Advanced Logic Programming

https://sewiki.iai.uni-bonn.de/teaching/lectures/alp/2017/

Dr. Günter Kniesel

Computer Science Department III
University of Bonn, Germany

gk@cs.uni-bonn.de

universitätbonn

Chapter 1.
Prolog in a Nutshell

- Last updated: July 10, 2017 -

Prolog

- Prolog stands for "Programming in Logic".
- It is the most common logic-based programming language.

Bits of history

- 1965
  - John Alan Robinson develops the resolution calculus – the formal foundation of automated theorem provers
- 1972
  - Alain Colmerauer (Marseilles) develops the first Prolog interpreter
- mid 70th
  - David D.H. Warren (Edinburg) develops first Prolog compiler
    - ⇨ Warren Abstract Machine (WAM) as compilation target → like Java byte code
- 1981-92
  - „5th Generation Project“ in Japan boosts adoption of Prolog world-wide

Prolog in a Nutshell

A *very* quick tour of basic Syntax and Semantics

It‘s all about truth

Predicates ▶ Map entities to truth values

Schema of a function declaration

Function symbol (a name) : Domain (a crossproduct of sets) → Codomain (a set)

A predicate declaration is the declaration of a boolean function.

Sample predicate declaration

```
isFatherOf: Person x Person → Bool
```

A predicate definition specifies concrete values from the domains

Predicate definition via truth table

```
(kurt,peter) → true
(peter,paul) → true
(peter,hans) → true
(peter,peter) → false
```

Facts are the Prolog syntax for a truth table that only contains the things that map to „true“

Predicate definition via facts

```
isFatherOf(kurt,peter).
isFatherOf(peter,paul).
isFatherOf(peter,hans).
```

Facts ▶ Statements that are true


```
isFatherOf('Kurt','Peter').
```

means

It is true that

'Kurt'
is father of
'Peter'

Rules ▶ Implications of what we know to be true


```
isGrandfatherOf('Kurt','Paul') :-
    isFatherOf('Kurt','Peter'),
    isFatherOf('Peter','Paul').
```

means

The truth of

'Kurt' is grandfather of 'Paul'

is implied by the truth of

'Kurt' is father of 'Peter'

and

'Peter' is father of 'Paul'

Variables▶ Placeholders for domain values


```
isGrandfatherOf(G,C) :-
    isFatherOf(G,F) ,
    isFatherOf(F,C) .
```

