JB-Prolog 2.1

for the MacIntosh

User's Manual
Language Reference Manual

jburse@clients.switch.ch

XLOG™, Postfach 423, CH-8042 Zürich

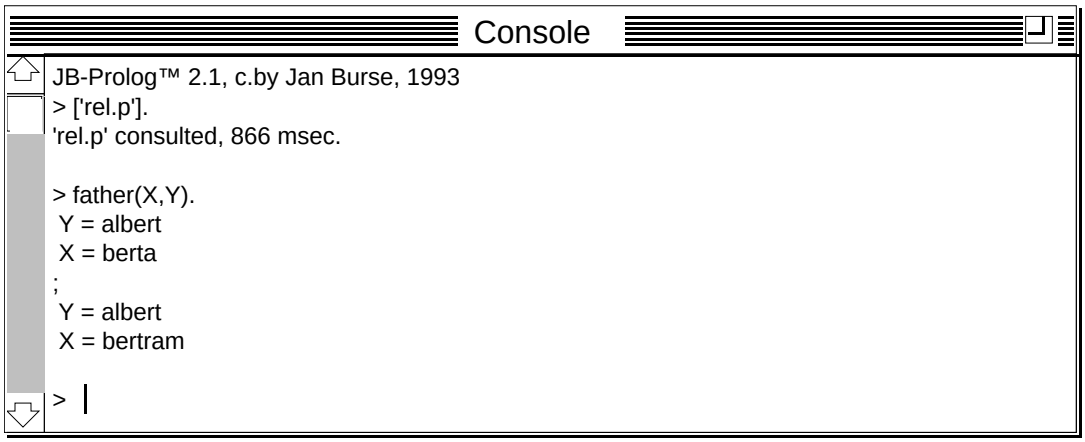
Contents

User's Manual

Introduction

The programming environment permits the user to directly communicate with the prolog interpreter through a console window and to edit, print and consult texts. The programming environment is a necessity to make the JB-Prolog 2.1 a self contained system.

