TM-0439

A GHC Language Processor in Prolog and GHC Sample Programs for Bench Marking.

by
S. Takagi, K. Ueda &
T. Chikayama

January, 1988

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Abstract

This memorandum is primarily prepared for Prolog benchmarking and contains the programs of a GHC language processor written in Prolog and several sample programs written in GHC.

The program of the GHC language processor can be considered as a typical example of Prolog programs and used for a material of studying Prolog programming styles and characteristics.

Furthermore, this program can be used as a practical GHC language processor on DEC-10 Prolog or similar environments to study a parallel logic programming language. GHC is classified as a committed-choice And-parallel logic programming language. The languages of this class have many interesting characteristics in programming styles, controlling of communication, algorithms, and so on.

To encourage the study of this kind and also for testing the installation of the GHC language processor, several sample programs written in GHC are also included. These programs are primarily written to study parallel algorithms and programming styles. Then, it will be used as an elementary text of learning parallel logic programming.

More complete documents of GHC and related research at ICOT are available in ICOT Technical Report and Memorandum.

1. Introduction

At ICOT, Prolog is heavily used for language processors. The user language ESP (Extended Self-contained Prolog) is initially developed on Prolog and bootstrapped to PSI (Personal Sequential Inference machine). PSI's operating and programming system SIMPOS is initially developed on the cross compiler written in Prolog. For the development of GHC (Guarded Horn Clauses) is also initiated from the Prolog version of GHC compiler/interpreter system. Not only these language processors, but also some more preprocessors and language converters are written in Prolog.

This heavy use of Prolog for language processors explains us enough to include a language processor in a collection of Prolog benchmark. I think the reasons why Prolog is used for these processors are almost the same as the case of Lisp. It is easy to write I/O routines and need no experience for reading sources and writing objects. It is easy to convert the source format into internal form and then object form. Prolog uses terms for everything as Lisp uses S-Expressions.

One of the difference between language processors and other application programs is that language processors need lots of I/O while usual application programs consumes CPU for data handling. We think this will help some different point of view for Prolog benchmark programs.

From the viewpoint of GHC parallel execution, some measurement has been done for these data programs. However, this processor itself is not profiled nor measured in the viewpoint of Prolog benchmark yet. Instead, here we show all the source program of GHC processor written in Prolog and the GHC source programs to be used for the benchmark.

The result of the execution of this GHC processor on some Prolog implementations will be reported as the result of the Prolog benchmark workshop at Los Angels, February 1988.

Any comments or discussions are welcome. Please send E-mail to
Internet: takagi@icot.jp
UUCP: {enea,inria,kddlab,mit-eddie,ukc}!icot!takagi

2. Source and Data

Complete source programs of GHC in Prolog and small programs in GHC are listed in the appendix. Brief documentation for each GHC program are also listed.

For each GHC program, read the documentation.

The programs are:

bpl	bestpath by layered stream
bpmon	bestpath by monitor
bpsc	bestpath by shortcircuit
gdpath	goodpath by layered stream
life1	life game program that prints the final result
life2	life game displaying the state changes
mxflw1	maximum flow calculation
mxflw2	maximum flow calculation
pascal	pascal's triangle
tep	a tiny propositional calculus solver
waltz	Waltz's labeling algorithm

All of these programs are tested on the following environment.
SUN 3/260 OS 3.3
GHC 1.9 on SICSTUS Prolog

Execution sample is listed also in appendix.

3. Comments

In the viewpoint of GHC benchmark, these programs behave differently. The result is very informative for writing pragmas, and also for improving the basic algorithm for the programs.

From the viewpoint of Prolog benchmark, these GHC programs are relatively small and some larger GHC programs will help more.

All the sources of GHC processor and GHC programs may be used for academic use only. If someone needs newer implementation of GHC, contact to K. Ueda of ICOT. (ueda%icot.jp@relay.cs.net)

4. Acknowledgments

I wish to thank the KL1-TG group members for writing these GHC programs and documentation although they are not used to writing in English. I would also like to thank Dr. F. Pereira of SRI, Mr. K. Sakai of ICOT, and Mr. Okumura of ICOT for suggesting these GHC programs.

5. References

K. Furukawa, S. Kunifuji, A. Takeuchi, and K. Ueda:
The Conceptual Specification of the Kernel Language Version 1,
ICOT TR-054, March 1984, 41p

Furukawa, Takeuchi, Miyazaki, Ueda, and Tanaka:
Kernel Language Version 1, ICOT, May 1985, 36p (in Japanese only)

S. Takagi ed.:
A Collection of KL1 Programs -- Part 1 --
ICOT TM-311, Dec 1987.

J. Tanaka, K. Ueda, T. Miyazaki, A. Takeuchi, Y. Matsumoto, and K. Furukawa:
Guarded Horn Clauses and Experiences with Parallel Logic Programming,
Proc. FJCC 1986, ACM and IEEE Computer Society, Nov. 2-6 1986, pp. 948-954

K. Ueda:
Guarded Horn Clauses, ICOT TR-103, 1985 and 1986
Also in Proc. Logic Programming '85, E. Wada (ed.),
Lecture Notes in Computer Science 221, Springer-Verlag, 1986, pp. 168-179
To appear in Concurrent Prolog: Collected Papers, Vol. 1,
E. Y. Shapiro (ed.), The MIT Press, 1987

K. Ueda:
Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard, ICOT TR-208, 1986 and 1987
Also to appear in Programming of Future Generation Computers,
M. Nivat and K. Fuchi (eds.), North-Holland, 1987

K. Ueda:
Guarded Horn Clauses, Doctoral Thesis, Information Engineering Course,
Faculty of Engineering, Univ. of Tokyo, 1986

K. Ueda:
Introduction to Guarded Horn Clauses, ICOT TR-209, 1986

K. Ueda:
Implementing GHC on Prolog, ICOT TM-366, 1987

Appendix:

GHC execution sample log

GHC compiler/Interpreter source for SICSTUS Prolog

GHC programs and brief documentations

bpl	bestpath by layered stream
bpmon	bestpath by monitor
bpsc	bestpath by shortcircuit
gdpath	goodpath by layered stream
life1	life game program that prints the final result
life2	life game displaying the state changes
mxflw1	maximum flow calculation
mxflw2	maximum flow calculation
pascal	pascal's triangle
tep	a tiny propositional calculus solver
waltz	Waltz's labeling algorithm

```
takagi@icot32[1.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('bp1.ghc').
bestPath/3, graph1/12, graph2/15, graph3/15, graph4/17, graph5/13, node/8,
announce/2, filter/7, lastFilter/6, minMerge/4, minM1/5, minM2/4, END.
[compiling /tmp/ghca14240...]
[/tmp/ghca14240 compiled in 177400 msec.]

yes
| ?- ghc bestPath(n0,n11,Path).
6020 msec.
Path = 13 - [[n0,n1,n2,n3,n4,n5,n11]] ?

yes
| ?- ^D[ End break ]
[ End of SICStus execution ]

user time 192.680000

takagi@icot32[2.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('bpmon.ghc').
bestpath/3, evaluator/5, path/6, p/7, edge/2, merge/3, END.
[compiling /tmp/ghca14247...]
[/tmp/ghca14247 compiled in 36819 msec.]

yes
| ?- ghc bestpath(n0,n8,Ans).
18459 msec.
Ans = 7 * [[n8,n2,n1,n0]] ?

yes
| ?- ghc bestpath(n15,n22,Ans).
21460 msec.
Ans = 24 * [[n22,n23,n17,n11,n10,n9,n15],[n22,n23,n17,n11,n5,n4,n3,n9,n15]] ?

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 82.000000
```

```
takagi@icot32[3.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('bpsc.ghc').
node/8, send_cp/6, closeOuts/1, bp/4, augment_edges/2, del/3, gen_nodes/5,
gen_node/8, merge_all/2, merge/3, length/2, length/3, write_result/2,
bp_sne/3, bp_ne/2, bp_ex1_n/1, bp_ex2_n/1, bp_ex3_s/1, END.
[compiling /tmp/ghca14430...]
[/tmp/ghca14430 compiled in 127340 msec.]

yes
| ?- bp_ex1_n([a,b,c,d,e,f,g,h]).
[Warning: The predicate bp_ex1_n / 1 is undefined]
1  1  Call: bp_ex1_n([a,b,c,d,e,f,g,h]) ? a
[ Execution aborted ]
| ?- ghc bp_ex1_n([a,b,c,d,e,f,g,h]).
a - 0 - []
b - 1 - [a]
c - 2 - [b,a]
d - 3 - [c,b,a]
e - 4 - [d,c,b,a]
f - 10 - [a]
g - 5 - [e,d,c,b,a]
h - 6 - [g,e,d,c,b,a]
280 msec.

yes
| ?- ghc bp_ex2_n([a,b,c,d,e,f,g,h,i,j,k,l]).
a - 0 - []
b - 10 - [a]
c - 1 - [a]
d - 11 - [e,f,g,c,a]
e - 10 - [f,g,c,a]
f - 8 - [g,c,a]
g - 7 - [c,a]
h - 18 - [e,f,g,c,a]
i - 17 - [f,g,c,a]
j - 29 - [h,e,f,g,c,a]
k - 27 - [i,f,g,c,a]
l - 30 - [k,i,f,g,c,a]
520 msec.

yes
| ?- ghc bp_ex3_s(n0).
n0 - 0 - []
n1 - 2 - [n0]
n2 - 5 - [n1,n0]
n3 - 9 - [n2,n1,n0]
n4 - 11 - [n3,n2,n1,n0]
n5 - 12 - [n4,n3,n2,n1,n0]
n6 - 1 - [n0]
n7 - 2 - [n6,n0]
n8 - 7 - [n2,n1,n0]
n9 - 10 - [n8,n2,n1,n0]
n10 - 14 - [n4,n3,n2,n1,n0]
n11 - 13 - [n5,n4,n3,n2,n1,n0]
n12 - 3 - [n6,n0]
n13 - 5 - [n7,n6,n0]
n14 - 10 - [n13,n7,n6,n0]
n15 - 17 - [n14,n13,n7,n6,n0]
n16 - 20 - [n10,n4,n3,n2,n1,n0]
n17 - 16 - [n11,n5,n4,n3,n2,n1,n0]
```

```
n18 - 7 - [n12,n6,n0]
n19 - 8 - [n13,n7,n6,n0]
n20 - 12 - [n19,n13,n7,n6,n0]
n21 - 19 - [n20,n19,n13,n7,n6,n0]
n22 - 21 - [n23,n17,n11,n5,n4,n3,n2,n1,n0]
n23 - 20 - [n17,n11,n5,n4,n3,n2,n1,n0]
n24 - 10 - [n18,n12,n6,n0]
n25 - 12 - [n24,n18,n12,n6,n0]
n26 - 17 - [n25,n24,n18,n12,n6,n0]
n27 - 21 - [n33,n32,n31,n30,n24,n18,n12,n6,n0]
n28 - 21 - [n34,n33,n32,n31,n30,n24,n18,n12,n6,n0]
n29 - 22 - [n23,n17,n11,n5,n4,n3,n2,n1,n0]
n30 - 12 - [n24,n18,n12,n6,n0]
n31 - 13 - [n30,n24,n18,n12,n6,n0]
n32 - 14 - [n31,n30,n24,n18,n12,n6,n0]
n33 - 17 - [n32,n31,n30,n24,n18,n12,n6,n0]
n34 - 19 - [n33,n32,n31,n30,n24,n18,n12,n6,n0]
n35 - 21 - [n34,n33,n32,n31,n30,n24,n18,n12,n6,n0]
2180 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 136.700000

takagi@icot32[4.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('gdpath.ghc').
goodPath/3, node/6, filter/3, END.
[compiling /tmp/ghca14458...]
[/tmp/ghca14458 compiled in 30020 msec.]

yes
| ?- ghc goodPath(a,h,Path).
79 msec.
Path = h * [e * [b * [a * begin,d * [g * []]],g * [d * [b * [a * begin]]]],
i * [],j * [f * [c * [a * begin]]]] ?

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 31.580000
```

```
takagi@icot32[5.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('life.ghc').
life_game/3, wr/3, make_mesh/4, make_mesh/6, make_node/4, mg/3, temp_node/2,
end_node/2, con/2, node/13, node_work1/8, node_work2/11, node_work3/8,
node_work4/13, send/2, my_life/4, life_test/1, END.
[compiling /tmp/ghca14461...]
[/tmp/ghca14461 compiled in 168820 msec.]

yes
| ?- ghc life_test(1).
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     1     1     1     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
459 msec.

yes
| ?- ghc life_test(5).
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     1     1     1     0
     0     0     0     0     0     0     0     0
2280 msec.

yes
| ?- ghc life_test(9).
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     1
     0     0     0     0     0     1     1     1
3840 msec.

yes
| ?- ghc life_test(13).
     1     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     1
     1     0     0     0     0     0     0     0
5519 msec.
```

```
yes
| ?- ghc life_test(17).
     0     1     0     0     0     0     0     0
     1     1     0     0     0     0     0     1
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     1     0     0     0     0     0     0     0
7180 msec.

yes
| ?- ghc life_test(21).
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     1     1     1     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
14219 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 211.860000

takagi@icot32[6.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('life2.ghc).
life_game/3, headder/2, headder1/3, headder2/3, home/1, erase/1, show/5,
show/4, start_node_ad/1, node_ad/2, make_mesh/2, make_mesh/4, make_mesh/6,
make_node1/5, make_node2/6, make_node3/6, make_node4/6, make_node5/6, mg/3,
temp_node/2, end_node/2, end_node1/3, con/2, node1/13, node_work1/9,
neighb_node_ad/2, node_work2/12, node_work3/9, node/14, node/15, node_work4/16,
send/4, send_first/5, send_all/4, my_life/4, life_test/1, life_test1/1, END.
[compiling /tmp/ghca14736...]
[/tmp/ghca14736 compiled in 172220 msec.]

yes
| ?- ghc life_test(8).

TRY THIS on VT100 !!!!
Lots of escape sequences are produced.

2939 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 188.060000
```

```
takagi@icot32[7.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('mxflw1.ghc').
ex1/1, nodes/5, entrance/2, exit/6, node/14, outFlow/4, END.
[compiling /tmp/ghca14800...]
[/tmp/ghca14800 compiled in 127460 msec.]

yes
| ?- ghc ex1(10).
pass([amount(2),link(1,d),link(1,c),link(1,a),link(0,entrance)])
pass([amount(2),link(2,d),link(2,b),link(1,a),link(0,entrance)])
return([amount(1),link(0,entrance)])
pass([amount(1),link(1,d),link(1,c),link(1,a),link(0,entrance)])
pass([amount(2),link(2,d),link(1,b),link(1,c),link(1,a),link(0,entrance)])
return([amount(2),link(0,entrance)])
remain(node(a,(0,closed),(2,closed)))
remain(node(c,(3,closed),(0,closed)))

maxFlow(7)


139 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 138.980000

takagi@icot32[8.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('mxflw2.ghc').
ex2/3, ex3/3, maxflow/5, genNodeList/3, findInitAllocPE/3, genNode/4,
find2Links/6, findOtherLink/4, initNode/15, findLinkToExit/2, entrance/3,
node/17, sendAlloc/4, exit/6, outFlow/3, outByPE/1, totalByPE/8,
outMsgByPE/4, END.
[compiling /tmp/ghca14832...]
[/tmp/ghca14832 compiled in 392280 msec.]
```

```
yes
| ?- ghc ex2(50, [node(f,2),node(g,3),node(p,1)], 1).
allocate(node(f),pe(2))
        :
allocate(node(q),pe(1))
push(50)
allocate(node(a),pe(1))
ope(pe(1),node(a),send_forward(50))
        :
ope(pe(2),node(l),send_forward(25))
ope(pe(2),node(l),send_forward(10),send_back(5))
allocate(node(m),pe(1))
ope(pe(2),node(n),send_back(10))
        :
ope(pe(1),node(q),send_forward(5))
pass([amount(15),link(1,e),link(1,d),link(1,c),link(2,g),link(1,l),link(1,k),
        link(1,b),link(1,a),link(0,entrance)])
        :
ope(pe(1),node(q),send_forward(5))
pass([amount(10),link(2,e),link(2,i),link(1,m),link(1,q),link(2,p),link(1,o),
        link(1,l),link(1,k),link(1,b),link(1,a),link(0,entrance)])
ope(pe(3),node(h),send_back(5))
        :
terminate(pe(1),node(p))
terminate(pe(1),node(q))
result(pe(1),node(a),ope(byFin(1),byBin(4)),status((65,closed),(45,closed)))
        :
result(pe(1),node(q),ope(byFin(4),byBin(1)),status((5,closed),(0,closed)))

maxFlow(40)
ope(pe(1),node([a,b,m,p,q]),byFin(15),byBin(8),total(23))
ope(pe(3),node([c,d,e,g,h,i]),byFin(20),byBin(5),total(25))
ope(pe(2),node([f,j,k,l,n,o]),byFin(11),byBin(7),total(18))
1639 msec.

yes
| ?- ghc ex3(200, [node(a4,2),node(c0,3),node(d3,4),
     node(h1,5),node(i5,6),node(l2,1),node(l3,2)], 1).
                :
maxFlow(83)
ope(pe(1),node([a0,a1,a2,a3,b0,b2,b3,b4,l0,l2,m0,m2,n0,n2,o0,o2,o3]),
        byFin(172),byBin(99),total(271))
ope(pe(2),node([a4,a5,b5,c5,d5,e4,e5,f4,f5,g5,h5,l3,l4,m3,m4,n3,o4]),
        byFin(124),byBin(84),total(208))
ope(pe(3),node([c0,c1,c2,d0,d1,d2,e0,e1,e2,f0,f1,g1,g2,h0,h2]),
        byFin(165),byBin(157),total(322))
ope(pe(4),node([c4,d3,d4,e3,f3,g3,h3,i2,i3,j3,k2,k3]),
        byFin(171),byBin(123),total(294))
ope(pe(6),node([g4,i4,i5,j4,j5,k4,k5,m5,n5,o5]),
        byFin(128),byBin(104),total(232))
ope(pe(5),node([h1,i0,i1,j0,j1,k0,k1,l1,o1]),byFin(104),byBin(30),total(134))
25960 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 445.320000
```

```
takagi@icot32[9.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('pascal.ghc').
pascal/0, pascal/2, pascal_go/3, max/3, result/3, result_write/3, next/3,
nextgo/3, pascal/4, pascal_data/4, new_pascal/5, make_pascal_data/2,
make_pascal_data/3, END.
[compiling /tmp/ghcal4851...]
[/tmp/ghcal4851 compiled in 27279 msec.]

yes
| ?- ghc pascal.
Binomial coefficient using Pascal's triangle
                                    87.4.3. by Eiji Sugino
Give me Some integer !
> 10.
1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1, END
Runtime : 19 msec
Max coefficient : 10
more ? y.
> 15.
1, 15, 105, 455, 1365, 3003, 5005, 6435, 6435, 5005, 3003, 1365, 455, 105, 15, 1,
Runtime : 39 msec
Max coefficient : 15
more ? n.
0 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 30.660000
```

```
takagi@icot32[10.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('tep.ghc').
tep/2, tep/4, tep1/2, lnot/7, rnot/7, land/7, rand/7, lor/7, ror/7, limpl/7,
rimpl/7, lequiv/7, requiv/7, decompose/3, decompose_decide/4,
decompose_decide1/3, decompose_decide1_left/2, decompose_decide1_right/2,
check_ans/3, check_ans1/3, decompose1/5, pp/3, pp_subproof/3, member/3,
merge/3, intersect/3, intersect_decide/4, tep/1, tep_go/1, END.
[compiling /tmp/ghcal4852...]
[/tmp/ghcal4852 compiled in 86500 msec.]

yes
| ?- ghc tep_go(1).
[[],[(~ ~ a <-> a)]]
requiv
     [[~ ~ a],[a]]
     lnot
          [[],[~ a,a]]
          rnot
               [[a],[a]]
               axiom
     [[a],[~ ~ a]]
     rnot
          [[~ a,a],[]]
          lnot
               [[a],[a]]
               axiom
119 msec.

yes
| ?- ghc tep_go(2).
[[],[~ (~ a /\ a)]]
rnot
     [[~ a /\ a],[]]
     land
          [[~ a,a],[]]
          lnot
               [[a],[a]]
               axiom
39 msec.

yes
```

```
| ?- ghc tep_go(3).
[[],[(~ (b /\ a) <-> ~ b \/ ~ a)]]
requiv
     [[~ (b /\ a)],[~ b \/ ~ a]]
     lnot
          [[],[b /\ a,~ b \/ ~ a]]
          ror
               [[],[b /\ a,~ b,~ a]]
               rnot
                    [[b],[b /\ a,~ a]]
                    rnot
                         [[a,b],[b /\ a]]
                         rand
                              [[a,b],[b]]
                              axiom
                              [[a,b],[a]]
                              axiom
     [[~ b \/ ~ a],[~ (b /\ a)]]
     rnot
          [[b /\ a,~ b \/ ~ a],[]]
          land
               [[b,a,~ b \/ ~ a],[]]
               lor
                    [[b,a,~ b],[]]
                    lnot
                         [[b,a],[b]]
                         axiom
                    [[b,a,~ a],[]]
                    lnot
                         [[b,a],[a]]
                         axiom
319 msec.

yes
```

```
| ?- ghc tep_go(4).
[[],[(~ (b \/ a) <-> ~ b /\ ~ a)]]
requiv
     [[~ (b \/ a)],[~ b /\ ~ a]]
     lnot
          [[],[b \/ a,~ b /\ ~ a]]
          ror
               [[],[b,a,~ b /\ ~ a]]
               rand
                    [[],[b,a,~ b]]
                    rnot
                         [[b],[b,a]]
                         axiom
                    [[],[b,a,~ a]]
                    rnot
                         [[a],[b,a]]
                         axiom
     [[~ b /\ ~ a],[~ (b \/ a)]]
     land
          [[~ b,~ a],[~ (b \/ a)]]
          lnot
               [[~ a],[b,~ (b \/ a)]]
               lnot
                    [[],[a,b,~ (b \/ a)]]
                    rnot
                         [[b \/ a],[a,b]]
                         lor
                              [[b],[a,b]]
                              axiom
                              [[a],[a,b]]
                              axiom
240 msec.

yes
| ?- ghc tep_go(5).
[[],[(b -> a <-> ~ a -> ~ b)]]
requiv
     [[(b -> a)],[(~ a -> ~ b)]]
     rimpl
          [[~ a,(b -> a)],[~ b]]
          lnot
               [[(b -> a)],[a,~ b]]
               rnot
                    [[b,(b -> a)],[a]]
                    limpl
                         [[b],[b,a]]
                         axiom
                         [[b,a],[a]]
                         axiom
     [[(~ a -> ~ b)],[(b -> a)]]
     rimpl
          [[b,(~ a -> ~ b)],[a]]
          limpl
               [[b],[~ a,a]]
               rnot
                    [[a,b],[a]]
                    axiom
               [[b,~ b],[a]]
               lnot
                    [[b],[b,a]]
                    axiom
239 msec.

yes
```

```
| ?- ghc tep_go(6).
[[],[(~ (b -> a) <-> b ∧ ~ a)]]
requiv
     [[~ (b -> a)],[b ∧ ~ a]]
     lnot
          [[],[(b -> a),b ∧ ~ a]]
          rimpl
               [[b],[a,b ∧ ~ a]]
               rand
                    [[b],[a,b]]
                    axiom
                    [[b],[a,~ a]]
                    rnot
                         [[a,b],[a]]
                         axiom
     [[b ∧ ~ a],[~ (b -> a)]]
     land
          [[b,~ a],[~ (b -> a)]]
          lnot
               [[b],[a,~ (b -> a)]]
               rnot
                    [[(b -> a),b],[a]]
                    limpl
                         [[b],[b,a]]
                         axiom
                         [[a,b],[a]]
                         axiom
179 msec.

yes
| ?- ghc tep_go(7).
[[],[(~ a ∨ b <-> a -> b)]]
requiv
     [[~ a ∨ b],[(a -> b)]]
     rimpl
          [[a,~ a ∨ b],[b]]
          lor
               [[a,~ a],[b]]
               lnot
                    [[a],[a,b]]
                    axiom
               [[a,b],[b]]
               axiom
     [[(a -> b)],[~ a ∨ b]]
     ror
          [[(a -> b)],[~ a,b]]
          rnot
               [[a,(a -> b)],[b]]
               limpl
                    [[a],[a,b]]
                    axiom
                    [[a,b],[b]]
                    axiom
180 msec.

yes
```

```
| ?- ghc tep_go(8).
[[],[[((a <-> b) <-> c) <-> (a <-> (b <-> c)))]]
requiv
     [[((a <-> b) <-> c)],[(a <-> (b <-> c))]]
     lequiv
          [[(a <-> b),c],[(a <-> (b <-> c))]]
          lequiv
               [[a,b,c],[(a <-> (b <-> c))]]
               requiv
                    [[a,a,b,c],[(b <-> c)]]
                    requiv
                         [[b,a,a,b,c],[c]]
                         axiom
                         [[c,a,a,b,c],[b]]
                         axiom
                    [[(b <-> c),a,b,c],[a]]
                    axiom
               [[c],[a,b,(a <-> (b <-> c))]]
               requiv
                    [[a,c],[a,b,(b <-> c)]]
                    axiom
                    [[(b <-> c),c],[a,b,a]]
                    lequiv
                         [[b,c,c],[a,b,a]]
                         axiom
                         [[c],[b,c,a,b,a]]
                         axiom
          [[],[(a <-> b),c,(a <-> (b <-> c))]]
          requiv
               [[a],[b,c,(a <-> (b <-> c))]]
               requiv
                    [[a,a],[b,c,(b <-> c)]]
                    requiv
                         [[b,a,a],[b,c,c]]
                         axiom
                         [[c,a,a],[b,c,b]]
                         axiom
                    [[(b <-> c),a],[b,c,a]]
                    axiom
               [[b],[a,c,(a <-> (b <-> c))]]
               requiv
                    [[a,b],[a,c,(b <-> c)]]
                    axiom
                    [[(b <-> c),b],[a,c,a]]
                    lequiv
                         [[b,c,b],[a,c,a]]
                         axiom
                         [[b],[b,c,a,c,a]]
                         axiom
     [[(a <-> (b <-> c))],[((a <-> b) <-> c)]]
     lequiv
          [[a,(b <-> c)],[((a <-> b) <-> c)]]
          lequiv
               [[a,b,c],[((a <-> b) <-> c)]]
               requiv
                    [[(a <-> b),a,b,c],[c]]
                    axiom
                    [[c,a,b,c],[(a <-> b)]]
                    requiv
                         [[a,c,a,b,c],[b]]
                         axiom
                         [[b,c,a,b,c],[a]]
                         axiom
               [[a],[b,c,((a <-> b) <-> c)]]
               requiv
```

```
                    [[(a <-> b),a],[b,c,c]]
                    lequiv
                         [[a,b,a],[b,c,c]]
                         axiom
                         [[a],[a,b,b,c,c]]
                         axiom
                    [[c,a],[b,c,(a <-> b)]]
                    axiom
               [[],[a,(b <-> c),((a <-> b) <-> c)]]
               requiv
                    [[b],[a,c,((a <-> b) <-> c)]]
                    requiv
                         [[(a <-> b),b],[a,c,c]]
                         lequiv
                              [[a,b,b],[a,c,c]]
                              axiom
                              [[b],[a,b,a,c,c]]
                              axiom
                         [[c,b],[a,c,(a <-> b)]]
                         axiom
                    [[c],[a,b,((a <-> b) <-> c)]]
                    requiv
                         [[(a <-> b),c],[a,b,c]]
                         axiom
                         [[c,c],[a,b,(a <-> b)]]
                         requiv
                              [[a,c,c],[a,b,b]]
                              axiom
                              [[b,c,c],[a,b,a]]
                              axiom
760 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 99.520000
```

```
takagi@icot32[11.3]> ../sicsghc
*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)
***  by Takashi Chikayama and Shigeyuki Takagi, ICOT
*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT

yes
| ?- ghccompile('waltz.ghc').
waltz1/0, waltz2/0, waltz3/0, waltz4/0, waltz1/1, waltz2/1, waltz3/1, ana/5,
conv/5, conv1/4, wr/1, waltz/3, waltz_go/2, p/5, p1/5, send/3, send1/2, ack/5,
ack_s/5, same/3, ack1/4, filtack/5, junctions/2, filter/4, filter/5, filter1/3,
append/3, filter2/5, END.
[compiling /tmp/ghca14853...]
[/tmp/ghca14853 compiled in 130920 msec.]

yes
| ?- ghc waltz1.
1 - a - [(in,p,out)]
2 - 1 - [(out,nil,in)]
3 - a - [(in,p,out)]
4 - f - [(p,p,p)]
5 - 1 - [(out,nil,in)]
6 - a - [(in,p,out)]
7 - 1 - [(out,nil,in)]
200 msec.

yes
| ?- ghc waltz2.
1 - a - [(in,p,out)]
2 - 1 - [(out,nil,in)]
3 - t - [(out,in,in)]
4 - 1 - [(out,nil,in)]
5 - a - [(in,p,out)]
6 - 1 - [(out,nil,in)]
7 - a - [(in,p,out)]
8 - 1 - [(out,nil,in)]
9 - f - [(p,p,p)]
10 - a - [(p,m,p)]
11 - 1 - [(out,nil,m)]
12 - a - [(in,p,out)]
13 - f - [(p,p,p)]
379 msec.

yes
| ?- ghc waltz3.
1 - a - [(in,p,out)]
2 - 1 - [(out,nil,in)]
3 - a - [(in,p,out)]
4 - 1 - [(out,nil,in)]
5 - a - [(in,p,out)]
6 - 1 - [(out,nil,in)]
7 - f - [(p,p,p)]
8 - a - [(p,m,p)]
9 - f - [(p,p,p)]
10 - a - [(p,m,p)]
11 - f - [(p,p,p)]
12 - a - [(p,m,p)]
13 - f - [(m,m,m)]
400 msec.

yes
| ?- ghc waltz4.
1 - a - [(in,p,out)]
2 - 1 - [(out,nil,in)]
3 - a - [(in,p,out)]
4 - 1 - [(out,nil,in)]
```

```
5 - a - [(in,p,out)]
6 - i - [(in,m,out)]
7 - a - [(in,p,out)]
8 - l - [(out,nil,in)]
9 - f - [(p,p,p)]
10 - a - [(p,m,p)]
11 - f - [(p,p,p)]
300 msec.

yes
| ?- ^D [ End break ]
[ End of SICStus execution ]

user time 143.040000
```