means

The truth of

G is grandfather of C

is implied by the truth of

G is father of F

and

F is father of

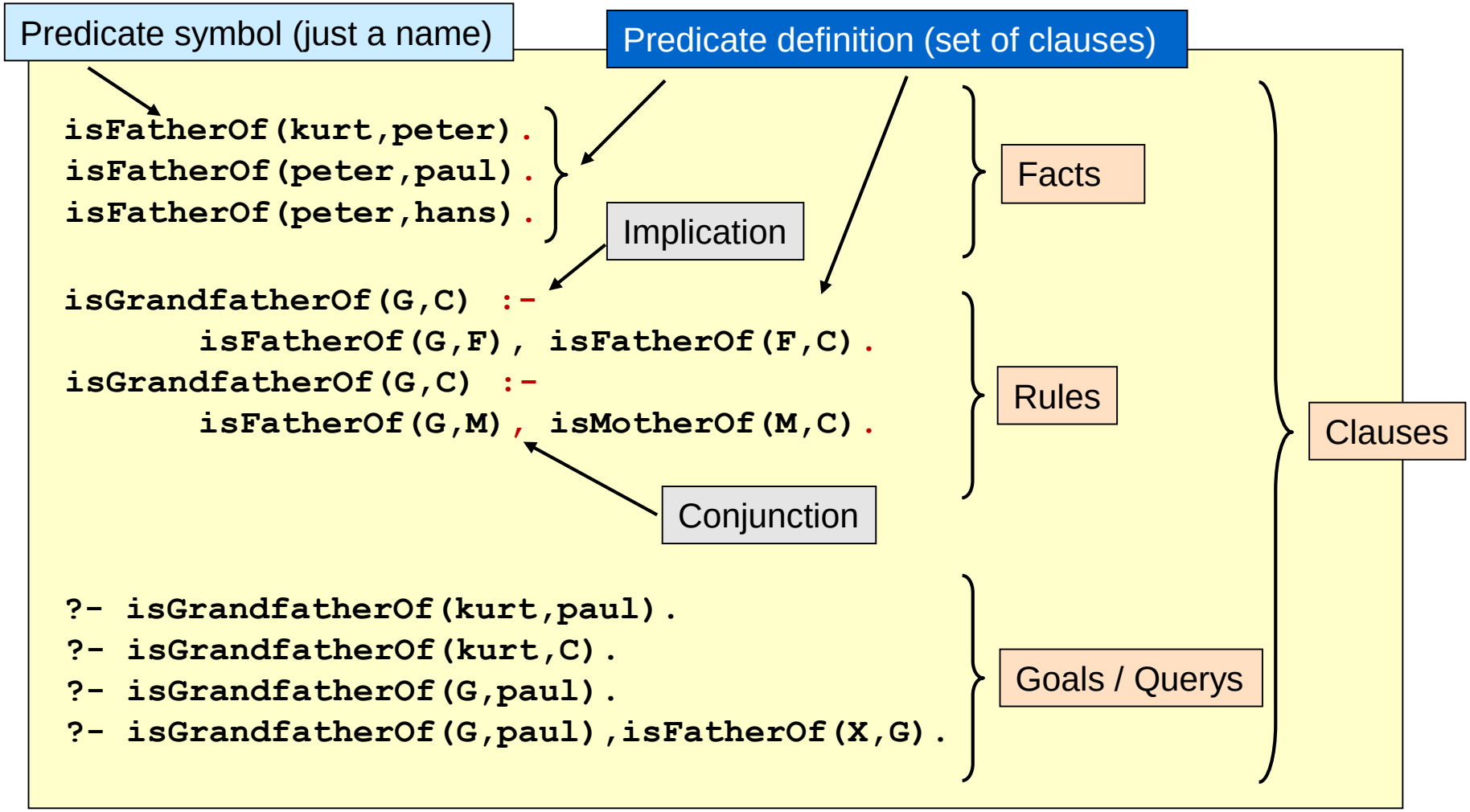
Prolog Syntax

Predicates

Clauses, Rules, Facts

Terms, Variables, Constants, Structures

Clauses ▸ Facts, Rules, Queries



Recursion

- Prolog predicates may be defined recursively

- A predicate is recursive if one or more rules in its definition refer to itself.

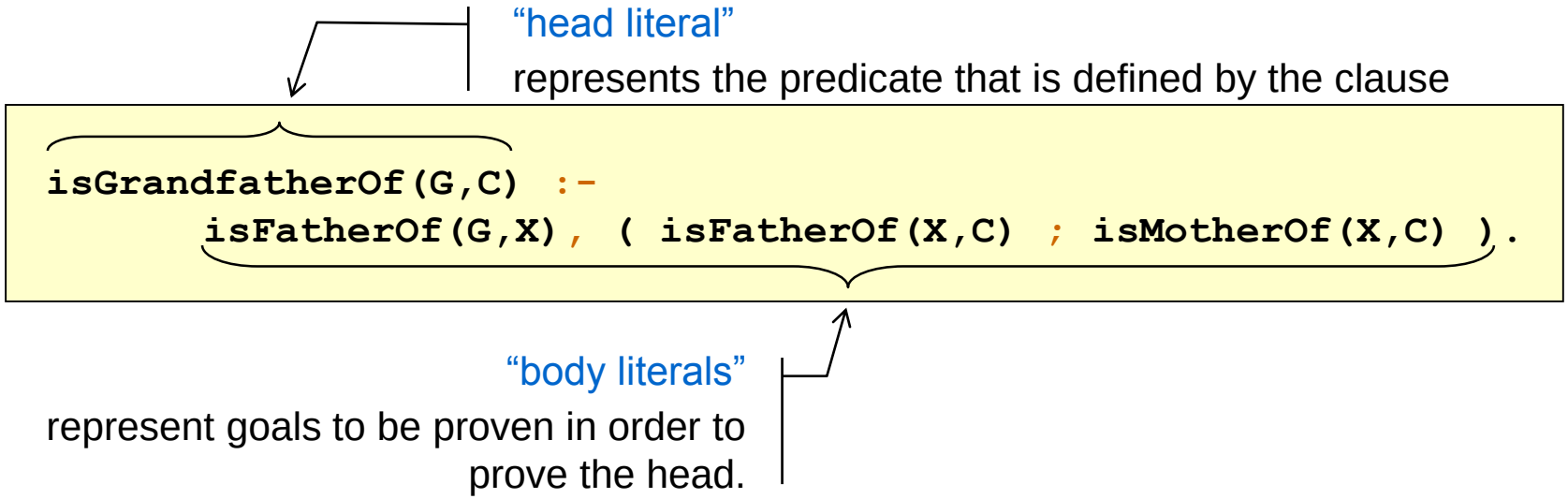
```
ancestor(Anc,Desc):- parent(Anc,Desc).
ancestor(Anc,Desc):- parent(Anc,X), ancestor(X,Desc).
```

- What does the definition of ancestor/2 by the above clauses mean?
  1. "Anc is a parent of Desc" *implies that* "Anc is an ancestor of Desc"
  2. "Anc is a parent of X *and* X is an ancestor of Desc" *implies that* "Anc is an ancestor of Desc"

- Homework
  - Try ancestor/2 together with your own parent/2 predicate definition
  - Does it give all the expected results?
  - Does it give only expected results?

Rules

- Rules consist of a head and a body.



- Facts are just syntactic sugar for rules with the body "true".

