Deductive Databases and Logic Programming (Winter 2003/2004)

Chapter 4: Built-In Predicates

- Binding Patterns (Modes)
- Range Restriction / Allowedness
- Prolog Built-In Predicates
- Built-In Predicates and Function Symbols

Objectives

After completing this chapter, you should be able to:

- define and explain binding pattern.
- check the allowedness of a clause.
- write Prolog programs using built-in predicates.
- explain how function symbols could be implemented with binding patterns.



1. Built-In Predicates, Binding Patterns

- 2. Important Built-In Predicates in Prolog
- 3. Range-Restriction, Allowedness
- 4. Function Symbols and Built-In Predicates



- A very pure Prolog Program contains only predicates that are defined by facts and rules.
- However, for larger, real-world applications, this is not very realistic.
- Already for simulating SQL-queries in Prolog, needs e.g. the standard arithmetic operators +, -, *, /, and the comparison operators =, \neq , <, >, \leq , \geq .
- For real programs, one needs also a mechanism for input/output etc.



- Theoretically, it might be possible to define e.g. < for all numbers that occur in the program by facts.
- But it would at least be tedious to enumerate all facts *X* < *Y* that might be important for a program.
- Therefore, Prolog systems and deductive database systems have certain predicates predefined by procedures in the system.
- E.g. for the query 3 < 5, the system does not look up facts and rules, but calls a built-in procedure written e.g. in C.



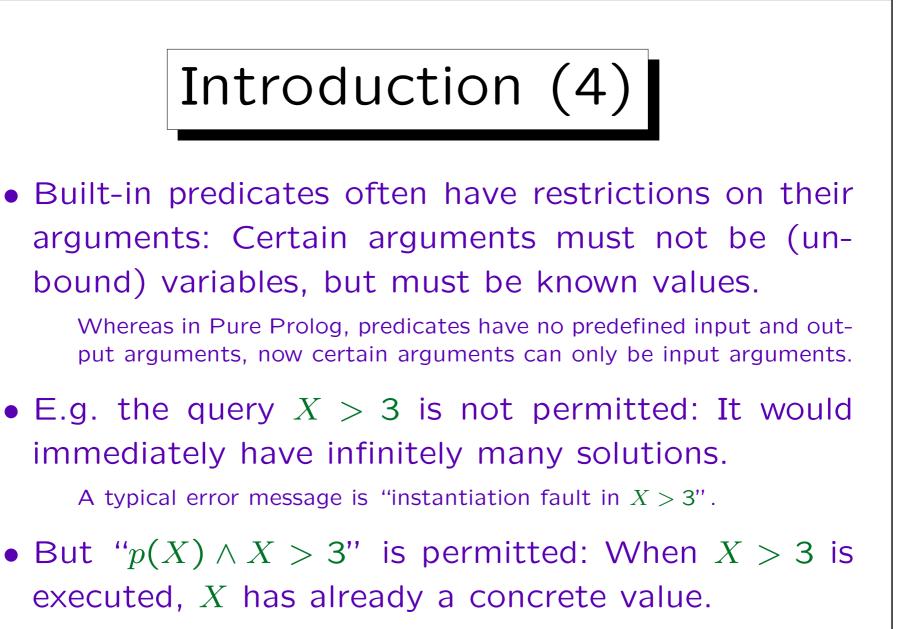
 Since built-in predicates are defined in the system, it is illegal to write a literal with a built-in predicate in the head of a rule, e.g.

 $X \leq Z \leftarrow X \leq Y \land Y \leq Z$. Error!

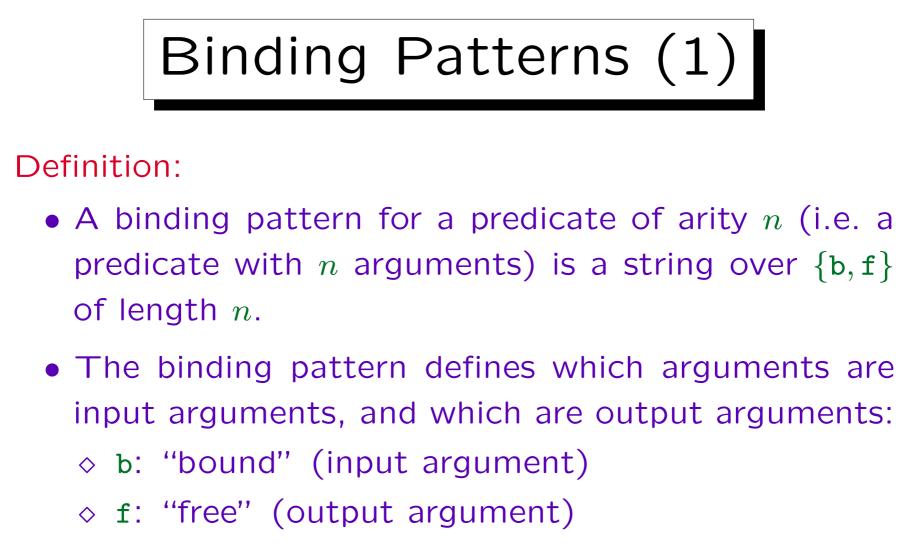
• Rules contribute to the definition of the predicate in their head, and the definition of built-in predicates cannot be changed.

Typical error message: "Attempt to modify static procedure \leq / 2."

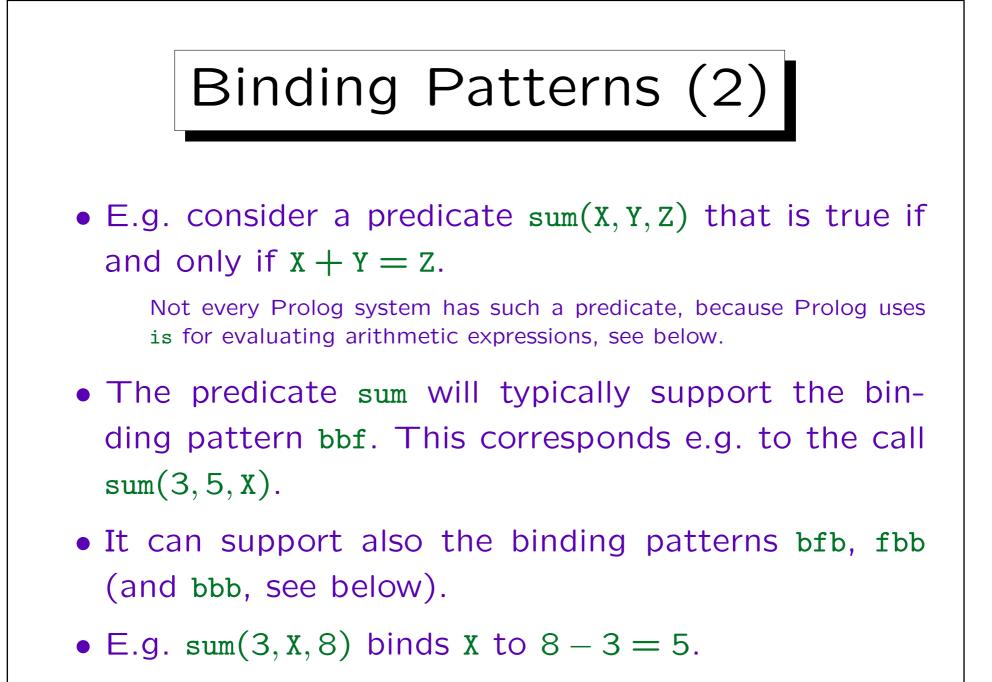
• Of course, one can use built-in predicates in the body of a rule (i.e. call them).



If p binds its argument to a value, e.g. " $p(X) \leftarrow$ " would not help.



• Binding patterns are not only important for built-in predicates, but can be specified for any predicate.





• If all three binding patterns are supported, a deductive DBMS will internally have three procedures:

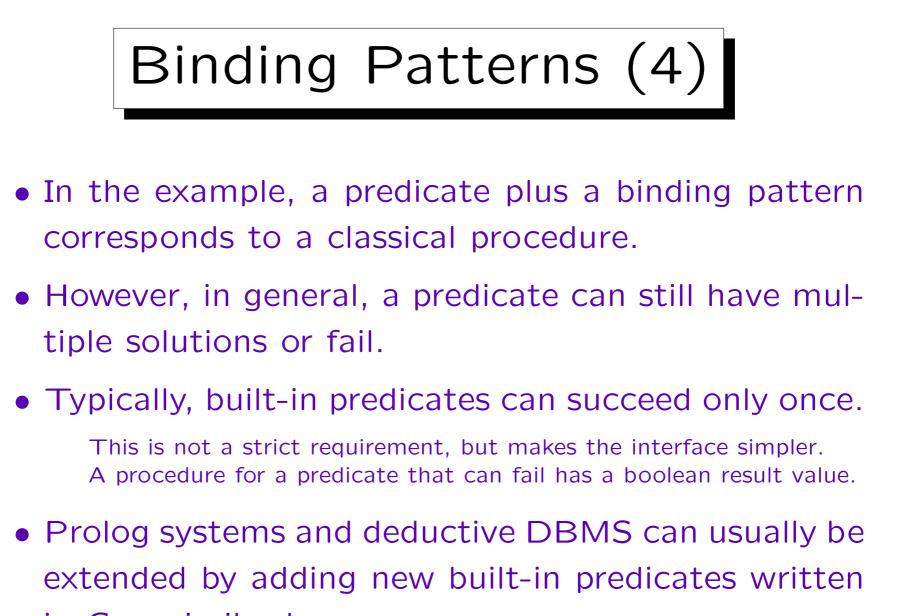
 \diamond sum_bbf(X, Y, var Z): begin Z := X + Y; end

 \diamond sum_bfb(X, var Y, Z): begin Y := Z - X; end

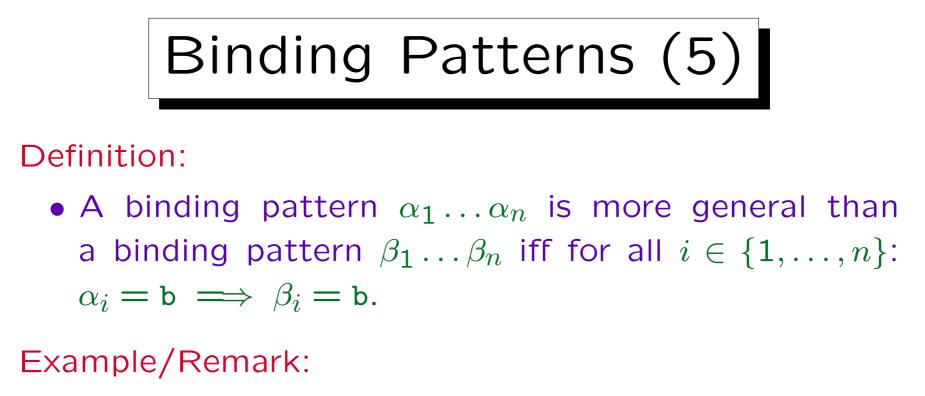
◇ sum_fbb(var X, Y, Z): begin X := Z - Y; end

• The compiler then selects the right procedure depending on the arguments.

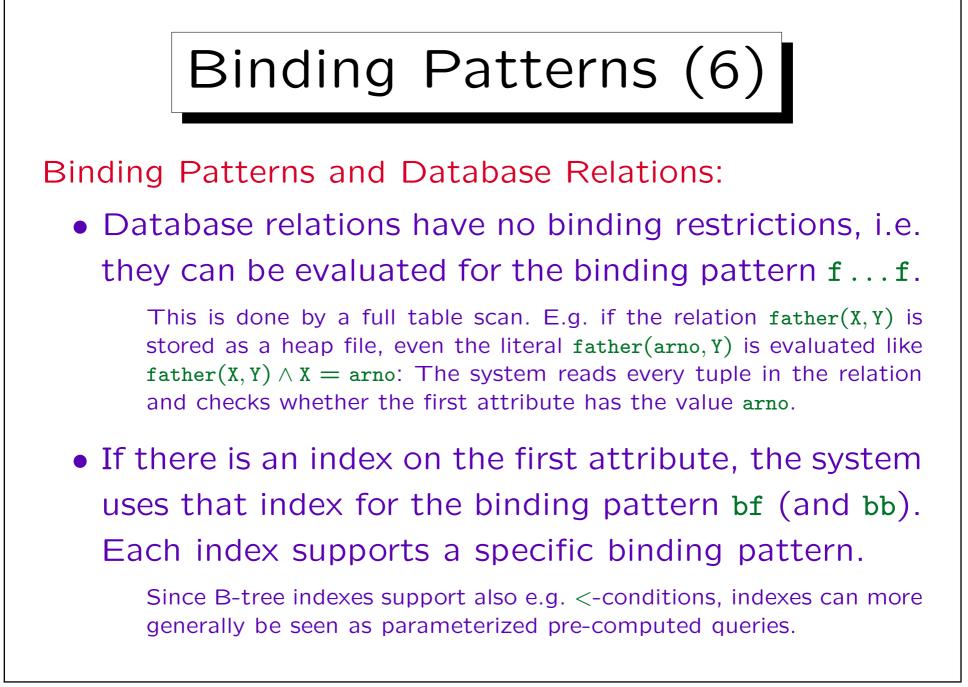
In Prolog, it is not always possible for the compiler to know whether a variable will be bound or free, therefore, there might be a runtime test to check which case applies.