bpl.doc

(0) Date: 27-Jul-87, written by A. Okumura
Modified: 25-dec-87 by S. Takagi

(1) Program name: bestpath_layered
(best path problem using the layered-stream method)

(2) Author: A. Okumura, ICOT 1st Lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the Problem:

A best path search problem on a network.

The network consists of nodes and arcs. Each arc connects two nodes. A node is connected to at least one arc. Each arc has non-negative cost. A path is a route from a given start node to a given end node through arvitrary arcs. Cost of a path is defined as adding each cost of arcs which constructs the path.

This problem is to find a minimum cost path between the start and the end node.

(5) Algorithm:

Node names and edge costs are propagated in sequences along edges. If a node receives a sequence which includes the name of the node itself, the sequence is eliminated. While the check, the total cost of the sequence is calculated. It is also propagated to other checking processes, and no more sequences which have higher costs than it is output from any nodes. If the goal node receives a sequence from the start node which does not include the name of the goal node, it is a candidate to be output. After the goal node receives the last input, the candidate sequence which has the lowest cost is output as an answer.

(6) Process structure:

One process is generated for each node. A node process prepares a pair of its node name and a layered stream as its output. The stream holds good paths from the start node to that node. The layered stream is made from the input from its neighbors by eliminating non-good paths. Non-good paths mean that the paths contain the current node name (i.e. some loop is in the path) or which have higher cost than a path already output.

(7) Pragma:

Not set yet.

(8) Program:

Top-level predicate is "bestPath/3".
This predicate defines the configuration of the graph,
generates a process for each node of the graph,
and layered streams are set up along the edges.

Predicates like "graphN/i" are the subsidual predicate of "bestPath/3".
They are for deviding the graph to smaller subgraphs so as to be compiled.

"node/8" produces the primary output bindings for each node.
The first clause of node/8 is for the starting node.
The second clause is for the goal node.

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

bestPath(S,G,Path) :- true |
        node(Min,Fin,S,G,n0,[2-N1,1-N6],N0,Path),
        node(Min,Fin,S,G,n1,[2-N0,3-N2,2-N7],N1,Path),
        node(Min,Fin,S,G,n6,[1-N0,1-N7,2-N12],N6,Path),
        node(Min,Fin,S,G,n7,[2-N1,1-N6,3-N13],N7,Path),
        node(Min,Fin,S,G,n12,[2-N6,3-N13,4-N18],N12,Path),
        node(Min,Fin,S,G,n13,[3-N7,3-N12,5-N14,3-N19],N13,Path),
        graph1(Min,Fin,S,G,Path,N1,N2,N12,N13,N14,N18,N19).

graph1(Min,Fin,S,G,Path,N1,N2,N12,N13,N14,N18,N19) :- true |
        node(Min,Fin,S,G,n2,[3-N1,4-N3,4-N3],N2,Path),
        node(Min,Fin,S,G,n3,[4-N2,2-N4,4-N9],N3,Path),
        node(Min,Fin,S,G,n8,[2-N2,3-N9,4-N14],N8,Path),
        node(Min,Fin,S,G,n9,[4-N3,3-N8,6-N10,8-N15],N9,Path),
        node(Min,Fin,S,G,n14,[4-N8,5-N13,7-N15],N14,Path),
        node(Min,Fin,S,G,n15,[8-N9,7-N14,9-N21],N15,Path),
        graph2(Min,Fin,S,G,Path,N3,N4,N9,N10,N12,N13,N15,N18,N19,N21).

graph2(Min,Fin,S,G,Path,N3,N4,N9,N10,N12,N13,N15,N18,N19,N21) :- true |
        node(Min,Fin,S,G,n4,[2-N3,1-N5,3-N10],N4,Path),
        node(Min,Fin,S,G,n5,[1-N4,1-N11],N5,Path),
        node(Min,Fin,S,G,n10,[3-N4,6-N9,2-N11,6-N16],N10,Path),
        node(Min,Fin,S,G,n11,[1-N5,2-N10,3-N17],N11,Path),
        node(Min,Fin,S,G,n16,[6-N10,4-N17,7-N22],N16,Path),
        node(Min,Fin,S,G,n17,[3-N11,4-N16,4-N23],N17,Path),
        graph3(Min,Fin,S,G,Path,N12,N13,N15,N16,N17,N18,N19,N21,N22,N23).

graph3(Min,Fin,S,G,Path,N12,N13,N15,N16,N17,N18,N19,N21,N22,N23) :- true |
        node(Min,Fin,S,G,n18,[4-N12,2-N19,3-N24],N18,Path),
        node(Min,Fin,S,G,n19,[3-N13,2-N18,4-N20],N19,Path),
        node(Min,Fin,S,G,n24,[3-N18,2-N25,2-N30],N24,Path),
        node(Min,Fin,S,G,n25,[2-N24,5-N26,2-N31],N25,Path),
        node(Min,Fin,S,G,n30,[2-N24,1-N31],N30,Path),
        node(Min,Fin,S,G,n31,[2-N25,1-N30,1-N32],N31,Path),
        graph4(Min,Fin,S,G,Path,N15,N16,N17,N19,N20,N21,N22,N23,N25,N26,N31,N32).

graph4(Min,Fin,S,G,Path,N15,N16,N17,N19,N20,N21,N22,N23,N25,N26,N31,N32) :- true
        node(Min,Fin,S,G,n20,[4-N19,7-N21,6-N26],N20,Path),
        node(Min,Fin,S,G,n21,[9-N15,7-N20,8-N27],N21,Path),
        node(Min,Fin,S,G,n26,[6-N20,5-N25,3-N32],N26,Path),
        node(Min,Fin,S,G,n27,[8-N21,3-N28,4-N33],N27,Path),
        node(Min,Fin,S,G,n32,[3-N26,1-N31,3-N33],N32,Path),
        node(Min,Fin,S,G,n33,[4-N27,3-N32,2-N34],N33,Path),
        graph5(Min,Fin,S,G,Path,N16,N17,N22,N23,N27,N28,N33,N34).

graph5(Min,Fin,S,G,Path,N16,N17,N22,N23,N27,N28,N33,N34) :- true |
        node(Min,Fin,S,G,n22,[7-N16,1-N23,5-N28],N22,Path),
        node(Min,Fin,S,G,n23,[4-N17,1-N22,2-N29],N23,Path),
        node(Min,Fin,S,G,n28,[5-N22,3-N27,2-N29,2-N34],N28,Path),
        node(Min,Fin,S,G,n29,[2-N23,2-N28,1-N35],N29,Path),
        node(Min,Fin,S,G,n34,[2-N28,2-N33,2-N35],N34,Path),
        node(Min,Fin,S,G,n35,[1-N29,2-N34],N35,Path).

node(Min,Fin,_,G,G,In,Out,Path) :- true |
        Out = nil, lastFilter(Min,Min,0-[G],nil,In,Path), announce(Path,Fin).
node(Min,_,S,_,S,_,Out,_) :- true | Out = S*begin.
node(Min0,Fin,S,G,N,In,Out,_) :- S \= N, G \= N |
        filter(Fin,N,Min,Min1,0,In,In1), Out = N*In1,
        minMerge(Fin,Min0,Min1,Min).

announce(_-_,Out) :- true | Out = fin.
```

"filter/7" is for filtering out paths that include loops.
"minMerge/4" suppress the output which has higher cost than the former one.
"lastFilter/6" outputs the path which has the lowest cost.

(9) Source file: bpl.ghc
    This Document: bpl.doc

(10) Examples:

Invocation:

```
bestPath(Start,Goal,Path).
        where Start is the starting node,
        Goal is the goal node,
        and we get all the best pathes between Start and Goal
        as the third argument Path.
```

For example, run bestPath(n0,n11,Path). We get the following result.

```
Path = 13-[[n0,n1,n2,n3,n4,n5,n11]]
```

(11) Evaluation data:

Not recorded here.

```
filter(fin,_,_,_,_,_,_) :- true | true.
filter(Fin,Node,[Min1,Min2|Mins],Min,Cost,In,Out) :- Min1 >= Cost |
        filter(Fin,Node,[Min2|Mins],Min,Cost,In,Out).
filter(Fin,_,[Min1|_],Min,Cost,_,Out) :- Min1 < Cost | Min = [], Out = [].
filter(Fin,Node,Mins,Min,Cost,[C-Node*In1|Ins],Out) :- true |
        filter(Fin,Node,Mins,Min,Cost,Ins,Out).
filter(Fin,Node,Mins,Min,Cost,[C-nil|Ins],Out) :- true |
        filter(Fin,Node,Mins,Min,Cost,Ins,Out).
filter(Fin,Node,Mins,Min,C0,[C-N*In1|Ins],Out) :- N \= Node |
        Cost := C0+C,
        filter(Fin,Node,Mins,M1,Cost,In1,Out1),
        filter(Fin,Node,Mins,M2,C0,  Ins,Outs),
        minMerge(Fin,M1,M2,Min),
        Out = [C-N*Out1|Outs].
filter(Fin,_,_,Min,Cost,begin,Out) :- true | Min = [Cost], Out = begin.
filter(Fin,_,_,Min,_,[],Out) :- true | Min = [], Out = [].


lastFilter([Min1,Min2|Mins],Min,Cost-Stack,Prev,In,Out) :- Min1 >= Cost |
        lastFilter([Min2|Mins],Min,Cost-Stack,Prev,In,Out).
lastFilter([Min1|_],Min,Cost-_,Prev,_,Out) :- Min1 < Cost |
        Out = Prev, Min = [].
lastFilter(_,Min,Cost-Stack,nil,begin,Out) :- true |
        Out = Cost-[Stack], Min = [Cost].
lastFilter(_,Min,Cost-Stack,Cost-Path0,begin,Out) :- true |
        Out = Cost-[Stack|Path0], Min = [].
lastFilter(_,Min,Cost-_,Cost0-Path0,begin,Out) :- Cost > Cost0 |
        Out = Cost0-Path0, Min = [].
lastFilter(_,Min,Cost-Stack,Cost0-_,begin,Out) :- Cost < Cost0 |
        Out = Cost-[Stack], Min = [Cost].
lastFilter(_,Min,_,Prev,[],Out) :- true | Out = Prev, Min = [].
lastFilter(Mins,Min,Stack,Prev,[_-nil|Ins],Out) :- true |
        lastFilter(Mins,Min,Stack,Prev,Ins,Out).
lastFilter(Mins,Min,Cost-Stack,Prev,[C-N*In1|Ins],Out) :- true |
        Cost1 := Cost+C,
        lastFilter(Mins,M1,Cost1-[N|Stack],Prev,In1,Mid),
        lastFilter(Mins,M2,Cost-Stack,Mid,Ins,Out),
        minMerge(_,M1,M2,Min).

minMerge(fin,_,_,_) :- true | true.
minMerge(Fin,[M|A],B,Out) :- true | Out = [M|Outs], minM1(Fin,M,A,B,Outs).
minMerge(Fin,A,[M|B],Out) :- true | Out = [M|Outs], minM1(Fin,M,A,B,Outs).
minMerge(Fin,[],In,Out) :- true | Out = In.
minMerge(Fin,In,[],Out) :- true | Out = In.

minM1(fin,_,_,_,_) :- true | true.
minM1(Fin,M0,[M|A],B,Out) :- M0 > M | Out = [M|Outs], minM1(Fin,M,A,B,Outs).
minM1(Fin,M0,A,[M|B],Out) :- M0 > M | Out = [M|Outs], minM1(Fin,M,A,B,Outs).
minM1(Fin,M0,[M|A],B,Out) :- M0 =< M | minM1(Fin,M0,A,B,Out).
minM1(Fin,M0,A,[M|B],Out) :- M0 =< M | minM1(Fin,M0,A,B,Out).
minM1(Fin,M0,[],In,Out) :- true | minM2(Fin,M0,In,Out).
minM1(Fin,M0,In,[],Out) :- true | minM2(Fin,M0,In,Out).

minM2(fin,_,_,_) :- true | true.
minM2(Fin,M0,[M|In],Out) :- M0 > M | Out = [M|Outs], minM2(Fin,M,In,Outs).
minM2(Fin,M0,[M|In],Out) :- M0 =< M | minM2(Fin,M0,In,Out).
minM2(Fin,_,[],Out) :- true | Out = [].
```

bpmon.doc

(0) Date: 3-Jun-87, written by K. Taki
Modified: 25-Dec-87 by S. Takagi

(1) Program name: bestpath

Bestpath program using a single monitor process

(2) Author: K. Taki, ICOT 4th Lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

A best path search problem on a network.
A network is constructed from nodes and arcs. Each arc connects two nodes. A node is connected to at least one arc. Each arc has a non-negative cost. A path is a route from a given start node to a given end node through arbitrary arcs. The cost of a path is defined as the total cost of arcs forming the path.
The problem is to find the minimum cost path when the start and end nodes are given.

(5) Algorithm:

In this path search algorithm, each growing path is an activity. It is called a process here.

A process has a state. The state contains path information and the cost of the growing path. The process has traversed the path and the cost is the sum of all the arc costs in the growing path.

The process forks traversing arcs and expands its path information.

There is another activity called a monitor process. The monitor holds the current best path information at any time. When a new best path is found, the monitor broadcasts the new minimum cost to all the processes.

i. Normal process activity for expanding the path:
When a process arrives at a node,
the process checks whether the node is the end node or not, and whether a broadcast message arrives or not.
If neither is the case,
the process checks whether the node has been visited the first time.
If not, the process terminates at once.
If that node has been visited the first time,
the process records the node number as its path information.
The process forks by n according to the number of connecting arcs, and newly forked processes travel along each connecting arc.

A new forked process adds the arc cost to its cost information, and the forked process visits the next node.

ii. When a process arrives at the end node:
When a process arrives at the end node and no broadcast message has arrived (it means that a new path has been completed).
The process reports the state information to the monitor process, then terminates itself.

iii. Monitor process activity:
When the monitor process receives the reported state information,
it compares the reported cost with its own recorded cost (minimum cost).
If the reported cost is higher,
the monitor process abandons the reported information.
If the reported cost is equal to the recorded cost,
the monitor adds the reported path information to its recorded
(best) path information.
If the reported cost is lower than the recorded cost,
the monitor changes the whole record with the reported information,
and broadcasts the reported cost to all the live processes.
(This means that the reported cost is the minimum detected
by the monitor.)

iv. When a broadcast message arrives at a process:
When a broadcast message (newly reported minimum cost) arrives
at a process, the process compares the broadcast cost
with its own cost.
If the broadcast cost is less than or equal to its own cost,
the process terminates itself at once.
(This means that there is a better path.)
If the broadcast cost is higher than its own cost,
the process continues to operate as long as its own cost
does not exceed the broadcast cost.
If its own cost exceeds the broadcast cost, the process terminates.

v. Termination:
When all the processes except the monitor terminate, the information
recorded in the monitor process is the best path information.

(6) Process structure:

There are one monitor process and many path processes.
Path processes fork according to the network graph.
Report streams of the forked path processes are merged
and connected to the monitor process.
The broadcast stream of the monitor process is shared by all path
processes.

(7) Pragma: Not supported

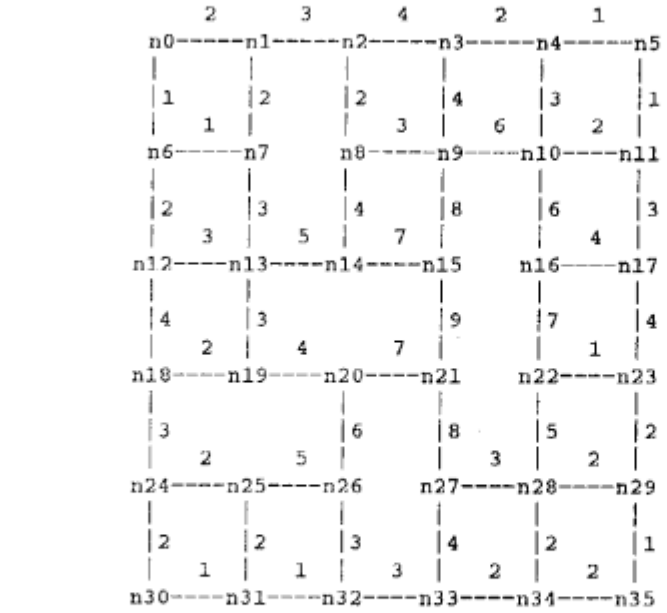
(8) Program:

bestpath/3:	Top level
evaluator/5:	Monitor process
path/6:	Path process
p/7:	This predicate checks loops. If the path has a loop, that path is terminated at once. If it does not, the connecting arc list (edge list) is selected and path/6 is recursively called.
edge/2:	Network data

(9) Source file: bpmon.ghc
This document: bpmon.doc

(10) Examples:

Network:



Start the program:

```
%-----------------------------------------------------------
%%% Example on GHC3 system

yes
| ?- ghc bestpath(n0,n8,Ans).

259 reductions and 54 suspensions in 34 cycles and 449 msec (576 rps)
The maximum number of reducible goals is 16/28 in the 25th cycle.
The maximum length of the queue is 29.

Ans = 7*[[n8,n2,n1,n0]]

yes
| ?- ghc bestpath(n15,n22,Ans).

4115 reductions and 666 suspensions in 86 cycles and 6700 msec (614 rps)
The maximum number of reducible goals is 114/207 in the 49th cycle.
The maximum length of the queue is 208.