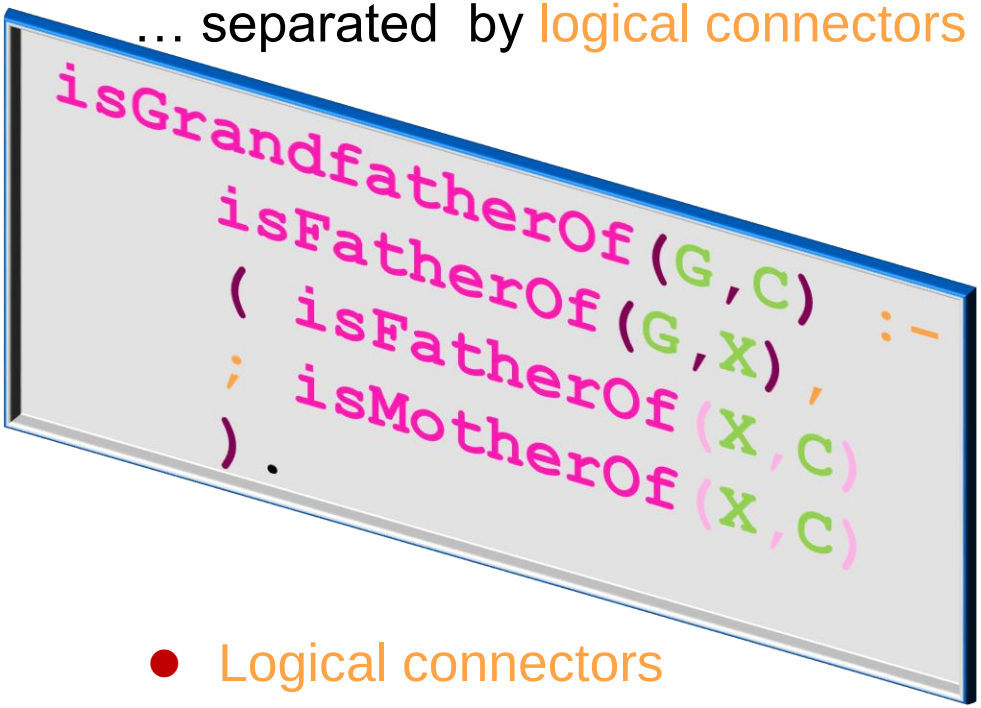
```
isFatherOf(peter,hans).
```

```
isFatherOf(peter,hans) :- true.
```

these clauses are equivalent

Program ▶ Clause ▶ Literal ▶ Argument

- Prolog programs consist of clauses (see previous slide)
- Clauses consist of literals …
  - Head literal
  - Zero or more body literals
- … separated by logical connectors



- Literals consist of
  - a predicate symbol
  - punctuation symbols
  - arguments
- Punctuation symbols
  - opening round brace "("
  - comma ","
  - closing round brace ")"
- Logical connectors
  - implication ":-"
  - conjunction ","
  - disjunction ";"

Terms

Terms are the arguments of literals. They may be

- Variables `X,Y, Father, Method, Type, _type, _Type, ...`
- Constants Numbers, Strings, ...
- Function terms `person(stan,laurel), +(1,*(3,4)), ...`

Terms are the only data structure in Prolog!

The only thing one can do with terms is unification with other terms!

→ All computation in Prolog is based on unification.

Variables ▶ Syntax

- Variables start with an upper case letter or an underscore '_'.

```
Country   Year   M   V   _45   _G107   _europe    _
```

- Anonymous Variables ('_')
  - For irrelevant values
  - “Does Peter have a father?” We neither care whether he has one or many fathers nor who the father is:

```
?- isFatherOf(_,peter).
```

Variables ▸Semantics

- The scope of a variable is the clause in which it appears
- Variables that appear only once in a clause are called singletons.
  - Mostly results of typos
  - SWI Prolog warns about singletons,
  - … unless you suppress the warnings
- All occurrences of the same variable in the same clause must have the same value!
  - Exception: the "anonymous variable" (the underscore)

```
isGrandfatherOf(G,C) :-
        isFatherOf(G,F),
        isFatherOf(F,C).
isGrandfatherOf(G,Child) :-
        isFatherOf(G,M),
        isMotherOf(M,Chil).


loves(romeo,juliet).
loves(john,eve).
loves(jesus,_Everybody).


?- classDefT(ID, _, 'Applet', _).
```

```
_Everybody
```

Intentional singleton variable, for which singleton warnings should be supressed.

Constants

- Numbers -17 -2.67e+021 0 1 99.9 512
- Atoms sequences of letters, digits or underscore characters '_' that
  - start with a lower case letter

  OR
  - are enclosed in simple quotes ('). If simple quotes should be part of an atom they must be doubled.

  OR
  - only contains special characters

```
ok:     peter   'Fritz'   'I don"t know!'    new_york    :-    -->
```

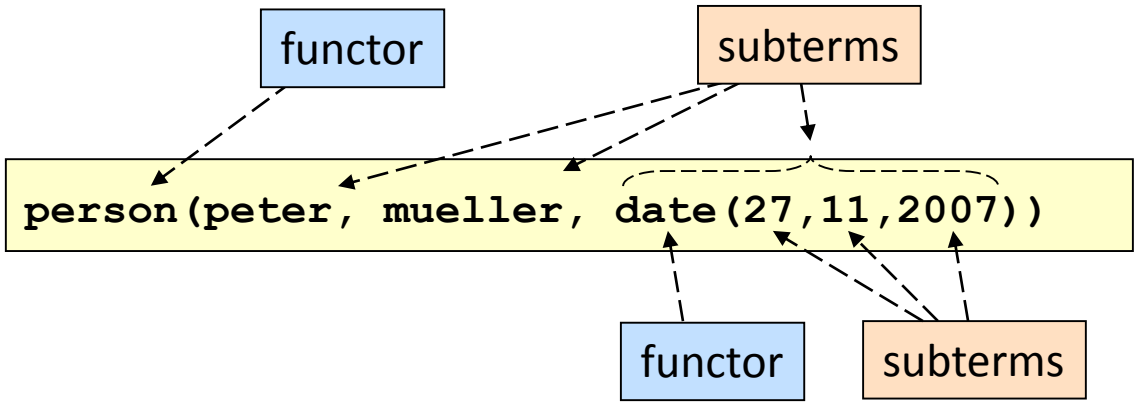
```
wrong:          Fritz     123     _xyz     new-york
```

- Remember: Prolog has no static typing!
  - So it is up to you to make sure you write what you mean.