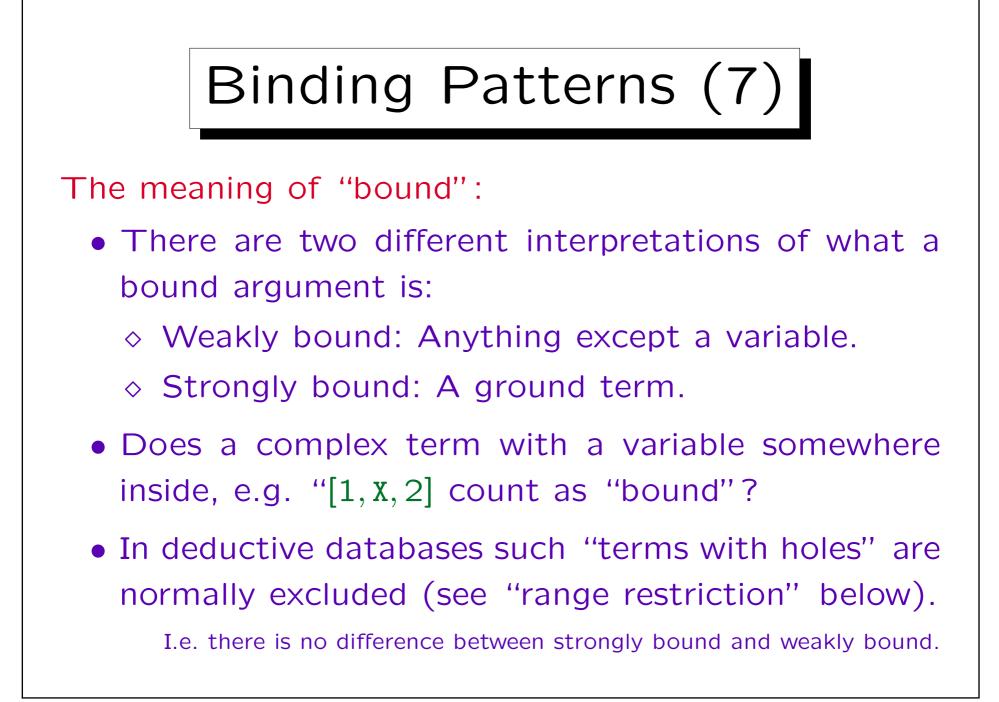


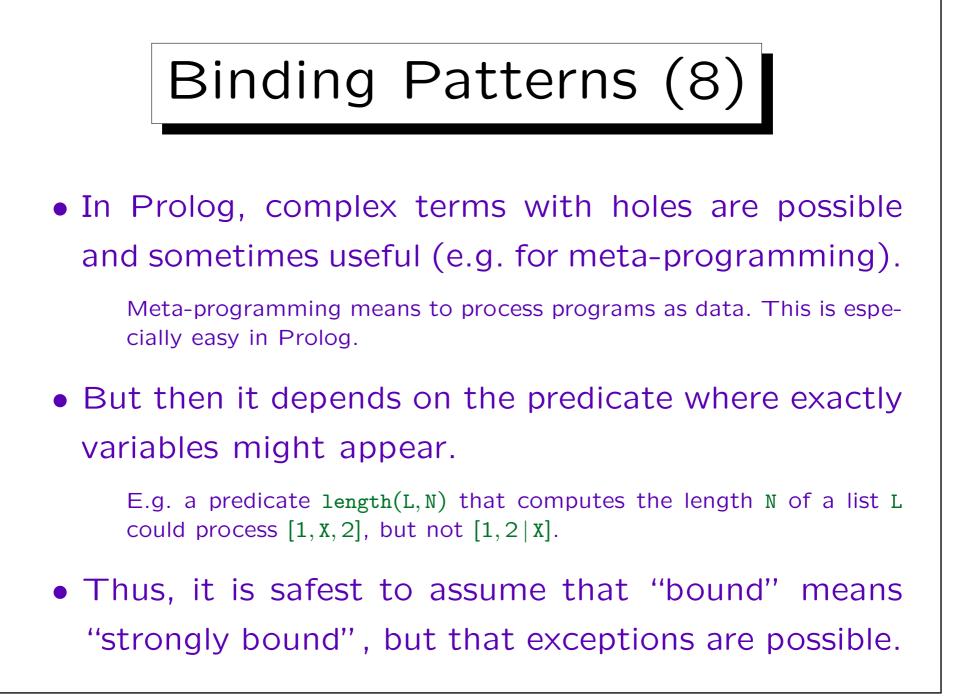
in C or similar languages.

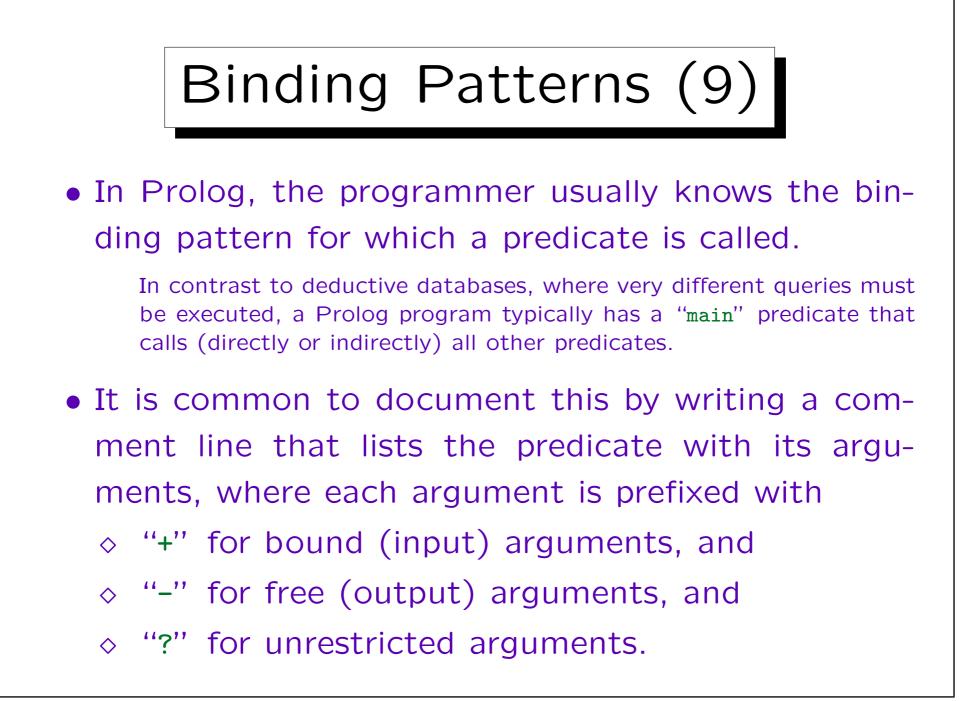


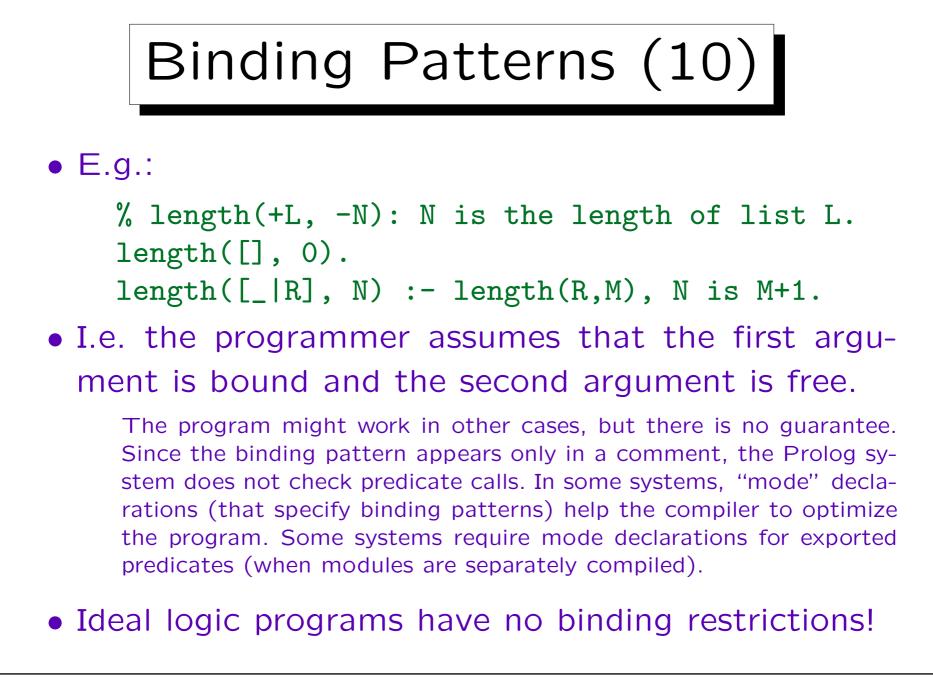
- The binding pattern bbf is more general than bbb.
- One can always use a procedure for a more general binding pattern.
- E.g. the compiler could transform sum(3,5,8) into sum(3,5,X) ∧ X = 8 (with a new variable X).



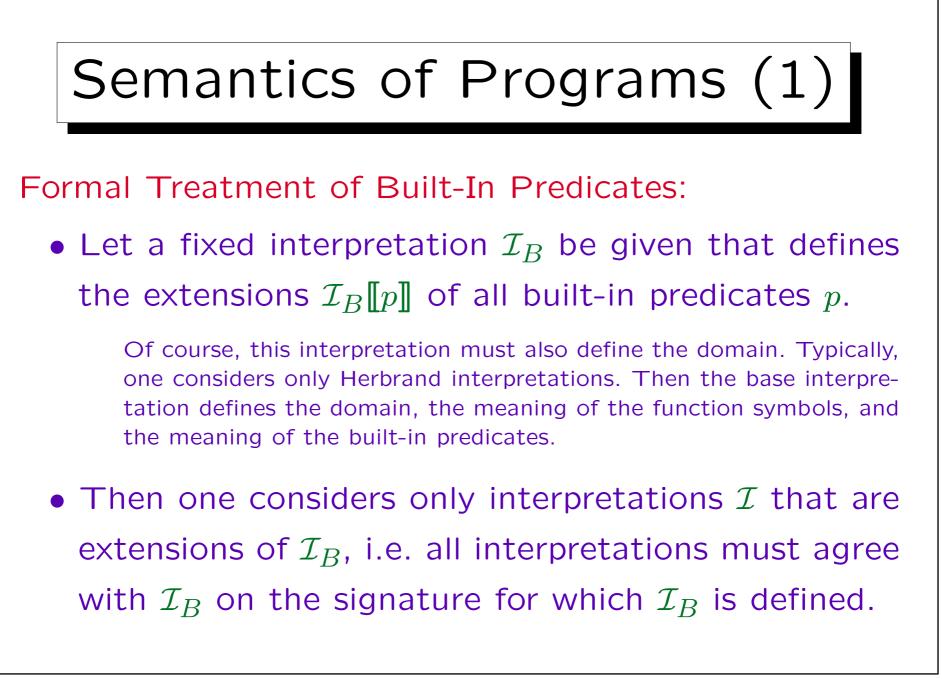








Stefan Brass: Deductive Databases and Logic Programming Univ. Halle, 2003



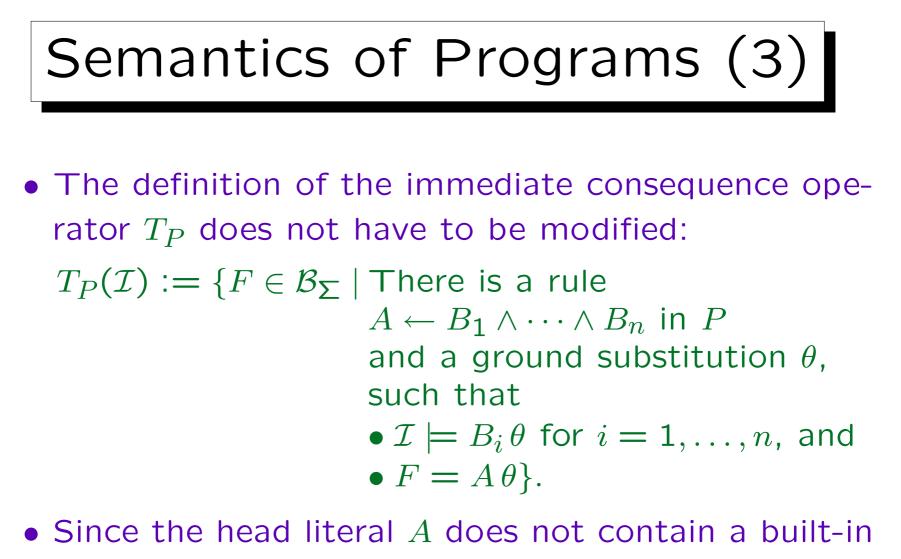
Semantics of Programs (2)

• One identifies an interpretation \mathcal{I} with the set of facts $p(t_1, \ldots, t_n)$ with $\mathcal{I} \models p(t_1, \ldots, t_n)$, where p is not a built-in predicate.

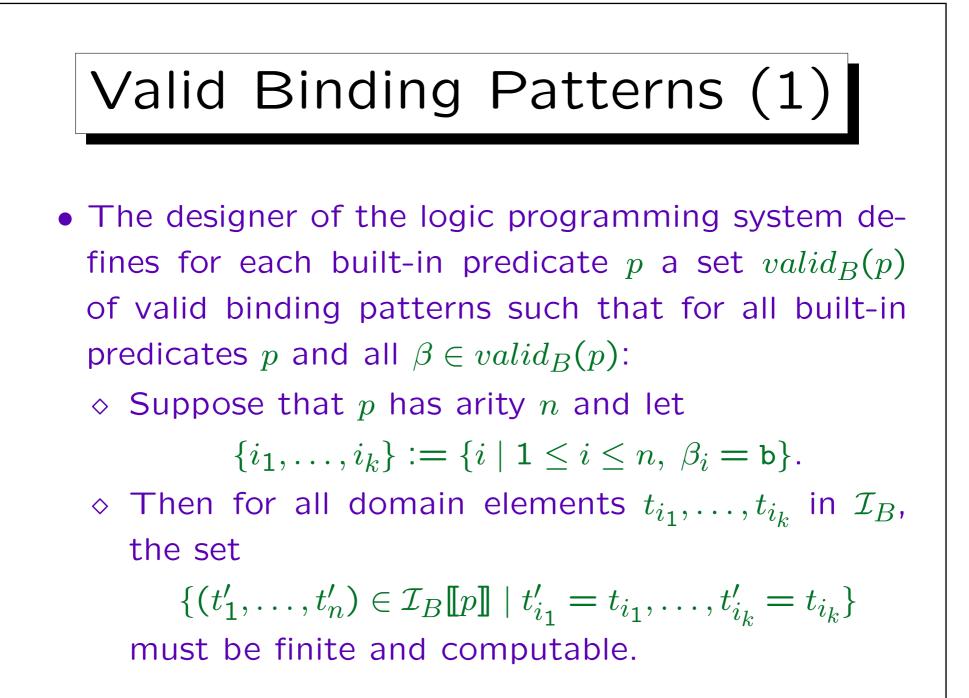
This is an extension of the corresponding convention for Herbrand interpretations. Since built-in predicates nearly always have an infinite extension, excluding them increases the chances that the set of facts is finite (and thus can be explicitly written down or explicitly stored). Of course, also predicates defined by rules can have infinite extensions.

• As before, the semantics of a program P is the least fixed point of the T_P -operator (see next slide).

This is the least model of P among the interpretations \mathcal{I} that extend \mathcal{I}_B .



predicate, one gets only facts about user-defined predicates.