Ans = 24*[[n22,n23,n17,n11,n5,n4,n3,n9,n15],[n22,n23,n17,n11,n10,n9,n15]]

%%%  This example shows that there are two best paths with the same cost.
```

(11) Evaluation data:

Measurement of bestpath problem in GHC3 '87.4.2 TAKI

route	suspensions reductions	cycles	msec	rps	max reducible goals at nth-cycle*1	*2 queue leng.
Taki-monitor-method						
n0-n7	57	13	17	98	581	9/11at11	11
n0-n14	576	113	51	876	657	31/43at25	48
n0-n21	4991	773	108	7935	628	144/260at55	260
n0-n28	23386	2868	135	38175	612	537/891at89	899
n0-n35	31681	4049	153	56952	336	1158/2146at91	2146
n4-n32	15899	2006	108	25732	617	391/720at64	720
n8-n26	6424	1038	120	10455	614	146/269at43	269
n13-n28	8334	1153	107	13815	603	173/299at74	314
Ichiyoshi-method (bestpath_termdet_ichiyoshi.ghc)						
n4-all	4592	858	393	23026	199	42/72at68	162
n8-all	4493	803	389	22183	202	42/135at87	164
n13-all	4778	1008	393	23525	203	44/158at112	160

*1 A/BatC: A = maximum number of reducible goals
B = number of goals (queue length) when A is measured
C = cycle number when A is measured

*2 queue leng.: maximum length of queue

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%-------- Bestpath problem --------
%       ( Taki-monitor-method )

bestpath(Start,Goal,Best_path) :- true |
        edge(Start,Next),
        path(Next,Goal,0*[Start],Path,Best_cost,10000),
        evaluator(Path,Best_cost,10000,[],Best_path).

evaluator([Total_cost*Path|Next],Best_cost,Cost_work,Path_work,Best_path) :-
        Total_cost < Cost_work |
        Best_cost = [Total_cost|NN],
        evaluator(Next,NN,Total_cost,[Path],Best_path).
evaluator([Total_cost*Path|Next],Best_cost,Cost_work,Path_work,Best_path) :-
        Total_cost = Cost_work |
        evaluator(Next,Best_cost,Cost_work,[Path|Path_work],Best_path).
evaluator([Total_cost*Path|Next],Best_cost,Cost_work,Path_work,Best_path) :-
        Total_cost > Cost_work |
        evaluator(Next,Best_cost,Cost_work,Path_work,Best_path).
evaluator([],Best_cost,Cost_work,Path_work,Best_path) :-  true |
        Best_cost = [],
        Best_path = Cost_work*Path_work.

% prior ;
path(Next,Goal,HIS,Path,[A_best_cost|Cost_next],Best_cost_work) :-
        true | path(Next,Goal,HIS,Path,Cost_next,A_best_cost).
% end prior ;

path(_,_,Acc_cost*_,P0,_,B_work) :-
        Acc_cost >= B_work | P0 = [].
path([Node*Cost|Next],Goal,Acc_cost*History,P0,B_cost,B_work) :-
        Acc_cost < B_work, Goal \= Node |
        p(History,Goal,Node,*(Acc_cost,History,Cost),P1,B_cost,B_work),
        path(Next,Goal,Acc_cost*History,P2,B_cost,B_work),
        merge(P1,P2,P0).
path([],_,_,P0,_,_) :- true | P0 = [].
path([Goal*Cost|Next],Goal,Acc_cost*History,P0,B_cost,B_work) :-
        Acc_cost < B_work |
        Total_cost := Acc_cost + Cost,
        P0 = [Total_cost*[Goal|History]|P1],
        path(Next,Goal,Acc_cost*History,P1,B_cost,B_work).

p([N1|Ns],Goal,N,HIS,P0,B_cost,B_work) :- N \= N1 |
        p(Ns,Goal,N,HIS,P0,B_cost,B_work).
p([],Goal,N,*(Acc_cost,History,Cost),P0,B_cost,B_work) :- true |
        edge(N,Next_next),
        New_acc_cost := Acc_cost + Cost,
        path(Next_next,Goal,New_acc_cost*[N|History],P0,B_cost,B_work).
p([N|_],_,N,_,P0,_,_) :- true | P0 = [].

edge(n0,Nodes) :- true | Nodes = [n1*2,n6*1].
edge(n1,Nodes) :- true | Nodes = [n0*2,n2*3,n7*2].
edge(n2,Nodes) :- true | Nodes = [n1*3,n3*4,n8*2].
edge(n3,Nodes) :- true | Nodes = [n2*4,n4*2,n9*4].
edge(n4,Nodes) :- true | Nodes = [n3*2,n5*1,n10*3].
edge(n5,Nodes) :- true | Nodes = [n4*1,n11*1].
edge(n6,Nodes) :- true | Nodes = [n0*1,n7*1,n12*2].
edge(n7,Nodes) :- true | Nodes = [n1*2,n6*1,n13*3].
edge(n8,Nodes) :- true | Nodes = [n2*2,n9*3,n14*4].
edge(n9,Nodes) :- true | Nodes = [n3*4,n8*3,n10*6,n15*8].
edge(n10,Nodes) :- true | Nodes = [n4*3,n9*6,n11*2,n16*6].
edge(n11,Nodes) :- true | Nodes = [n5*1,n10*2,n17*3].
edge(n12,Nodes) :- true | Nodes = [n6*2,n13*3,n18*4].
```

```
edge(n13,Nodes) :- true | Nodes = [n7*3,n12*3,n14*5,n19*3].
edge(n14,Nodes) :- true | Nodes = [n8*4,n13*5,n15*7].
edge(n15,Nodes) :- true | Nodes = [n9*8,n14*7,n21*9].
edge(n16,Nodes) :- true | Nodes = [n10*6,n17*4,n22*7].
edge(n17,Nodes) :- true | Nodes = [n11*3,n16*4,n23*4].
edge(n18,Nodes) :- true | Nodes = [n12*4,n19*2,n24*3].
edge(n19,Nodes) :- true | Nodes = [n13*3,n18*2,n20*4].
edge(n20,Nodes) :- true | Nodes = [n19*4,n21*7,n26*6].
edge(n21,Nodes) :- true | Nodes = [n15*9,n20*7,n27*8].
edge(n22,Nodes) :- true | Nodes = [n16*7,n23*1,n28*5].
edge(n23,Nodes) :- true | Nodes = [n17*4,n22*1,n29*2].
edge(n24,Nodes) :- true | Nodes = [n18*3,n25*2,n30*2].
edge(n25,Nodes) :- true | Nodes = [n24*2,n26*5,n31*2].
edge(n26,Nodes) :- true | Nodes = [n20*6,n25*5,n32*3].
edge(n27,Nodes) :- true | Nodes = [n21*8,n28*3,n33*4].
edge(n28,Nodes) :- true | Nodes = [n22*5,n27*3,n29*2,n34*2].
edge(n29,Nodes) :- true | Nodes = [n23*2,n28*2,n35*1].
edge(n30,Nodes) :- true | Nodes = [n24*2,n31*1].
edge(n31,Nodes) :- true | Nodes = [n25*2,n30*1,n32*1].
edge(n32,Nodes) :- true | Nodes = [n26*3,n31*1,n33*3].
edge(n33,Nodes) :- true | Nodes = [n27*4,n32*3,n34*2].
edge(n34,Nodes) :- true | Nodes = [n28*2,n33*2,n35*2].
edge(n35,Nodes) :- true | Nodes = [n29*1,n34*2].


merge([A|X],Y,Z) :- A \= [] | Z = [A|W], merge(Y,X,W).
merge(X,[A|Y],Z) :- A \= [] | Z = [A|W], merge(Y,X,W).
merge([],Y,Z)    :- true    | Z = Y.
merge(X,[],Z)    :- true    | Z = X.
```

bpsc.doc

(0) Date: 2-Jul-87, written by N. Ichiyoshi
Modified: 25-Dec-87 by S. Takagi

(1) Program name: bestpath
(bestpath program with the shortcircuit method as termination detection)

(2) Author: N. Ichiyoshi

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

Given a network with a non-negative cost assigned to each edge, and a start node, find for each node the path with the minimum accumulated cost from the starting node.

(5) Algorithm:

The nodes in the network send a path and cost information packet cp(Cost,Path) to the adjacent nodes. At all time, each node keeps:
(1) C (the minimum cost found so far from the starting node), and
(2) P (a path that realizes cost C).

It maintains the above information as follows. When a node receives a new path and cost information packet cp(Cost,Path) from any one of its neighboring nodes, it does the following. If Cost >= C, then the node simply ignores Cost and Path. Otherwise, it updates C and P to Cost and Path, and sends cost and path information packets to its neighboring nodes. Node (n) sends a path and cost information cp(Cost+EC,[n|Path]) where EC is the cost of the edge from n to n' and [n|Path] is Path extended by appending n to the top.

The above procedure is repeated until there are no path and cost information packets in the network. The finiteness of the network guarantees that this state is reached in a finite time. (There are only a finite number of non-circular paths in the network, and circular paths are guaranteed to be discarded because of the algorithm and the non-negative costs of the edges.)

This program detects this state by the shortcircuit technique. Shortcircuit switches are attached to all packets. The shortcircuit switch is closed when the packet is discarded. It splits into serially connected switches when the packet spawns child packets. The circuit attached to the initial packet is closed when all descendant circuits are closed, i.e. when all descendant packets are discarded.

(6) Process structure:

One node in the network has one corresponding node process. A node process is connected to its neighboring node processes by one input and one output stream.

(7) Pragma: Not attached

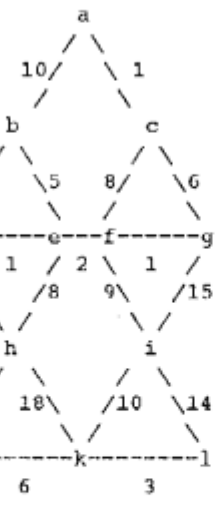
```
(8) Program:

(main part)
        node/8: Node process
        send_cp/6:      Subroutine for sending path/cost information packets
                to neighboring nodes
        closeOuts/1:    Closes a stream
        bp/4:           Top level goal
(network generation)
        augment_edges/2: Augments edge information to include both-way streams
        del/3:          Deletes an element from a list
        gen_nodes/5:    Node generation routine
        gen_node/8:     Makes a node process out of node information
(utility predicates)
        merge_all/2:    n-way merger
        merge/3:        Standard merger
        length/2,length/3: Gives the list length
        write_result/2: Writes out stream elements
        bp_sne/3:       Runs the program with the start node, set of nodes,
                        and set of edges given
        bp_ne/2:        Runs the program with the set of nodes
                        and set of edges given
(test program)
        bp_ex1_n/1:     First example
        bp_ex2_n/1:     Second example
        bp_ex3_s/1:     Third example

(9) Source file:   bpsc.ghc     247 lines (111 lines of which are test data)
    This Document: bpsc.doc

(10) Examples:

Invocation:
(To run example 1 with start node a)
        :- ghc bp_ex1_n([a,b,c,d,e,f,g,h]).
(To run example 2 with start node a)
        :- ghc bp_ex2_n([a,b,c,d,e,f,g,h,i,j,k,l]).
(To run example 3 with start node n0)
        :- ghc bp_ex3_s(n0).

(11) Evaluation data: Not recorded
```

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Best Path Problem
%
%       N. Ichiyoshi (according to Ohki-san's idea of using short circuit)
%       April 1987
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       node process            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% node(In,Outs,Cout,C,P, End,R0-R,Cs,N)
%
%       In      : input stream (including the control stream)
%       Outs    : output streams
%       C       : minimum cost so far
%       P       : path with cost C
%       End     : termination flag
%       R0-R    : result d-list
%       Cs      : costs of outgoing edges (constant)
%       N       : node identifier (constant)
%
node([cp(Cost,Path,T0-T)|In],Outs,C,P, End,R0-R, Cs,N) :-
    Cost >= C |
    T0 = T,
    node(In,Outs,C,P, End,R0-R,Cs,N).
node([cp(Cost,Path,T0-T)|In],Outs,C,P, End,R0-R,Cs,N) :-
    Cost < C |
    send_cp(Outs,Cs,Cost,[N|Path], T0-T, NewOuts),
    node(In,NewOuts,Cost,Path, End,R0-R,Cs,N).
node(_,Outs,C,P, end,R0-R,_,N) :- true |
    R0 = [N-C-P|R],
    closeOuts(Outs).

send_cp([],_,_,_, T0-T, NewOuts) :- true | T0 = T, NewOuts = [].
send_cp([Out|Outs],[C|Cs],Cost,NewPath, T0-T, NewOuts) :-
    NewCost := Cost+C |
    Out = [cp(NewCost,NewPath,T0-T1)|Out1],
    NewOuts = [Out1|NewOuts1],
    send_cp(Outs,Cs,Cost,NewPath, T1-T, NewOuts1).

closeOuts([]) :- true | true.
closeOuts([Out|Outs]) :- true | Out = [], closeOuts(Outs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Process generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% bp(S,Ns,Es, R)
%
%       S       : start node
%       Ns      : list of nodes (a node has a constant value)
%       Es      : list of edges (an edge is of form e(Node1-Node2,Cost))
%       R       : result (list of nodes with the minimum cost from start node
%                 and a path with the minimum cost)
%
bp(S,Ns,Es, R) :- true |
    augment_edges(Es, AEs),
    del(S,Ns, Ms),
```

```
    gen_nodes([S|Ms],AEs,End,R-[],S).

augment_edges([], AEs) :- true | AEs = [].
augment_edges([e(N1-N2,C)|Es], AEs) :- true |
    AEs = [e(N1-N2,C,_-_)|AEs1],
    augment_edges(Es, AEs1).

del(N,[M|Ns], Ms) :- N = M | Ms = Ns.
del(N,[M|Ns], Ms) :- N \= M | Ms = [M|Ms1], del(N,Ns, Ms1).
del(N,[], Ms) :- true | Ms = [].

gen_nodes([N|Ns],AEs,End,R0-R,S) :- true |
    gen_node(N,AEs,[],[],[],End,R0-R1,S),
    gen_nodes(Ns,AEs,End,R1-R,S).
gen_nodes([],_,_,R0-R,_) :- true | R0 = R.

gen_node(N,[e(N-_,Cost,Self-Other)|AEs],Ins,Outs,Costs,End,RR,S) :- true |
    Ins1 = [Self|Ins],
    Outs1 = [Other|Outs],
    Costs1 = [Cost|Costs],
    gen_node(N,AEs,Ins1,Outs1,Costs1,End,RR,S).
gen_node(N,[e(_-N,Cost,Other-Self)|AEs],Ins,Outs,Costs,End,RR,S) :- true |
    Ins1 = [Self|Ins],
    Outs1 = [Other|Outs],
    Costs1 = [Cost|Costs],
    gen_node(N,AEs,Ins1,Outs1,Costs1,End,RR,S).
gen_node(N,[e(O1-O2,_,_)|AEs],Ins,Outs,Costs,End,RR,S) :-
    N \= O1, N \= O2 |
    gen_node(N,AEs,Ins,Outs,Costs,End,RR,S).
gen_node(N,[],Ins,Outs,Costs,End,RR,S) :- N \= S |
    length(Outs, EC),
    merge_all(Ins, In),
    node(In,Outs,99999,?, End,RR,Costs,N).
gen_node(N,[],Ins,Outs,Costs,End,RR,S) :- N = S |
    length(Outs, EC),
    merge_all([[cp(0,[],End-end)]|Ins], In),
    node(In,Outs,99999,?, End,RR,Costs,N).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Utility
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

merge_all([], Out) :- true | Out = [].
merge_all([In|Ins], Out) :- true |
    merge(In,Out1, Out),
    merge_all(Ins, Out1).

merge([],Ys, Zs) :- true | Zs = Ys.
merge(Xs,[], Zs) :- true | Zs = Xs.
merge([X|Xs],Ys, Zs) :- true | Zs = [X|Zs1], merge(Xs,Ys, Zs1).
merge(Xs,[Y|Ys], Zs) :- true | Zs = [Y|Zs1], merge(Xs,Ys, Zs1).

length(L, N) :- true | length(L, 0,N).

length([], K,N) :- true | N = K.
length([_|L], K,N) :- K1 := K+1 | length(L, K1,N).

write_result([], O) :- true | O = [].
write_result([X|Xs], O) :- true |
    O = [write(X),nl|NewO], write_result(Xs, NewO).

%
% Bp_sne is given the start node, list of nodes and list of edges.
% Bp_ne is given list of nodes and list of edges. (The start node becomes
%  the first node in the list of nodes.)
%
```

```
bp_unc(Start,Nodes,Edges) :- true |
    bp(Start,Nodes,Edges, Result),
    write_result(Result, 0),
    outstream(0).

bp_ne(Nodes,Edges) :- Nodes = [Start|_] |
    bp_sne(Start,Nodes,Edges).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Three examples
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```


```
bp_ex1_n(Nodes) :-
    true |
    bp_ne(Nodes,
      [e(a-f,10),e(a-g,10),e(a-b,1),
       e(b-g,10),e(b-c,1),
       e(c-h,10),e(c-d,1),
       e(d-e,1),
       e(e-h,10),e(e-f,10),e(e-g,1),
       e(g-h,1)
      ]).
```




```
bp_ex2_n(Nodes) :-
    true |
    bp_ne(Nodes,
      [e(a-b,10),e(a-c,1),
       e(b-d,3),e(b-e,5),
       e(c-f,8),e(c-g,6),
       e(d-e,1),e(d-h,12),
```

```
        e(e-f,2),e(e-h,8),
        e(f-g,1),e(f-i,9),
        e(g-i,15),
        e(h-j,11),e(h-k,18),
        e(i-k,10),e(i-l,14),
        e(j-k,6),
        e(k-l,3)
       ]).

bp_ex3_s(Start) :-
    true |
    bp_sne(Start,
      [n0,n1,n2,n3,n4,n5,n6,n7,n8,n9,
       n10,n11,n12,n13,n14,n15,n16,n17,n18,n19,
       n20,n21,n22,n23,n24,n25,n26,n27,n28,n29,
       n30,n31,n32,n33,n34,n35
      ],
      [e(n0-n1,2),e(n0-n6,1),
       e(n1-n2,3),e(n1-n7,2),
       e(n2-n3,4),e(n2-n8,2),
       e(n3-n4,2),e(n3-n9,4),
       e(n4-n5,1),e(n4-n10,3),
       e(n5-n11,1),
       e(n6-n7,1),e(n6-n12,2),
       e(n7-n13,3),
       e(n8-n9,3),e(n8-n14,4),
       e(n9-n10,6),e(n9-n15,8),
       e(n10-n11,2),e(n10-n16,6),
       e(n11-n17,3),
       e(n12-n13,3),e(n12-n18,4),
       e(n13-n14,5),e(n13-n19,3),
       e(n14-n15,7),
       e(n15-n21,8),
       e(n16-n17,4),e(n16-n22,7),
       e(n17-n23,4),
       e(n18-n19,2),e(n18-n24,3),
       e(n19-n20,4),
       e(n20-n21,7),e(n20-n26,6),
       e(n21-n27,8),
       e(n22-n23,1),e(n22-n28,5),
       e(n23-n29,2),
       e(n24-n25,2),e(n24-n30,2),
       e(n25-n26,5),e(n25-n31,2),
       e(n26-n32,3),
       e(n27-n28,3),e(n27-n33,4),
       e(n28-n29,2),e(n28-n34,2),
       e(n29-n35,1),
       e(n30-n31,1),
       e(n31-n32,1),
       e(n32-n33,3),
       e(n33-n34,2),
       e(n34-n35,2)
      ]).
```

gdpath.doc

(0) Date: 16-Jul-87, written by K. Wada
Modified: 27-Jul-87 by A. Okumura
Modified: 25-Dec-87 by S. Takagi

(1) Program name: goodpath_layered
(good path problem using the layered-stream method)

(2) Author: A. Okumura, ICOT 1st Lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

The good path problem is to find all acyclic paths from the start node to the goal node in a given graph.

(5) Algorithm:

Node names are propagated in sequences along edges. When a node receives a sequence which includes the name of the node, the sequence is eliminated. If the goal node receives a sequence from the start node which does not include the name of the goal node, the sequence is output as an answer.

(6) Process structure:

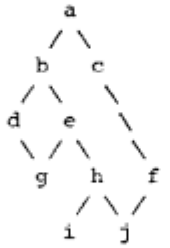
One process is generated for each node. A node process prepares a pair consisting of its node name and a layered stream as an output. The stream holds good paths from the start node to that node. The layered stream is made from the input from its neighbors by eliminating paths which contain the current node name.

(7) Pragma: Not set yet

(8) Program:

Top-level predicate is goodPath/3.
This predicate defines the configuration of the graph,
generates a process for each node of the graph,
and sets up layered streams along the edges.
node/6 produces the primary output bindings for each node.
The first clause of node/6 is for the starting node.
The second clause is for the goal node.
filter/3 is for filtering out paths that include loops.

The following graph is defined in this program:



(9) Source file: gdpath.ghc
This document: gdpath.doc

(10) Examples:

Invocation:

goodPath(Start,Goal,Path).
where Start is the starting node,
Goal is the goal node,
and all good paths between Start and Goal are obtained
as the third argument, Path.

For example, run goodPath(a,h,Path). The following result is obtained.

```
Path = h*[e*[b*[a*begin,d*[g*[]]],
             g*[d*[b*[a*begin]] ] ],
          i*[],
          j*[f*[c*[a*begin]]]       ].
```

It means that good paths between nodes a and h are :

a-b-e-h,
a-b-d-g-e-h,
a-c-f-j-h.

Note: The output of the value of Path is not formatted as shown above.
The formatting has been done by hand for legibility.

(11) Evaluation data:

Not recorded here.

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%  goodpath problem using layered stream

goodPath(Start, Goal, Path) :- true |
        node(Start, Goal, a, [B,C],   A, Path),
        node(Start, Goal, b, [A,E,D], B, Path),
        node(Start, Goal, c, [A,F],   C, Path),
        node(Start, Goal, d, [B,G],   D, Path),
        node(Start, Goal, e, [B,G,H], E, Path),
        node(Start, Goal, f, [C,J],   F, Path),
        node(Start, Goal, g, [D,E],   G, Path),
        node(Start, Goal, h, [E,I,J], H, Path),
        node(Start, Goal, i, [H],     I, Path),
        node(Start, Goal, j, [F,H],   J, Path).

node(Start, _, Start, _, Out, _) :- true | Out = Start*begin.
node(_, Goal, Goal, In, Out, Path) :- true |
        Out = [], Path = Goal*In.
node(Start, Goal, Node, In, Out, _) :- Start \= Node, Goal \= Node |
        Out = Node*In1, filter(In, Node, In1).

filter(begin, _, Out) :- true | Out = begin.
filter([], _, Out) :- true | Out = [].
filter([Node*_|In], Node, Out) :- true | filter(In, Node, Out).
filter([[]|In], Node, Out) :- true | filter(In, Node, Out).
filter([N*Ns|In], Node,Out) :- Node \= N |
        Out = [N*Ns1|Out1], filter(Ns, Node, Ns1), filter(In, Node, Out1).
```

life.doc

(0) Date: 1987-Jul-18, written by E. Sugino
Modified: 25-Dec-87 by S. Takagi

(1) Program name: life

(2) Author: E.Sugino, ICOT 4th lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem: The life game is a popular game.

(5) Algorithm:

Any node waits for the state of all neighbors.
A node changes its state as follows:
When a node is alive and there are 2 or 3 live nodes around the node, the node is alive at the next time period.
When a node was dead and there were 3 live nodes around it, the node becomes live at the next time period.
In other cases, the node becomes dead.

(6) Process structure:

Each node is a process. They are connected to each other.

(7) Pragma: None

(8) Program:

life_game/3: A predicate to invoke a life game pattern.
The first argument (M) is the number of cycles in the pattern.
The second argument is a list of integer 0 or 1, which makes a pattern with M cycles. The last argument (N) is a life cycle, and the pattern of the N-th time period displayed.
wr/3: Writes a pattern.

The following predicates make a mesh.
make_mesh, make_node, mg, temp_node, con,
parts of node (clauses in which the 11th argument is wait)

The node processes are nodes of the pattern.

The rest of the nodes make themselves the nodes of the next time period. They are over at the life cycle, and a token, result(S,St), is sent through the control stream of the nodes to obtain the last pattern.

(9) Source file: life.ghc
This document: life.doc

(10) Example:

A sample program glider:

```
        ?- ghc life_test(1).     % A glider: Original pattern
                                 % at the home position
        ?- ghc life_test(5).     % Every 4 cycles, the glider appears
                                 % at another position.
        ?- ghc life_test(9).

        ..................