Function Terms ('Structures')

Function terms are terms that are composed of other terms

- akin to "records" in Pascal (or objects without any behavior in Java)



- Arbitrary nesting allowed
- No static typing: `person(1,2,'a')` is legal!
- Function terms are not function calls! They do not yield a result!!!

Notation for function symbols: Functor/Arity, e.g. `person/3, date/3`

Example ▶ Employees

Without function terms

```
employee( tom, jones, 5000, 10 ).
employee( tim, james, 7000, 0  ).
employee( tam, junes, 6500, 15 ).
```

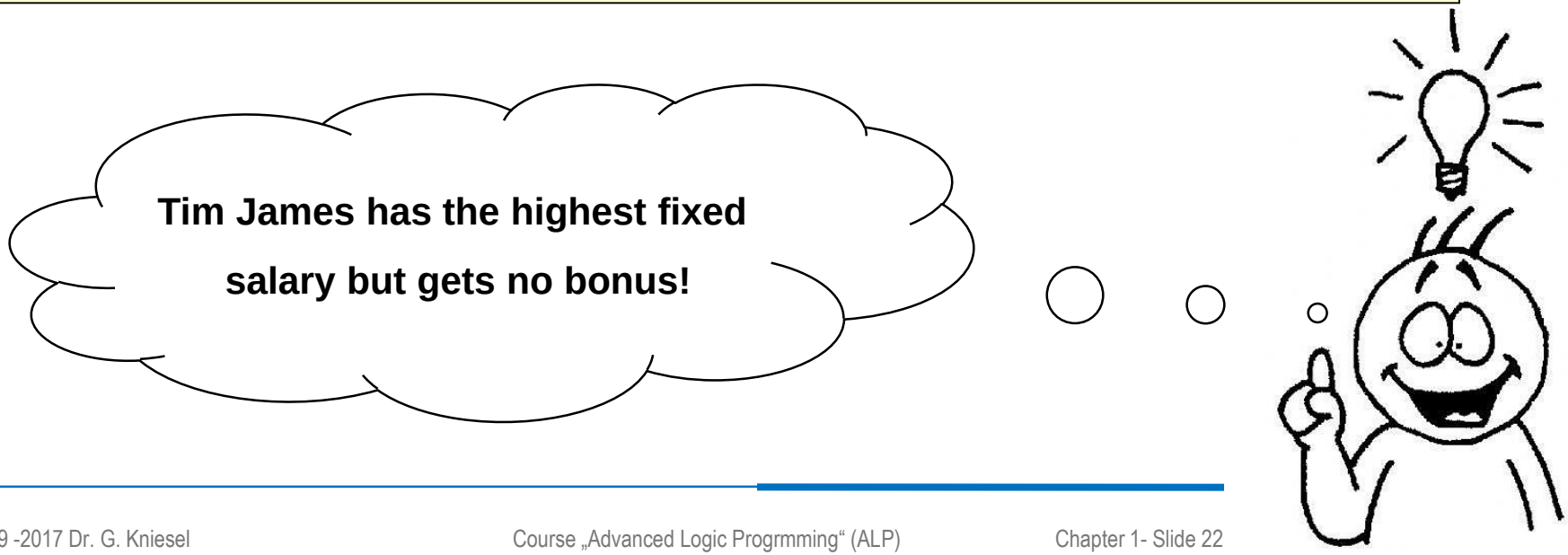
Concise but clumsy!

What do the arguments represent?

Example ▶ Employees

With function terms

```
employee( who(tom, jones), salary(5000, bonus(10,%)) ).
employee( who(tim, james), salary(7000, bonus( 0,%)) ).
employee( who(tam, junes), salary(6500, bonus(15,%)) ).
```



Lists – Recursive Structures with special Syntax

- Lists are denoted by square brackets "[]"

```
[]   [1,2,a]   [1,[2,a],c]
```

- The pipe symbol "|" delimits the initial elements of the list from its „tail“

```
[1|[2,a]]   [1,2|[a]]   [Head|Tail]
```

Strings

- Strings are enclosed in double quotes (")
  - `"Prolog"` (with double quotes) is a string
  - `'Prolog'` (with simple quotes) is an atom
  - `Prolog` (without any quotes) is a variable