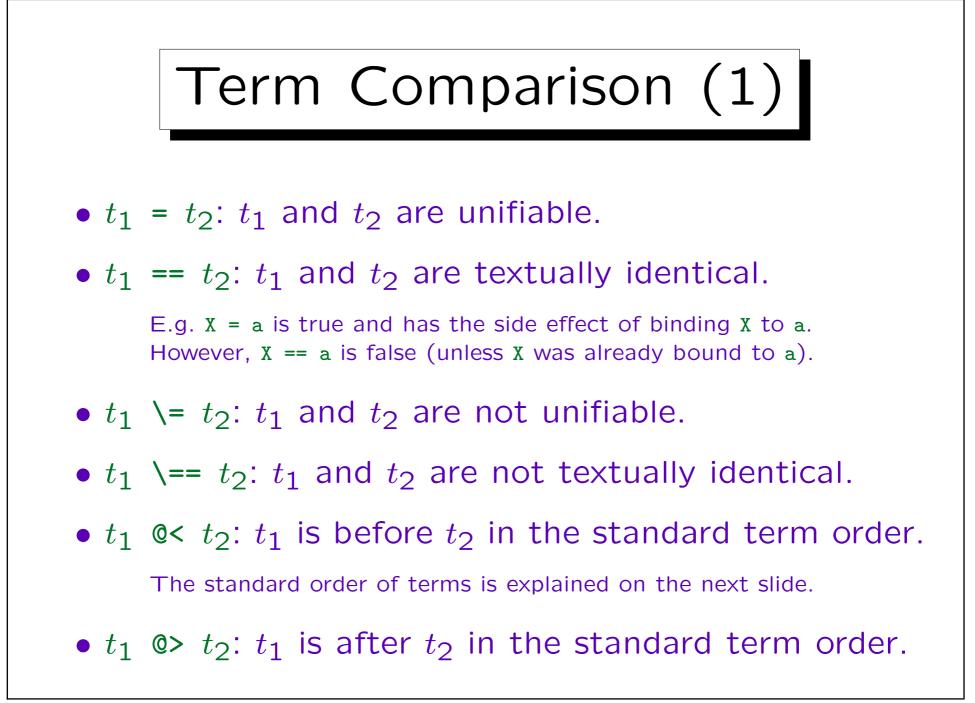


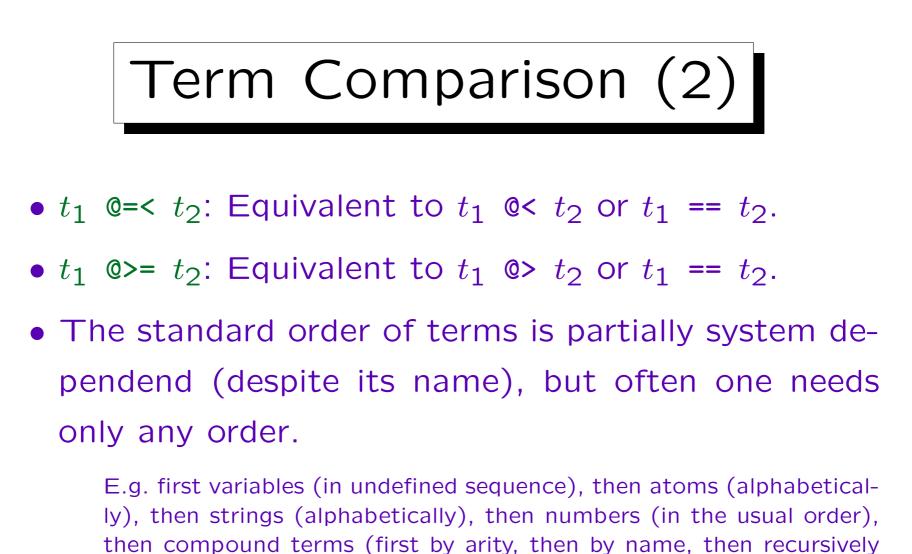
- I.e. the requirement is that given any values for the bound arguments, it must be effectively possible to compute values for the other arguments, and to compute all such solutions.
- Together with the range-restriction defined below, this ensures that each single application of the T_P operator is computable and has a finite result.

Of course, in the limit (minimal model), it is still possible that userdefined predicates have infinite extensions. Which is also bad, because it means that the computation does not terminate.



- 1. Built-In Predicates, Binding Patterns
- 2. Important Built-In Predicates in Prolog
- 3. Range-Restriction, Allowedness
- 4. Function Symbols and Built-In Predicates

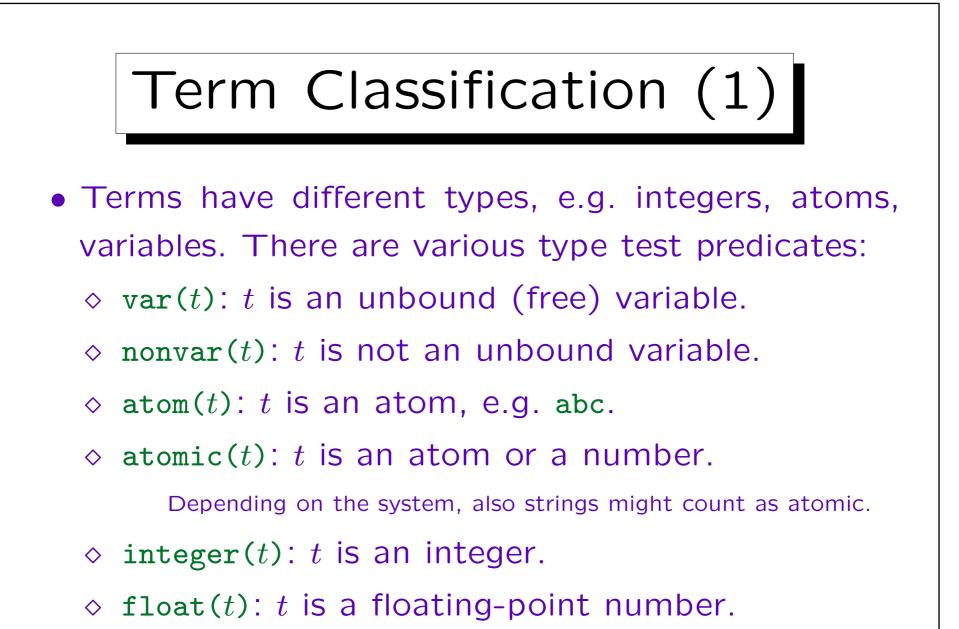




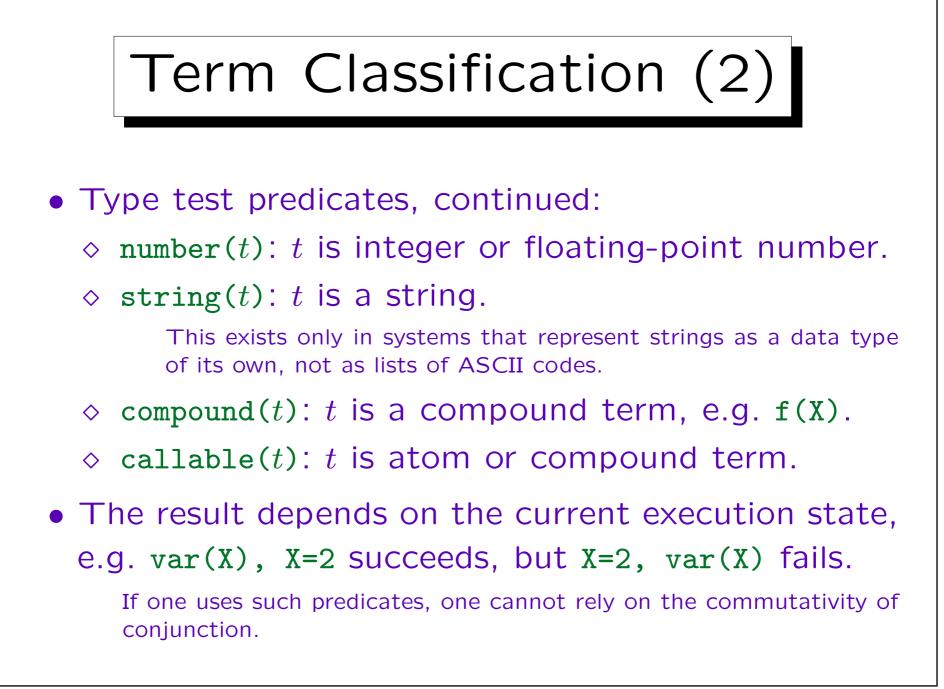
by arguments from left to right).

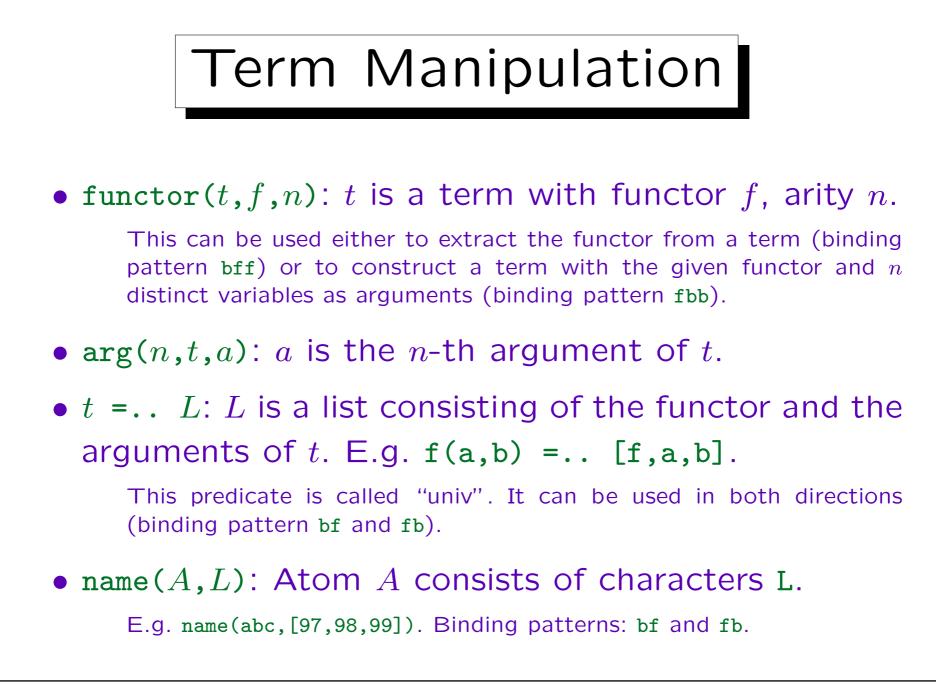
• compare(o, t_1, t_2): Binds o to <, =, or >.

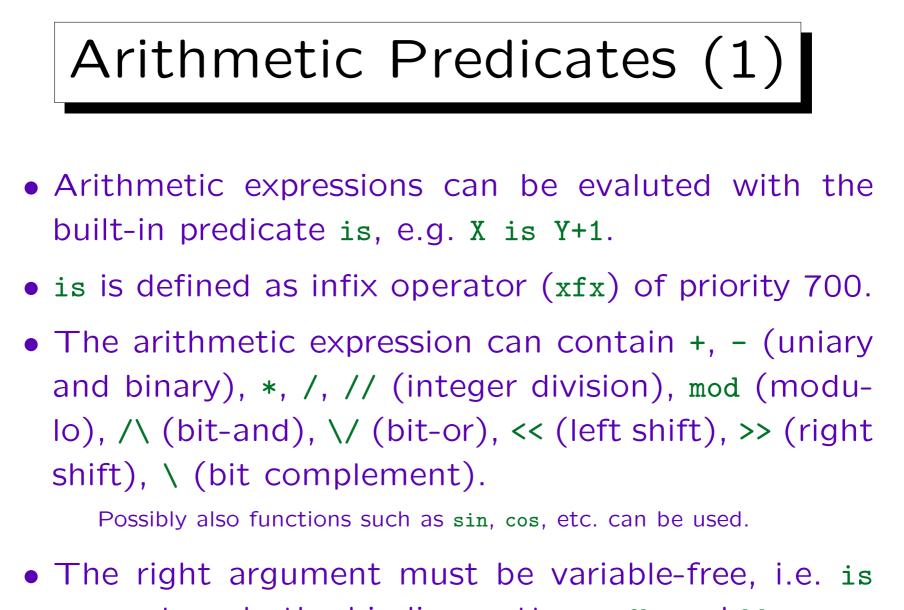
4 - 25

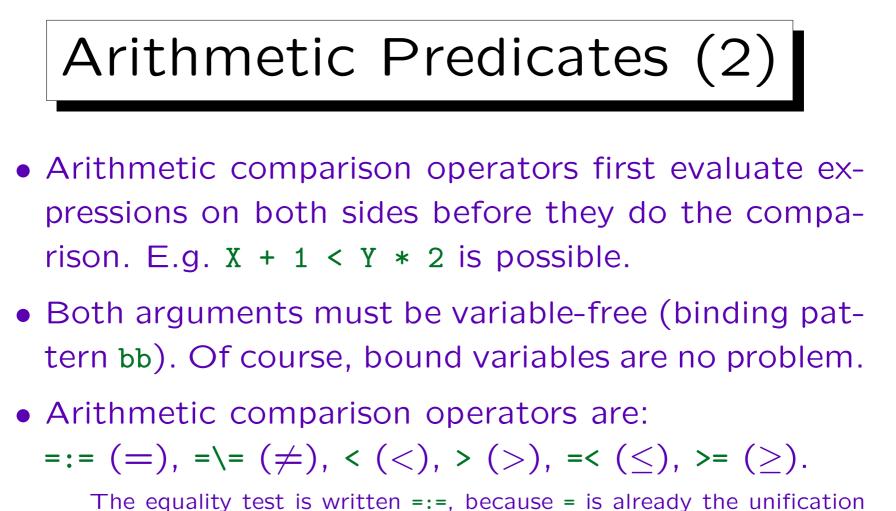


Depending on the system, this might also be called real(t).