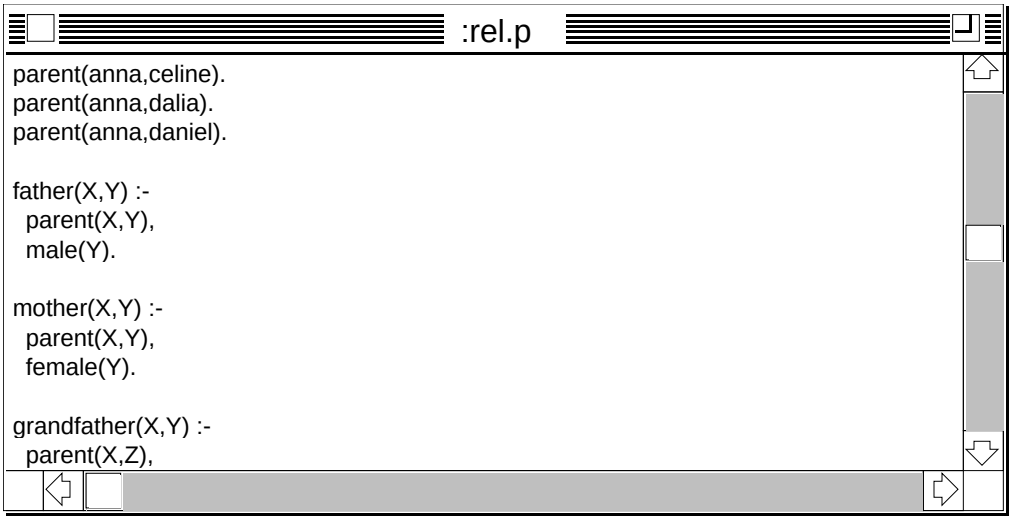
The Console Window



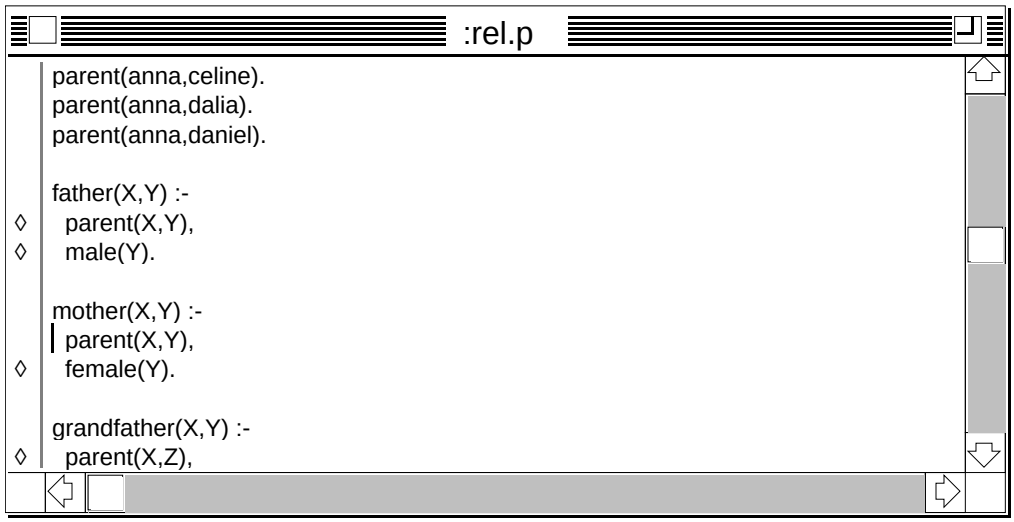
In the console window you can directly type queries and submit them to the interpreter. If the prompt > appears, you can enter a query. Do not forget to terminate the query by a period and a carriage return. The interpreter will then successively search for answer substitutions. On success the interpreter displays the variable bindings. Type a semicolon followed by a carriage return to get a next solution or type a carriage return alone to abort the query.

The editing of the console is restricted. The area above the last line is read-only. You can scroll and select this area, but you get a beep, if you try to modify it. On the other hand as long as you do not terminate your the last line by a carriage return or ctrl-D, you can always correct it. A ctrl-D has the effect of an end-of-file.

The Text Windows

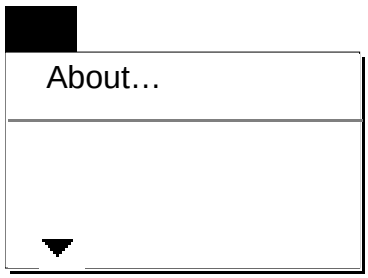

:rel.p
parent(anna,celine).
parent(anna,dalia).
parent(anna,daniel).
father(X,Y) :-
parent(X,Y),
male(Y).
mother(X,Y) :-
parent(X,Y),
female(Y).
grandfather(X,Y) :-
parent(X,Z),

In a text window you can display and modify a text. Use the **File** menu to store and print texts. Use the **Edit** menu to cut and paste texts. Use the **Search** menu to find and replace strings, or to go to a specific line. Use the **Prolog** menu to consult a text. Use the **Windows** menu to bring a text window to front. Use the **Font** menu to change the font name and font size. If you have modified a text, it is considered dirty. If you close a dirty text, JB-Prolog 2.1 asks you to save it. If you save or revert a text, the dirty flag is reset.

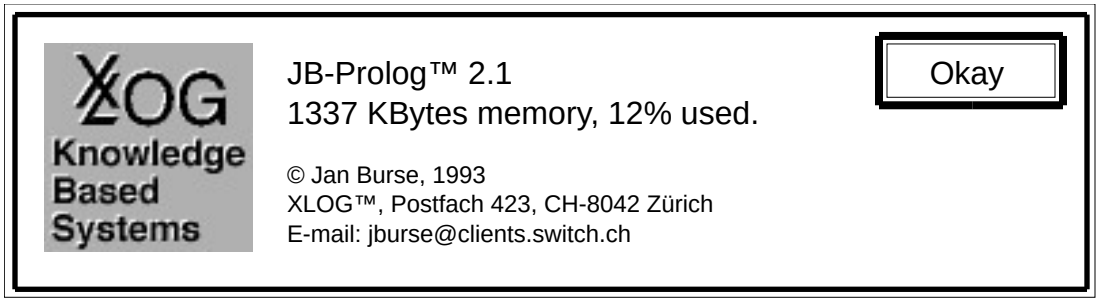


If you click the left bottom square the text window is extended by a spy bar. The signs ◊ and 11 indicate traceable lines, to see them you have to consult the text at least once. A 11 sign marks a line where the interpreter has to stop. You can toggle between ◊ and 11 by clicking on the sign.

