(300, xfx, [son, es_un]).
  - :- op(300, fx, le_gusta).
  - :- op(200, xfy, y).
  - :- op(100, fy, famoso).
- ¿Cuáles de los siguientes son términos bien formados? ¿Cuáles son los principales operadores? Proporcione las inferencias.
  - X es_un mago
  - harry y ron y hermione son amigos
  - harry es_un mago y le_gusta el quidditch
  - dumbledore es_un mago famoso