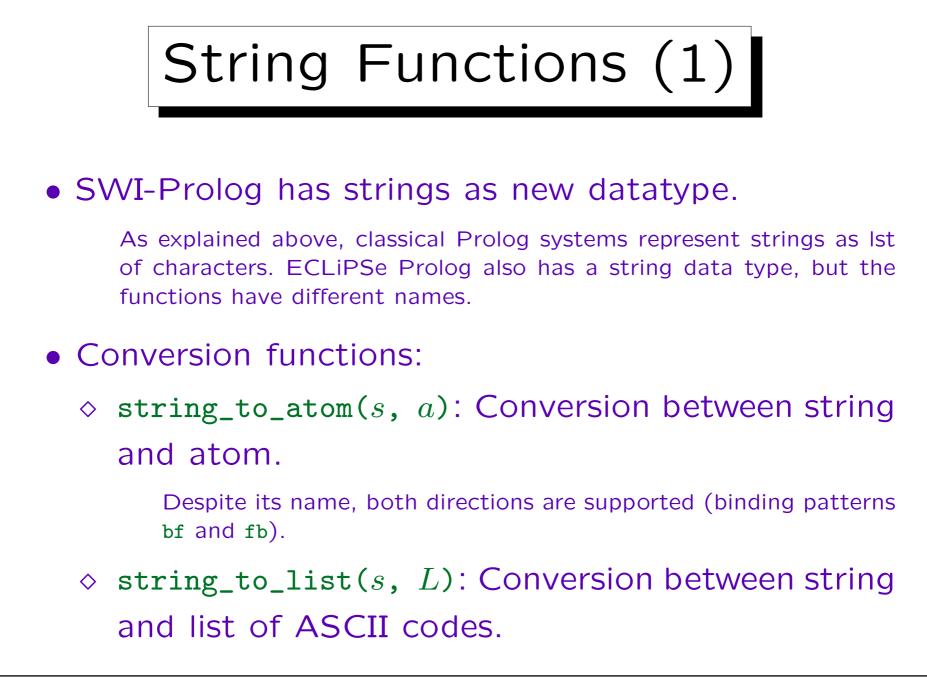


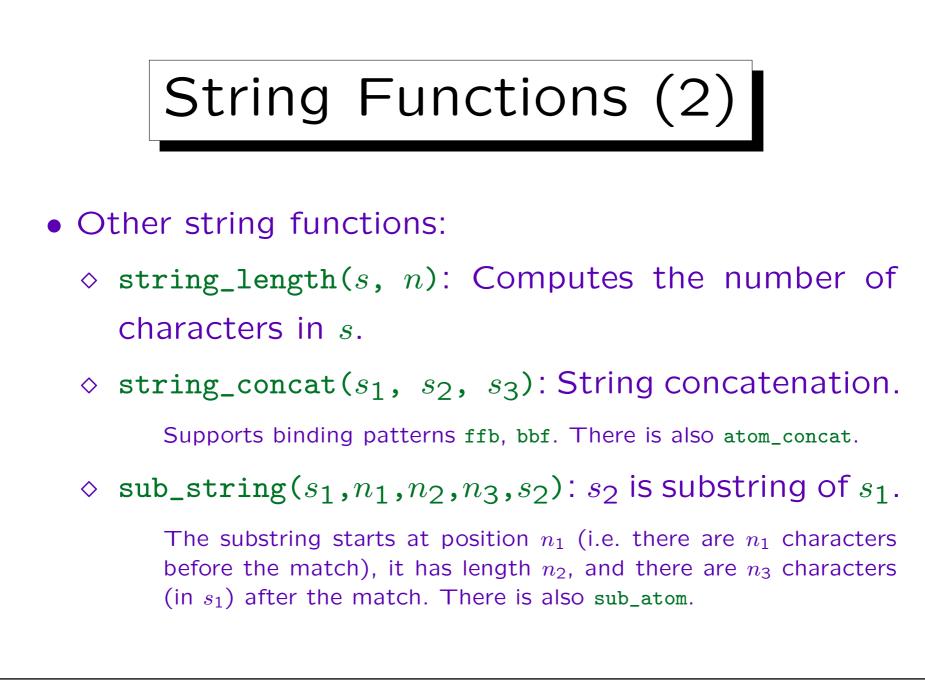






(which does not evaluate arithmetic expressions). In the same way, inequality is written =\=, because $\ means$ "does not unify with". Note that \leq is written =<, because the Prolog designers wanted to save the arrow <= for other purposes.







• Define a predicate to compute the Fibonacci numbers:

$$f(n) := \begin{cases} 1 & n = 0, n = 1\\ f(n-1) + f(n-2) & n \ge 2. \end{cases}$$

- Define sum(X, Y, Z) that holds iff X + Y = Z and can handle the binding patterns bbf, bfb, fbb, bbb.
- Define a predicate makeground(t) that binds all variables that appear in t to x.

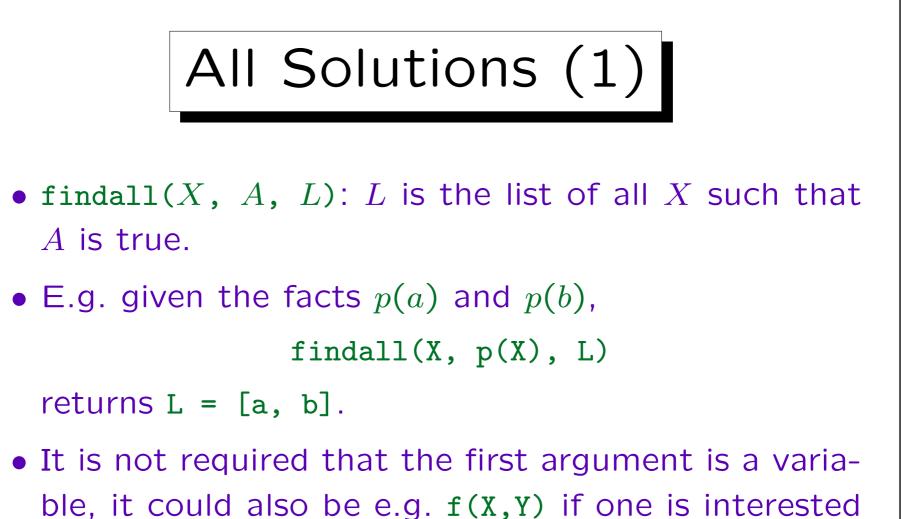
Constructed Goals

 Proof goals can be dynamically constructed, i.e. can be computed at runtime.

In purely compiled languages, that is not possible.

- call(A): Executes the literal A.
- In many Prolog systems, one can write simply X instead of call(X).

But it might be clearer to use call.



in bindings for both variables.

I.e. in general, the result list contains the instantiation of the first argument whenever a solution to the second argument was found.

All Solutions (2)

• bagof(X, A, L): (similar to findall).

The different lies in the treatment of variables that occur in A, but do not occur in X. findall treats them as existentially quantified, i.e. it does not bind them, and findall can succeed only once. In contrast, bagof binds such variables to a value and collects then only solutions with this value. Upon backtracking, one can also get other solutions. For example, suppose that p is defined by the facts p(a,1), p(a,2), p(b,3). Then findall(X, p(Y,X), L) would bind L=[1,2,3]. However, bagof(X, p(Y,X), L) would succeed two times: One for Y=a and L=[1,2], and once for Y=b and L=[3].

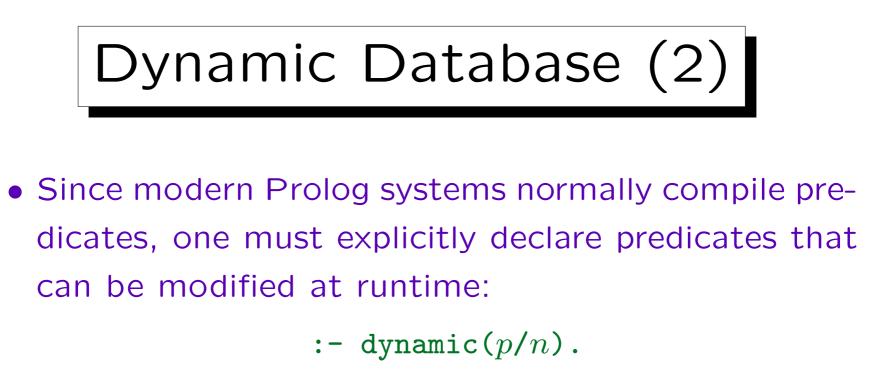
setof(X, A, L): As bagof, but the result list is ordered and does not contain duplicates.

Stefan Brass: Deductive Databases and Logic Programming Univ. Halle, 2003

Dynamic Database (1)

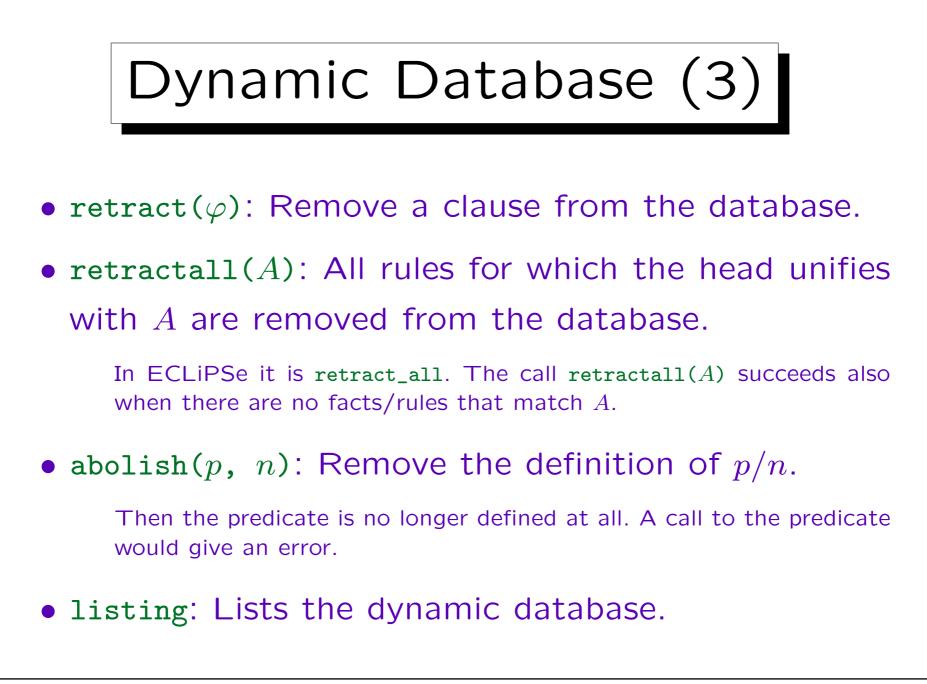
- Prolog systems permit that the definition of certain predicates is modified at runtime.
- E.g. if a database relation is represented as a set of facts, one can insert and delete facts.
- Such changes persist even when Prolog backtracks to find another solution.

Input/output and modifications of the dynamic database are the only changes that are not undone upon backtracking.



assert(φ): The clause φ is inserted into the dynamic database.

Normally, φ will be a fact, but it is also possible to assert rules. Some Prolog systems guarantee that the new clause is appended at the end of the predicate definition, but officially, there is no guarantee about the order unless one uses asserta (insert at the beginning) or assertz (insert at the end).

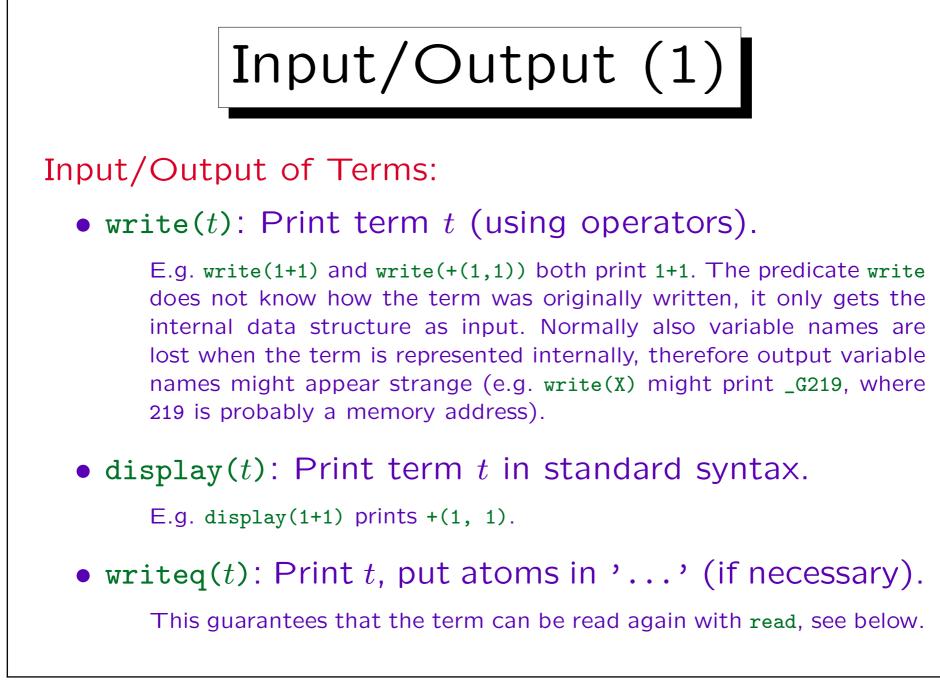


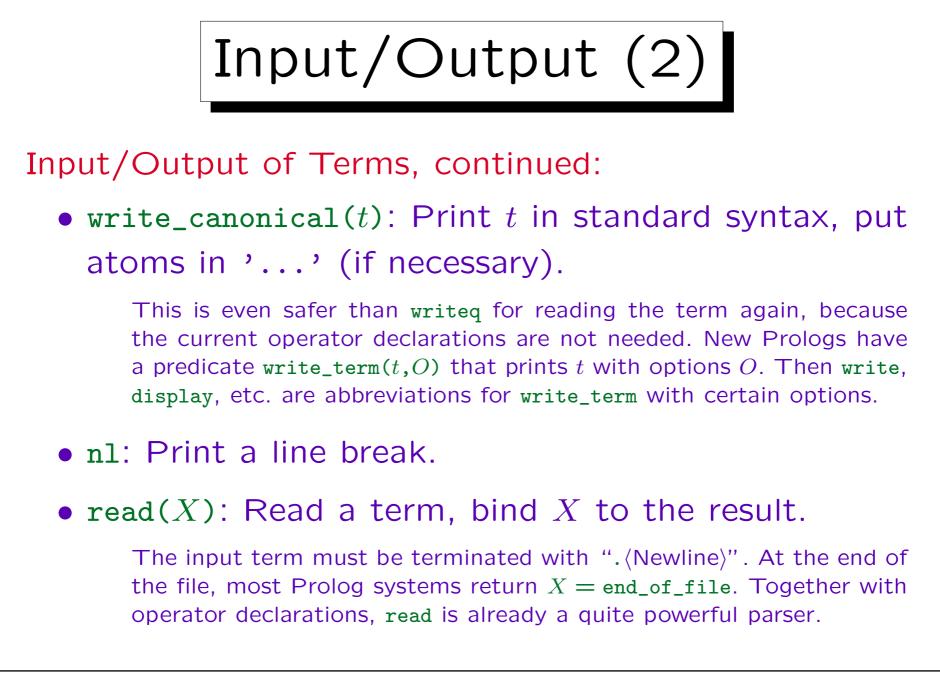


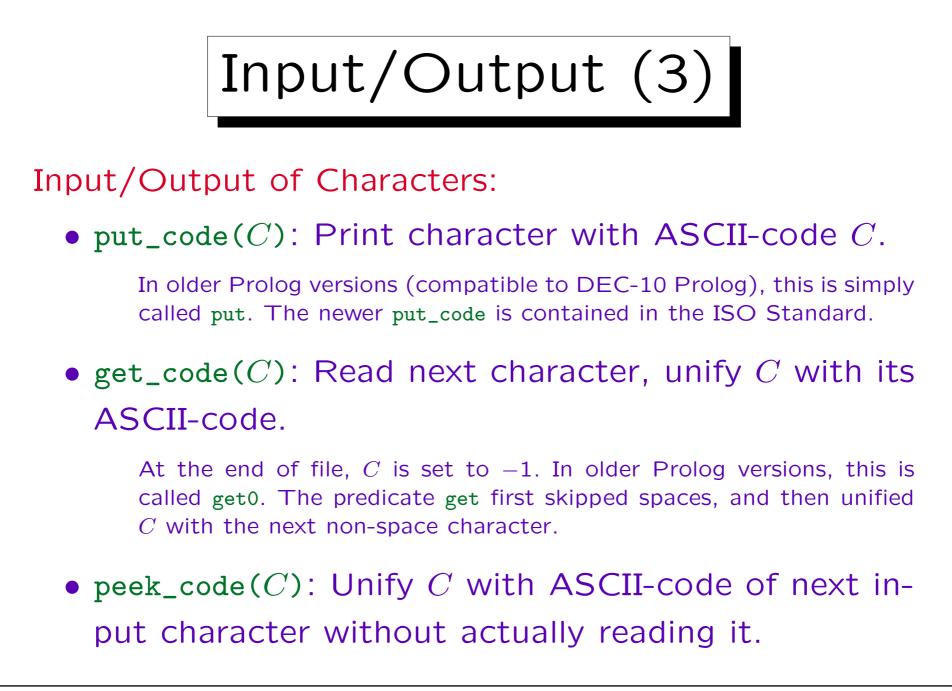
Exercise:

- Define a predicate next(N), that generates unique numbers, i.e. the first call returns 1, the second call returns 2, and so on.
- Define a predicate all_solutions that works like findall.