Terms, again

- Terms are constants, variables or structures

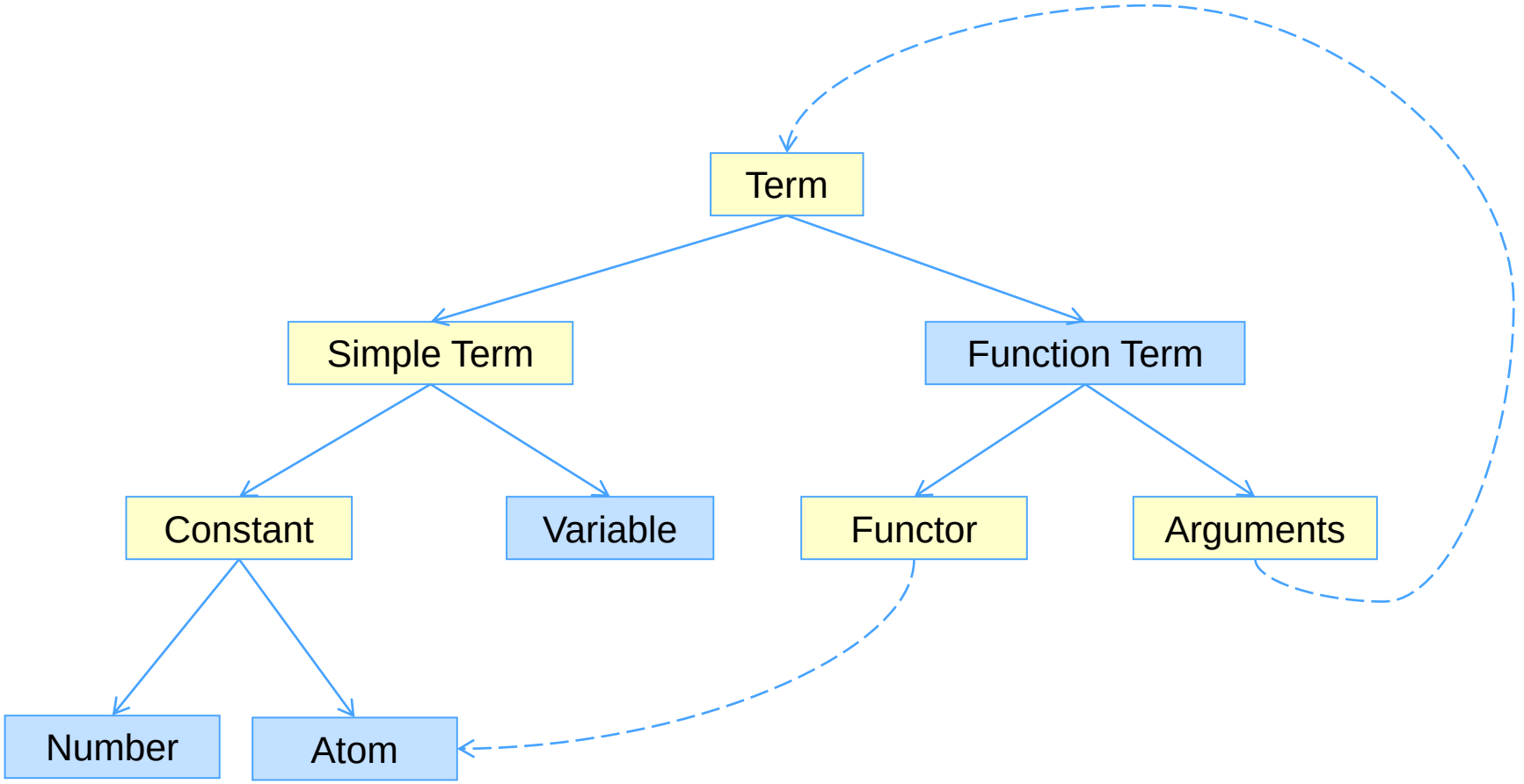
```
peter
27
MM
[europe, asia, africa | Rest]
person(peter, Name, date(27, MM, 2007))
```

- A ground term is a variable-free term

```
person(peter, mueller, date(27, 11, 2007))
```

Terms: Summary

Relations between the different kinds of term



Test Yourself

How would you represent the following hospital information?

- Patient Tom gets Aspirine at 8:00, 16:00 and 22:00 with water.
- Patient Tim gets Dimethylamine at 8:00
  and Insuline at 14:00 and 20:00
- … imagine more patients, each with different medications

You have 2 minutes to think about it for yourself with paper and pencil.