        ?- ghc life_test(21).
```

If you want to try another pattern, use life_game/2.
See life_test/1.

(11) Evaluation data: Not recorded

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          LIFE GAME  Program     1987.2.24. by E.Sugino                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

life_game(X,List,Max) :- true | make_mesh(X,List,Max,Result),
        wr(1,X,Result).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%        Writer            %
%%%%%%%%%%%%%%%%%%%%%%%%%%
wr(N,M,[A|B]) :- N =< M,prolog((tab(5),write(A))),N1 := N + 1 | wr(N1,M,B).
wr(N,M,[A|B]) :- N > M,prolog((nl,tab(5),write(A))) | wr(2,M,B).
wr(_,_,[]) :- prolog(nl) | true.

%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Mesh maker         %
%%%%%%%%%%%%%%%%%%%%%%%%%%
make_mesh(Max,[Status|Next],Maxcycle,Result) :- true |
        make_node((_,_,E,SE,S,SW,W,_),Status,Maxcycle,Contr1-C1),
        con(E,W),
        make_mesh(2,Max,(SE,S,SW),Next,C1-C2,Maxcycle),
        temp_node(C2,Contr),
        mg(Contr,[result([end|Result],Result)],Contr1).

make_mesh(K,Max,(NW,N,NE),[Status|Status_list],C1-C3,Maxcycle) :-
        K =< Max, K1 := K + 1 |
        con(E,W),con(N,N1),con(NW,NW1),con(NE,NE1),
        make_node((N1,NE1,E,SE,S,SW,W,NW1),Status,Maxcycle,C1-C2),
        make_mesh(K1,Max,(SE,S,SW),Status_list,C2-C3,Maxcycle).
make_mesh(K,Max,(NW,N,NE),Status,C1-C2,Maxcycle) :- K > Max |
        C2=[end_line(Status,(NW,N,NE))|C1].

%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Node maker         %
%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% node(Nr,NEr,Er,SEr,Sr,SWr,Wr,NWr,Contr,Status,Cycle,Maxcycle,Work)
%
%        Nr,...,NWr : Receive_stream - Send_streame
%        Contr      : Controle stream   RECEIVE - SEND
%        Status     : the status of the node
%        Cycle      : Life cycle of the node
%        Maxcycle   : Max life cycle
%        Work       : D-List for collecting datas

make_node((N,NE,E,SE,S,SW,W,NW),Status,Maxcycle,Contr) :- true |
        N  = Nr - Ns, NE = NEr - NEs, E = Er - Es,
        SE = SEr - SEs, S = Sr - Ss, SW = SWr - SWs,
        W = Wr - Ws, NW = NWr - NWs,
        node(Nr-Ns,NEr-NEs,Er-Es,SEr-SEs,Sr-Ss,SWr-SWs,Wr-Ws,NWr-NWs,Contr,
                Status,(wait),Maxcycle,Work-Work).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Merge              %
%%%%%%%%%%%%%%%%%%%%%%%%%%
mg([],X,Y) :- true | X = Y.
mg(X,[],Y) :- true | X = Y.
mg([X|Y],Z,L) :- true | L = [X|LL],mg(Y,Z,LL).
mg(Z,[X|Y],L) :- true | L = [X|LL],mg(Z,Y,LL).


%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%       Temp Node       %
%%%%%%%%%%%%%%%%%%%%%%%%%%

temp_node([end_line(Status,(NW,N,NE))|Cn],Contr) :- true |
        Contr = [schizo(Status,(NW,N,NE)) | C1],
        temp_node(Cn,C1).
temp_node([schizo(Status,XNE,(YNW,YN,YC1-YCn)) | XCn],Contr) :- Status \= [] |
        Contr = [set_ne(XNE)],
        YC1 = [schizo1(Status,(YNW,YN)) | Cz],
        mg(C1,XCn,Cz),
        temp_node(YCn,C1).
temp_node([schizo([],XNE,(YNW,YN,YC1-YCn)) | XCn],Contr) :- true |
        Contr = [set_ne(XNE)],
        YC1 = [end(YNW,YN) | XCn],
        end_node(YCn,[]).
temp_node([result(A,B) |C],Contr) :- true |
        Contr = [result(A,B)|CC],
        temp_node(C,CC).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%       End Node        %
%%%%%%%%%%%%%%%%%%%%%%%%%%

end_node([],S) :- true | S = [].
end_node([end|C],S) :- true | end_node(C,S).
end_node([result([R1|R2],Rt)|C],T) :- R1 \= end |
        T = Rt,
        end_node(C,[R1|R2]).
end_node([result([end|R],Rt)|C],T) :- true |
        T = Rt.

%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Connecter       %       It connects one hands to another.
%%%%%%%%%%%%%%%%%%%%%%%%%%
con(X,Y) :- true | X = A-B,Y=B-A.

%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Node            %
%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%
%  schizo/2     .... for the first node of the first array

node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,[schizo(Status,(NW,N,NE))|Contr]-Next,
                Self,(wait),Maxcycle,Work) :- true |
        con(NW,XNW),con(N,XN),
        node(XN,_,XE1,XSE1,XS,XSW,XW,XNW,Contr-Next1,Self,(wait),Maxcycle,Work),
        node_work1(NE,XE,XSE,XE1,XSE1,Status,Maxcycle,Next-Next1).

node_work1(NE,XE,XSE,XE1,XSE1,[Stat|Status],Maxcycle,Next-Next1) :- true |
        con(XE1,YW),con(NE,XNE),
        make_node((_,XNE,XE,XSE,YS,YSW,YW,_),Stat,Maxcycle,YC1-YC2),
        Next = [schizo(Status,YSW,(XSE1,YS,YC1-YC2)) | Next1].

%%%%%%%%%%%%%%%%
%   schizo/3    ..... for the general nodes

node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,
                [schizo(Status,NEx,(NWy,Ny,YC1-YC2)) | Contr]-Next,
                Self,(wait),Max,Work) :- true |
        con(XNE1,NEx),
        node(XN,XNE1,XE1,XSE1,XS,XSW,XW,XNW,Contr-Next1,Self,0,Max,Work),
        node_work2(XNE,XE,XSE,XE1,XSE1,Ny,NWy,YC1-YC2,Status,Max,Next-Next1).

node_work2(XNE,XE,XSE,XE1,XSE1,Ny,NWy,YC1-YC2,
                                        [Stat|Status],Max,Next-Next1) :- true |
```

```
        con(Ny,YN),con(XE1,YW),con(NWy,YNW),
        make_node({YN,XNE,XE,XSE,YS,YSW,YW,YNW),Stat,Max,YC2-YC3),
        Next = [schizo(Status,YSW,(XSE1,YS,YC1-YC3)) | Next1].

%%%%%%%%%%%%%%%%%
%   set_ne        .... set the North-West hand (for the first line)

node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,[set_ne(NEx)|Contr]-Next,
                Self,(wait),Maxcycle,Work) :- true |
        con(NEx,XNE),
        node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Contr-Next,Self,0,Maxcycle,Work).

%%%%%%%%%%%%%%%%%
%   schizo1/2     ... for the arrays except the first on

node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,[schizo1(Status,(NW,N))|Contr]-Next,
                Self,(wait),Maxcycle,Work) :- true |
        con(N,XN),con(NW,XNW),
        node(XN,_,XE1,XSE1,XS,XSW,XW,XNW,Contr-Next1,Self,(wait),Maxcycle,Work),
        node_work3(XNE,XE,XSE,XE1,XSE1,Status,Maxcycle,Next-Next1).

node_work3(XNE,XE,XSE,XE1,XSE1,[Stat|Status],Maxcycle,Next-Next1) :- true |
        con(XE1,YW),
        make_node({_,XNE,XE,XSE,YS,YSW,YW,_),Stat,Maxcycle,YC1-YC2),
        Next = [schizo(Status,YSW,(XSE1,YS,YC1-YC2)) | Next1].

%%%%%%%%%%%%%%%%%
%   end/2         for the first node of the last array

node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,[end(NW,N)|Contr]-Next,
                Self,(wait),Maxcycle,Work) :- true |
        con(N,XN),con(NW,XNW),
        node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Contr-Next1,Self,0,Maxcycle,Work),
        Next = [end|Next1].

%%%%%%%%%%%%%%%%%
%   end           ... for the last array

node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,[end|Contr]-Next,
                Self,(wait),Maxcycle,Work) :- true |
        node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Contr-Next1,Self,0,Maxcycle,Work),
        Next = [end|Next1].

%%%%%%%%%%%%%%%%%%
%   going next life cycle

node([N|XN]-Ns,[NE|XNE]-NEs,[E|XE]-Es,[SE|XSE]-SEs,
                [S|XS]-Ss,[SW|XSW]-SWs,[W|XW]-Ws,[NW|XNW]-NWs,
        Contr,Self,Life,Max,Work) :- Life < Max, NewLife := Life + 1 |
        my_life([N,NE,E,SE,S,SW,W,NW],Self,NewSelf,0),
        node_work4(XN-Ns,XNE-NEs,XE-Es,XSE-SEs,XS-Ss,XSW-SWs,XW-Ws,XNW-NWs,
                        Contr,NewSelf,NewLife,Max,Work).

node_work4(XN-Ns,XNE-NEs,XE-Es,XSE-SEs,XS-Ss,XSW-SWs,XW-Ws,XNW-NWs,
                Contr,NewSelf,NewLife,Max,Work) :- true |
        send([Ns-XNs,NEs-XNEs,Es-XEs,SEs-XSEs,
                Ss-XSs,SWs-XSWs,Ws-XWs,NWs-XNWs],NewSelf),
        node(XN-XNs,XNE-XNEs,XE-XEs,XSE-XSEs,XS-XSs,XSW-XSWs,XW-XWs,XNW-XNWs,
                        Contr,NewSelf,NewLife,Max,Work).

%%%%%%%%%%%%%%%%%%%%%%
%   dead

node(_,_,_,_,_,_,_,_,Contr-Next,Self,Life,Max,Work-Wt) :- Life >= Max |
        Wt = [Self|St],
        Next = [result(Work,St)|Contr].
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%   collect datas

node(A,B,C,D,E,F,G,H,[result(S,St)|Contr]-Next,Self,(wait),Max,Work-Wt) :- true |
        St = Work,
        node(A,B,C,D,E,F,G,H,Contr-Next,Self,(wait),Max,S-Wt).
node(A,B,C,D,E,F,G,H,[result(S,St)|Contr]-Next,Self,Life,Max,Work-Wt) :-
        Life < Max |
        St = Work,
        node(A,B,C,D,E,F,G,H,Contr-Next,Self,Life,Max,S-Wt).

%%%%%%%%%%%%%%%%%%%%%%%%%
%    for the first time  ... when the node's life cycle is 0

node(N-Ns,NE-NEs,E-Es,SE-SEs,S-Ss,SW-SWs,W-Ws,NW-NWs,
                                Cont-Next,Self,0,Max,Work) :- true |
        send([Ns-Nsn,NEs-NEsn,Es-Esn,SEs-SEsn,
                Ss-Ssn,SWs-SWsn,Ws-Wsn,NWs-NWsn],Self),
        node(N-Nsn,NE-NEsn,E-Esn,SE-SEsn,S-Ssn,SW-SWsn,W-Wsn,NW-NWsn,
                Cont-Next,Self,1,Max,Work).

%%%%%%%%%%%%%%%%%%%%%%%%%
%       Send            %       send my status to the neighbours
%%%%%%%%%%%%%%%%%%%%%%%%%
send([],_) :- true | true.
send([A-B|C],S) :- true | A = [S|B],send(C,S).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Rule of Life Game       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       mys_life(Neighbours_list,
%                Me,
%                NewStatus,
%                Number_of_Black_neighbours)

my_life([],1,Return,2) :- true | Return = 1.
my_life([],1,Return,3) :- true | Return = 1.
my_life([],0,Return,3) :- true | Return = 1.
my_life([],1,Return,N) :- N < 2 | Return = 0.
my_life([],1,Return,N) :- N > 3 | Return = 0.
my_life([],0,Return,N) :- N < 3 | Return = 0.
my_life([],0,Return,N) :- N > 3 | Return = 0.
my_life([A|B],Self,Return,Sum) :- Sum1 := Sum + A |
        my_life(B,Self,Return,Sum1).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                Test Program for This game                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       The patter    *     moves by 4 cyles.  (as if it is a Glider !)
%                      *
%                     ***
%

life_test(N) :- true |
        life_game(8,   [0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,
                        0,0,0,0,1,0,0,0,
                        0,0,0,0,0,1,0,0,
                        0,0,0,1,1,1,0,0,
                        0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0],
```

N).

life2.doc

(0) Date: 1987-Jul-18, written by E. Sugino
    Modified: 25-Dec-87 by S. Takagi

(1) Program name: life2

(2) Author: E. Sugino, ICOT 4th lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

The life game is a popular game.

(5) Algorithm: See life.doc

(6) Process structure: See life.doc

(7) Pragma: None

(8) Program:

    It is almost the same as the program in life.ghc.
    The following points differ.

    (a) It displays the status of the mesh with Prolog functions.
        Each node has its own coordinates on the mesh so that
        it can be displayed on the terminal.
    (b) The parts of node which are used to make the mesh,
        are renamed node1.
    (c) When you use life cycle 0, the original pattern is displayed.
        (You must use life cycle 1 in LIFE.)

(9) Source file: life2.ghc
    This document: life2.doc

(10) Example:

    Compile the program, and try it.

    ?- ghc life_test(8).

    8 is the number of the life cycle.
    You can substitute any positive integer for 8.

(11) Evaluation data: Not recorded

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       LIFE GAME  Program      1987.4.16. by E.Sugino                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

/*
module life.

 :- public life_game/3 ,
           life_test/1 ,life_test1/1 .
*/


life_game(X,List,Max) :-  true |
     headder(R,Max),
     make_mesh(R,(X,List,Max,Result)).

headder(R,Max) :- true | erase(R1),headder1(R1,R,Max).
headder1(end,R,Max) :- true | home(R2),headder2(R2,R,Max).
headder2(end,R,Max) :-
     prolog(
          (
          nl,
          write('Life Game Now Starts ! '),              %
          write(Max),                                    % Comment
          (Max > 1,write(' cycles') ; write(' cycle'))   %   on PSI
          ,nl)) |                                        %
/*
     write('Life Game Now Starts ! ') |
*/
     R = end.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Tool                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
home(R) :- prolog(write('^[[H')) | R = end.             % com. on PSI
/*
home(R) :- esp_call(move,home,ok) | R = end.
*/

erase(R) :- prolog(write('^[[2J')) | R = end.           % com. on PSI
/*
erase(R) :- esp_call(erase,a,ok) | R = end.
*/

show(X,Y,S,S,End) :- true | End = end.
show(X,Y,S,S1,End) :- S =\= S1 | show(X,Y,S,End).

show(X,Y,1,E) :- wait(X),wait(Y),                                    %
     prolog((nl,                                                     %
          write('^[['),write(Y),write(';'),write(X),write('H'),      %
          put("*"),ttyflush)) |                                      % Comment
     E = end.                                                        %   on PSI
show(X,Y,0,E) :- wait(X),wait(Y),                                    %
     prolog((nl,                                                     %
          write('^[['),write(Y),write(';'),write(X),write('H'),      %
          put("-"),ttyflush)) |                                      %
     E = end.                                                        %
/*
show(X,Y,1,E) :- wait(X),wait(Y), esp_call(move,(X,Y),ok),
                 esp_call(write_term,('*'),ok) |
     E = end.
show(X,Y,0,E) :- wait(X),wait(Y), esp_call(move,(X,Y),ok),
```

```
                    esp_call(write_term,('-'),ok) |
      E = end.
*/

start_node_ad(X) :- true |
            X = (5,5).
%             X = (2,2).
node_ad(K,YY) :-
            Y := 5 + (K - 1) * 3 | YY =(5,Y).
%             Y := 2 + (K - 1) * 2 | YY =(2,Y).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%     Mesh maker     %
%%%%%%%%%%%%%%%%%%%%%%%%%%

make_mesh(end,(X,List,Max,Result)) :- true |
      make_mesh(X,List,Max,Result).
make_mesh(Max,[Status|Next],Maxcycle,Result) :- true |
      start_node_ad(XY),
      make_node1(SE_S_SW,Status,Maxcycle,Contr1-C1,XY),
      make_mesh(2,Max,SE_S_SW,Next,C1-C2,Maxcycle),
      temp_node(C2,Contr),
      mg(Contr,[result([end|Result],Result)],Contr1).

make_mesh(K,Max,(NW,N,NE),[Status|Status_list],C1-C3,Maxcycle) :-
      K =< Max, K1 := K + 1 |
      node_ad(K,YY),
      make_node2((N,NE,NW),SE_S_SW,Status,Maxcycle,C1-C2,YY),
      make_mesh(K1,Max,SE_S_SW,Status_list,C2-C3,Maxcycle).
make_mesh(K,Max,(NW,N,NE),Status,C1-C2,Maxcycle) :- K > Max |
      C2=[end_line(Status,(NW,N,NE))|C1].

%%%%%%%%%%%%%%%%%%%%%%%%%%
%     Node maker     %
%%%%%%%%%%%%%%%%%%%%%%%%%%

make_node1(SE_S_SW,Status,Maxcycle,Contr,XY) :- true |
      SE_S_SW = (SEr - SEs, Sr - Ss, SWr - SWs),
      node1(Contr,_-_,_-_,Er-Es,SEr-SEs,Sr-Ss,SWr-SWs,Es-Er,_-_,
            Status,Maxcycle,Work-Work,XY).

make_node2((Ns-Nr,NEs-NEr,NWs-NWr),SE_S_SW,Status,Maxcycle,Contr,XY) :- true |
      SE_S_SW = (SEr - SEs, Sr - Ss, SWr - SWs),
      node1(Contr,Nr-Ns,NEr-NEs,Er-Es,SEr-SEs,Sr-Ss,SWr-SWs,Es-Er,NWr-NWs,
            Status,Maxcycle,Work-Work,XY).

make_node3((NEs-NEr,Er-Es,SEr-SEs,Ws-Wr),(S,SW),Status,Maxcycle,Contr,XY)
    :- true |
      S = Sr - Ss, SW = SWr - SWs,
      node1(Contr,_-_,NEr-NEs,Er-Es,SEr-SEs,Sr-Ss,SWr-SWs,Wr-Ws,_-_,
            Status,Maxcycle,Work-Work,XY).

make_node4((Ns-Nr,NEr-NEs,Er-Es,SEr-SEs,Ws-Wr,NWs-NWr),(S,SW),
                                        Status,Maxcycle,Contr,XY) :- true |
      S = Sr - Ss, SW = SWr - SWs,
      node1(Contr,Nr-Ns,NEr-NEs,Er-Es,SEr-SEs,Sr-Ss,SWr-SWs,Wr-Ws,NWr-NWs,
        Status,Maxcycle,Work-Work,XY).

make_node5((NEr-NEs,Er-Es,SEr-SEs,Ws-Wr),(S,SW),Status,Maxcycle,Contr,XY)
   :- true |
      S = Sr - Ss, SW = SWr - SWs,
      node1(Contr,Nr-Ns,NEr-NEs,Er-Es,SEr-SEs,Sr-Ss,SWr-SWs,Wr-Ws,NWr-NWs,
            Status,Maxcycle,Work-Work,XY).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%        Merge          %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%
mg([],X,Y) :- true | X = Y.
mg(X,[],Y) :- true | X = Y.
mg([X|Y],Z,L) :- true | L = [X|LL],mg(Y,Z,LL).
mg(Z,[X|Y],L) :- true | L = [X|LL],mg(Z,Y,LL).


%%%%%%%%%%%%%%%%%%%%%%%%%
%       Temp Node         %
%%%%%%%%%%%%%%%%%%%%%%%%%

temp_node([end_line(Status,(NW,N,NE))|Cn],Contr) :- true |
     Contr = [schizo(Status,(NW,N,NE)) | C1],
     temp_node(Cn,C1).
temp_node([schizo(Status,XNE,(YNW,YN,YC1-YCn)) | XCn],Contr) :- Status \= [] |
     Contr = [set_ne(XNE)],
     YC1 = [schizo1(Status,(YNW,YN)) | Cz],
     mg(C1,XCn,Cz),
     temp_node(YCn,C1).
temp_node([schizo([],XNE,(YNW,YN,YC1-YCn)) | XCn],Contr) :- true |
     Contr = [set_ne(XNE)],
     YC1 = [end(YNW,YN) | XCn],
     end_node(YCn,[]).
temp_node([result(A,B) |C],Contr) :- true |
     Contr = [result(A,B)|CC],
     temp_node(C,CC).

%%%%%%%%%%%%%%%%%%%%%%%%%
%      End Node           %
%%%%%%%%%%%%%%%%%%%%%%%%%

end_node([],S) :- true | S = [].
end_node([end|C],S) :- true | end_node(C,S).
end_node([result([R1|R2],Rt)|C],T) :- R1 \= end |
     T = Rt,
     end_node(C,[R1|R2]).
end_node([result([end|R],Rt)|C],T) :- true | home(End),end_node1(End,T,Rt).
end_node1(end,T,Rt) :- true | T=Rt.

%%%%%%%%%%%%%%%%%%%%%%%%%
%     Connecter           %      It connects one hands to another.
%%%%%%%%%%%%%%%%%%%%%%%%%
% con(X,Y) :- true | X = A-B,Y=B-A.
con(A-B,C-D) :- true | A=D,B=C.

%%%%%%%%%%%%%%%%%%%%%%%%%
%      Node               %
%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%
%  schizo/2      .... for the first node of the first array

node1([schizo(Status,(NW,N,NE))|Contr]-Next,XN,XNE,XE,XSE,XS,XSW,XW,XNW,
          Self,Maxcycle,Work,XY) :- true |
     con(NW,XNW),con(N,XN),
     node1(Contr-Next1,XN,_-_,C-D,A-B,XS,XSW,XW,XNW,Self,Maxcycle,Work,XY),
     node_work1(NE,XE,XSE,C-D,A-B,Status,Maxcycle,Next-Next1,XY).

%%%%%%%%%%%%%%%
%   schizo/3      ..... for the general nodes

node1([schizo(Status,NEx,(NWy,Ny,YC1-YC2)) | Contr]-Next,
        XN,XNE,XE,XSE,XS,XSW,XW,XNW,
          Self,Max,Work,XY) :- true |
     con(NEx,C-D),
     node(XN,C-D,E-F,A-B,XS,XSW,XW,XNW,Contr-Next1,Self,0,Max,Work,XY),
```

```
     node_work2(XNE,XE,XSE,E-F,A-B,Ny,NWy,YC1-YC2,Status,Max,Next-Next1,XY).

%%%%%%%%%%%%%%%%%
%   set_ne        .... set the North-West hand (for the first line)

node1([set_ne(NEx)|Contr]-Next,XN,XNE,XE,XSE,XS,XSW,XW,XNW,
          Self,Maxcycle,Work,XY) :- true |
     con(NEx,XNE),
     node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Contr-Next,Self,0,Maxcycle,Work,XY).

%%%%%%%%%%%%%%%%%
%   schizo1/2     ... for the arrays except the first one

node1([schizo1(Status,(NW,N))|Contr]-Next,XN,XNE,XE,XSE,XS,XSW,XW,XNW,
          Self,Maxcycle,Work,XY) :- true |
     con(N,XN),con(NW,XNW),
     node1(Contr-Next1,XN,_-_,C-D,A-B,XS,XSW,XW,XNW,Self,Maxcycle,Work,XY),
     node_work3(XNE,XE,XSE,C-D,A-B,Status,Maxcycle,Next-Next1,XY).

%%%%%%%%%%%%%%%%%
%   end/2         for the first node of the last array

node1([end(NW,N)|Contr]-Next,XN,XNE,XE,XSE,XS,XSW,XW,XNW,
          Self,Maxcycle,Work,XY) :- true |
     con(N,XN),con(NW,XNW),
     node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Contr-Next1,Self,0,Maxcycle,Work,XY),
     Next = [end|Next1].

%%%%%%%%%%%%%%%%%
%   end           ... for the last array

node1([end|Contr]-Next,XN,XNE,XE,XSE,XS,XSW,XW,XNW,
          Self,Maxcycle,Work,XY) :- true |
     node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Contr-Next1,Self,0,Maxcycle,Work,XY),
     Next = [end|Next1].

%%%%%%%%%%%%%%%%%%%%%%%
%   collect datas

node1([result(S,St)|Contr]-Next,A,B,C,D,E,F,G,H,Self,Max,Work-Wt,XY) :-
     true |
     St = Work,
     node1(Contr-Next,A,B,C,D,E,F,G,H,Self,Max,S-Wt,XY).

node_work1(NE,XE,XSE,XE1,XSE1,[Stat|Status],Maxcycle,Next-Next1,XY) :-
     true |
     neighb_node_ad(XY,NXY),
     make_node3((NE,XE,XSE,XE1),(YS,YSW),Stat,Maxcycle,YC1-YC2,NXY),
     Next = [schizo(Status,YSW,(XSE1,YS,YC1-YC2)) | Next1].

neighb_node_ad((X,Y),XY) :- X1 := X + 6 | XY = (X1,Y).


node_work2(NE,E,SE,XE,XSE,N,NW,YC1-YC2,
                          [Stat|Status],Max,Next-Next1,(X,Y)) :-
     X1 := X + 6 |
     make_node4((N,NE,E,SE,XE,NW),(YS,YSW),Stat,Max,YC2-YC3,(X1,Y)),
     Next = [schizo(Status,YSW,(XSE,YS,YC1-YC3)) | Next1].


node_work3(NE,E,SE,W,XSE1,[Stat|Status],Maxcycle,Next-Next1,(X,Y)) :-
     X1 := X + 6 |
     make_node5((NE,E,SE,W),(YS,YSW),Stat,Maxcycle,YC1-YC2,(X1,Y)),
     Next = [schizo(Status,YSW,(XSE1,YS,YC1-YC2)) | Next1].

%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%     for the first time  ... when the node's life cycle is 0

node(N,NE,E,SE,S,SW,W,NW,Cont,Self,0,Max,Work,XY) :-
      XY = (X,Y) |
      show(X,Y,Self,End),
      send_first(End,[N,NE,E,SE,S,SW,W,NW],
                     [Nr,NEr,Er,SEr,Sr,SWr,Wr,NWr],Send,Self),
      node(Nr,NEr,Er,SEr,Sr,SWr,Wr,NWr,Send,Cont,Self,1,Max,Work,XY).