Of course, you should not use findall or bagof, but the dynamic database. For simplicity, you can assume that the goal does not call recursively all_solutions. You need the predicate fail that is logically false (triggers backtracking).

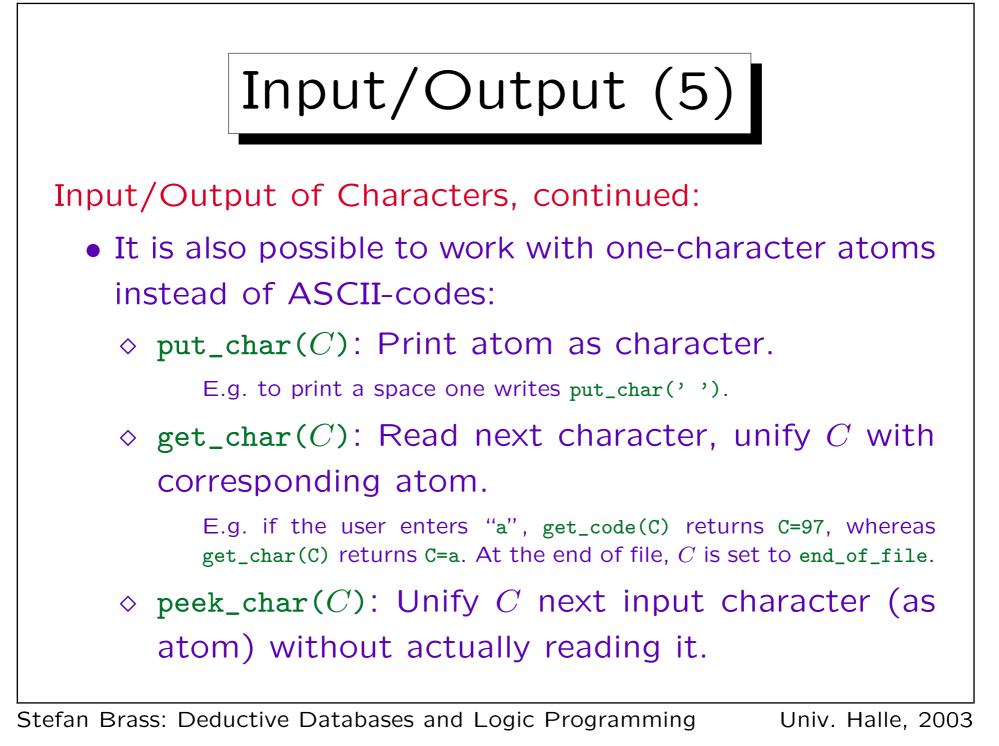






Input/Output (4)

	0	1	2	3	4	5	6	7	8	9
0	NULL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US		!	11	#	\$	%	&	>
40	()	*	+	,	-	•	1	0	1
50	2	3	4	5	6	7	8	9	•	•
60	<	=	>	?	Q	Α	В	С	D	E
70	F	G	Η	I	J	K	L	Μ	Ν	0
80	Р	Q	R	S	Т	U	V	W	Х	Y
90	Z	Γ	\backslash]	^	_	6	a	b	С
100	d	е	f	g	h	i	j	k	1	m
110	n	0	р	q	r	S	t	u	V	W
120	x	У	Ζ	{		}	~	DEL		

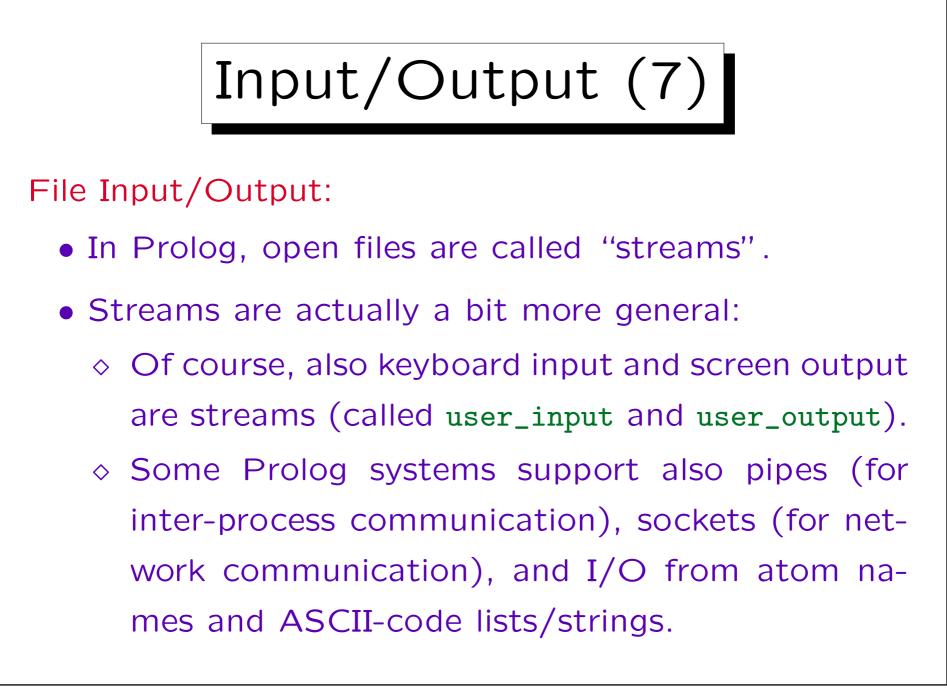


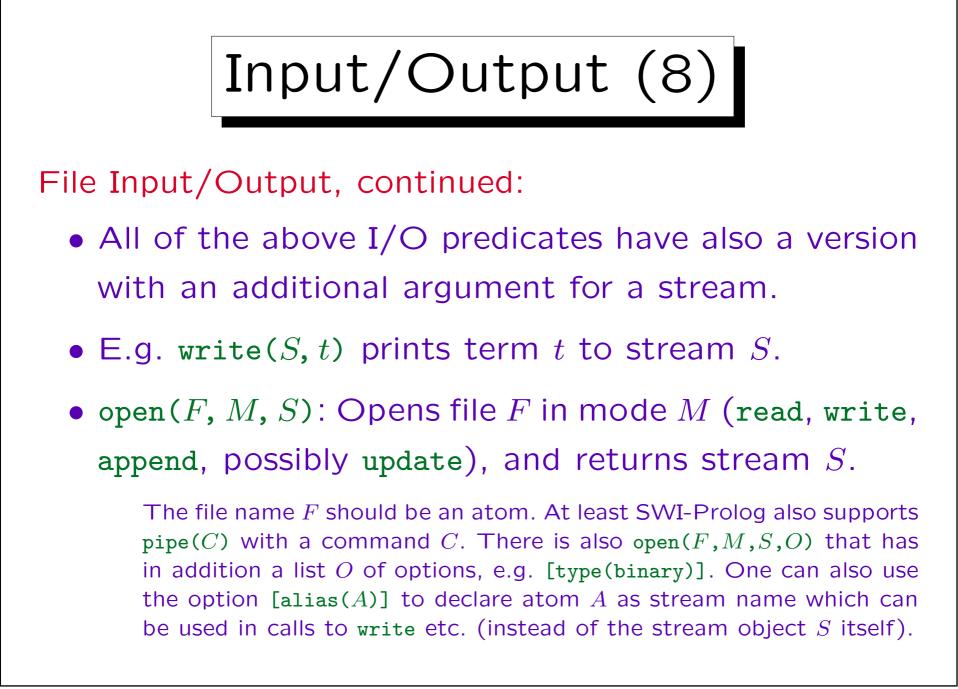


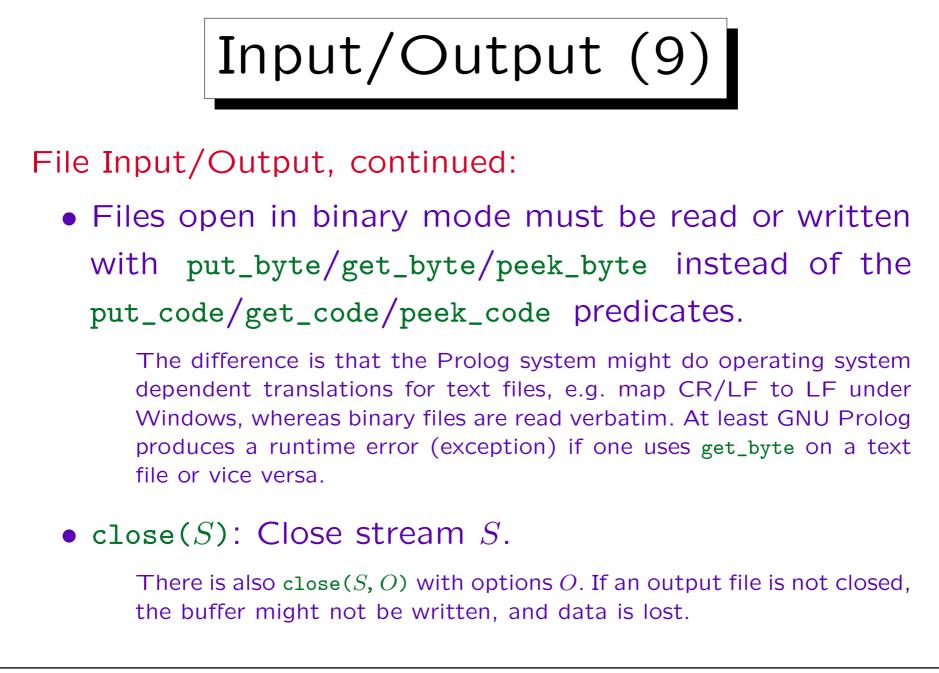
Input/Output of Characters, continued:

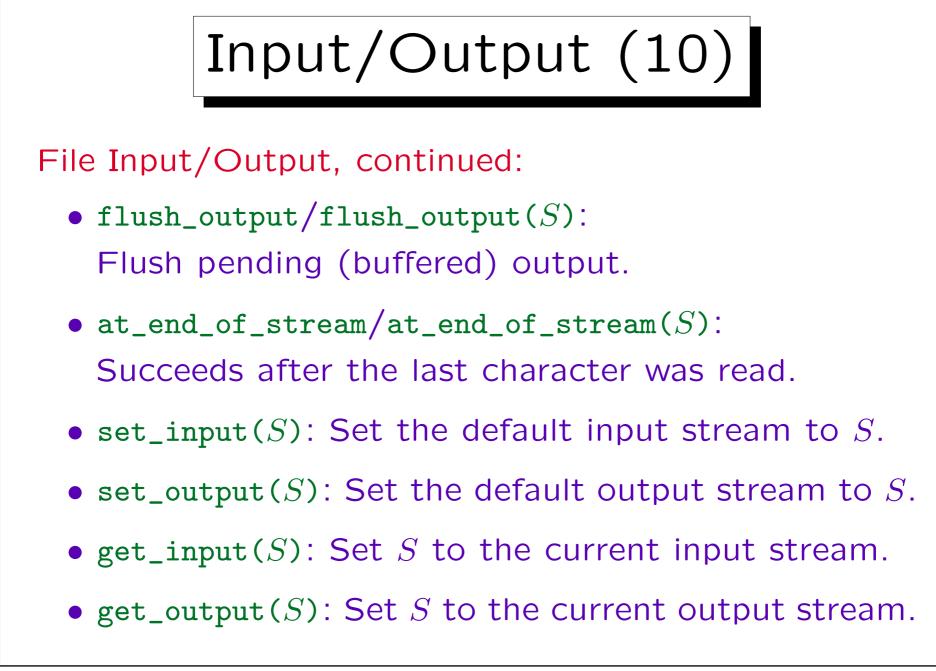
- The above get_*-predicates normally wait for an entire line of input from the keyboard.
- In contrast to read, it is not necessary to finish the input with ".", Enter/Return suffices.
- Every Prolog system has a way to read characters without buffering, but that is system dependent.

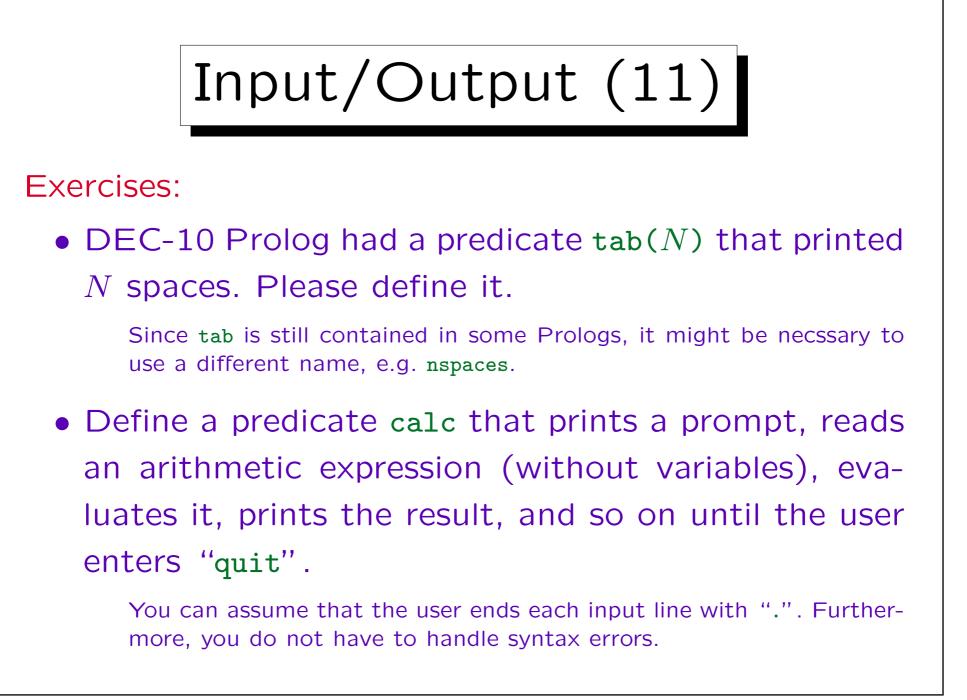
E.g. in SWI Prolog, use get_single_char. In GNU Prolog, use get_key.













- A, B: A and B (conjunction).
- A; B: A or B (disjunction).

Conjunction binds stronger than disjunction (";" has priority 1100, "," has priority 1000). One can use parentheses if necessary. Disjunction is not strictly needed, one can use several rules instead.

- true: True (always succeeds).
- fail: False (always fails).