The Apple Menu



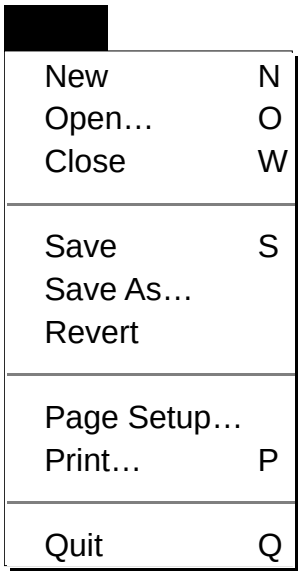
About…: Displays the dialog below.



The dialog displays the version of your prolog system, the maximum available memory and the usage of the memory in percent. It also displays the paper and e-mail address where you should direct questions concerning JB-Prolog 2.1.

Album etc.: Opens a desk accessory.

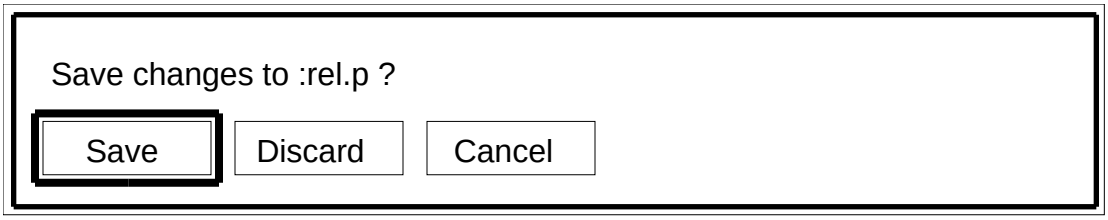
The File Menu



New: Opens a new text window. The name of the text window will be Untitled-# where # is a new number and the text of the text window will be empty.

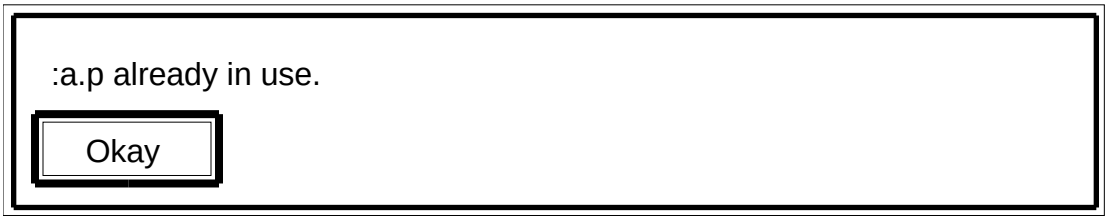
Open…: You can choose an existing file. If a text window with the same name already exists the text window comes to front. Otherwise a text window with the name and the corresponding text is created.

Close: If the text of the front most text window is not dirty the text window is removed. Otherwise the dialog below is displayed.



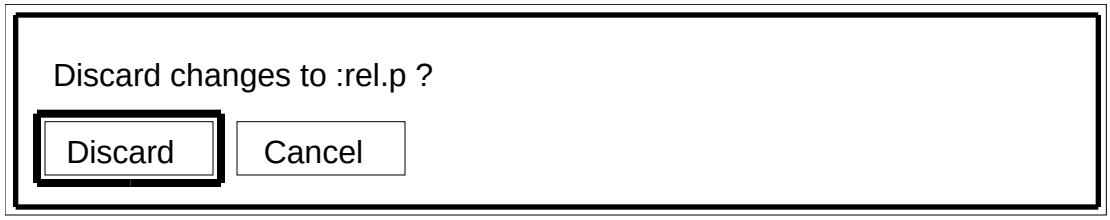
If you select **Cancel** nothing is done. If you select **Save** the text is saved and the text window is removed. If you select **Discard** the text window is removed without saving.

Save: If the front most text window is not untitled the text is saved. Otherwise you can enter a name. If no other text window with the same name already exists text window is renamed and the text is saved. Otherwise the dialog below is displayed and nothing is done.



Save As…: You you enter a name. If no other text window with the same name already exists the text window is renamed and the text is saved. Otherwise the dialog above is displayed and nothing is done.