%%%%%%%%%%%%%%%%%%
%    going next life cycle

node([N|XN],[NE|XNE],[E|XE],[SE|XSE],[S|XS],[SW|XSW],[W|XW],[NW|XNW],Send,
                                          Contr,Self,Life,Max,Work,XY) :-
      0 < Life,Life =< Max,
      NewLife := Life + 1 |
      my_life([N,NE,E,SE,S,SW,W,NW],Self,NewSelf,0),
      node_work4(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Send,
                                  Contr,NewSelf,NewLife,Max,Work,XY,Self).

%%%%%%%%%%%%%%%%%%%%%%
%    dead

node(_,_,_,_,_,_,_,_,Contr-Next,Self,Life,Max,Work-Wt,(X,Y)) :-
      Life > Max |
      Wt = [Self|St],
      Next = [result(Work,St)|Contr].

node(A,B,C,D,E,F,G,H,Sd,[result(S,St)|Contr]-Next,Self,Life,Max,Work-Wt,XY) :-
      0 < Life,Life =< Max |
      St = Work,
      node(A,B,C,D,E,F,G,H,Sd,Contr-Next,Self,Life,Max,S-Wt,XY).

node_work4(XN,XNE,XE,XSE,XS,XSW,XW,XNW,Send,
           Contr,NewSelf,NewLife,Max,Work,(X,Y),Self) :-
      true |
      show(X,Y,NewSelf,Self,End),
      send(End,Send,NewSend,NewSelf),
      node(XN,XNE,XE,XSE,XS,XSW,XW,XNW,NewSend,
                                  Contr,NewSelf,NewLife,Max,Work,(X,Y)).

%%%%%%%%%%%%%%%%%%%%%%%%
%     Send           %     send my status to the neighbours
%%%%%%%%%%%%%%%%%%%%%%%%

send(end,Send,NewSend,Self) :- true | Send = [Self|NewSend].

send_first(end,In,Out,Send,Self) :- true | send_all(In,Out,Send,Self).
send_all([],[],_,_) :- true | true.
send_all([A-B|C],[AA|CC],Send,Self) :- true |
        B = [Self|Send],AA=A,
        send_all(C,CC,Send,Self).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     Rule of Life Game     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     my_life(Neighbours_list,
%           Mc,
%           NewStatus,
%           Number_of_Black_neighbours)

my_life([],1,Return,2) :- true | Return = 1.
my_life([],1,Return,3) :- true | Return = 1.
my_life([],0,Return,3) :- true | Return = 1.
my_life([],1,Return,N) :- N < 2 | Return = 0.
```

```
my_life([],1,Return,N) :- N > 3 | Return = 0.
my_life([],0,Return,N) :- N < 3 | Return = 0.
my_life([],0,Return,N) :- N > 3 | Return = 0.
my_life([A|B],Self,Return,Sum) :- Sum1 := Sum + A |
     my_life(B,Self,Return,Sum1).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Test Program for This game           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%     The patter    *          moves by 4 cyles.  (as if it is a Glider !)
%                    *
%                  ***
%

 life_test(N) :- true |
      life_game(8,   [0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,0,0,
                0,0,0,0,1,0,0,0,
                0,0,0,0,0,1,0,0,
                0,0,0,1,1,1,0,0,
                0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,0,0],
           N).

 life_test1(N) :- true |
        life_game(6,   [0,0,0,0,0,0,
                        0,0,1,0,0,0,
                        0,1,1,0,0,0,
                        0,0,0,1,0,0,
                        0,0,0,0,0,0,
                        0,0,0,0,0,0],
                N).


/*
end.
*/
```

mxflw1.doc

(0) Date: 1987-July-30, written by Kazuaki Rokusawa
Modified: 25-Dec-87 by S. Takagi

(1) Program name: mxflw1

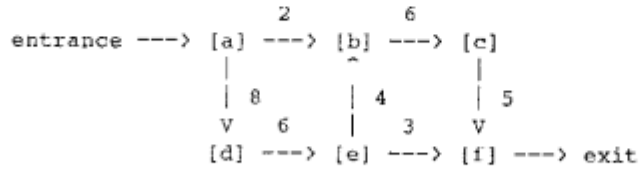
(2) Author: Kazuaki Rokusawa

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

Finding the maximum amount of flow through the network.
The network is composed of nodes and one-directional arcs.
Each node has up to two arcs each for output and input.
The flow through each arc is limited.

Example:



node = [a], [b], [c], [d], [e], [f]

Node [b] has 1 arc for output and 2 arcs for input.
Node [e] has 2 arcs for output and 1 arc for input.

Node [b] can send flow 6 to node [c].
Node [e] can send flow 4 to node [b] and 3 to node [f].

The maximum flow of this network is 8.

(5) Algorithm:

Each node works as follows.

On receiving input flow, the node tries to send the same amount of flow.

When the input flow is beyond the limit of the output flow, the node sends as much as possible and returns the remainder to the sender node.

When the input flow is less than the limit of the output flow, the node sends all the input. If that node has two output arcs, there is some preference to decide the amount for each output.

When the sum of the flow to the exit and the flow back to the entrance comes to the same amount as the original flow to the entrance, each node stops processing.

Sending flow 10 to node [a] on the example of (4), the following occur.

Node [a] sends flow 2 to [b] and 8 to [d].
On receiving input flow 2 from [a], node [b] sends 2 to [c].
This flow 2 reaches exit through nodes [c] and [f].

On receiving input flow 8, node [d] sends 6 to [e] and returns flow 2 to [a].
Returned flow 2 is returned to the entrance.

Node [e] receives 6 and sends 4 to [b] and 2 to [f]
(when the arc to [b] has a higher priority than the one to [f]).
Flow 2 reaches the exit through node [f].

Flow 4 reaches node [c] through node [b].
Since node [c] has already been sent 2,
it sends 3 to [f] and returns 1 to [b].
Flow 3 reaches the exit through node [f].

On receiving this backward input flow 1, node [b] returns
again to node [e].
Since node [e] has sent only 2 to node [f],
it receives backward input flow 1 and sends it to [f].
Total flow 8 reaches exit in all.

The sum of the flow which arrived at exit and returned to entrance
becomes equal to the original input.
Each node stops processing.

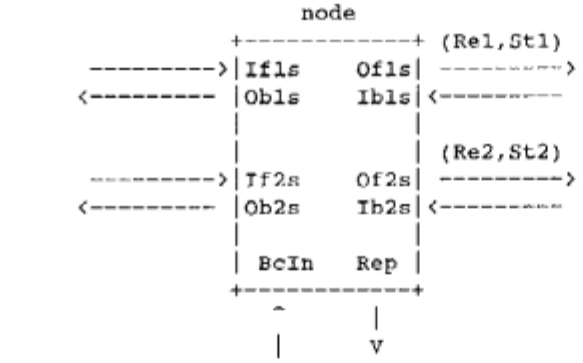
(6) Process structure:

The main processes are:

node: The node process which sends or receives a flow.
This process creates the same number of nodes.
The figure below shows this process.

exit: Detecting the end

outFlow:Display process



If1s and If2s, Of1s and Of2s, Ib1s and Ib2s, Ob1s and Ob2s are all
forward or backward input or output streams. "f" and "b" mean forward
and backward. "I" and "O" mean input and output.

Re1 and Re2 indicate the amount of flow that the node can send through
forward output streams Of1s and Of2s.

St1 and St2 have the value open or closed. open indicates that the
output stream still has space and the node can send more flow through
this stream. closed indicates that the output stream is already full
and the node cannot send through this stream any more.

The flow forms as follows.

```
 [amount(Amount-of-flow),link(1 or 2,Node-name),
                         link( , ),...,link(0,entrance)]
```

1 means that the node received the flow through If1s, while 2 means through If2s.

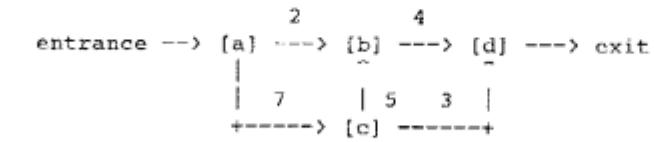
In example (4), flow 2, which reaches the exit through nodes [a], [b], [c], and [f], forms as follows.

```
 [amount(2),link(2,f),link(1,c),link(2,b),link(1,a)].
```

(7) Pragma: None

(8) Program:

The network is as follows.



The main predicates used in this program are:

ex1/1:	Top level predicate to start this program
entrance/2:	Sends the original flow to the entrance
exit/6:	Detects the end
node/14:	The main process which sends or receives a flow. This process works as: Waits for the input-flow Changes the status Sends the flow
outFlow/4:	Displays the result

(9) Source file: mxflw1.ghc
This document: mxflw1.doc

(10) Examples:

Compile the program.

```
        ?- ghccompile('mxflw1.ghc').<CR>
```

Execute the program.

```
        ?- ghc ex1(10).<CR>      % The first argument of ex1/1 is the amount
                                 % of the original flow to the entrance.
```

(11) Evaluation data:

The results of the execution on the GHC3 system are:

```
| ?- ghc ex1(10).
pass([amount(2),link(2,d),link(2,b),link(1,a),link(0,entrance)])
        % This output shows the flow to exit.
return([amount(1),link(0,entrance)])
        % This output shows the flow back to entrance.
pass([amount(2),link(1,d),link(1,c),link(1,a),link(0,entrance)])
pass([amount(2),link(2,d),link(1,b),link(1,c),link(1,a),link(0,entrance)])
pass([amount(1),link(1,d),link(1,c),link(1,a),link(0,entrance)])
return([amount(2),link(0,entrance)])
remain(node(a,(0,closed),(2,closed)))
        % This output is a report of the node in which Re1>0 or Re2>0
        % after execution.
remain(node(c,(3,closed),(0,closed)))

maxFlow(7)
        % This output shows the maximum flow.

61 reductions and 15 suspensions in 18 cycles and 456 msec (133 rps)
The maximum number of reducible goals is 7/9 at the 9th cycle.
The maximum length of the queue is 9.

yes
| ?- ghc ex1(7).
pass([amount(2),link(2,d),link(2,b),link(1,a),link(0,entrance)])
pass([amount(2),link(2,d),link(1,b),link(1,c),link(1,a),link(0,entrance)])
pass([amount(3),link(1,d),link(1,c),link(1,a),link(0,entrance)])
remain(node(a,(0,closed),(2,open)))
remain(node(c,(3,closed),(0,closed)))

maxFlow(7)

48 reductions and 17 suspensions in 17 cycles and 346 msec (138 rps)
The maximum number of reducible goals is 6/6 at the 11th cycle.
The maximum length of the queue is 8.
```

(12) References

L. Hellerstein and E. Shapiro,
Implementing Parallel Algorithms in Concurrent Prolog:
    The MAXFLOW Experience,
Journal of Logic Programming, volume 3, number 2, July 1986, pp.157-184

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%
%        MAXFLOW
%       ---------
%
% example. 1      ( 4 nodes )
%
%                          2           4           7
%     entrance --> [ a ] ---> [ b ] ---> [ d ] ---> exit
%                    |           A           A
%                    |   7       | 5     3   |
%                    +-------> [ c ] --------+
%
% ex1( LimFlow ) :- true |
%       entrance( LimFlow, Fsa ),
%       node( a, (2,open), (7,open),   Fsa, [],  Fab, Fac, Bab, Bac, Bsa, _,
%                                                BcIn, RepHead, RepNext1 ),
%       node( b, (4,open), (0,closed), Fcb, Fab, Fbd, _,   Bbd, [],  Bcb, Bab,
%                                                BcIn, RepNext1, RepNext2 ),
%       node( c, (5,open), (3,open),   Fac, [],  Fcb, Fcd, Bcb, Bcd, Bac, _,
%                                                BcIn, RepNext2, RepNext3 ),
%       node( d, (7,open), (0,closed), Fcd, Fbd, Fdg, _,    [], [],  Bcd, Bbd,
%                                                BcIn, RepNext3, [] ),
%       exit( LimFlow, 0, Fdg, Bsa, BcIn, Prs ),
%       outFlow( Prs, RepHead, 0, Os ),
%       outstream( Os ).
%
ex1( LimFlow ) :- true |
       entrance( LimFlow, Fsa ),
       nodes( Fsa, Fdg, Bsa, BcIn, RepHead ),
       exit( LimFlow, 0, Fdg, Bsa, BcIn, Prs ),
       outFlow( Prs, RepHead, 0, Os ),
       outstream( Os ).
nodes( Fsa, Fdg, Bsa, BcIn, RepHead ) :- true |
       node( a, (2,open), (7,open),   Fsa, [],  Fab, Fac, Bab, Bac, Bsa, _,
                                                BcIn, RepHead, RepNext1 ),
       node( b, (4,open), (0,closed), Fcb, Fab, Fbd, _,   Bbd, [],  Bcb, Bab,
                                                BcIn, RepNext1, RepNext2 ),
       node( c, (5,open), (3,open),   Fac, [],  Fcb, Fcd, Bcb, Bcd, Bac, _,
                                                BcIn, RepNext2, RepNext3 ),
       node( d, (7,open), (0,closed), Fcd, Fbd, Fdg, _,    [], [],  Bcd, Bbd,
                                                BcIn, RepNext3, [] ).
%
%        MAXFLOW          - main part -
%       ---------
%
%                  +-----------------------------+
%                  |                             |
%                  |                             V
%   entrance ---> node <--> ... <--> node <--> exit --> outFlow --> outstream
%                  A                  A         |          A
%                  |                  |         V          |
%                  V                  V      all nodes  all nodes
%                 node <--> ... <--> node
%                  A                  A
%                  |       ,          |
%                  :                  :
%
%                      stream diagram
%                     ----------------
%
%                         node
%                       +------------+ (Rel,Stl)
```

```
%              --------->|If1s    Of1s| --------->
%             <--------- |Ob1s    Ib1s|<---------
%                        |            |
%                        |            | (Re2,St2)
%              --------->|If2s    Of2s| --------->
%             <--------- |Ob2s    Ib2s|<---------
%                        |            |
%                        | BcIn   Rep |
%                        +------------+
%                           A      |
%                           |      V
%
%               Ifs  : input  forward  stream
%               Ofs  : output forward  stream
%               Ibs  : input  backward stream
%               Obs  : output backward stream
%               Re   : remainder
%               St   : status ( open / closed )
%               BcIn : broadcast in
%               Reph : reply head
%               Rept : reply tail
%
%                    node specification
%                   --------------------
%
%       entrance
%      ----------
entrance( LimFlow, Ofs ) :- true |
       Ofs = [[amount(LimFlow),link(0,entrance)]].
%
%       exit
%      ------
% continue       ( LimFlow > NextSum )
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
              Ifs = [If|RestIfs], If = [amount(Flow)|_],
              NextSum := Sum+Flow, LimFlow > NextSum |
       Prs = [pass(If)|NextPrs],
       exit( LimFlow, NextSum, RestIfs, Irs, BcOut, NextPrs ).
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
              Irs = [Ir|RestIrs], Ir = [amount(Flow)|_],
              NextSum := Sum+Flow, LimFlow > NextSum |
       Prs = [return(Ir)|NextPrs],
       exit( LimFlow, NextSum, Ifs, RestIrs, BcOut, NextPrs ).
%
% terminate       ( LimFlow = NextSum )
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
              Ifs = [If|RestIfs], If = [amount(Flow)|_],
              NextSum := Sum+Flow, LimFlow = NextSum |
       BcOut = end, Prs = [pass(If)].
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
              Irs = [Ir|RestIrs], Ir = [amount(Flow)|_],
              NextSum := Sum+Flow, LimFlow = NextSum |
       BcOut = end, Prs = [return(Ir)].
%
%       node
%      ------
% 1 --> 1
%       ( Flow < Rel )
node( Node, (Rel,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
              St1 = open, Flow < Rel |
       NextRel := Rel-Flow,
       Of1s = [ [amount(Flow),link(1,Node)|RestIf] | NextOf1s ],
       node( Node, (NextRel,St1), (Re2,St2),
                     RestIf1s, If2s, NextOf1s, Of2s,
```

```
                    Ib1s,     Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re1 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                              Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
              St1 = open, Flow = Re1 |
       Of1s = [ [amount(Flow),link(1,Node)|RestIf] ],
       node( Node, (0,closed), (Re2,St2),
                    RestIf1s, If2s, _,    Of2s,
                    Ib1s,     Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 1 ---> 1 & call again
%       ( Flow > Re1 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                              Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
              St1 = open, Flow > Re1 |
       RestFlow :=  Flow-Re1,
       Of1s = [ [amount(Re1),link(1,Node)|RestIf] | NextOf1s ],
       AgainIf1s = [ [amount(RestFlow)|RestIf] | RestIf1s ],
       node( Node, (0,closed), (Re2,St2),
                    AgainIf1s, If2s, NextOf1s, Of2s,
                    Ib1s,      Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
% 2 --> 1
%       ( Flow < Re1 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                              Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
              St1 = open, Flow < Re1 |
       NextRe1 := Re1-Flow,
       Of1s = [ [amount(Flow),link(2,Node)|RestIf] | NextOf1s ],
       node( Node, (NextRe1,St1), (Re2,St2),
                    If1s, RestIf2s, NextOf1s, Of2s,
                    Ib1s,     Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re1 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                              Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
              St1 = open, Flow = Re1 |
       Of1s = [ [amount(Flow),link(2,Node)|RestIf] ],
       node( Node, (0,closed), (Re2,St2),
                    If1s, RestIf2s, _,    Of2s,
                    Ib1s,     Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 2 ---> 1 & call again
%       ( Flow > Re1 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                              Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
              St1 = open, Flow > Re1 |
       RestFlow :=  Flow-Re1,
       Of1s = [ [amount(Re1),link(2,Node)|RestIf] | NextOf1s ],
       AgainIf2s = [ [amount(RestFlow)|RestIf] | RestIf2s ],
       node( Node, (0,closed), (Re2,St2),
                    If1s, AgainIf2s, NextOf1s, Of2s,
                    Ib1s,      Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
% 1 --> 2
%       ( Flow < Rc2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                              Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
              St1 = closed, St2 = open, Flow < Re2 |
       NextRe2 := Re2-Flow,
       Of2s = [ [amount(Flow),link(1,Node)|RestIf] | NextOf2s ],
       node( Node, (Re1,St1), (NextRe2,St2),
                          RestIf1s, If2s, Of1s, NextOf2s,
                    Ib1s,     Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re2 )
```

```
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
              St1 = closed, St2 = open, Flow = Re2 |
       Of2s = [ [amount(Flow),link(1,Node)|RestIf] ],
       node( Node, (Re1,St1), (0,closed),
                           RestIf1s, If2s, Of1s, _,
                     Ib1s, Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 1 --+--> 2
%     |
% 1<--+
%       ( Flow > Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
              St1 = closed, St2 = open, Flow > Re2 |
       RestFlow := Flow-Re2 ,
       Of2s = [ [amount(Re2),link(1,Node)|RestIf] ],
       Ob1s = [ [amount(RestFlow)|RestIf]|NextOb1s ],
       node( Node, (Re1,St1), (0,closed),
                     RestIf1s, If2s, Of1s,     _,
                     Ib1s,     Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 --> 2
%       ( Flow < Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
              St1 = closed, St2 = open, Flow < Re2 |
       NextRe2 := Re2-Flow,
       Of2s = [ [amount(Flow),link(2,Node)|RestIf] | NextOf2s ],
       node( Node, (Re1,St1), (NextRe2,St2),
                           If1s, RestIf2s, Of1s, NextOf2s,
                     Ib1s, Ib2s,     Ob1s, Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
              St1 = closed, St2 = open, Flow = Re2 |
       Of2s = [ [amount(Flow),link(2,Node)|RestIf] ],
       node( Node, (Re1,St1), (0,closed),
                     If1s, RestIf2s, Of1s, _,
                     Ib1s, Ib2s,     Ob1s, Ob2s, BcIn, Reph, Rept ).
% 2 --+--> 2
%     |
% 2<--+
%       ( Flow > Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
              St1 = closed, St2 = open, Flow > Re2 |
       RestFlow := Flow-Re2,
       Of2s = [ [amount(Re2),link(2,Node)|RestIf] ],
       Ob2s = [ [amount(RestFlow)|RestIf]|NextOb2s ],
       node( Node, (Re1,St1), (0,closed),
                     If1s, RestIf2s, Of1s, _,
                     Ib1s,     Ib2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
%  +-- 1
%  |
%  +-->2
%       ( Flow < Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib1s = [Ib|RestIb1s], Ib = [amount(Flow)|RestIf],
              St2 = open, Flow < Re2 |
       NextRe1 := Re1+Flow, NextRe2 := Re2-Flow, Of2s = [Ib|NextOf2s],
       node( Node, (NextRe1,closed), (NextRe2,St2),
```

```
                          If1s,     If2s, _,     NextOf2s,
                       RestIb1s, Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib1s = [Ib|RestIb1s], Ib = [amount(Flow)|RestIf],
              St2 = open, Flow = Re2 |
       NextRe1 := Re1+Flow, Of2s = [Ib],
       node( Node, (NextRe1,closed), (0,closed),
                          If1s,     If2s, _,     _,
                       RestIb1s, Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 1<--+-- 1
%     |
%     +-->2
%       ( Flow > Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib1s = [Ib|RestIb1s], Ib = [amount(Flow),link(1,Node)|RestIb],
              St2 = open, Flow > Re2 |
       NextRe1 := Re1+Flow, RestFlow := Flow-Re2,
       Of2s = [[amount(Re2),link(1,Node)|RestIb]],
       Ob1s = [[amount(RestFlow)|RestIb]|NextOb1s],
       node( Node, (NextRe1,closed), (0,closed),
                          If1s,     If2s, _,         _,
                       RestIb1s, Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
%     +-- 1
%     |
% 2<--+-->2
%       ( Flow > Re2 )
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib1s = [Ib|RestIb1s], Ib = [amount(Flow),link(2,Node)|RestIb],
              St2 = open, Flow > Re2 |
       NextRe1 := Re1+Flow, RestFlow := Flow-Re2,
       Of2s = [[amount(Re2),link(2,Node)|RestIb]],
       Ob2s = [[amount(RestFlow)|RestIb]|NextOb2s],
       node( Node, (NextRe1,closed), (0,closed),
                          If1s,     If2s, _,     _,
                       RestIb1s, Ib2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1 <-- 1
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib1s = [[amount(Flow),link(1,_)|RestIb]|RestIb1s],
              St2 = closed |
       NextRe1 := Re1+Flow, Ob1s = [[amount(Flow)|RestIb]|NextOb1s],
       node( Node, (NextRe1,closed), (Re2,St2),
                          If1s, If2s, _,  _,
                       RestIb1s, Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 <-- 1
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib1s = [[amount(Flow),link(2,_)|RestIb]|RestIb1s],
              St2 = closed |
       NextRe1 := Re1+Flow, Ob2s = [[amount(Flow)|RestIb]|NextOb2s],
       node( Node, (NextRe1,closed), (Re2,St2),
                          If1s, If2s, _,  _,
                       RestIb1s, Ib2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1 <-- 2
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib2s = [[amount(Flow),link(1,_)|RestIb]|RestIb2s] |
       NextRe2 := Re2+Flow, Ob1s = [[amount(Flow)|RestIb]|NextOb1s],
       node( Node, (Re1,St1), (NextRe2,closed),
                          If1s, If2s, _,  _,
                       Ib1s, RestIb2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 <-- 2
```

```
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              Ib2s = [[amount(Flow),link(2,_)|RestIb]|RestIb2s] |
       NextRe2 := Re2+Flow, Ob2s = [[amount(Flow)|RestIb]|NextOb2s],
       node( Node, (Re1,St1), (NextRe2,closed),
                              If1s, If2s, _,  _,
                     Ib1s, RestIb2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1 --+
%     |
% 1<--+
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If1s = [If|RestIf1s], St1 = closed, St2 = closed |
       Ob1s = [If|NextOb1s],
       node( Node, (Re1,St1), (Re2,St2),
                              RestIf1s, If2s, _,         _,
                     Ib1s,      Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 --+
%     |
% 2<--+
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              If2s = [If|RestIf2s], St1 = closed, St2 = closed |
       Ob2s = [If|NextOb2s],
       node( Node, (Re1,St1), (Re2,St2),
                              If1s, RestIf2s, _,     _,
                     Ib1s, Ib2s,      Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1,2 <-- end, terminate
node( Node, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
              BcIn = end |
       Reph = [node(Node,(Re1,St1),(Re2,St2))|Rept].
%
%       outFlow
%       ---------
outFlow( [Pr|RestPrs], Reps, SumFlow, Os ) :- Pr = pass([amount(Flow)|_]) |
       NextSumFlow := SumFlow + Flow, Os = [write(Pr),nl|NextOs],
       outFlow( RestPrs, Reps, NextSumFlow, NextOs ).
outFlow( [Pr|RestPrs], Reps, SumFlow, Os ) :- Pr = return(_) |
       Os = [write(Pr),nl|NextOs],
       outFlow( RestPrs, Reps, SumFlow, NextOs ).
outFlow( Prs, [Rep|RestReps], SumFlow, Os ) :- Rep = node(_,(0,_),(0,_)) |
       outFlow( Prs, RestReps, SumFlow, Os ).
outFlow( Prs, [Rep|RestReps], SumFlow, Os ) :- Rep = node(_,(Re1,_),(Re2,_)),
              Re1+Re2 > 0 |
       Os = [write(remain(Rep)),nl|NextOs],
       outFlow( Prs, RestReps, SumFlow, NextOs ).
outFlow( [], [], SumFlow, Os ) :- true |
       Os = [nl,write(maxFlow(SumFlow)),nl,nl,nl].
```

mxflw2.doc

(0) Date: 1987-July-30, written by Kazuaki Rokusawa
Modified: 25-dec-87 by S. Takagi

(1) Program name: mxflw2

(2) Author: Kazuaki Rokusawa

(3) Runs on: GHC on DEC 2065

(4) Description of the problem: See mxflw1.doc.