Using Prolog for Logic Puzzles

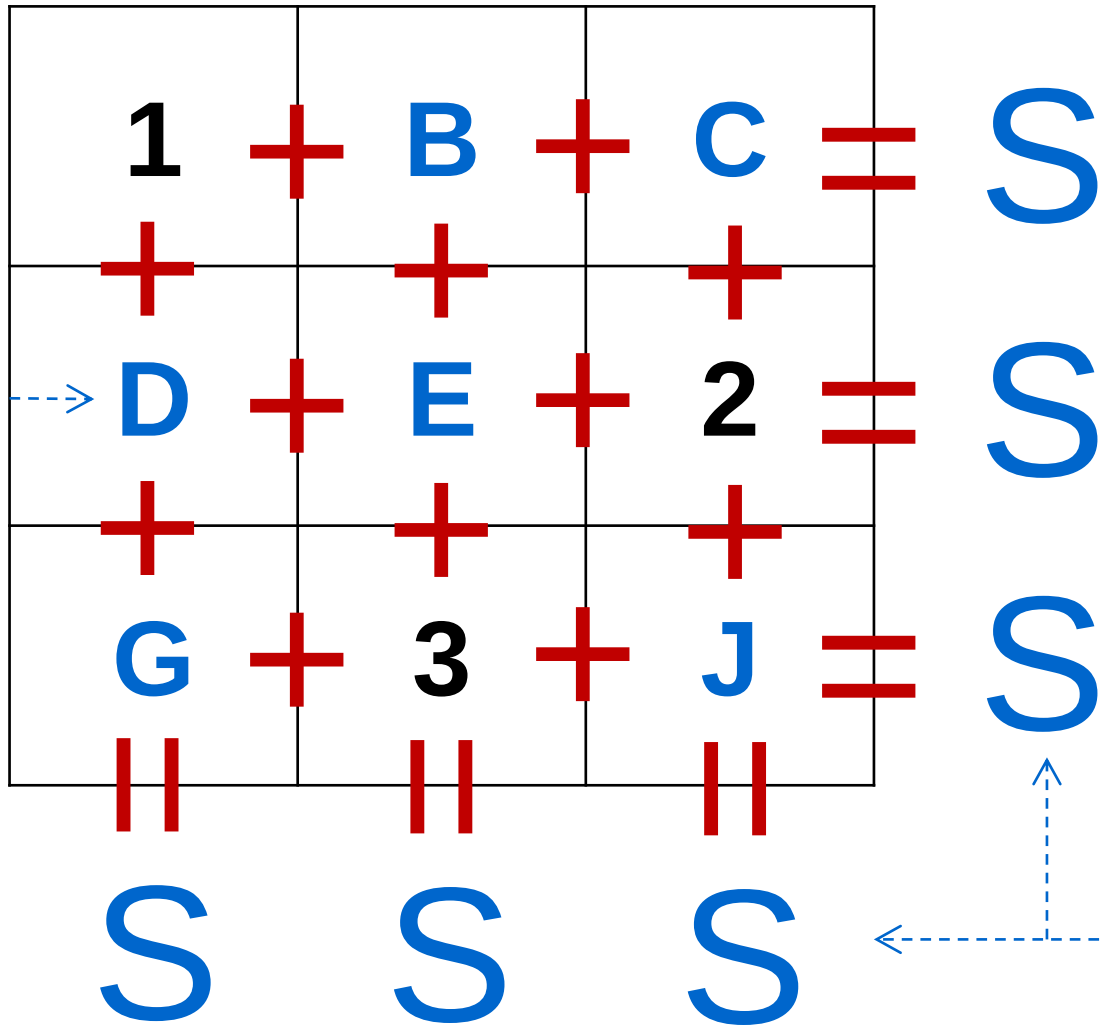
The Magic Square

Magic Square

1		
		2
	3	

Can I fill the square so that each line (row or column) has the same sum?

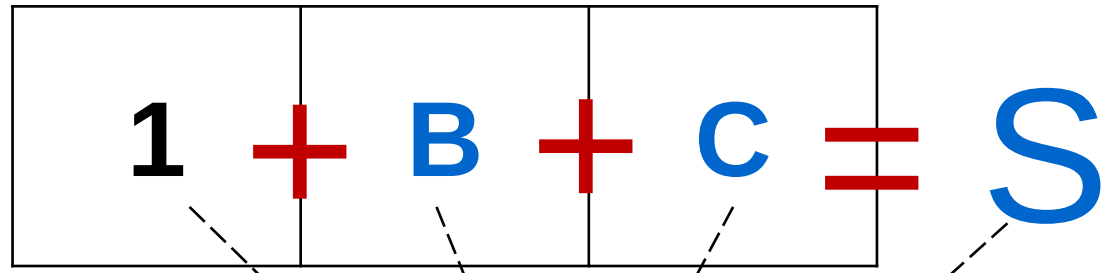
Abstract Problem Statement



Represent an unknown cell value by a variable

Same variable S: Same sum for each row and each column

State the Problem (1) ▸ Sum of a Line




```
line([X,Y,Z],S):-
        S is X+Y+Z.
```

Represent a line as a list of cell values

built-in predicate: left-hand-side is the result of evaluating the right-hand-side

„S“ is the sum of the cell values in a line

State the Problem (2) ▸ Magic Square

```
%% magic( +Square )
%
%  Square is a list of 9 integers, with the first 3
%  representing the first line of the square,
%  the next 3 representing the second line, etc.
%  A square is magic if all lines (rows or columns)
%  have the same sum S.

magic( [A,B,C, D,E,F, G,H,I] ):-

   % rows (horizontal lines):

   line([A,B,C],S), line([D,E,F],S), line([G,H,I],S),

   % columns (vertical lines):

   line([A,D,G],S), line([B,E,H],S), line([C,F,I],S).
```

State the Problem (3) ▸ Solution of Magic Square

```
%% solve_square( Square)
%
%  Square is a list of 9 cells, with the first 3
%  representing the first line of the square,
%  the next 3 representing the second line, etc.
%  A cell is either a variable or an integer.
%  The square is magic if the variables can be replaced
%  by values from 1 to 9 so that magic(Square) is true.

solve_square(Square) :-

   % Replace variables in Square with values from 1 to 9

   permutation([1,2,3,4,5,6,7,8,9], Square),

   % ... so that the resulting square is magic

   magic(Square).
```