Revert: Displays the dialog below.



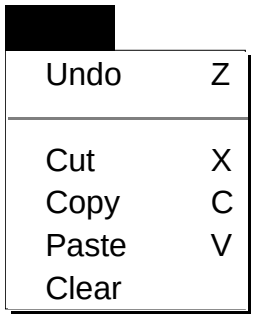
If you select **Discard** the last saved text of the front most text window is reload. If you select **Cancel** nothing is done. The command is disabled if the front most text window is untitled.

Page Setup…: Displays the page setup dialog. Make sure that you have chosen the right printer with the Chooser desk accessory.

Print…: Displays the print dialog and then prints the text of the front most text window. Make sure that you have chosen the right printer with the Chooser desk accessory.

Quit: Successively tries to close every open text window. Terminates JB-Prolog 2.1 if no open text window remains.

The Edit Menu



Undo: Not yet implemented.

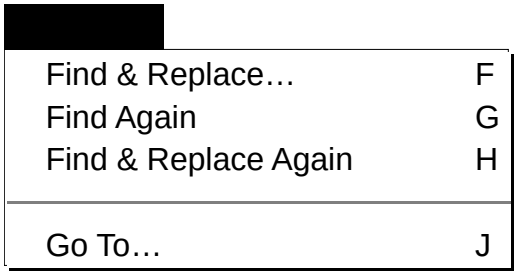
Cut: Copies the selected text into the clipboard and clears the selected text.

Copy: Copies the selected text into the clipboard.

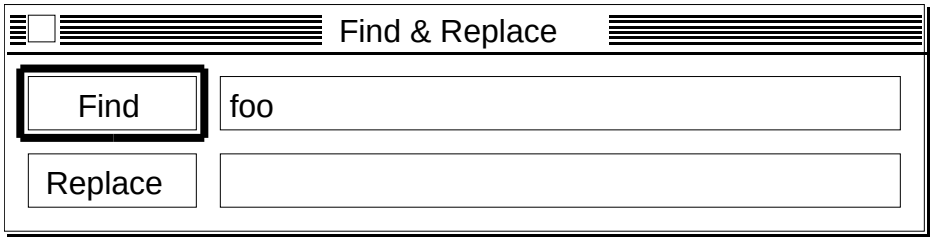
Paste: Pastes the clipboard into the selected text.

Clear: Clears the selected text.

The Search Menu



Find & Replace…: Displays the dialog below.

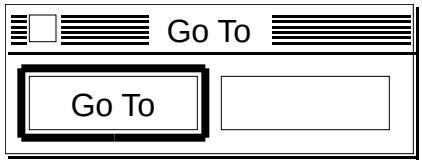


Enter a find string and a replace string. Select **Find** to search for the find string. Select **Replace** to replace the selected text by the replace string and search for the find string.

Find Again: Searches for the last entered find string.

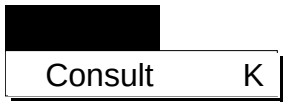
Find & Replace Again: Replaces the selected text by the last entered replace string and searches for the last entered find string.

Go To…: Displays the dialog below.

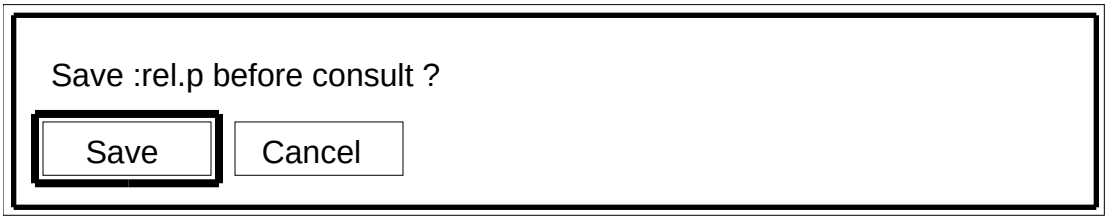


Enter a line number. Select **Go To** to highlight the corresponding line.

The Prolog Menu

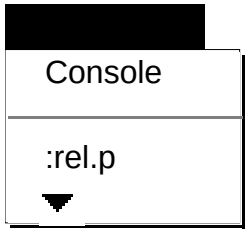


Consult:.If the text of the front most text window is not dirty it is consulted. Otherwise the dialog below is displayed.