(5) Algorithm: See mxflw1.doc.

(6) Process structure: See mxflw1.doc.

(7) Pragma:

The pragma (PE-number) is attached to each node process. The user can attach it to some or all node processes. When attaching it to some node processes, the program attaches the pragma to other node processes.

(8) Program:

There are two networks. One (example 2) includes 17 nodes, and the other (example 3) includes 80 nodes (see the source file).

The main predicates used in this program are:

ex2/3, ex3/3: Top level predicates to start this program
ex2 -- example 2, ex3 -- example 3

maxflow/5: Main part of this program

genNodeList/3, findInitAllocPE/3, genNode/4, find2Links/6, findOtherLink/4, initNode/15, findLinkToExit/2:
Predicates that generate node process

entrance/3: Sends the initial flow to the entrance

node/17: The main process which sends and receives a flow.
This process works as:
Waits for the input-flow
Changes the status
Sends the flow

exit/6: Detects the end

outFlow/3, outByPE/1, totalByPE/8, outMsgByPE/4:
Displays the result

(9) Source file: mxflw2.ghc
This file: mxflw2.doc

(10) Examples:

Compile the program.

```
        ?- ghccompile('mxflw2.ghc').<CR>
```

Execute the program.

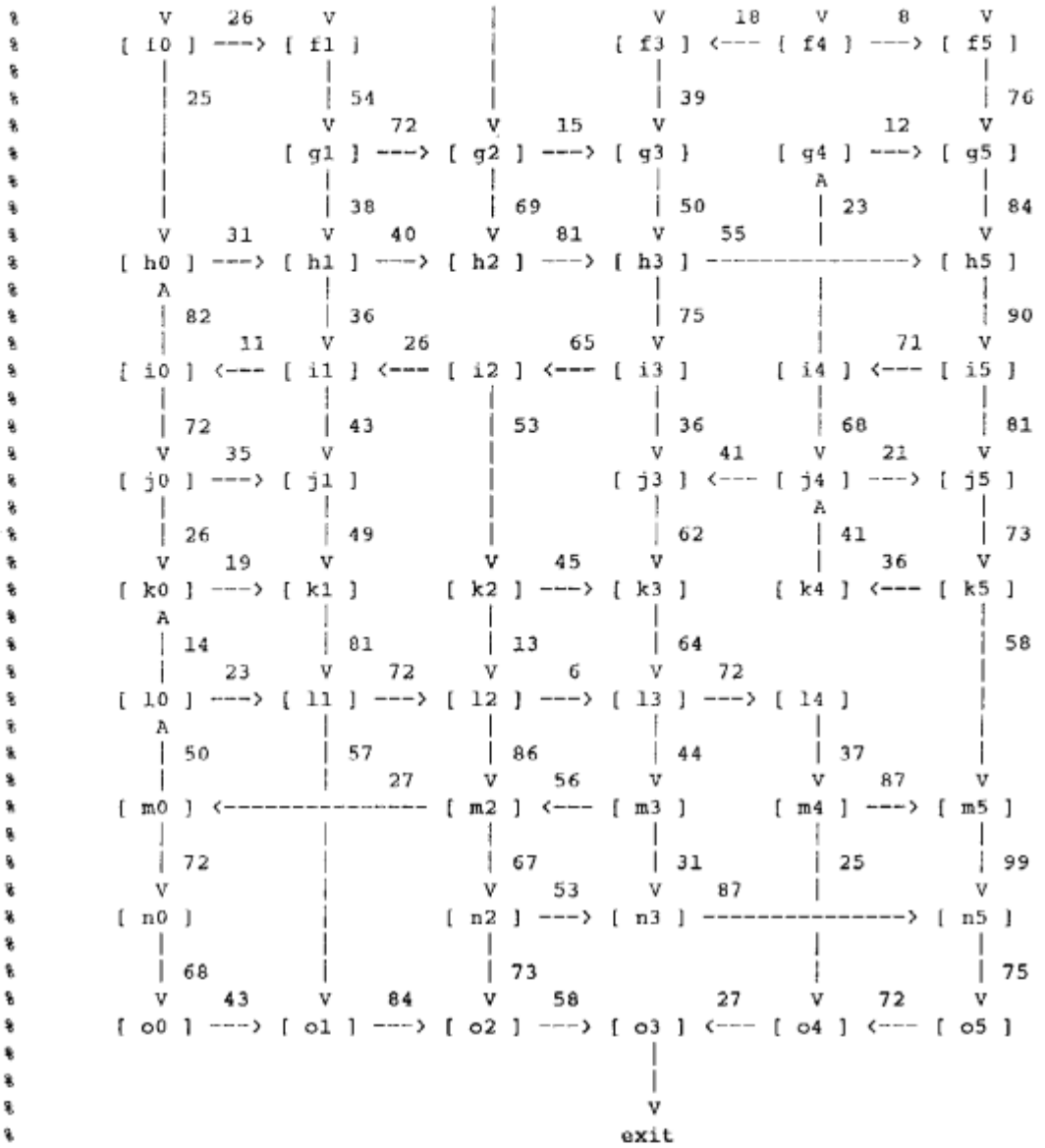
```
        ?- ghc ex2(LimFlow, InitAllocList, EntrancePE).<CR>

        % where LimFlow is the amount of initial flow to the entrance,
        % InitAllocList is a list of Pragma (PE-number),
        % EntrancePE is the Pragma(PE-number) attached to the entrance node.

        % Example of InitAllocList:
        %       InitAllocList = [node(k,3),node(h,1),node(q,2)]
        % This means attaching PE#3 to node k, PE#1 to node h,
        %  and PE#2 to node q.
```

(11) Evaluation data:

The results of the execution on the GHC3 system are as follows.

```
| ?- ghc ex2(50, [node(f,2),node(g,3),node(p,1)], 1).
        :
maxFlow(40)
        :
639 reductions and 104 suspensions in 81 cycles and 9530 msec (67 rps)
The maximum number of reducible goals is 28/44 at the 20th cycle.
The maximum length of the queue is 44.

yes
| ?- ghc ex3(200, [node(a4,2),node(c0,3),node(d3,4),
|                  node(h1,5),node(i5,6),node(l2,1),node(l3,2)], 1).
        :
maxFlow(83)
        :
7984 reductions and 720 suspensions in 409 cycles and
 xwd(0,-78733) msec (119680 rps)
The maximum number of reducible goals is 81/82 at the 321th cycle.
The maximum length of the queue is 153.

xwd(0,-78733) means that the value is more than 18 bits and is
represented in the 2 half-words format.
```

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%
%        MAXFLOW
%        -------
%
% example. 2      ( 17 nodes )
%
%                          100                          15          20
%     entrance --> [ a ] ---> [ b ]       [ c ] ---> [ d ] ---> [ e ] ---> exit
%                    |                      A                      A
%                    | 50          | 120     | 20                   | 30
%                    V    20       |         |    40          5     |
%                  [ f ] ---------------> [ g ] ---> [ h ] ---> [ i ]
%                    |                      A          |           A
%                    | 10          |        | 25       | 20        | 20
%                    V    80       V   40   |          |           |
%                  [ j ] ---> [ k ] ---> [ l ]         |         [ m ]
%                               |            |          |           A
%                               | 10         | 10       |           | 25
%                               V            V    10    V    25     |
%                             [ n ]        [ o ] ---> [ p ] ---> [ q ]
%
ex2( LimFlow, InitAllocList, EntrancePE ) :- true |
        NodeSpecList = [node(a,out(100-b, 50-f)), node(b,out(120-k,   [])),
                        node(c,out( 15-d,   [])), node(d,out( 20-e,   [])),
                        node(e,out( 50-exit,[])), node(f,out( 20-g, 10-j)),
                        node(g,out( 20-c, 40-h)), node(h,out(  5-i, 20-p)),
                        node(i,out( 30-e,   [])), node(j,out( 80-k,   [])),
                        node(k,out( 40-l, 10-n)), node(l,out( 25-g, 10-o)),
                        node(m,out( 20-i,   [])), node(n,out(   [],   [])),
                        node(o,out( 10-p,   [])), node(p,out( 25-q,   [])),
                        node(q,out( 25-m,   []))],
        EntranceNode = a,
        maxflow( LimFlow, NodeSpecList, EntranceNode,
                                                InitAllocList, EntrancePE ).
%
%-------------------------------------------------------------------------------
%
% example. 3      ( 80 nodes )
%
%       entrance
%          |
%          |
%          V    31          40          62          75          89
%       [ a0 ] ---> [ a1 ] ---> [ a2 ] ---> [ a3 ] ---> [ a4 ] ---> [ a5 ]
%          |                      A                      A           |
%          | 76                   | 41                   | 30        | 61
%          V     80               |    52          15    |    15     V
%       [ b0 ] ----------------> [ b2 ] ---> [ b3 ] ---> [ b4 ] ---> [ b5 ]
%          |                      A          |           A           |
%          | 63                   | 14       | 61        | 20        | 72
%          V    70          40    |          |           |    17     V
%       [ c0 ] ---> [ c1 ] ---> [ c2 ]       |        [ c4 ] ---> [ c5 ]
%                      |            |         |           A           |
%                      | 37         | 28      |           | 25        | 68
%                49    V      3     V    46   V     25    |    32     V
%       [ d0 ] <--- [ d1 ] <--- [ d2 ] ---> [ d3 ] ---> [ d4 ] ---> [ d5 ]
%          |            |            A          |                       |
%          | 74         | 22         | 7       | 77                    | 53
%          V    45      V    40      |     61  V      9          23     V
%       [ e0 ] ---> [ e1 ] ---> [ e2 ] <--- [ e3 ] <--- [ e4 ] <--- [ e5 ]
%          |            |            |          |           |           |
%          | 36         | 13         | 99       | 48        | 35        | 61
```




```
ex3( LimFlow, InitAllocList, EntrancePE ) :- true |
        NodeSpecList = [node(a0,out(31-a1,76-b0)), node(a1,out(40-a2,    [])),
                        node(a2,out(62-a3,    [])), node(a3,out(75-a4,    [])),
                        node(a4,out(89-a5,    [])), node(a5,out(61-b5,    [])),
                        node(b0,out(80-b2,63-c0)),
                        node(b2,out(41-a2,52-b3)), node(b3,out(15-b4,61-d3)),
                        node(b4,out(30-a4,15-b5)), node(b5,out(72-c5,    [])),
                        node(c0,out(70-c1,    [])), node(c1,out(40-c2,37-d1)),
                        node(c2,out(14-b2,28-d2)),
                        node(c4,out(20-b4,17-c5)), node(c5,out(68-d5,    [])),
                        node(d0,out(74-e0,    [])), node(d1,out(49-d0,22-e1)),
                        node(d2,out( 3-d1,46-d3)), node(d3,out(25-d4,77-e3)),
                        node(d4,out(25-c4,32-d5)), node(d5,out(53-e5,    [])),
                        node(e0,out(45-e1,36-f0)), node(e1,out(40-e2,13-f1)),
                        node(e2,out( 7-d2,99-g2)), node(e3,out(61-e2,48-f3)),
                        node(e4,out( 9-e3,35-f4)), node(e5,out(23-e4,61-f5)),
                        node(f0,out(26-f1,25-h0)), node(f1,out(54-g1,    [])),
                                                   node(f3,out(39-g3,    [])),
                        node(f4,out(18-f3, 8-f5)), node(f5,out(76-g5,    [])),
                                                   node(g1,out(72-g2,38-h1)),
                        node(g2,out(15-g3,69-h2)), node(g3,out(50-h3,    [])),
                        node(g4,out(12-g5,    [])), node(g5,out(84-h5,    [])),
                        node(h0,out(31-h1,    [])), node(h1,out(40-h2,36-i1)),
```

```
                        node(h2,out(81-h3,   [])), node(h3,out(55-h5,75-i3)),
                                                   node(h5,out(90-i5,   [])),
                        node(i0,out(72-j0,82-h0)), node(i1,out(11-i0,43-j1)),
                        node(i2,out(26-i1,53-k2)), node(i3,out(65-i2,36-j3)),
                        node(i4,out(23-g4,68-j4)), node(i5,out(71-i4,81-j5)),
                        node(j0,out(35-j1,26-k0)), node(j1,out(49-k1,   [])),
                                                   node(j3,out(62-k3,   [])),
                        node(j4,out(41-j3,21-j5)), node(j5,out(73-k5,   [])),
                        node(k0,out(19-k1,   [])), node(k1,out(81-l1,   [])),
                        node(k2,out(45-k3,13-l2)), node(k3,out(64-l3,   [])),
                        node(k4,out(41-j4,   [])), node(k5,out(36-k4,58-m5)),
                        node(l0,out(14-k0,23-l1)), node(l1,out(72-l2,57-o1)),
                        node(l2,out( 6-l3,86-m2)), node(l3,out(72-l4,44-m3)),
                        node(l4,out(37-m4,   [])),
                        node(m0,out(50-l0,72-n0)),
                        node(m2,out(27-m0,67-n2)), node(m3,out(56-m2,31-n3)),
                        node(m4,out(87-m5,25-o4)), node(m5,out(99-n5,   [])),
                        node(n0,out(68-o0,   [])),
                        node(n2,out(53-n3,73-o2)), node(n3,out(87-n5,   [])),
                                                   node(n5,out(75-o5,   [])),
                        node(o0,out(43-o1,   [])), node(o1,out(84-o2,   [])),
                        node(o2,out(58-o3,   [])), node(o3,out(85-exit, [])),
                        node(o4,out(27-o3,   [])), node(o5,out(72-o5,   [])),
        EntranceNode = a0,
        maxflow( LimFlow, NodeSpecList, EntranceNode,
                                                InitAllocList, EntrancePE ).
%
%-------------------------------------------------------------------------------
%
%        MAXFLOW         - main part -
%       ---------
%
%       NodeSpecList = [node(name,out(amount1-out1,amount2-out2)),...]
%
%       < example>
%
%       NodeSpecList = [node(a,out(2-b,7-c)),node(b,out(4-d,[])),
%                       node(c,out(5-b,3-d)),node(d,out(7-exit,[]))]
%
%                         2          4          7
%                  [ a ] ---> [ b ] ---> [ d ] ---> exit
%                    |          A          A
%                    |   7      | 5    3   |
%                    +------> [ c ] -------+
%
maxflow( LimFlow, NodeSpecList, EntranceNode, InitAllocList, EntrancePE ) :-
                true |
        Ent = (node(_,out(_-EntranceNode,[])),
                      channel(ToEntrance,_,BackToExit,_),_),
        genNodeList( NodeSpecList, InitAllocList, NodeList ),
        genNode( NodeList, [Ent|NodeList], BcIn, RepHead ),
        entrance( LimFlow, EntrancePE, ToEntrance ),
        findLinkToExit( NodeList, ToExit ),
        exit( LimFlow, 0, ToExit, BackToExit, BcIn, Prs ),
        outFlow( Prs, RepHead, 0 ),
        outByPE( RepHead ).
%
%-------------------------------------------------------------------------------
%       process generation
%-------------------------------------------------------------------------------
%
%       genNodeList
%      -------------
%
%       NodeSpecList ----> NodeList
%
```

```
%       NodeList = [(node(...),channel(Of1s,Of2s,Ib1s,Ib2s),pe(PE#)),...]
%                ( where node(...) is the same as node(...) in NodeSpecList )
%
genNodeList( [OneNodeSpec|RestNodeSpecList], InitAllocList, NodeList ) :-
                OneNodeSpec = node(Node,_) |
        findInitAllocPE( Node, InitAllocList, InitAllocPE ),
        OneNode = (OneNodeSpec,channel(Of1s,Of2s,Ib1s,Ib2s),pe(InitAllocPE)),
        NodeList = [OneNode|NextNodeList],
        genNodeList( RestNodeSpecList, InitAllocList, NextNodeList ).
genNodeList( [], _, NodeList ) :- true |
        NodeList = [].
%
%       findInitAllocPE
%       ---------------
%
%       find the initially allocated PE according to InitAllocList
%
findInitAllocPE( Node, InitAllocList, InitAllocPE ) :-
                InitAllocList = [node(Node,PE)|_] |
        InitAllocPE = PE.
findInitAllocPE( Node, InitAllocList, InitAllocPE ) :-
                InitAllocList = [node(OtherNode,_)|RestInitAllocList],
                Node \= OtherNode |
        findInitAllocPE( Node, RestInitAllocList, InitAllocPE ).
findInitAllocPE( _, InitAllocList, InitAllocPE ) :- InitAllocList = [] |
        InitAllocPE = notAllocated.
%
%       genNode
%       -------
%
%       generation of "node" process
%
%       ANodeList = [entrance-spec|NodeList]    ( see maxflow predicate )
%
%                Of1s, Ib1s
%       [node] ------------>
%          |
%          |     Of2s, Ib2s
%          +---------------->
%
genNode( [OneNode|RestNodeList], ANodeList, BcIn, Reph ) :-
                OneNode = (node(Node,out(Amount1-_,Amount2-_)),
                           channel(Of1s,Of2s,Ib1s,Ib2s),pe(AllocPE)) |
        find2Links( Node, ANodeList, If1s, If2s, Ob1s, Ob2s ),
        initNode( AllocPE, Node, (Amount1,open), (Amount2,open),
              If1s,If2s, Of1s,Of2s, Ib1s,Ib2s, Ob1s,Ob2s, BcIn, Reph,Rept ),
        genNode( RestNodeList, ANodeList, BcIn, Rept ).
%
%                Of1s, Ib1s
%       [node] ------------>
%
genNode( [OneNode|RestNodeList], ANodeList, BcIn, Reph ) :-
                OneNode = (node(Node,out(Amount1-_,[])),
                           channel(Of1s,_,Ib1s,_),pe(AllocPE)) |
        find2Links( Node, ANodeList, If1s, If2s, Ob1s, Ob2s ),
        initNode( AllocPE, Node, (Amount1,open), (0,closed),
              If1s,If2s, Of1s,_, Ib1s,[], Ob1s,Ob2s, BcIn, Reph,Rept ),
        genNode( RestNodeList, ANodeList, BcIn, Rept ).
%
%       [node]
%          |
%          |     Of2s, Ib2s
%          +---------------->
%
genNode( [OneNode|RestNodeList], ANodeList, BcIn, Reph ) :-
                OneNode = (node(Node,out([],Amount2-_)),
```

```
                            channel(_,Of2s,_,Ib2s),pe(AllocPE)) |
        find2Links( Node, ANodeList, If1s, If2s, Ob1s, Ob2s ),
        initNode( AllocPE, Node, (0,closed), (Amount2,open),
              If1s,If2s, _,Of2s, [],Ib2s, Ob1s,Ob2s, BcIn, Reph,Rept ),
        genNode( RestNodeList, ANodeList, BcIn, Rept ).
%
%       [node]          ( having no output-link )
%
genNode( [OneNode|RestNodeList], ANodeList, BcIn, Reph ) :-
                OneNode = (node(Node,out([],[])),_,pe(AllocPE)) |
        find2Links( Node, ANodeList, If1s, If2s, Ob1s, Ob2s ),
        initNode( AllocPE, Node, (0,closed), (0,closed),
              If1s,If2s, _,_, [],[], Ob1s,Ob2s, BcIn, Reph,Rept ),
        genNode( RestNodeList, ANodeList, BcIn, Rept ).
%
genNode( [], _, _, Reph ) :- true |
        Reph = [].
%
%       findLink
%       ----------
%
%       find the link to node(Node)
%
%                       Of1s = If1s, Ib1s = Ob1s
%       [other-node] -------------------------> [Node]
%                                                 A
%                                                 |
%                       Of2s = If2s, Ib2s = Ob2s  |
%       [other-node] ------------------------------+
%
find2Links( Node, [OneNode|RestANodeList], If1s, If2s, Ob1s, Ob2s ) :-
                OneNode = (node(_,out(_-Node,_)),channel(Of1s,_,Ib1s,_),_) |
        If1s = Of1s, Ob1s = Ib1s,
        findOtherLink( Node, RestANodeList, If2s, Ob2s ).
find2Links( Node, [OneNode|RestANodeList], If1s, If2s, Ob1s, Ob2s ) :-
                OneNode = (node(_,out(_,_-Node)),channel(_,Of2s,_,Ib2s),_) |
        If1s = Of2s, Ob1s = Ib2s,
        findOtherLink( Node, RestANodeList, If2s, Ob2s ).
find2Links( Node, [OneNode|RestANodeList], If1s, If2s, Ob1s, Ob2s ) :-
                OneNode = (node(_,out(_-OutNode1,_-OutNode2)),_,_),
                OutNode1 \= Node, OutNode2 \= Node |
        find2Links( Node, RestANodeList, If1s, If2s, Ob1s, Ob2s ).
find2Links( Node, [OneNode|RestANodeList], If1s, If2s, Ob1s, Ob2s ) :-
                OneNode = (node(_,out(_-OutNode1,[])),_,_), OutNode1 \= Node |
        find2Links( Node, RestANodeList, If1s, If2s, Ob1s, Ob2s ).
find2Links( Node, [OneNode|RestANodeList], If1s, If2s, Ob1s, Ob2s ) :-
                OneNode = (node(_,out([],_-OutNode2)),_,_), OutNode2 \= Node |
        find2Links( Node, RestANodeList, If1s, If2s, Ob1s, Ob2s ).
find2Links( Node, [OneNode|RestANodeList], If1s, If2s, Ob1s, Ob2s ) :-
                OneNode = (node(_,out([],[])),_,_) |
        find2Links( Node, RestANodeList, If1s, If2s, Ob1s, Ob2s ).
find2Links( Node, [], If1s, If2s, Ob1s, Ob2s ) :- true |
        If1s = [], If2s = [], Ob1s = _, Ob2s = _.
%
findOtherLink( Node, [OneNode|RestANodeList], Ifs, Obs ) :-
                OneNode = (node(_,out(_-Node,_)),channel(Of1s,_,Ib1s,_),_) |
        Ifs = Of1s, Obs = Ib1s.
findOtherLink( Node, [OneNode|RestANodeList], Ifs, Obs ) :-
                OneNode = (node(_,out(_,_-Node)),channel(_,Of2s,_,Ib2s),_) |
        Ifs = Of2s, Obs = Ib2s.
findOtherLink( Node, [OneNode|RestANodeList], Ifs, Obs ) :-
                OneNode = (node(_,out(_-OutNode1,_-OutNode2)),_,_),
                OutNode1 \= Node, OutNode2 \= Node |
        findOtherLink( Node, RestANodeList, Ifs, Obs ).
findOtherLink( Node, [OneNode|RestANodeList], Ifs, Obs ) :-
                OneNode = (node(_,out(_-OutNode1,[])),_,_), OutNode1 \= Node |
```