Use the Solution ▶ Solve the magic square

1	B	C
D	E	2
G	3	J

```
?- solve_square([1,B,C,D,E,2,G,3,J]). <---
B = 8,
C = 6,
D = 9,
E = 4,
G = 5,
J = 7
```

---- first solution --->

1	8	6
9	4	2
5	3	7

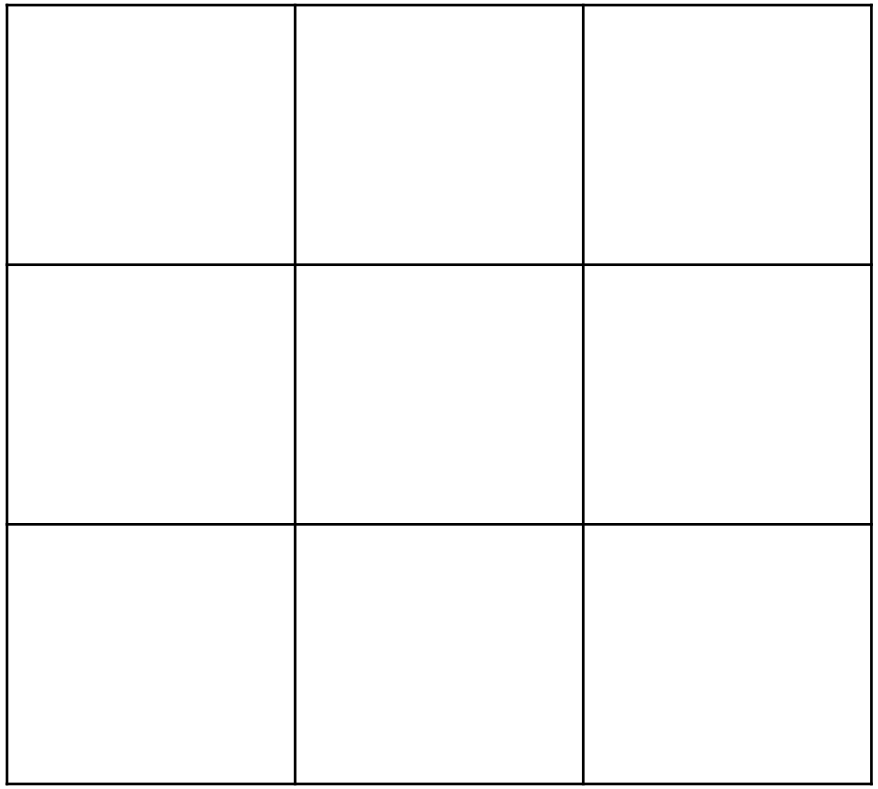
```
; <-- ask for next solution ----
B = 5,
C = 9,
D = 6,
E = 7,
G = 8,
J = 4
```

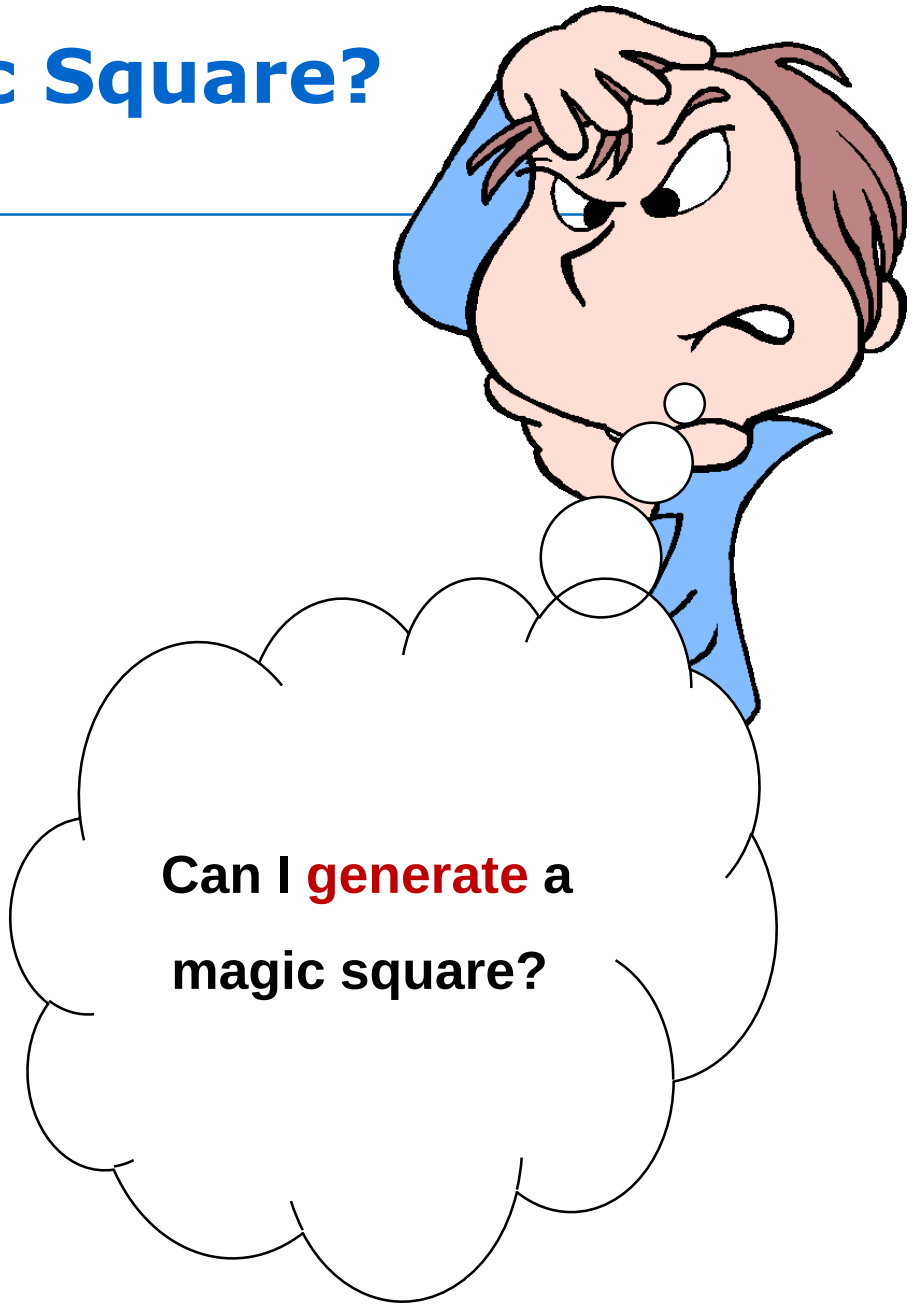
-- second solution -->

1	5	9
6	7	2
8	3	4

```
; <-- ask for next solution ----
false.
```

How to Create a Magic Square?