If you select **Cancel** nothing is done. If you select **Save** the text is saved before consulting it.

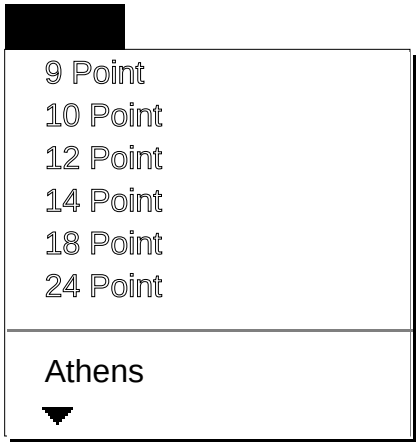
The Windows Menu



Console:.Brings the console window to front.

:rel.p etc.:.Brings the text window to front.

The Font Menu



9 Point etc.: Sets the font size.

Athens etc.: Sets the font name.

Introduction

JB-Prolog 2.1 is a slim and powerful prolog interpreter. It is currently available for the MacIntosh where it comes with a programming environment described in the User's Manual. In the following a brief summary of the main characteristics of the language implementation is given. The reader is assumed familiar with the basic concepts of prolog.

Datastructures

The available datastructures are strings, integers, floats, variables and functions. For functions dictionary entries are generated that represent name/arity pairs. Dictionary entries are not reclaimed and therefore you should use strings instead of atoms whenever possible.

integer: A sequence of digits preceded by an optional sign, i.e. **123**, **-2**, etc. Integers can be arbitrary long.

strings: A sequence of chars enclosed in apostrophes, i.e. **'abc'**, **'Hello World!'**, etc. Strings can be arbitrary long.

floats: A sequence of digits with a period in it preceded by an optional sign, i.e. **3.1415**, **-1000.0**, etc. 16 digits are significant.

variables: An identifier which starts with a capital letter or an underscore, i.e. **_1**, **X**, **Head**, etc. Single underscores can be used for anonymous variables.

functions: A special identifier or an identifier which starts with a small letter, followed by an optional list of arguments, i.e. **-(7)**, **foo(X,bar)**, **blue**, etc.

Special identifiers are the characters **!**, |, **,**, **;** or a sequences of the characters #, *****, +, **-**, **.**, **:**, <, =, >, **?**, \, /, **@**, **&**, ~, ^,.`. Quotation is used for strings only, i.e. 'f'(a,b) is not correct and 'f' is a strings but not an atom. Prefix, infix, set and list notation are supported, i.e. **-a**, **a+b**, **{a}**, **[a,b|c]**.

Predicates

The aim of a prolog program is to define predicates by means of rules. Synonymously you can call predicates, relations or tables. A predicate is written as a function, i.e. a small identifier followed by an optional list of arguments. The following kinds of predicates are distinguished:

undef: If a predicate is seen for the first time the predicate is considered undefined. You cannot perform calls or database operations on an undefined

predicate.

dynamic: If you consult rules for a predicate or if you call **dynamic/1** for a predicate explicitly a predicate becomes dynamic. You can perform calls and database operations on a dynamic predicate.

builtin: A predicate that is defined internally is a built-in.You cannot consult rules for a built-in and you cannot perform database operations on a built-in. You can only perform calls on a built-in.

Programs

A program is a collection of rules and directives. A rule has the form **Head :- Tail** or **Head** alone, whereas a directive has the form **:- Goal**. During a consult the old rules that were previously consulted are first removed and then new rules are added to the database and directives are executed once:

[F_1,...,F_n]	Consult the files F_1, …, F_n.

Control

The JB-Prolog 2.1 interpreter supports standard backtracking and the cut. Blocked goals are not supported. The cut in a conjunction, disjunction or findall resets the surrounding predicate. The cut in a negation or conditional behaves locally.

!	Cut. Succeeds and reset surrounding predicate on redo.
X, Y	And. Succeeds whenever **X** and then **Y** succeed.
X; Y	Or. Succeeds whenever **X** or else **Y** succeed.
\+ X	Not. As defined by **(\+ X) :- X, !, fail; true**.
X -> Y; Z	If-then-else. As defined by **(X -> Y ; Z) :- X, !, Y; Z**.
true	True. Succeed and fail on redo.
repeat	Repeat. Succeed always.
fail	Fail. Fail on call.
findall(X,P,L)	Findall. Succeed with the list of all **X** such that **P** in **L**.