```
        findOtherLink( Node, RestANodeList, Ifs, Obs ).
findOtherLink( Node, [OneNode|RestANodeList], Ifs, Obs ) :-
                OneNode = (node(_,out([],_-OutNode2)),_,_), OutNode2 \= Node |
        findOtherLink( Node, RestANodeList, Ifs, Obs ).
findOtherLink( Node, [OneNode|RestANodeList], Ifs, Obs ) :-
                OneNode = (node(_,out([],[])),_,_) |
        findOtherLink( Node, RestANodeList, Ifs, Obs ).
findOtherLink( Node, [], Ifs, Obs ) :- true |
        Ifs = [], Obs = _.
%
%       initNode
%      -----------
%
%       initial generation of "node" process
%
%       ---  initially allocated  ---
%
initNode( AllocPE, Node, (Re1,St1), (Re2,St2),
            If1s,If2s,Of1s,Of2s, Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                AllocPE \= notAllocated |
        InitIf1s = [alloc(AllocPE)|If1s],
        node( notAllocated, Node, 0, 0, (Re1,St1), (Re2,St2),
              InitIf1s,If2s,Of1s,Of2s,Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ).
%
%       ---  initially not allocated  ---
%
initNode( AllocPE, Node, (Re1,St1), (Re2,St2),
            If1s,If2s,Of1s,Of2s, Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                AllocPE = notAllocated |
        node( notAllocated, Node, 0, 0, (Re1,St1), (Re2,St2),
              If1s,If2s,Of1s,Of2s,Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ).
%
%       findLinkToExit
%      ----------------
%
%       find the link to "exit"
%
%                       Of1s or Of2s = ToExit
%       [other-node] ----------------------> exit
%
findLinkToExit( [OneNode|RestNodeList], ToExit ) :-
                OneNode = (node(_,out(_-OutNode1,_-OutNode2)),_,_),
                OutNode1 \= exit, OutNode2 \= exit |
        findLinkToExit( RestNodeList, ToExit ).
findLinkToExit( [OneNode|RestNodeList], ToExit ) :-
                OneNode = (node(_,out(_-OutNode1,[])),_,_), OutNode1 \= exit |
        findLinkToExit( RestNodeList, ToExit ).
findLinkToExit( [OneNode|RestNodeList], ToExit ) :-
                OneNode = (node(_,out([],_-OutNode2)),_,_), OutNode2 \= exit |
        findLinkToExit( RestNodeList, ToExit ).
findLinkToExit( [OneNode|RestNodeList], ToExit ) :-
                OneNode = (node(_,out([],[])),_,_) |
        findLinkToExit( RestNodeList, ToExit ).
findLinkToExit( [OneNode|RestNodeList], ToExit ) :-
                OneNode = (node(_,out(_-exit,_)),channel(Of1s,_,_,_),_) |
        ToExit = Of1s.
findLinkToExit( [OneNode|RestNodeList], ToExit ) :-
                OneNode = (node(_,out(_,_-exit)),channel(_,Of2s,_,_),_) |
        ToExit = Of2s.
findLinkToExit( [], _ ) :- Msg = 'illegal NodeSpecList',
                prolog( ( write( Msg ), nl ) ) |
        true.
%
%-------------------------------------------------------------------------------
%       process specification
%-------------------------------------------------------------------------------
```

```
%
%                    +-----------------------------+
%                    |                             |
%                    |                             V
%     entrance ---> node <--> ... <--> node <--> exit --> outFlow --> outstream
%                    A                   A         |          A
%                    |                   |         V          |
%                    V                   V      all nodes  all nodes
%                   node <--> ... <--> node
%                    A                   A
%                    |                   |
%                    :                   :
%
%                         stream diagram
%                        ----------------
%
%                              node
%                            +-------------+ (Re1,St1)
%                 --------->|If1s     Of1s| --------->
%               <---------  |Ob1s     Ib1s|<---------
%                           |             |
%                           |             | (Re2,St2)
%                 --------->|If2s     Of2s| --------->
%               <---------  |Ob2s     Ib2s|<---------
%                           |             |
%                           | BcIn    Rep |
%                           +-------------+
%                              A       |
%                              |       V
%
%                  Ifs  : input  forward  stream
%                  Ofs  : output forward  stream
%                  Ibs  : input  backward stream
%                  Obs  : output backward stream
%                  Rc   : remainder
%                  St   : status ( open / closed )
%                  BcIn : broadcast in
%                  Reph : reply head
%                  Rept : reply tail
%
%                       node specification
%                      --------------------
%
%        entrance
%       ----------
entrance( LimFlow, EntrancePE, Ofs ) :- Msg = push(LimFlow),
                prolog( ( write( Msg ), nl ) ) |
        Ofs = [alloc(EntrancePE),[amount(LimFlow),link(0,entrance)]].
%
%        node
%       ------
%
% allocation to PE specified in the message
%
%      receiving "alloc-message" from Ifls
%                                     ~
%      ---  first receiving  ---
%
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ifls = [If|RestIfls], If = alloc(AllocPE), PE = notAllocated,
                Msg = allocate(node(Node),pe(AllocPE)),
                prolog( ( write(Msg), nl ) ) |
        sendAlloc( AllocPE, St1, Of1s, NextOf1s ),
        sendAlloc( AllocPE, St2, Of2s, NextOf2s ),
        node( AllocPE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2),
```

```
                        RestIfls, If2s, NextOfls, NextOf2s,
                        Ibls,     Ib2s, Obls,     Ob2s, BcIn, Reph, Rept ).
%
%       --- already received ---
%
node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2), Ifls,If2s,Ofls,Of2s,
                                    Ibls,Ib2s,Obls,Ob2s, BcIn, Reph, Rept ) :-
                Ifls = [If|RestIfls], If = alloc(AllocPE), PE \= notAllocated |
        node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2),
                        RestIfls, If2s, Ofls, Of2s,
                        Ibls,     Ib2s, Obls, Ob2s, BcIn, Reph, Rept ).
%
%       receiving "alloc-message" from If2s
%
%       --- first receiving ---
%
node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2), Ifls,If2s,Ofls,Of2s,
                                    Ibls,Ib2s,Obls,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = alloc(AllocPE), PE = notAllocated,
                Msg = allocate(node(Node),pe(AllocPE)),
                prolog( ( write(Msg), nl ) ) |
        sendAlloc( AllocPE, Stl, Ofls, NextOfls ),
        sendAlloc( AllocPE, St2, Of2s, NextOf2s ),
        node( AllocPE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2),
                        Ifls, RestIf2s, NextOfls, NextOf2s,
                        Ibls,     Ib2s, Obls,     Ob2s, BcIn, Reph, Rept ).
%
%       --- already received ---
%
node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2), Ifls,If2s,Ofls,Of2s,
                                    Ibls,Ib2s,Obls,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = alloc(AllocPE), PE \= notAllocated |
        node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2),
                        Ifls, RestIf2s, Ofls, Of2s,
                        Ibls,     Ib2s, Obls, Ob2s, BcIn, Reph, Rept ).
%
sendAlloc( AllocPE, St, Ofs, NextOfs ) :- St = open |
        Ofs = [alloc(AllocPE)|NextOfs].
%
sendAlloc( AllocPE, St, Ofs, NextOfs ) :- St = closed |
        Ofs = NextOfs.
%
% general processing
%
%       receiving [amount(Flow),link(_,_),...] message
%
% 1 --> 1
%       ( Flow < Rel )
node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2), Ifls,If2s,Ofls,Of2s,
                                    Ibls,Ib2s,Obls,Ob2s, BcIn, Reph, Rept ) :-
                Ifls = [If|RestIfls], If = [amount(Flow)|RestIf],
                Stl = open, Flow < Rel, NextRel := Rel-Flow, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ofls = [ [amount(Flow),link(1,Node)|RestIf] | NextOfls ],
        node( PE, Node, NewIfc, Ibc, (NextRel,Stl), (Re2,St2),
                        RestIfls, If2s, NextOfls, Of2s,
                        Ibls,     Ib2s, Obls,     Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Rel )
node( PE, Node, Ifc, Ibc, (Rel,Stl), (Re2,St2), Ifls,If2s,Ofls,Of2s,
                                    Ibls,Ib2s,Obls,Ob2s, BcIn, Reph, Rept ) :-
                Ifls = [If|RestIfls], If = [amount(Flow)|RestIf],
                Stl = open, Flow = Rel, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ofls = [ [amount(Flow),link(1,Node)|RestIf] ],
```

```
        node( PE, Node, NewIfc, Ibc, (0,closed), (Re2,St2),
                        RestIf1s, If2s, _,     Of2s,
                        Ib1s,     Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 1 ---> 1 & call again
%       ( Flow > Rel )
node( PE, Node, Ifc, Ibc, (Rel,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
                St1 = open, Flow > Rel, RestFlow :=  Flow-Rel, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Rel)),
                prolog( ( write(Msg), nl ) ) |
        Of1s = [ [amount(Rel),link(1,Node)|RestIf] | NextOf1s ],
        AgainIf1s = [ [amount(RestFlow)|RestIf] | RestIf1s ],
        node( PE, Node, NewIfc, Ibc, (0,closed), (Re2,St2),
                        AgainIf1s, If2s, NextOf1s, Of2s,
                        Ib1s,      Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
% 2 --> 1
%       ( Flow < Rel )
node( PE, Node, Ifc, Ibc, (Rel,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
                St1 = open, Flow < Rel, NextRel := Rel-Flow, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of1s = [ [amount(Flow),link(2,Node)|RestIf] | NextOf1s ],
        node( PE, Node, NewIfc, Ibc, (NextRel,St1), (Re2,St2),
                        If1s, RestIf2s, NextOf1s, Of2s,
                        Ib1s,      Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Rel )
node( PE, Node, Ifc, Ibc, (Rel,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
                St1 = open, Flow = Rel, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of1s = [ [amount(Flow),link(2,Node)|RestIf] ],
        node( PE, Node, NewIfc, Ibc, (0,closed), (Re2,St2),
                        If1s, RestIf2s, _,     Of2s,
                        Ib1s,      Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 2 ---> 1 & call again
%       ( Flow > Rel )
node( PE, Node, Ifc, Ibc, (Rel,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
                St1 = open, Flow > Rel, RestFlow :=  Flow-Rel, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Rel)),
                prolog( ( write(Msg), nl ) ) |
        Of1s = [ [amount(Rel),link(2,Node)|RestIf] | NextOf1s ],
        AgainIf2s = [ [amount(RestFlow)|RestIf] | RestIf2s ],
        node( PE, Node, NewIfc, Ibc, (0,closed), (Re2,St2),
                        If1s, AgainIf2s, NextOf1s, Of2s,
                        Ib1s,      Ib2s, Ob1s,     Ob2s, BcIn, Reph, Rept ).
% 1 --> 2
%       ( Flow < Re2 )
node( PE, Node, Ifc, Ibc, (Rel,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
                St1 = closed, St2 = open, Flow < Re2, NextRe2 := Re2-Flow,
                NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [ [amount(Flow),link(1,Node)|RestIf] | NextOf2s ],
        node( PE, Node, NewIfc, Ibc, (Rel,St1), (NextRe2,St2),
                        RestIf1s, If2s, Of1s, NextOf2s,
                        Ib1s,      Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re2 )
```

```
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ifls = [If|RestIfls], If = [amount(Flow)|RestIf],
                St1 = closed, St2 = open, Flow = Re2, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [ [amount(Flow),link(1,Node)|RestIf] ],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (0,closed),
                        RestIf1s, If2s, Of1s, _,
                        Ib1s, Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 1 --+--> 2
%     |
% 1<--+
%       ( Flow > Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If1s = [If|RestIf1s], If = [amount(Flow)|RestIf],
                St1 = closed, St2 = open, Flow > Re2, RestFlow :=  Flow-Re2,
                NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),
                                        send_forward(Re2),send_back(RestFlow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [ [amount(Re2),link(1,Node)|RestIf] ],
        Ob1s = [ [amount(RestFlow)|RestIf]|NextOb1s ],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (0,closed),
                        RestIf1s, If2s, Of1s,     _,
                        Ib1s,     Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 --> 2
%       ( Flow < Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
                St1 = closed, St2 = open, Flow < Re2, NextRe2 := Re2-Flow,
                NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [ [amount(Flow),link(2,Node)|RestIf] | NextOf2s ],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (NextRe2,St2),
                        If1s, RestIf2s, Of1s, NextOf2s,
                        Ib1s, Ib2s,     Ob1s, Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
                St1 = closed, St2 = open, Flow = Re2, NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [ [amount(Flow),link(2,Node)|RestIf] ],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (0,closed),
                        If1s, RestIf2s, Of1s, _,
                        Ib1s, Ib2s,     Ob1s, Ob2s, BcIn, Reph, Rept ).
% 2 --+--> 2
%     |
% 2<--+
%       ( Flow > Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                  Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], If = [amount(Flow)|RestIf],
                St1 = closed, St2 = open, Flow > Re2, RestFlow :=  Flow-Re2,
                NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),
                                        send_forward(Re2),send_back(RestFlow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [ [amount(Re2),link(2,Node)|RestIf] ],
        Ob2s = [ [amount(RestFlow)|RestIf]|NextOb2s ],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (0,closed),
```

```
                        If1s, RestIf2s, Of1s, _,
                        Ib1s,     Ib2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
%  +-- 1
%  |
%  +-->2
%       ( Flow < Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                    Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib1s = [Ib|RestIb1s], Ib = [amount(Flow)|RestIf],
                St2 = open, Flow < Re2,
                NextRe1 := Re1+Flow, NextRe2 := Re2-Flow, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [Ib|NextOf2s],
        node( PE, Node, Ifc, NewIbc, (NextRe1,closed), (NextRe2,St2),
                        If1s,     If2s, _,     NextOf2s,
                        RestIb1s, Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
%       ( Flow = Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                    Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib1s = [Ib|RestIb1s], Ib = [amount(Flow)|RestIf],
                St2 = open, Flow = Re2, NextRe1 := Re1+Flow, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),send_forward(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [Ib],
        node( PE, Node, Ifc, NewIbc, (NextRe1,closed), (0,closed),
                        If1s,     If2s, _,     _,
                        RestIb1s, Ib2s, Ob1s, Ob2s, BcIn, Reph, Rept ).
% 1<--+-- 1
%     |
%     +-->2
%       ( Flow > Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                    Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib1s = [Ib|RestIb1s], Ib = [amount(Flow),link(1,Node)|RestIb],
                St2 = open, Flow > Re2,
                NextRe1 := Re1+Flow, RestFlow := Flow-Re2, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),
                                    send_forward(Re2),send_back(RestFlow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [[amount(Re2),link(1,Node)|RestIb]],
        Ob1s = [[amount(RestFlow)|RestIb]|NextOb1s],
        node( PE, Node, Ifc, NewIbc, (NextRe1,closed), (0,closed),
                        If1s,     If2s, _,         _,
                        RestIb1s, Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
%     +-- 1
%     |
% 2<--+-->2
%       ( Flow > Re2 )
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                    Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib1s = [Ib|RestIb1s], Ib = [amount(Flow),link(2,Node)|RestIb],
                St2 = open, Flow > Re2,
                NextRe1 := Re1+Flow, RestFlow := Flow-Re2, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),
                                    send_forward(Re2),send_back(RestFlow)),
                prolog( ( write(Msg), nl ) ) |
        Of2s = [[amount(Re2),link(2,Node)|RestIb]],
        Ob2s = [[amount(RestFlow)|RestIb]|NextOb2s],
        node( PE, Node, Ifc, NewIbc, (NextRe1,closed), (0,closed),
                        If1s,     If2s, _,     _,
                        RestIb1s, Ib2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1 <-- 1
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                    Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib1s = [[amount(Flow),link(1,_)|RestIb]|RestIb1s],
```

```
                St2 = closed, NextRe1 := Re1+Flow, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),send_back(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ob1s = [[amount(Flow)|RestIb]|NextOb1s],
        node( PE, Node, Ifc, NewIbc, (NextRe1,closed), (Re2,St2),
                        If1s, If2s, _,  _,
                        RestIb1s, Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 <-- 1
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib1s = [[amount(Flow),link(2,_)|RestIb]|RestIb1s],
                St2 = closed, NextRe1 := Re1+Flow, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),send_back(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ob2s = [[amount(Flow)|RestIb]|NextOb2s],
        node( PE, Node, Ifc, NewIbc, (NextRe1,closed), (Re2,St2),
                        If1s, If2s, _,  _,
                        RestIb1s, Ib2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1 <-- 2
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib2s = [[amount(Flow),link(1,_)|RestIb]|RestIb2s],
                NextRe2 := Re2+Flow, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),send_back(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ob1s = [[amount(Flow)|RestIb]|NextOb1s],
        node( PE, Node, Ifc, NewIbc, (Re1,St1), (NextRe2,closed),
                        If1s, If2s, _,  _,
                        Ib1s, RestIb2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 <-- 2
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                Ib2s = [[amount(Flow),link(2,_)|RestIb]|RestIb2s],
                NextRe2 := Re2+Flow, NewIbc := Ibc+1,
                Msg = ope(pe(PE),node(Node),send_back(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ob2s = [[amount(Flow)|RestIb]|NextOb2s],
        node( PE, Node, Ifc, NewIbc, (Re1,St1), (NextRe2,closed),
                        If1s, If2s, _,  _,
                        Ib1s, RestIb2s, Ob1s, NextOb2s, BcIn, Reph, Rept ).
% 1 --+
%     |
% 1<--+
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If1s = [If|RestIf1s], St1 = closed, St2 = closed,
                If = [amount(Flow)|_], NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_back(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ob1s = [If|NextOb1s],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (Re2,St2),
                        RestIf1s, If2s, _,        _,
                        Ib1s,     Ib2s, NextOb1s, Ob2s, BcIn, Reph, Rept ).
% 2 --+
%     |
% 2<--+
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                If2s = [If|RestIf2s], St1 = closed, St2 = closed,
                If = [amount(Flow)|_], NewIfc := Ifc+1,
                Msg = ope(pe(PE),node(Node),send_back(Flow)),
                prolog( ( write(Msg), nl ) ) |
        Ob2s = [If|NextOb2s],
        node( PE, Node, NewIfc, Ibc, (Re1,St1), (Re2,St2),
                        If1s, RestIf2s, _,     _,
                        Ib1s, Ib2s,     Ob1s, NextOb2s, BcIn, Reph, Rept ).
```

```
%
% termination of "node" process
%
%     detecting   "BcIn = end"
%
node( PE, Node, Ifc, Ibc, (Re1,St1), (Re2,St2), If1s,If2s,Of1s,Of2s,
                                    Ib1s,Ib2s,Ob1s,Ob2s, BcIn, Reph, Rept ) :-
                BcIn = end,
                Msg = terminate(pe(PE),node(Node)),
                prolog( ( write(Msg), nl ) ) |
        Ope = ope(byFin(Ifc),byBin(Ibc)),
        Status = status((Re1,St1),(Re2,St2)),
        Reph = [result(pe(PE),node(Node),Ope,Status)|Rept].
%
%       exit
%      ------
% continue      ( LimFlow > NextSum )
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
                Ifs = [If|RestIfs], If = [amount(Flow)|_],
                NextSum := Sum+Flow, LimFlow > NextSum |
        Prs = [pass(If)|NextPrs],
        exit( LimFlow, NextSum, RestIfs, Irs, BcOut, NextPrs ).
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
                Irs = [Ir|RestIrs], Ir = [amount(Flow)|_],
                NextSum := Sum+Flow, LimFlow > NextSum |
        Prs = [return(Ir)|NextPrs],
        exit( LimFlow, NextSum, Ifs, RestIrs, BcOut, NextPrs ).
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
                Ifs = [If|RestIfs], If = alloc(_) |
        exit( LimFlow, Sum, RestIfs, Irs, BcOut, Prs ).
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
                Irs = [Ir|RestIfs], Ir = alloc(_) |
        exit( LimFlow, Sum, Ifs, RestIrs, BcOut, Prs ).
%
% terminate     ( LimFlow = NextSum )
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
                Ifs = [If|RestIfs], If = [amount(Flow)|_],
                NextSum := Sum+Flow, LimFlow = NextSum |
        BcOut = end, Prs = [pass(If)].
exit( LimFlow, Sum, Ifs, Irs, BcOut, Prs ) :-
                Irs = [Ir|RestIrs], Ir = [amount(Flow)|_],
                NextSum := Sum+Flow, LimFlow = NextSum |
        BcOut = end, Prs = [return(Ir)].
%
%       outFlow  --  output flow (pass,return) and result on each node
%      ---------
outFlow( [Pr|RestPrs], Reps, SumFlow ) :-
                Pr = pass([amount(Flow)|_]), NextSumFlow := SumFlow + Flow,
                prolog( ( write( Pr ), nl ) ) |
        outFlow( RestPrs, Reps, NextSumFlow ).
outFlow( [Pr|RestPrs], Reps, SumFlow ) :-
                Pr = return(_),
                prolog( ( write( Pr ), nl ) ) |
        outFlow( RestPrs, Reps, SumFlow ).
outFlow( Prs, [Rep|RestReps], SumFlow ) :-
                prolog( ( write( Rep ), nl ) ) |
        outFlow( Prs, RestReps, SumFlow ).
outFlow( [], [], SumFlow ) :- Msg = maxFlow(SumFlow),
                prolog( ( nl, write(Msg), nl ) ) |
        true.
%
%       outByPE  --  output result on each PE
%      ---------
outByPE( Reps ) :- Reps = [result(pe(PE),_,_,_)|_] |
        totalByPE( PE, Reps, 0, 0, SumIfc, SumIbc, AllocNodes, NewReps ),
        outMsgByPE( PE, SumIfc, SumIbc, AllocNodes ),
```

```
        outByPE( NewReps ).
outByPE( Reps ) :- Reps = [] |
        true.
totalByPE( PE, [Rep|RestReps], SumIfc, SumIbc, FinalSumIfc, FinalSumIbc,
                                                    AllocNodes, NewReps ) :-
                Rep = result(pe(PE),node(Node),ope(byFin(Ifc),byBin(Ibc)),_),
                NewSumIfc := SumIfc+Ifc, NewSumIbc := SumIbc+Ibc |
        AllocNodes = [Node|NextAllocNodes],
        totalByPE( PE, RestReps, NewSumIfc,NewSumIbc, FinalSumIfc,FinalSumIbc,
                                                   NextAllocNodes, NewReps ).
totalByPE( PE, [Rep|RestReps], SumIfc, SumIbc, FinalSumIfc, FinalSumIbc,
                                                    AllocNodes, NewReps ) :-
                Rep = result(pe(OtherPE),_,_,_), PE \= OtherPE |
        NewReps = [Rep|NextNewReps],
        totalByPE( PE, RestReps, SumIfc, SumIbc, FinalSumIfc, FinalSumIbc,
                                                   AllocNodes, NextNewReps ).
totalByPE( PE, [], SumIfc, SumIbc, FinalSumIfc, FinalSumIbc,
                                               AllocNodes, NewReps ) :- true |
        FinalSumIfc = SumIfc, FinalSumIbc = SumIbc,
        AllocNodes = [], NewReps = [].
outMsgByPE( PE, SumIfc, SumIbc, AllocNodes ) :-
                SumIc := SumIfc+SumIbc,
                Msg = ope(pe(PE),node(AllocNodes),
                        byFin(SumIfc),byBin(SumIbc),total(SumIc)),
                prolog( ( write(Msg), nl ) ) |
        true.
```

pascal.doc

(0) Date: 1987-Jul-18, written by E. Sugino
    Modified: 25-Dec-87 by S. Takagi

(1) Program name: pascal

(2) Author: E. Sugino, ICOT 4th lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

Binomial coefficients are calculated using Pascal's triangle.
Pascal's triangle is:

```
                    1 1
                  1 2 1
                1 3 3 1
              1 4 6 4 1
            ...........
        1   a   b ... b a   1
      1 1+a a+b ... b+a a+1 1
    ........................
```

(5) Algorithm:

The (N)th layer of coefficients is calculated from the (N-1)th layer. When there is no (N-1)th layer, it is calculated from the (N-2)th layer. If the (N-2)th layer is not calculated yet, the (N-3)th layer is calculated. This algorithm is recursively applied. Once the (N)th layer is calculated, it is kept as a process.

(6) Process structure:

Layers of coefficients which were already calculated are kept as processes. These processes are waiting for a message from the same stream. The message requires a certain number of layers. When this number matches the number of process layers, the coefficient value which the process is keeping is sent in reply. When the number is larger than any number of layers which is currently kept, the process that currently has the largest number of layers will start calculating the next layers.

(7) Pragma: None

(8) Program:

Top-level loop:
pascal/0, pascal/3, pascal_go, max,
result, result_write, next, nextgo

Starter: pascal/4

Data process: pascal_data

Calculator: new_pascal, make_pascal_data

(9) Source file: pascal.ghc
    This document: pascal.doc

(10) Example:

Compile the program, and try it.