Use the Solution ▶ Generate magic square

```
?- solve_square([A,B,C,D,E,F,G,H,J]). <---
A = 1
B = 6,
C = 8,
D = 9,
E = 2,
F = 4,
G = 5,
H = 7,
J = 3
;
... 71 other squares
;
false.
```

Ask for the most general square: Variables in all cells!

---- first square --->

1	6	8
9	2	4
5	7	3

The same program solves and generates magic squares!

No need to rewrite!

Awesome!

Try it!!!

Predicates versus Functions

Difference of predicates and functions

Input Modes

Predicates versus Functions (1)

- In the functional programming language Haskell the following definition of the **`isFatherOf`** predicate is illegal:

```
isFatherOf x | x==frank   = peter
isFatherOf x | x==peter   = paul
isFatherOf x | x==peter   = hans
           x | otherwise = dummy
```

✘

- In a functional language predicates must be modeled as boolean functions:

```
isFatherOf x y| x==frank y==peter = True
isFatherOf x y| x==peter y==paul  = True
isFatherOf x y| x==peter y==hans  = True
           x y| otherwise         = False
```

✓

Predicates versus Functions (2)

- Function application in Haskell must not contain any variables!
- Only the following “checks” are legal:

```
isFatherOf frank peter   ⟶ True
isFatherOf kurt peter    ⟶ False
```

- In Prolog each argument of a goal may be a variable!
- So each predicate can be used / queried in many different input modes:

```
?- isFatherOf(kurt,peter).   ⟶ Yes

?- isFatherOf(kurt,X).       ⟶ Yes
                               X = paul;
                               X = hans

?- isFatherOf(paul,Y).       ⟶ No

?- isFatherOf(X,Y).          ⟶ Yes
                               X = frank, Y = peter;
                               X = peter, Y= paul;
                               X = peter, Y=hans;
                               No
```

Predicates versus Functions (3)

- Haskell is based on functions
  - Length of a list in Haskell

```
length([ ])  = 0
length(x:xs) = length(xs) + 1
```

- Prolog is based on predicates
  - Length of a list in Prolog:

```
length([ ], 0).
length([X|Xs],N) :- length(Xs,M), N is M+1.
```

```
?- length([1,2,a],Length).
   Length = 3
```

← used in mode (ground, free)

```
?- length(List,3).
   List = [_G330, _G331, _G332]
```

← used in mode (free, ground)

<--- Most general list with 3 elements

Documentation

Modes

Determinism

PlDoc

Documenting Predicates Properly

- Predicates are more general than functions
  - There is not one unique result but many, depending on the input

- So resist temptation to document predicates as if they were functions!
  - Don't write this:

```
% The predicate length(List, Int) returns in Arg2
% the number of elements in the list Arg1.
```

  - Better write this instead:

```
% The predicate length(List, Int) succeeds iff Arg2 is
% the number of elements in the list Arg1.
```

Documenting Invocation Modes

- Invocation mode of an argument
  - "–" means "is a free variable at invocation time"
  - "+" means "is non-variable at invocation time"
  - "?" means "don't care whether free or not at invocation time"

- Document behaviour that depends on the invocation mode!

```
%% length(+List, ?Int) is det
% length(?List, +Int) is det
% length(-List, -Int) is nondet
%
% The predicate length(List, Int) succeeds iff Arg2 is
% the number of elements in the list Arg1.
%
 length([ ],0).
 length([X|Xs],N) :- length(Xs,N1), N is N1+1.
```

Documenting Determinism

Determinism	Predicate behaviour
failure	Always fails
semidet	Fails or succeeds exactly once without a choice-point
det	Succeeds exactly once without a choice point
nondet	No constraints on the number of times the predicate succeeds and whether or not it leaves choice-points on the last success.
multi	As nondet, but succeeds at least one time.

- Document also the determinism!

```
%% length(+List, ?Int) is det
% length(?List, +Int) is det
% length(-List, -Int) is nondet
%
% The predicate length(List, Int) succeeds iff Arg2 is
% the number of elements in the list Arg1.
%
 length([ ],0).
 length([X|Xs],N) :- length(Xs,N1), N is N1+1.
```

Modes and Determinism in SWI-Prolog

Note that

- SWI-Prolog uses a richer set of mode and determinism annotations: http://www.swi-prolog.org/pldoc/man?section=modes
- In the SWI-Prolog documentation, the ‚?‘ mode has a different meaning
- In this course we use the annotations specified on the previous slides.

Tip

- Have a look at the full PlDoc package (akin to JavaDoc) an use it extensively
  http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/pldoc.html%27)

Chapter Summary

- Prolog Syntax
  - Programs, clauses, literals
  - Terms, variables, constants
  - Recusion
- Application Example
  - Solving logic puzzles
- Relations versus Functions
  - Input modes
- Extending the syntax
  - Operator definitions