Obviously, this can only be interesting with previous side effects (or the cut). Examples are shown in the next chapter.

• repeat: Always succeeds, also on backtracking.

This can be defined as repeat. repeat :- repeat.



• !: Ignore all previous alternatives in this predicate activation (cut, see next Chapter).

This means that no further rules for the same predicate will be tried, and no further solutions for all body literals to the left of the cut.

• $A \rightarrow B_1$; B_2 : If A, then B_1 , else B_2 .

This really means $(A \rightarrow B_1)$; B_2 . The arrow "->" has priority 1050, disjunction ";" has priority 1100.

• $A \rightarrow B$: If A, then B, else fail.

This is equivalent to A, !, B.

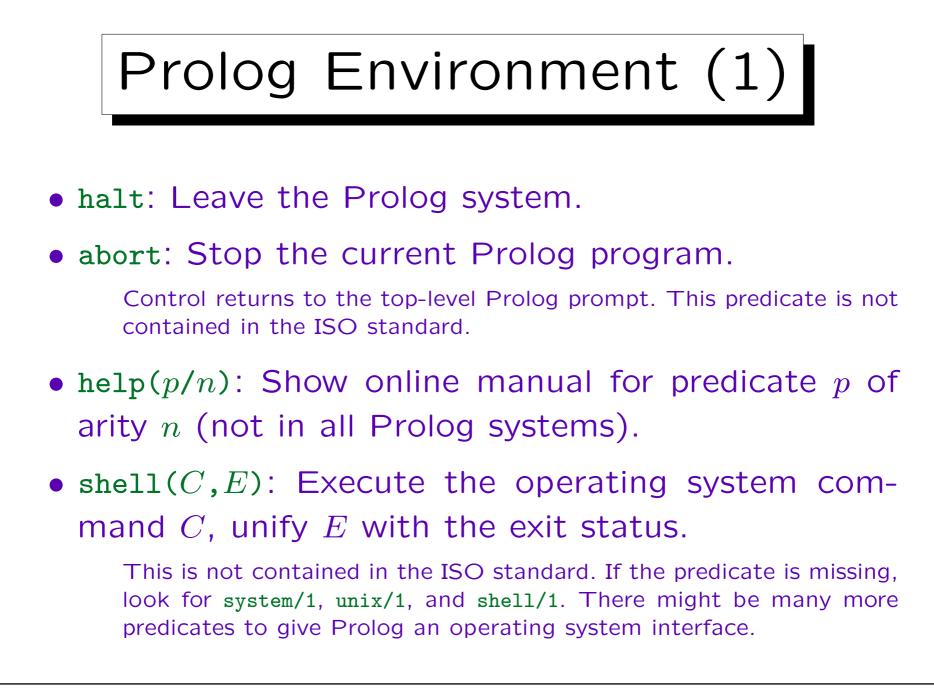
• once(A): Compute only first solution for A.

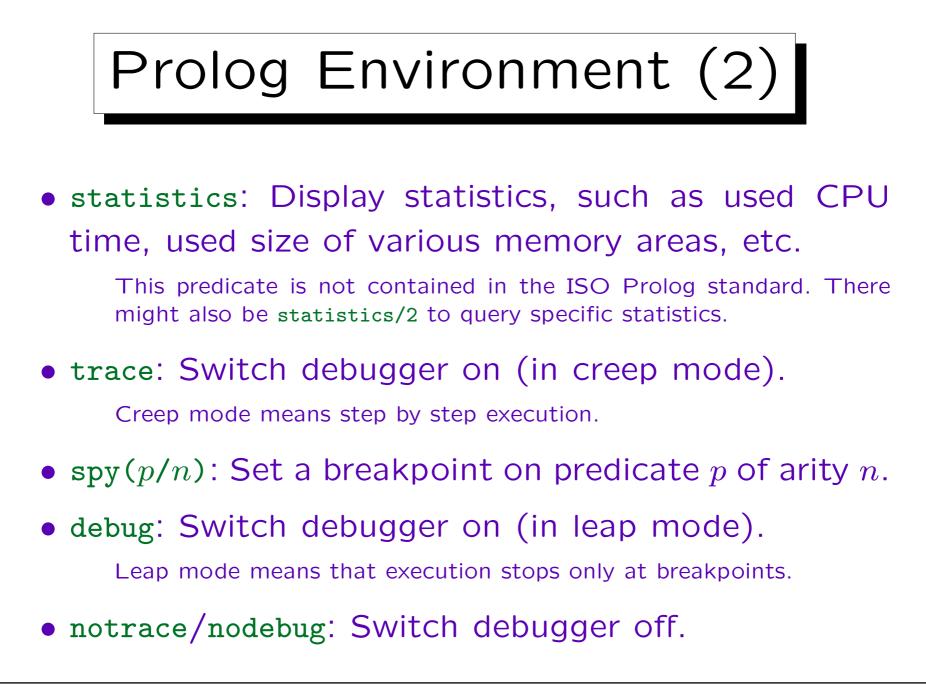
Stefan Brass: Deductive Databases and Logic Programming

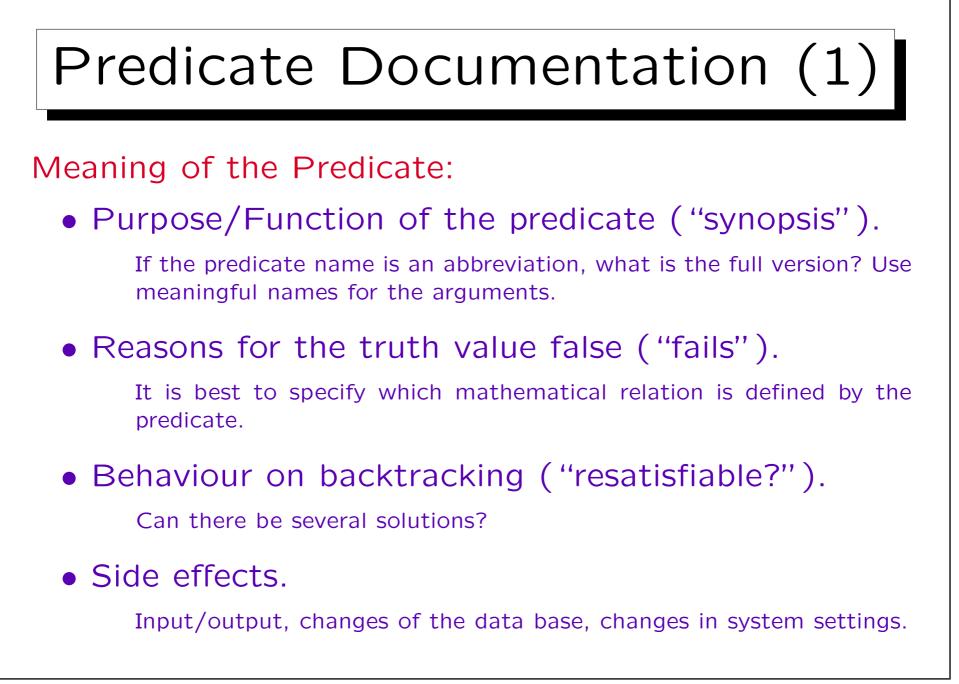
Control (3)

• \uparrow A: A is not provable (fails).

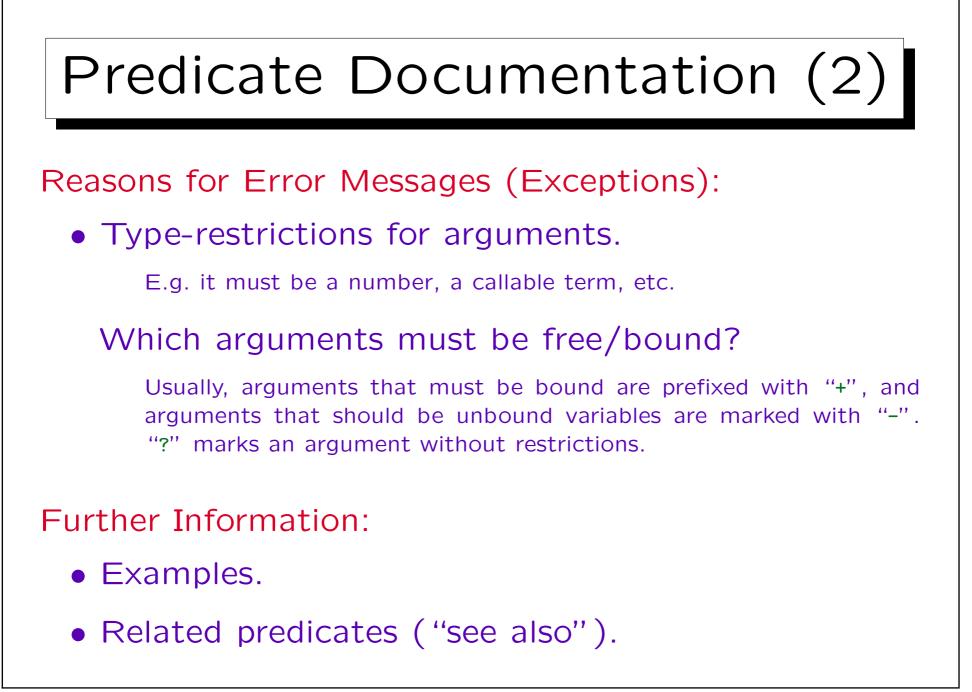
This is called negation as failure. It is not the logical negation, because Prolog permits only to write down positive knowledge. Negation as failure behaves non-monotonically, whereas classical predicate logic is monotonic: If one adds formulas, one can at least prove everything that was provable earlier. Some Prologs also understand not A.







Stefan Brass: Deductive Databases and Logic Programming





- 1. Built-In Predicates, Binding Patterns
- 2. Important Built-In Predicates in Prolog
- 3. Range-Restriction, Allowedness
- 4. Function Symbols and Built-In Predicates

Motivation (1)

- In deductive databases, query evaluation is done by applying the T_P -operator iteratively to compute the minimal model (with certain optimizations).
 - Computing the entire minimal model would not be goal-directed. Of course, one should compute only facts that are important for the query. This problem is solved by the magic set transformation: Given a logic program and a query, it computes a new logic program that has the same answer, but implies only facts relevant to the query. See Chapter 6.
- The allowed rules must be restricted so that immediate consequences can effectively be computed.



• For example, computing immediate consequences for a rule like the following would be difficult:

 $p(X,Y) \leftarrow q(X).$

• The possible values for Y depend on the domain: All data values can be inserted, often this set infinite, and maybe not even explicitly known.

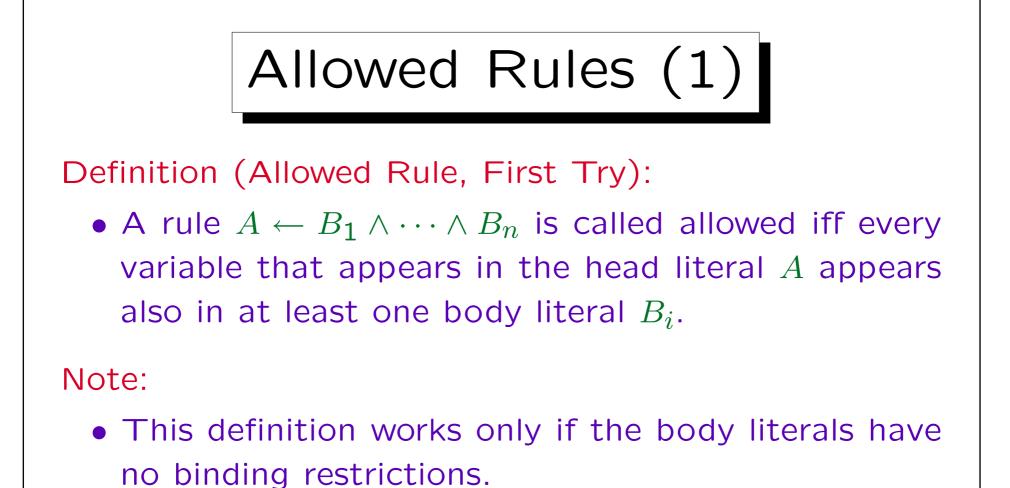
Normally, one works with the Herbrand universe that consists of all terms which can be constructed from the constants and function symbols appearing in the program. Then one can add a completely unrelated fact, in which a new constant appears, and thereby change the extension of p. That is a strange behaviour.

Motivation (3)

- Given e.g. q(a), one could derive the "fact" p(a, Y).
- One problem with this is that variables cannot be easily represented in database relations.

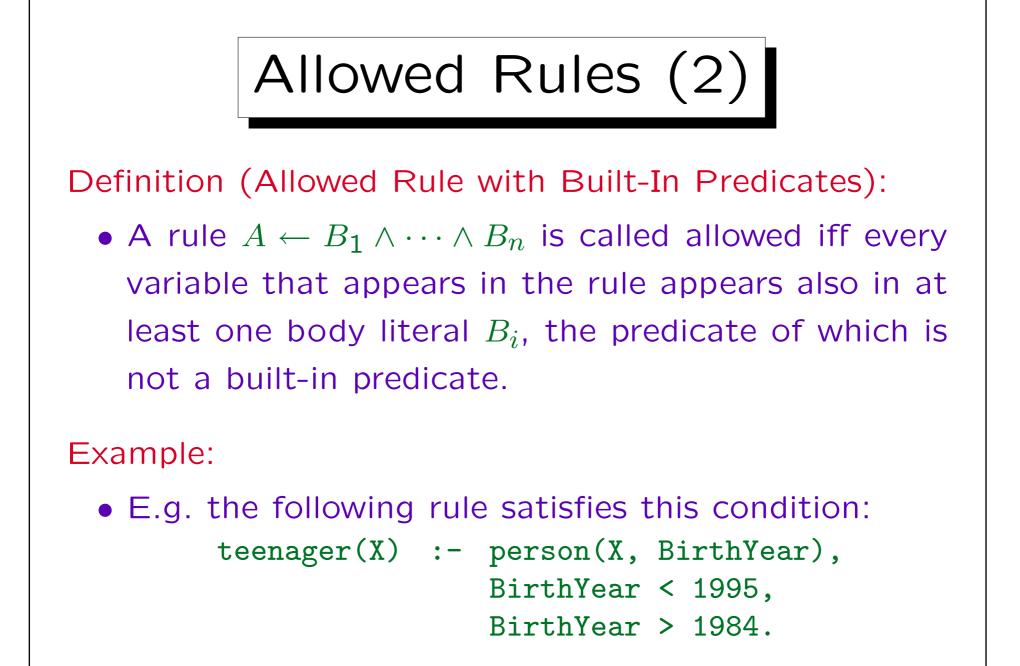
As long as one does not use function symbols, derived predicates should correspond to views in relational databases. Furthermore, at least some prototypes did actually use a relational database system for query evaluation: Then storing an intermediate result in a temporary relation for p is at least difficult.