Basics

The following predicates perform basic operations on the datastructures of JB-Prolog 2.1. They test, construct and destruct strings, integers, floats and functions. Note that **name/2** and **=../2** work with strings.

var(X)	Succeeds if **X** is a variable.
string(X)	Succeeds if **X** is a string.
integer(X)	Succeeds if **X** is an integer.
float(X)	Succeeds if **X** is a float.
function(X)	Succeeds if **X** is a function.
builtin(X)	Succeeds if **X** is a built-in predicate.
name(S,L)	Succeeds if **L** is the character list of the string **S**.
X=..[S|L]	Succeeds if string **S** is functor and **L** arguments of **X**.
X= Y	Succeeds if **X** and **Y** unify.
X==Y	Succeeds if **X** and **Y** are identical.
X\==Y	Succeeds if **X** and **Y** are not identical.
X<Y	Succeeds if **X** is lexically before **Y**.
X=<Y	Succeeds if **X** is lexically before or identical **Y**.
X>Y	Succeeds if **X** is lexically after **Y**.
X>=Y	Succeeds if **X** is lexically after or identical **Y**.

X is Y	Succeeds if **Y** evaluates to **X**.
sort(L,R)	Succeeds with the sorted list **L** in **R**.

Prolog works with uninterpreted functions, therefore 1+2=3 fails. Use **is/2** to evaluate expressions. The supported functions are.

-X	Change sign: float, integer.
abs(X)	Absolute value: float, integer.
int(X)	Convert to integer: float, string.
float(X)	Convert to float: integer, string.
str(X)	Convert to string: integer, float.
X + Y	Addition: integer x integer, float x float. Concat: string x string.
X - Y	Subtraction: integer x integer, float x float.

X * Y	Multiplication: integer x integer, float x float.
X / Y	Division: integer x integer, float x float.
X mod Y	Modulo: integer x integer.
min(X,Y)	Minimum: integer x integer, float x float, string x string.
max(X,Y)	Maximum: integer x integer, float x float, string x string.
sin(X)	Sinus: float.
cos(X)	Cosinus: float.
tan(X)	Tangens: float.
atan(X)	Arcus tangens: float.
ln(X)	Logarithmus naturalis: float.
exp(X)	Exponential function: float.
sqrt(X)	Square root: float.

Input and Output

To perform basic output the JB-Prolog 2.1 system maintains a current input stream and a current output stream. You can operate on these by the following predicates.

tell(S)	Set current output stream to **F**.
told	Set current output stream to previous one.
write(X)	Write term **X** to current output stream.
put(S)	Write string **S** to current output stream.
nl	Write a carriage return to current output stream.
flush	Flush the current output stream.
see(S)	Set current input stream to **F**.
seen	Set current input stream to previous one.
read(X)	Read term **X** from current input stream.
get(X)	Read string **S** from current input stream.

Operators

Prefix and infix operators of the modes **fx**, **fy**, **xfx**, **yfx**, **xfy** and **yfy** are supported. Postfix operators are not supported. Use **op/3** to defined your own operators.

op(N,M,S)	Define **S** an operator of mode **M** and level **N**.

Put operators with level above or equal 1000 in parentheses, i.e. **write((X;Y))** instead of **write(X;Y)**. The following operators are predefined.

400,yfx	*, /, **mod**
500,fx	+, **-**
500,yfx	+, **-**
700,xfx	<, =<, =, **=..**, ==, >, >=, \==, **is**
900,fx	\+
1000,yfx	**,**

1100,xfy	**->**, **;**
1200,fx	**:-**
1200,xfx	**:-**

Database

The database stores rules on the heap. A rule of the form **Head** alone is called a fact and is an abbreviation for the rule **Head :- true**. The database uses a simple hash function on every argument to speedup the access of rules and facts.

dynamic(X)	Declare **X** dynamic.
asserta(R)	Assert rule **R** on top.

assertz(R)	Assert rule **R** on bottom.
clause(R)	Retrieve rules matching **R**.
retract(R)	Retrieve and delete rules matching **R**.

