Metalevel		Syntactic level			Runtime level				
Syntax	Description	Syntax	Description	And also	If the runtime state is...	...then one says	...but it would be clearer to say	One can also state that...	The following lingo also exists
Used in manual descriptions		Used in source code and queries		Just by looking at the syntax we can decide...	All depends on the variables from the syntactic level momentarily denote.				
		- A "**term**" is an assembly of function symbols, constant symbols and variables. - A "**variable**" is "a symbol from the set of variables". - This corresponds to the usage of "**term**" and "**variable**" in mathematical logic. - In logic, the variable ranges over a domain which is further constrained by logic equations and inequations. - In Prolog, going forward in a proof, the instantiation of a **syntactic level variable** becomes less and less general as the **runtime level term** it is bound to contains less and less **runtime level variables**. "Maximum instantiation" is reached if the runtime level term is ground.			- A "**term**" can be thought of a implemented through a directed acyclic graph. - A runtime level "**variable**" is an empty, childless node that may take up content later. - A syntactic level "**variable**" is a designator for (a reference to) a node of a directed acyclic graph. The subgraph reachable from that node is the syntactic level term to which the syntactic level variable is bound. Note that if that subgraph os printed out, empty nodes are given "temporary variable names". As such, there really are no "unbound variables" at all. - We denote the fact that a variable X of the syntactic level denotes a node in graph of the runtime level with "--->"			
`foo(Term)`	Whatever appears between the parentheses of foo(.) in source code is designated by *Term*.								
		`foo(X)`	"**X** is a *term* and it is a *variable*" (N.B.: it is a variable of the syntactic level)	**ground(X)** outcome depends on runtime **var(X)** outcome depends on runtime					
					X ---> {}	"**X** is *unbound*" "**X** is *uninstantiated*" "**X** is an *unbound variable*" "**X** is a *variable*" (N.B. it is a variable both of the syntactic level and the runtime level) "**X** is *var*" "**X** is *free*" (avoid this! "free" should be reserved for the meaning of "not bound by a quantifier" in mathematical logic)	"**X** designates an empty node"	"**X** is *nonground*"	"**X** is a *partial term*"
					X ---> a	"**X** is *bound*" (to an atom) "**X** is *instantiated*"	"**X** designates a node containing the atom 'a'"	"**X** is *ground*"	
					X ---> f(a)	"**X** is *bound*" (to a compound term)	"**X** designates a node containing the functor 'f/1' with a child node containing the atom 'a'."	"**X** is *ground*"	
					X ---> f(Z), Z ---> a	"**X** is *bound*" (to a compound term) and "**Z** is *bound*" (to an atom)	"**X** designates a node containing the functor 'f/1' with a child node containing the atom 'a'. That node is also designated by **Z**"	"**X** is *ground*" "**Z** is *ground*"	
					X ---> f(Z), Z ---> {}	"**X** is *bound*" (to a compound term) and "**Z** is *unbound*"	"**X** designates a node containing the functor 'f/1' with an empty child node That node is also designated by **Z**"	"**X** is *nonground*" "**Z** is *nonground*"	"**X** is a *partial term*"
		`foo(a)`	"**a** is a *term* and it is an *atom*"	**ground(a)** succeeds **var(a)** fails					
		`foo(f(a))`	"**f(a)** is a *term* and it is a *compound term*" (it is "*compound*")	**ground(f(a))** succeeds **var(f(a))** fails					
		`foo(f(X))`	"**f(X)** is a *term*, it is *compound* and it contains a *variable*, **X**"	**ground(f(X))** outcome depends on runtime **var(f(X))** outcome depends on runtime					