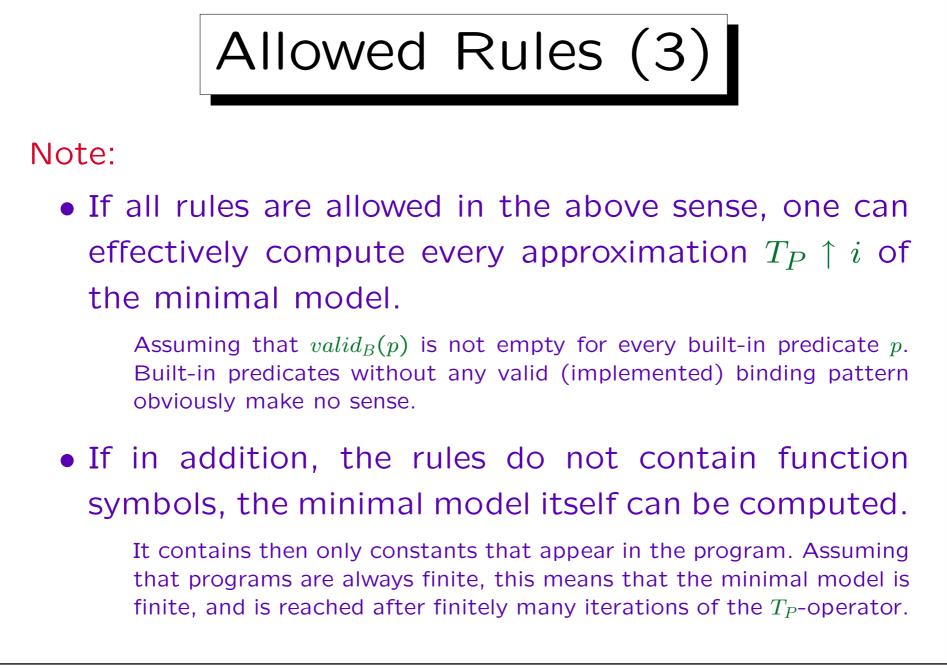
 In contrast to Prolog, deductive databases normally use only one-directional, restricted form of unification ("matching"): Variables appear only in rules, body literals are matched with variable-free facts.

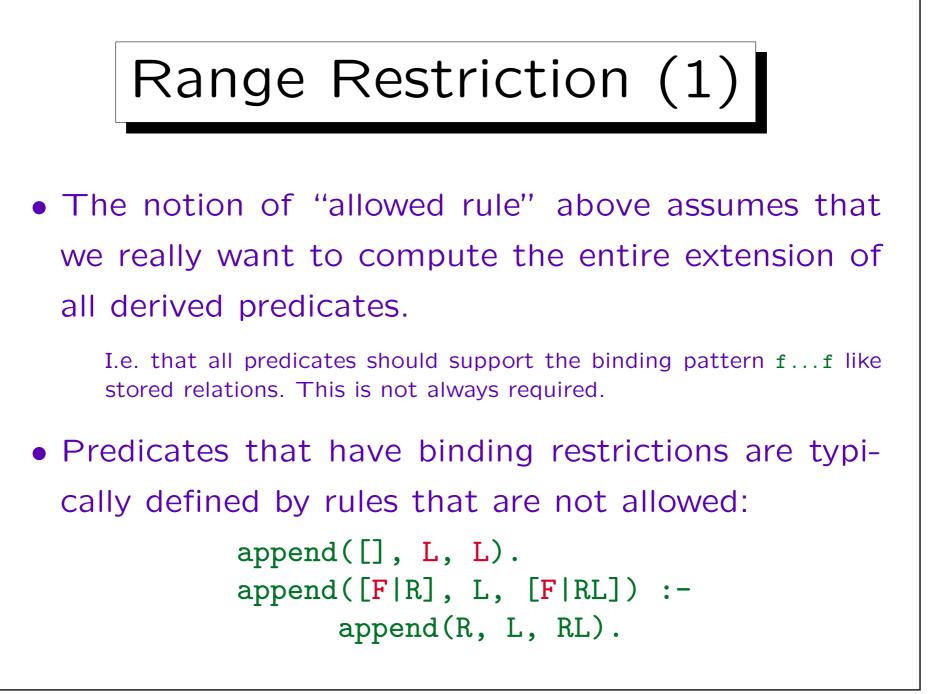


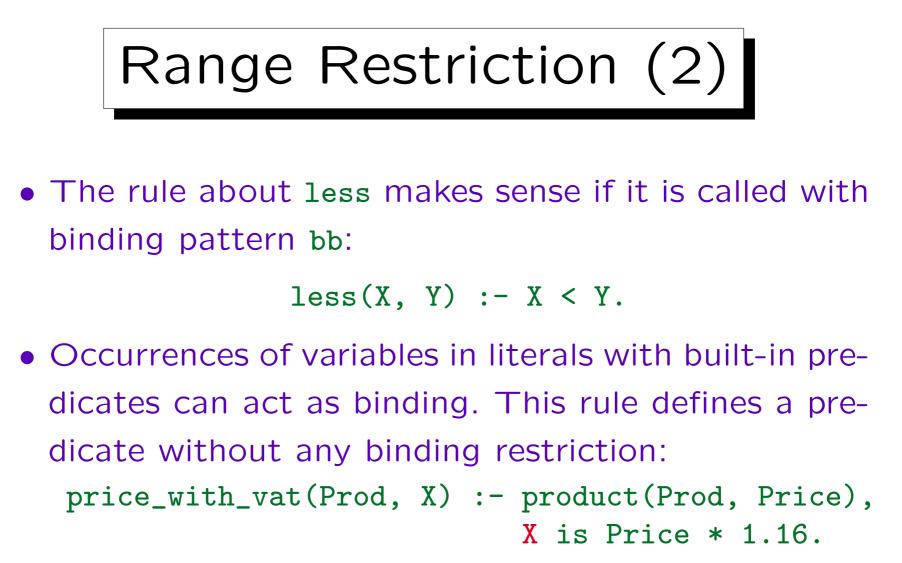
• E.g. one cannot compute all consequences of the following rule, although it statisfies the condition:

less(X, Y) :- X < Y.

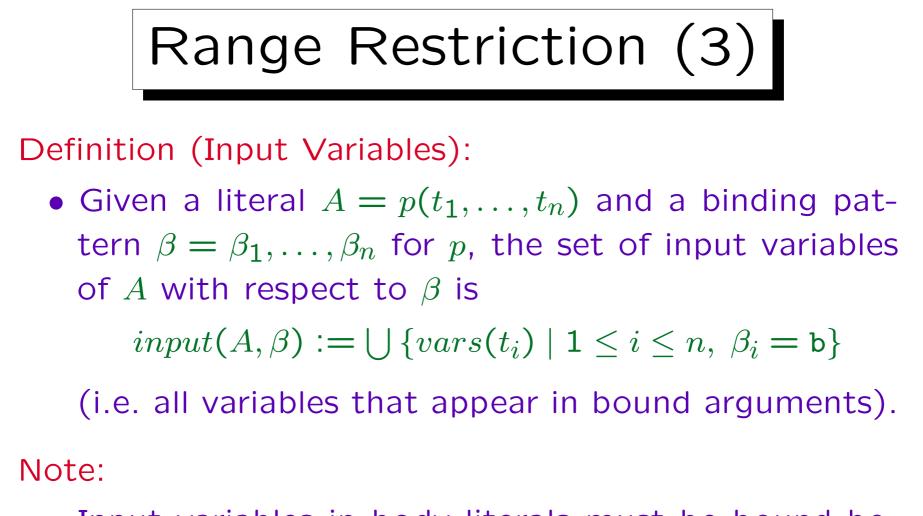




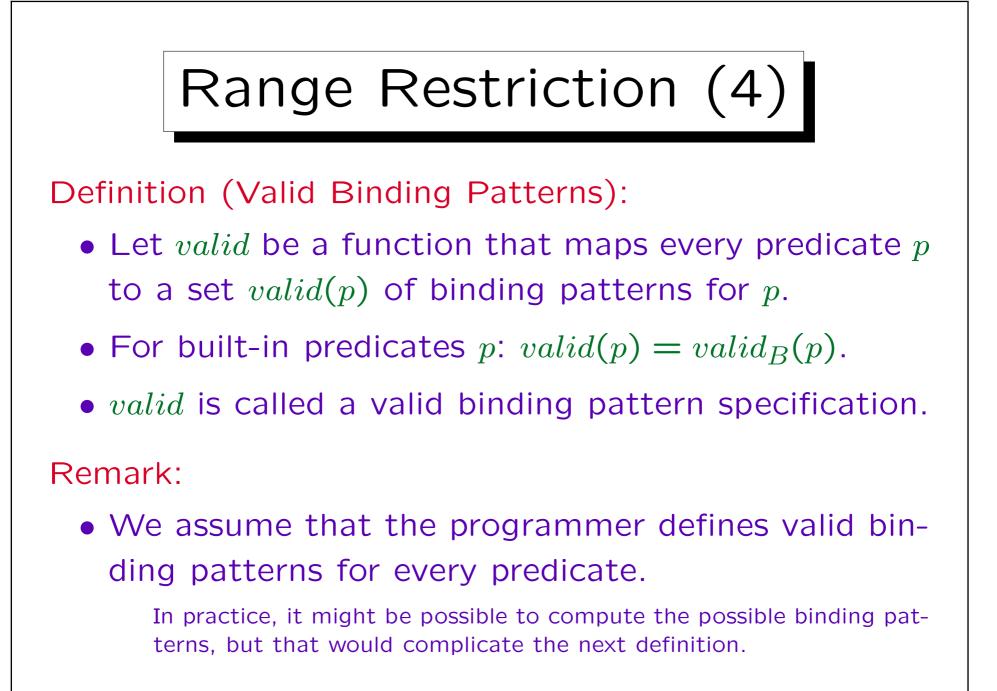


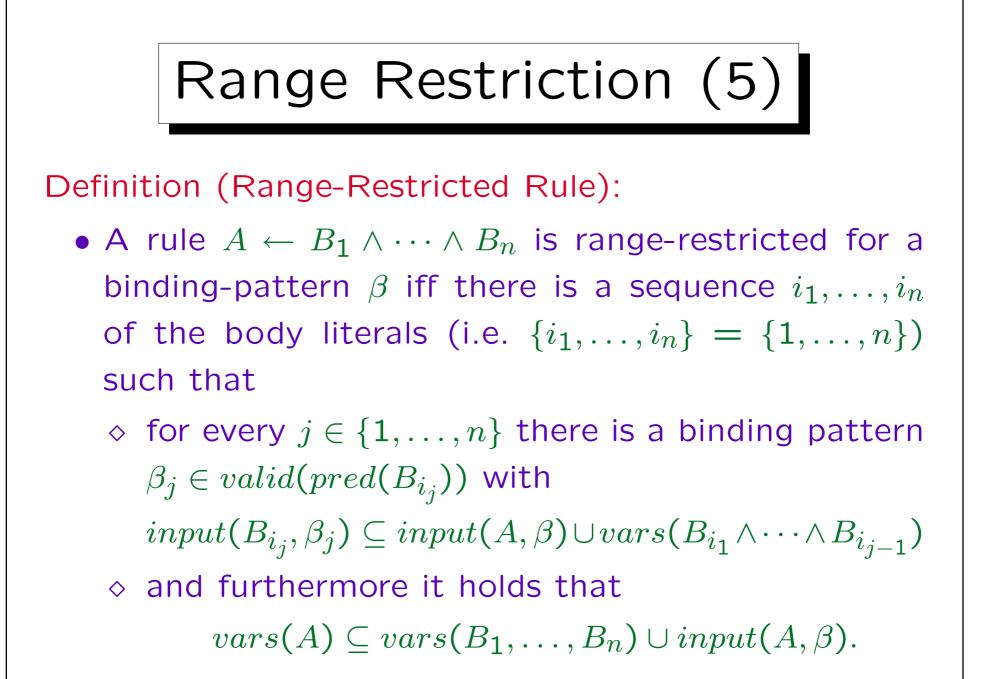


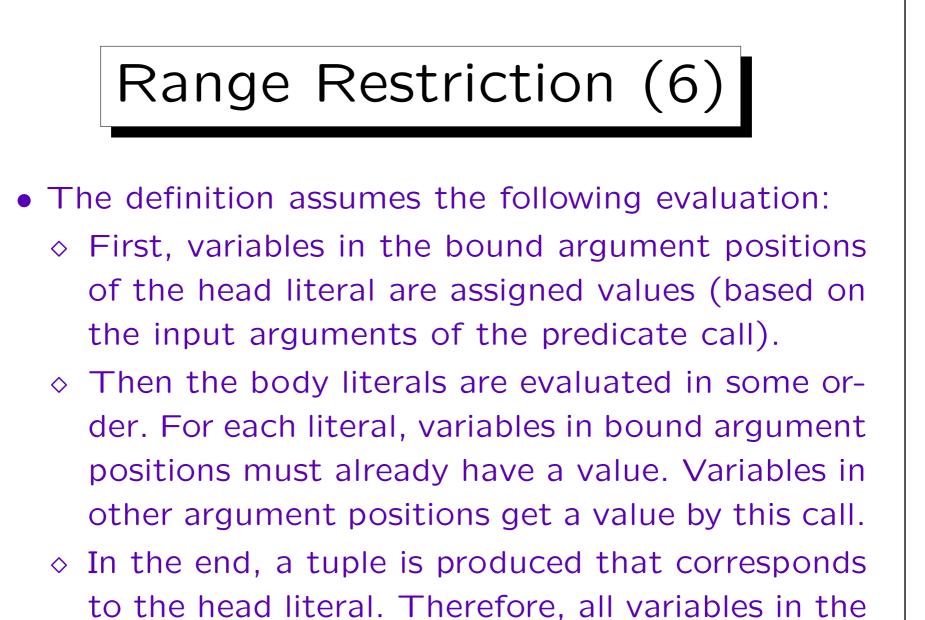
• All this shows that the allowedness requirement is too restrictive.



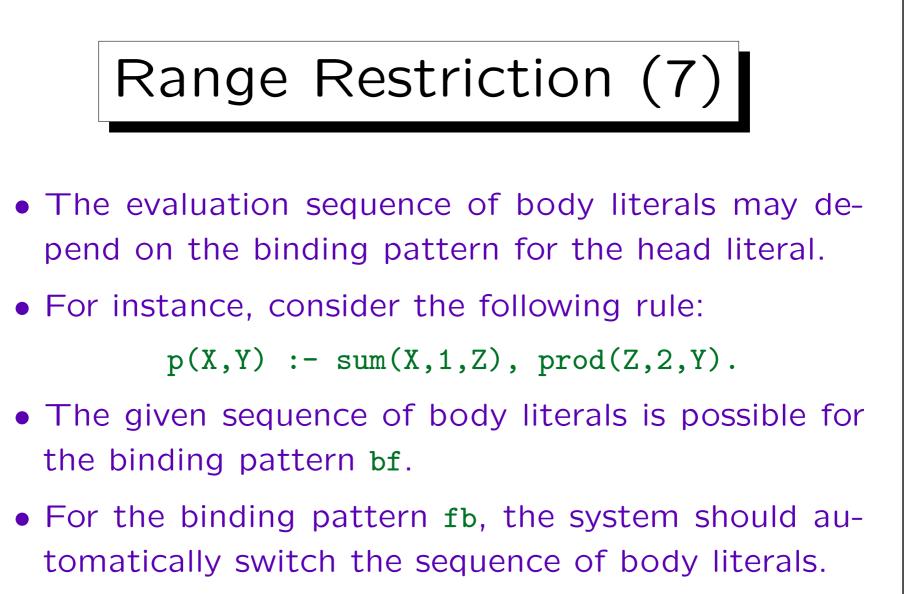
• Input variables in body literals must be bound before the literal can be called. Input variables in head literals are bound when the rule is executed.