Tail Recursion

In general a rule is tail recursive if it has the form **Head :- $Guard_1$, …, $Guard_n$, !, Tail**. Because of the cut, in case of a call of **Tail**, the interpreter may replace **Head** by **Tail** instead of generating a new invocation for **Tail**. The JB-Prolog 2.1 interpreter is able to apply this optimization if **Tail** is dynamic and if the guards **$Guard_1$, …, $Guard_n$** are among the following predicates.

var/1	**string/1**	**integer/1**	**float/1**	**function/1**
builtin/1	**name/2**	**=../2**	**=/2**	**==/2**
\==/2	**</2**	**=</2**	**>/2**	**>=/2**
is/2				

Grammar Rules

A program may also contain grammar rules of the form **Head --> Tail**. In a consult, grammar rules are converted to normal rules before they are added to the database. Every non-terminal is thus extended by two parameters in the front to pass a difference list. The following constructs are supported:

{G}	Action. **G** is executed.
X, Y	Sequence. **X** and then **Y** are parsed.
[T_1,...,T_n]	Terminals. Terminals **T_1**,…,**T_n** are parsed.
!	Cut. Backtracking control.

Debugging

The debugger of JB-Prolog 2.1 is line oriented and not predicate oriented. Hence you will not find predicates to spy or unspy a predicate. You can control the debugger through the programming environment and through the following predicates:

trace	Continues execution till next traceable line.
notrace	Continues with normal execution.
stop	Stops execution and exits current block.
exec(P)	Calls **P** once in a new block.

If execution is interrupted, the interpreter highlights the traceable line where execution stopped and displays a debugger line on the console. A typical debugger line is:

12 Call: foo(_1,bar)

12 is the depth of execution measured in traceable lines that were passed. **Call** is

the current port reached and **foo(_1,bar)** is the current goal. To continue execution you have to type an option followed by a carriage return right after the debugger line. You can choose between the following options:

<void>	Creep. Continues execution till next traceable line.
l	Leap. Continues with normal execution.
s	Skip. Continues execution till current level is reached again.
a	Abort. Stops execution and exits current block.

Compatibility

JB-Prolog 2.1 has a reduced set of built-ins. If you have problems with missing predicates try to define the predicates yourself. The following definitions may help you in doing so.

```
:- op(700,xfx,'@<').
:- op(700,xfx,'@>').
:- op(700,xfx,'@=<').
:- op(700,xfx,'@>=').
:- op(700,xfx,'=:=').
:- op(200,xfy,'^').

nonvar(X) :- \+var(X).
atom(X) :- function(X), X=..[_].
compound(X) :- function(X), X=..[_,_|_].
number(X) :- integer(X); float(X).
atomic(X) :- string(X);number(X);atom(X).

X @< Y :- X < Y.
X @> Y :- X > Y.
X @=< Y :- X =< Y.
X @>= Y :- X >= Y.
X =:= Y :- Z is X, Z is Y.

atom_chars(A,L) :- var(A), !, name(S,L), A=..[S].
atom_chars(A,L) :- A=..[S], name(S,L).
number_chars(N,L) :- var(X), member(47,L),!, name(S,L), N is float(S).
number_chars(N,L) :- var(X),!, name(S,L), N is int(S).
number_chars(N,L) :- S is str(N), name(S,L).

length([],0).
length([_|Tail],Len) :- var(List), !, Len1 is Len-1, length(Tail,Len1).
length([_|Tail],Len1) :- length(Tail,Len), Len1 is Len+1.

functor(Head,Name,Arity) :-
 var(Head), !, length(List,Arity), Head=..[Name|List].
functor(Head,Name,Arity) :-
 Head=..[Name|List], length(List,Arity).

reconsult(X) :- [X]. /* not exactly */
consult(X) :- [X]. /* not exactly */
assert(X) :- assertz(X).
abolish(N,A) :- functor(X,N,A), (retract((X:-_)), fail; true).
listing(N,A) :- functor(X,N,A),
  (clause((X:-Y)),
    (Y==true -> write(X); write((X :- Y))), put('.'), nl, fail;
```

```
    true).

X^P :- P.
setof(X,P,L) :- findall(X,P,H), sort(H,L). /* not exactly */
```

Predicate Index