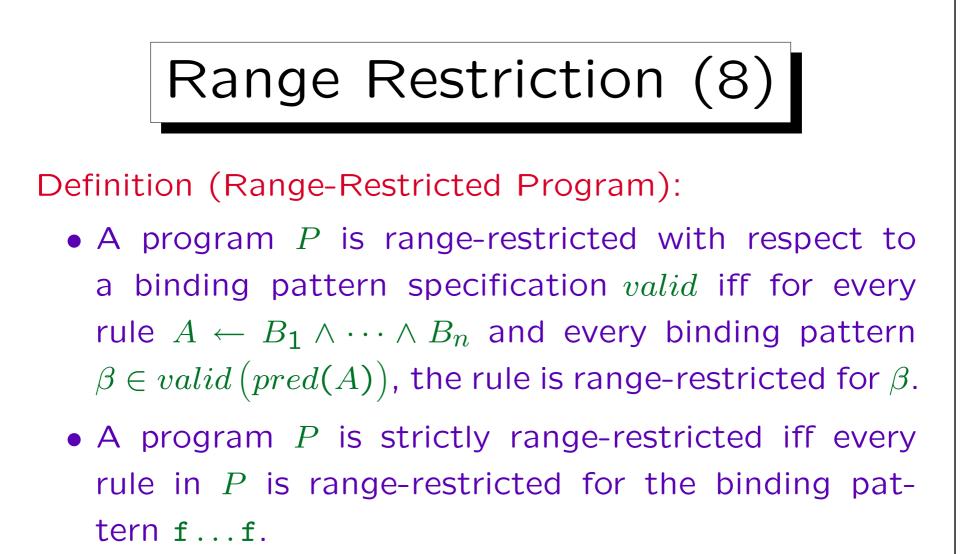




head literal must now have a value.



The Datalog programmer does not necessarily know the binding pattern. Furthermore, it would be bad style to double the rule.

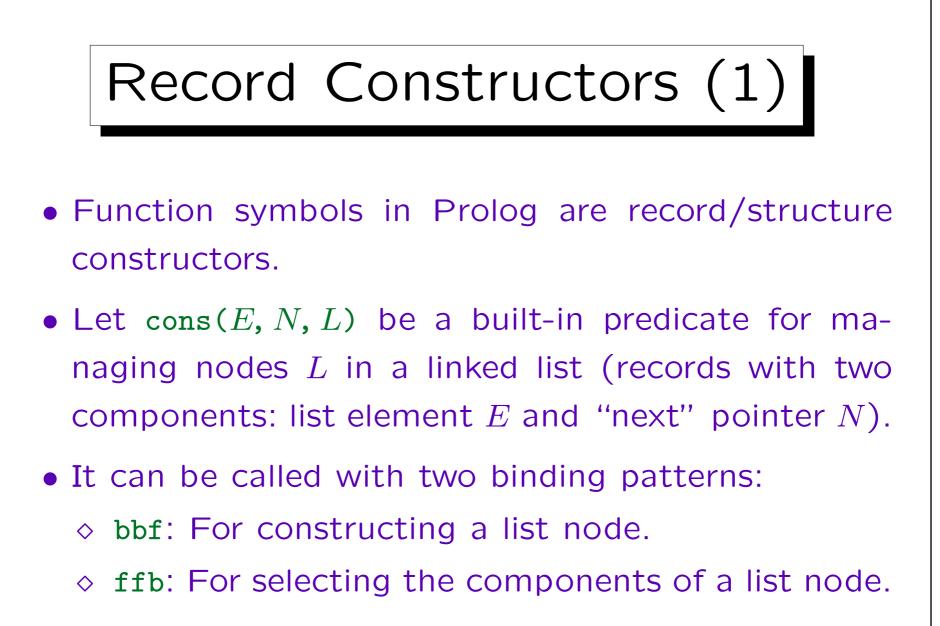


Strict range restriction is the requirement for the T_P -operator to be directly executable. As we will see, the magic set transformation turns a range-restricted program into a strictly range-restricted program.

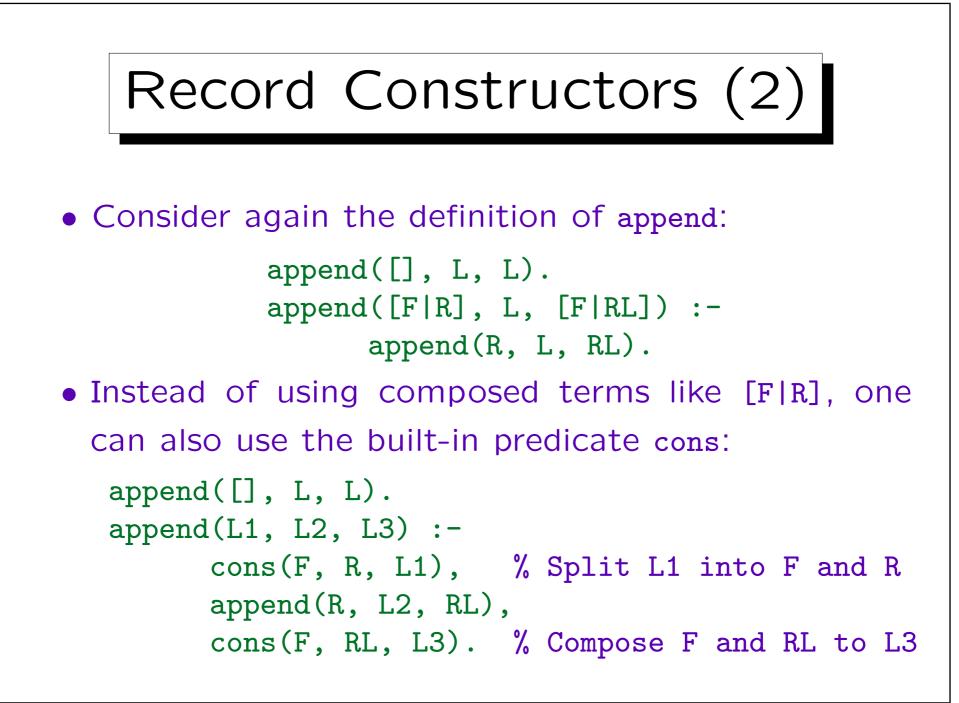


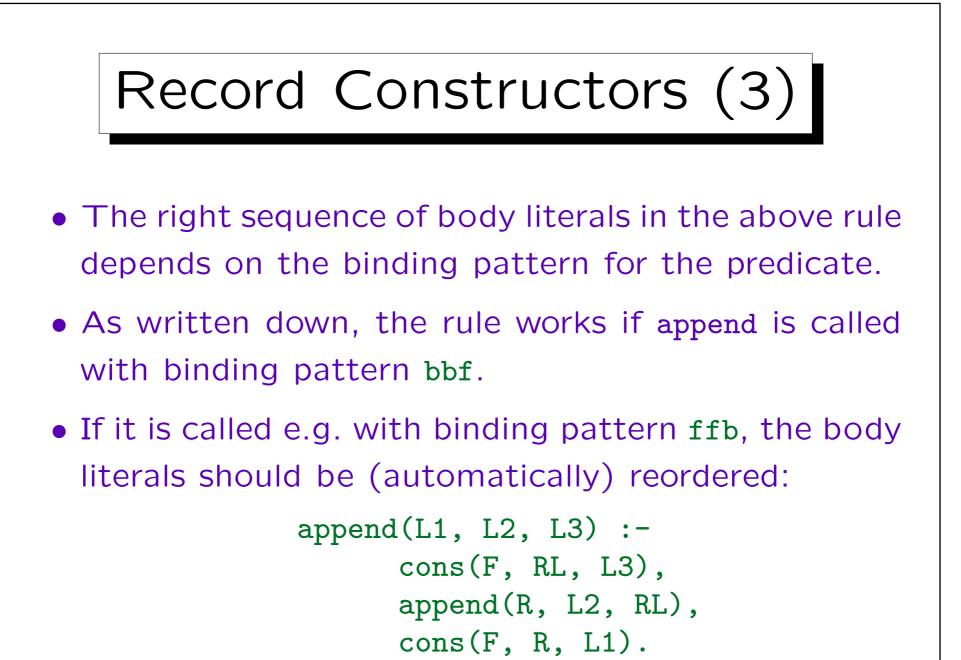
- 1. Built-In Predicates, Binding Patterns
- 2. Important Built-In Predicates in Prolog
- 3. Range-Restriction, Allowedness

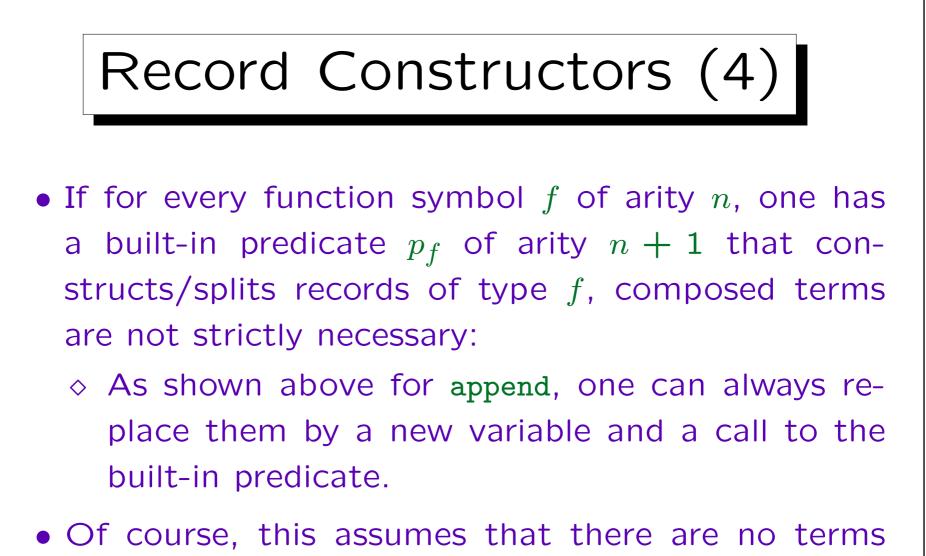
4. Function Symbols and Built-In Predicates



• Note that cons(E, N, L) actually means L=[E|N].

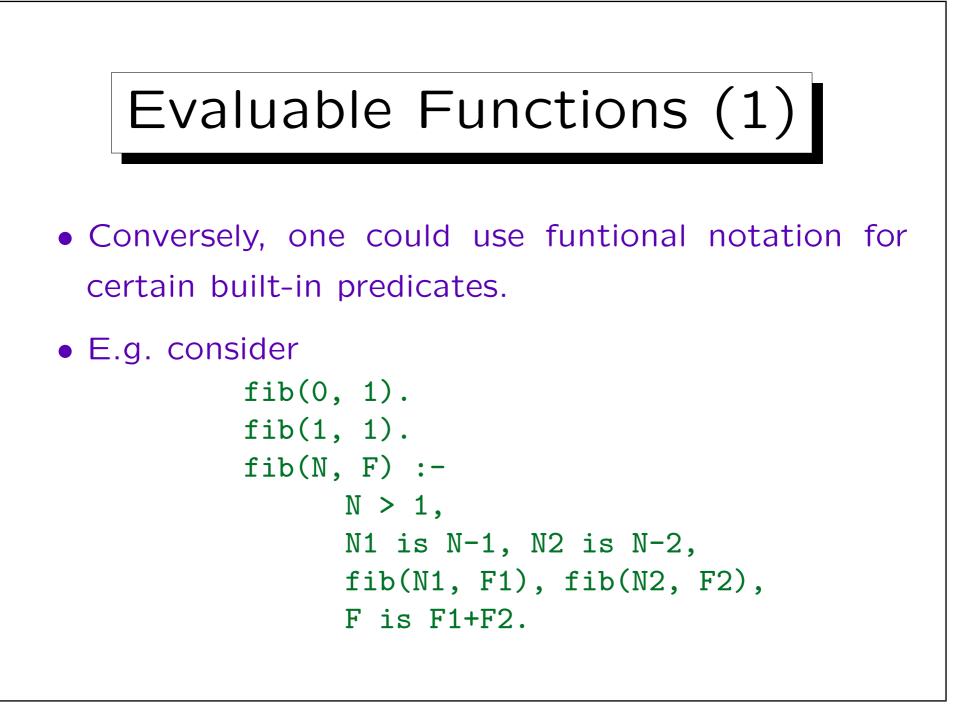


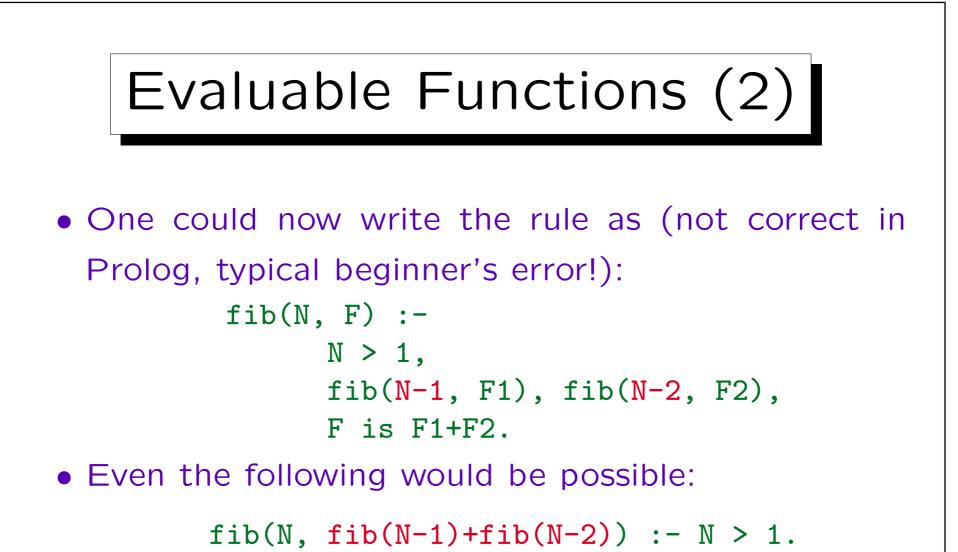




with "holes" (variables) in them.

In deductive databases, this is normally the case.





• A preprocessor could translate both back to the standard predicate notation.