```
?- ghc pascal.
Binomial coefficient using Pascal's triangle
                              87.4.3. by Eiji Sugino
Give me Some integer !
> 10. <CR>                    % You can insert any positive integer.
```

(11) Evaluation data: Not recorded

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%
%       Binomial coefficient generator using Pascal's triangle
%                               87.4.3.  by Eiji Sugino
%       This Program can be used less than (X + Y)^19
%       so you should give a number that is from 1 to 19 .
%

%%%%%%%%%%%%%%%%%%%%%%%%%
%       PASCAL          %       Top Level procedure
%%%%%%%%%%%%%%%%%%%%%%%%%
pascal :-
        prolog((write('Binomial coefficient using Pascal''s triangle'),nl,
                write('                              87.4.3. by Eiji Sugino'),nl,
                write('Give me Some integer !'),nl)) |
        pascal_data(Stream,1,[1,1],1),
        pascal(Stream,1).

%%%%%%%%%%%%%%%%%       Loop
pascal(Stream,Max) :-
        prolog((prompt(X,'> '),read(N))) | pascal_go(N,Stream,Max).

pascal_go(N,Stream,Max) :-
        prolog((integer(N), statistics(runtime,_))) |
        pascal(N,Result,Stream,S),
        max(N,Max,M),
        result(Result,S,M).
pascal_go(N,Stream,Max) :- prolog((\+(integer(N)))) | pascal(Stream,Max).

max(N,M,X) :- N =< M | X = M.
max(N,M,X) :- N >= M | X = N.

%%%%%%%%%%%%%%%%%       Result writer
result(Result,Stream,N) :- true |
        result_write(Result,End,N),
        next(End,Stream,N).

result_write([],E,N) :- prolog((statistics(runtime,[_,Time]),
                                write('END'),nl,
                                write('Runtime : '),
                                write(Time),write(' msec'),nl,
                                write('Max coefficient : '),
                                write(N),nl)) |
        E = end.
result_write([A|B],E,N) :- wait(A),
                           prolog((write(A),write(', '))) |
        result_write(B,E,N).

%%%%%%%%%%%%%%%%%       End of one cycle
next(end,Stream,N) :- prolog((prompt(_,'more ? '),read(X))) |
        nextgo(X,Stream,N).

nextgo(n,Stream,_) :- true| Stream = [].
nextgo('N',Stream,_) :- true | Stream = [].
nextgo(y,Stream,N) :- true | pascal(Stream,N).
nextgo('Y',Stream,N) :- true | pascal(Stream,N).
nextgo(X,Stream,N) :- X \= n, X \= y, X \= 'N', X \= 'Y' |
        next(end,Stream,N).

%%%%%%%%%%%%%%%%%%%%      Main routine
pascal(N,Result,Stream,S) :- prolog(integer(N)) |
        Stream = [data(N,Result)|S].
```

```
%%%%%%%%%%%%%%%%%%%%      Pascal's triangle data
pascal_data([data(N,D)|S],N,Data,Max) :- true |
        D = Data,
        pascal_data(S,N,Data,Max).
pascal_data([data(N,D)|S],M,Data,Max) :- N =\= M,N =< Max |
        pascal_data(S,M,Data,Max).
pascal_data([data(N,D)|S],Max,Data,Max) :- Max < N,M1 := Max + 1 |
        new_pascal(M1,N,Data,D,S),
        pascal_data(S,Max,Data,N).
pascal_data([data(N,D)|S],M,Data,Max) :-  Max < N, M < Max |
        pascal_data(S,M,Data,N).
pascal_data([],_,_,_) :- true | true.

%%%%%%%%%%%%%%%%%%%%      Data creator
new_pascal(N,N,Data,D,Stream) :- true |
        make_pascal_data(Data,D),
        pascal_data(Stream,N,D,N).
new_pascal(N,M,Data,D,Stream) :- N < M, N1 := N+1 |
        make_pascal_data(Data,Data1),
        pascal_data(Stream,N,Data1,M),
        new_pascal(N1,M,Data1,D,Stream).

make_pascal_data([F1,F2|Data],New) :- Nf2 := F1+F2 |
        New = [F1,Nf2|New1],
        make_pascal_data([F2|Data],New1,[Nf2,F1]).
make_pascal_data([N],New,E) :- true | New = [N].
make_pascal_data([A,A|C],New,E) :- B := A+A | New = [B|E].
make_pascal_data([A,B|C],New,E) :- A > B | New = E.
make_pascal_data([A,B|C],New,E) :- A < B, D := A+B |
        New = [D|New1],
        make_pascal_data([D|C],New1,[D|E]).
```

tep.doc

(0) Date: 3-Jul-87, written by M. Koshimura
Modified: 25-Dec-87 by S. Takagi

(1) Program name: tep

(2) Author: M. Koshimura, ICOT 4th Lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

A tiny propositional calculus solver by LK (Logistischer Kalkul).

A problem is represented by a sequent. A sequent consists of 2 parts.
One is the left hand side (LHS). The other is the right hand side (RHS).
For example,

```
        a&(a->b) --> b
```

means that if a and a implies b is true, then b is true.
Here a&(a->b) is the LHS and b is the RHS.
In general, both the LHS and RHS consist of a sequence of formulas.

Let ai, bi be a formula, then a1,...,an --> b1,...,bm is a sequent.
Here, the LHS is a1,...,an , and the RHS is b1,...,bm.
This means that (a1 and ... and an) implies (b1 or ... or bm).
The sequent a1,...,an --> b1,...,bm is called an axiom
if there exists i,j such that ai = bj.

The following logical symbols are supported.

~	Strong (classical) negation
\/	Disjunction
/\	Conjunction
->	Implication
<->	Equivalence

This system can prove predicates such as:

```
        --> (~ ~a) <-> a
        --> (b -> a) <-> (~a -> ~b)
```

(5) Algorithm:

The program keeps two different lists, which present the LHS and RHS,
i.e. a problem is represented by these two lists.

First,
Check the intersection of the LHS and RHS.
If the intersection is not empty, this is an axiom, and returns true.
If the intersection is empty, continue to the next step.
Second,
Try to decompose the problem to subproblem(s), according to the
decomposition rules. The subproblems have an &-relation.
If succeeds, recursively apply the algorithm to the subproblem(s).
If fails, it is not provable in the algorithm.

(6) Process structure:

Run in one process when crunching the term. When two or more
subproblems are obtained, these subproblems are processed by newly
created processes. The original process merges the results of the
forked processes.

(7) Pragma: Not supported

(8) Program:

The program consists of the decompose part and fork part.
tep/2 is the top level.

The decompose part checks whether the problem is an axiom,
using intersect/3.

Next, it tries to decompose the problem, using a decompose predicate,
such as lnot/7 or rnot.

l means the LHS, and r means the RHS. '***' of l'***', r'***' means
a propositional symbol, which l'***' (r'***') tries to decompose.

If two or more decomposing rules can be applied to the problem, it is
necessary to choose one. This is done by decompose_decidel/3.

The fork part forks the processes which correspond to each decomposed
subproblem(s). This part is included in tepl/2.

The rest is a pretty print routine for the proof tree.
If the procedure fails to prove the problem,
the printed proof tree includes the atom, fail.

(9) Source file: tep.ghc
This document: tep.doc

(10) Examples:

Invocation:

```
        tep_go(N).      where N is the trial problem number.
                        N = 1 to 8 are effective.
                        The more tries the more complicated sequent.

        test number 1:  --> (~ ~a) <-> a
        test number 2:  --> ~ (~a /\ a)
        test number 3:  --> ~ (b /\ a) <-> (~b \/ ~a)
        test number 4:  --> ~ (b \/ a) <-> (~b /\ ~a)
        test number 5:  --> (b -> a) <-> (~a -> ~b)
        test number 6:  --> (~ (b -> a)) <-> (b /\ ~a)
        test number 7:  --> (~a \/ b) <-> (a -> b)
        test number 8:  --> ((a<->b) <-> c) <-> (a <-> (b<->c))
```

(11) Evaluation data: Not recorded

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

:- op(200, fy,[~]).
:- op(500,yfx,[\/]).
:- op(500,yfx,[/\]).
:- op(1050,xfy,[->]).
:- op(1060,xfx,[<->]).
%************************************************
%
% tep(+Sequent, -Proof)
%
%************************************************
tep(Seq, Proof) :- true |
        decompose(Seq, Inf,UpSeqs),
        tep(Inf, Seq, UpSeqs, Proof).
%
% tep(+Inf, +Sequent, +UpperSequents, -Proof)
%
tep(axiom, Seq, _, Proof) :- true | Proof = [axiom, Seq].
tep(fail, Seq, _, Proof) :- true | Proof = [fail, Seq].
tep(Inf, Seq, UpSeqs, Proof) :- Inf \= axiom, Inf \= fail |
        Proof = [Inf, Seq, Proof1],
        tep1(UpSeqs, Proof1).

%
% tep1(+UpperSequents, -Proof)
%
tep1([Seq|Seqs], Proof) :- true |
        Proof = [SProof|Proof1],
        tep(Seq, SProof),
        tep1(Seqs, Proof1).
tep1([], Proof) :- true | Proof = [].

%
% lnot(+Left,+Right, +LeftHead,-LeftTail, +LeftHead,-LeftTail, -UpperSequents)
%
lnot([(~A)|Left],Right, LHead,TLeft, _,_, Ans) :- true |
        TLeft = Left,
        Ans = [[LHead,[A|Right]]].
lnot([Other|Left],Right, LHead,TLeft, _,_, Ans) :- Other \= (~A) |
        TLeft = [Other|CLHead],
        lnot(Left,Right, LHead,CLHead, _,_, Ans).

%
% rnot(+Left,+Right, +RightHead,-RightTail, +RightHead,-RightTail,
%                       -UpperSequents)
%
rnot(Left,[(~A)|Right], RHead,TRight, _,_, Ans) :- true |
        TRight = Right,
        Ans = [[[A|Left],RHead]].
rnot(Left,[Other|Right], RHead,TRight, _,_, Ans) :- Other \= (~A) |
        TRight = [Other|CRHead],
        rnot(Left,Right, RHead,CRHead, _,_, Ans).

land([(A/\B)|Left],Right, LHead,TLeft, _,_, Ans) :- true |
        TLeft = [A,B|Left],
        Ans = [[LHead,Right]].
land([Other|Left],Right, LHead,TLeft, _,_, Ans) :- Other \= (A/\B) |
        TLeft = [Other|CLHead],
        land(Left,Right, LHead,CLHead, _,_, Ans).

rand(Left,[(A/\B)|Right], RHead,TRight, RHead1,TRight1, Ans) :- true |
        TRight = [A|Right],
        TRight1 = [B|Right],
```

```
        Ans = [[Left,RHead],[Left,RHead1]].
rand(Left,[Other|Right], RHead,TRight, RHead1,TRight1, Ans) :-
        Other \= (A/\B) |
        TRight = [Other|CRHead],
        TRight1 = [Other|CRHead1],
        rand(Left,Right, RHead,CRHead, RHead1,CRHead1, Ans).

lor([(A\/B)|Left],Right, LHead,TLeft, LHead1,TLeft1, Ans) :- true |
        TLeft = [A|Left],
        TLeft1 = [B|Left],
        Ans = [[LHead,Right],[LHead1,Right]].
lor([Other|Left],Right, LHead,TLeft, LHead1,TLeft1, Ans) :-  Other \= (A\/B) |
        TLeft = [Other|CLHead],
        TLeft1 = [Other|CLHead1],
        lor(Left,Right, LHead,CLHead, LHead1,CLHead1, Ans).

ror(Left,[(A\/B)|Right], RHead,TRight, _,_, Ans) :- true |
        TRight = [A,B|Right],
        Ans = [[Left,RHead]].
ror(Left,[Other|Right], RHead,TRight, _,_, Ans) :- Other \= (A\/B) |
        TRight = [Other|CRHead],
        ror(Left,Right, RHead,CRHead, _,_, Ans).

limpl([(A->B)|Left],Right, LHead,TLeft, LHead1,TLeft1, Ans) :- true |
        TLeft = Left,
        TLeft1 = [B|Left],
        Ans = [[LHead,[A|Right]],[LHead1,Right]].
limpl([Other|Left],Right, LHead,TLeft, LHead1,TLeft1, Ans) :- Other \= (A->B) |
        TLeft = [Other|CLHead],
        TLeft1 = [Other|CLHead1],
        limpl(Left,Right, LHead,CLHead, LHead1,CLHead1, Ans).

rimpl(Left,[(A->B)|Right], RHead,TRight, _,_, Ans) :- true |
        TRight = [B|Right],
        Ans = [[[A|Left],RHead]].
rimpl(Left,[Other|Right], RHead,TRight, _,_, Ans) :- Other \= (A->B) |
        TRight = [Other|CRHead],
        rimpl(Left,Right, RHead,CRHead, _,_, Ans).

lequiv([(A<->B)|Left],Right, LHead,TLeft, LHead1,TLeft1, Ans) :- true |
        TLeft = [A,B|Left],
        TLeft1 = Left,
        Ans = [[LHead,Right],[LHead1,[A,B|Right]]].
lequiv([Other|Left],Right, LHead,TLeft, LHead1,TLeft1, Ans) :-
        Other \= (A<->B) |
        TLeft = [Other|CLHead],
        TLeft1 = [Other|CLHead1],
        lequiv(Left,Right, LHead,CLHead, LHead1,CLHead1, Ans).

requiv(Left,[(A<->B)|Right], RHead,TRight, RHead1,TRight1, Ans) :- true |
        TRight = [B|Right],
        TRight1 = [A|Right],
        Ans = [[[A|Left],RHead],[[B|Left],RHead1]].
requiv(Left,[Other|Right], RHead,TRight, RHead1,TRight1, Ans) :-
        Other \= (A<->B) |
        TRight = [Other|CRHead],
        TRight1 = [Other|CRHead1],
        requiv(Left,Right, RHead,CRHead, RHead1,CRHead1, Ans).

%
% decompose(+Sequent, -Inference,-UpperSequent)
%
decompose([Left,Right], Inf,UpSeqs) :- true |
        intersect(Left,Right, Ans),
        decompose_decide(Ans, [Left,Right], Inf,UpSeqs).
```

```
%
% decompose_decide(+Result, +Sequent, -Inference,-UpperSequent)
%
decompose_decide(true, _, Inf,UpSeqs) :- true |
        Inf = axiom,
        UpSeqs = [].
decompose_decide(fail, [Left,Right], Inf,UpSeqs) :- true |
        decompose_decide1(Left,Right, Ans),
        decompose1(Ans, Left,Right, Inf,UpSeqs).

%
% decompose_decide1(+Left,+Right, -Answer)
%
decompose_decide1(Left,Right, Ans) :- true |
        decompose_decide1_left(Left, Ans1),
        decompose_decide1_right(Right, Ans2),
        check_ans(Ans1,Ans2, Ans).

%
% decompose_decide1_left(+Left, -Answer)
%
decompose_decide1_left([], Ans) :- true | Ans = [].
decompose_decide1_left([(~A)|Left], Ans) :- true |
        Ans = [(10,lnot)|Ans1],
        decompose_decide1_left(Left, Ans1).
decompose_decide1_left([(A/\B)|Left], Ans) :- true |
        Ans = [(10,land)|Ans1],
        decompose_decide1_left(Left, Ans1).
decompose_decide1_left([(A\/B)|Left], Ans) :- true |
        Ans = [(20,lor)|Ans1],
        decompose_decide1_left(Left, Ans1).
decompose_decide1_left([(A->B)|Left], Ans) :- true |
        Ans = [(20,limpl)|Ans1],
        decompose_decide1_left(Left, Ans1).
decompose_decide1_left([(A<->B)|Left], Ans) :- true |
        Ans = [(20,lequiv)|Ans1],
        decompose_decide1_left(Left, Ans1).
decompose_decide1_left([A|Left], Ans) :-
        A \= (~B), A \= (B/\C), A \= (B\/C), A \= (B->C), A \= (B<->C)|
        decompose_decide1_left(Left, Ans).

%
% decompose_decide1_right(+Right, -Answer)
%
decompose_decide1_right([], Ans) :- true | Ans = [].
decompose_decide1_right([(~A)|Right], Ans) :- true |
        Ans = [(10,rnot)|Ans1],
        decompose_decide1_right(Right, Ans1).
decompose_decide1_right([(A/\B)|Right], Ans) :- true |
        Ans = [(20,rand)|Ans1],
        decompose_decide1_right(Right, Ans1).
decompose_decide1_right([(A\/B)|Right], Ans) :- true |
        Ans = [(10,ror)|Ans1],
        decompose_decide1_right(Right, Ans1).
decompose_decide1_right([(A->B)|Right], Ans) :- true |
        Ans = [(10,rimpl)|Ans1],
        decompose_decide1_right(Right, Ans1).
decompose_decide1_right([(A<->B)|Right], Ans) :- true |
        Ans = [(20,requiv)|Ans1],
        decompose_decide1_right(Right, Ans1).
decompose_decide1_right([A|Right], Ans) :-
        A \= (~B), A \= (B/\C), A \= (B\/C), A \= (B->C), A \= (B<->C)|
        decompose_decide1_right(Right, Ans).

%
% check_ans(+Answer,+Answer, -Answer)
```

```
%
check_ans(Ans1,Ans2, Ans) :- true |
        merge(Ans1,Ans2, Ans3),
        check_ans1(Ans3, cand(1000,fail), Ans).

%
% check_ans1(+Answer, +Candidate, -Answer)
%
check_ans1([(N,OP)|Ans1], cand(CN,COP), Ans) :- N < CN |
        check_ans1(Ans1, cand(N,OP), Ans).
check_ans1([(N,OP)|Ans1], cand(CN,COP), Ans) :- N >= CN |
        check_ans1(Ans1, cand(CN,COP), Ans).
check_ans1([], cand(CN,COP), Ans) :- true | Ans = COP.

%
% decompose1(+Inference, +Left,+Right, -Inference,-UpperSequents)
%
decompose1(lnot, Left,Right, Inf,UpSeqs) :- true |
        Inf = lnot,
        lnot(Left,Right, Left1,Left1, Left2,Left2, UpSeqs).
decompose1(rnot, Left,Right, Inf,UpSeqs) :- true |
        Inf = rnot,
        rnot(Left,Right, Right1,Right1, Right2,Right2, UpSeqs).
decompose1(land, Left,Right, Inf,UpSeqs) :- true |
        Inf = land,
        land(Left,Right, Left1,Left1, Left2,Left2, UpSeqs).
decompose1(ror, Left,Right, Inf,UpSeqs) :- true |
        Inf = ror,
        ror(Left,Right, Right1,Right1, Right2,Right2, UpSeqs).
decompose1(rimpl, Left,Right, Inf,UpSeqs) :- true |
        Inf = rimpl,
        rimpl(Left,Right, Right1,Right1, Right2,Right2, UpSeqs).
decompose1(rand, Left,Right, Inf,UpSeqs) :- true |
        Inf = rand,
        rand(Left,Right, Right1,Right1, Right2,Right2, UpSeqs).
decompose1(lor, Left,Right, Inf,UpSeqs) :- true |
        Inf = lor,
        lor(Left,Right, Left1,Left1, Left2,Left2, UpSeqs).
decompose1(limpl, Left,Right, Inf,UpSeqs) :- true |
        Inf = limpl,
        limpl(Left,Right, Left1,Left1, Left2,Left2, UpSeqs).
decompose1(lequiv, Left,Right, Inf,UpSeqs) :- true |
        Inf = lequiv,
        lequiv(Left,Right, Left1,Left1, Left2,Left2, UpSeqs).
decompose1(requiv, Left,Right, Inf,UpSeqs) :- true |
        Inf = requiv,
        requiv(Left,Right, Right1,Right1, Right2,Right2, UpSeqs).
decompose1(fail, Left,Right, Inf,UpSeqs) :- true | Inf = fail.

%%%%%%%%%%%%%%%%%%%%
%
% Pretty Print
%
%%%%%%%%%%%%%%%%%%%%
pp([axiom,Sequent],Tab,HOut-TOut) :- true |
        HOut = [tab(Tab), write(Sequent), nl, tab(Tab), write(axiom), nl|TOut].
pp([fail,Sequent],Tab,HOut-TOut) :- true |
        HOut = [tab(Tab), write(Sequent), nl, tab(Tab), write(fail), nl |TOut].
pp([Inf,Sequent,Proof],Tab,HOut-TOut) :-
        Inf \= axiom, Inf \= fail |
        HOut = [tab(Tab), write(Sequent), nl, tab(Tab), write(Inf), nl| HOut1],
        Tab1 := Tab+5,
        pp_subproof(Proof,Tab1,HOut1-TOut).

pp_subproof([Proof|RestProof],Tab,HOut-TOut) :- true |
        pp(Proof,Tab,HOut-HOut1),
```

```
        pp_subproof(RestProof,Tab,HOut1-TOut).
pp_subproof([],_,HOut-TOut) :- true | HOut = TOut.

%%%%%%%%%%%%%%%%%%%%
%
% Library
%
%%%%%%%%%%%%%%%%%%%%
member(X,[X|Xs],Ans) :- true | Ans = true.
member(A,[X|Xs],Ans) :- A \= X | member(A,Xs,Ans).
member(_,[],Ans) :- true | Ans = fail.

merge([A|X], Y, Z) :- true | Z = [A|Zr], merge(Y, X, Zr).
merge(X, [A|Y], Z) :- true | Z = [A|Zr], merge(Y, X, Zr).
merge([], X, Z) :- true | Z = X.
merge(X, [], Z) :- true | Z = X.

intersect([X|Xs],Ys,Ans) :- true |
        member(X,Ys,Ans1),
        intersect_decide(Ans1,Xs,Ys,Ans).
intersect([],_,Ans) :- true | Ans = fail.

intersect_decide(true,_,_,Ans) :- true | Ans = true.
intersect_decide(fail,Xs,Ys,Ans) :- true | intersect(Xs,Ys,Ans).


tep(Sequent) :- true |
        tep(Sequent,Proof),
        pp(Proof,0,NewProof-[]),
        outstream(NewProof).

tep_go(1) :- true |
        tep([[],
            [((~ ~ a) <-> a)]]).
tep_go(2) :- true |
        tep([[],
            [~ (~ a /\ a)]]).
tep_go(3) :- true |
        tep([[],
            [(~ (b/\a) <-> (~ b \/ ~ a))]]).
tep_go(4) :- true |
        tep([[],
            [(~ (b\/a) <-> (~ b /\ ~ a))]]).
tep_go(5) :- true |
        tep([[],
            [((b -> a) <-> (~ a -> ~ b))]]).
tep_go(6) :- true |
        tep([[],
            [(~ (b -> a) <-> (b /\ ~a))]]).
tep_go(7) :- true |
        tep([[],
            [((~ a \/ b) <-> (a -> b))]]).
tep_go(8) :- true |
        tep([[],
            [(((a<->b) <-> c) <-> (a <-> (b<->c)))]]).
```

waltz.doc

(0) Date: 18-Jul-87, written by E. Sugino
    Modified: 25-Dec-87 by S. Takagi

(1) Program name: waltz

(2) Author: E. Sugino, ICOT 4th lab.

(3) Runs on: GHC on DEC 2065

(4) Description of the problem:

The labeling problem is to analyze a line drawing.
All lines are classified into three types.

a) A border line which has a direction.
   The way to determine a direction is:
   when you walk along a line looking at an object, your forward direction is decided as one direction. One direction is going out from the node, the other is coming into the node.
   These are described as 'out' and 'in' in this program.
b) Convex line which is described as 'p' ('+')
c) Concave line which is described as 'm' ('-')

Generally speaking, nodes (junctions) of an object are classified into 9 types. Only four types:

'arrow' /|\ , 'l' \/ , 'fork' Y , and 't' -|

are enough to classify objects on the assumption of:

a) No cracks or shadows;
b) All of the vertices are made with only three planes;
c) The properties of junctions of any line drawing are not changed when the viewpoint is moved.

(5) Algorithm:

Each node has several possible candidates for the type of junctions.
Each node sends the possible types of its own hands to the neighbors.
Candidates for the types of nodes will be restricted to fewer types by the neighbors' types. When candidates for the types of a node are restricted to fewer types, the node sends the restricted possible types of its own hands to the neighbors. Finally, any node will be restricted to only one type (if the line drawing is correct).

(6) Process structure:

Each node is a process. They are connected to each other.

(7) Pragma: None

(8) Program:

waltz1, waltz2, waltz3 and waltz4 are sample programs.
The line drawings in these programs are seen in
  'Preira Si ga teiansita 5tu no mondai nituite' (PSM-O-A-KL1-003).

waltz/3 needs the information of the junctions and the borders.
    The first argument is a list for information of the junctions,
    any element of which is the form:
        p(type of the node,
          left node number,
          back node number,
          right node number,
          node number)
    The second argument is a list of information of the borders,
    which consists of border node numbers.

Hands of junctions are connected by conv in ana.
conv1 gives the information that
    if a junction is a border,
    the types of its hands must be in or out.
Junctions are released as processes by waltz_go.
p1 is the first process
    which sends the information of its type to the neighbors.
p is a process for a node.
    When its node type is restricted to only one,
    the node decides its type.
p waits for information from neighbors.
    When a message has come, p checks whether its node type
    candidates can be restricted by the information.
    Then p replies by sending its new candidates to the neighbors
    if the candidates are restricted.

(9) Source file: waltz.ghc
    This document: waltz.doc

(10) Example:

Sample programs are invoked as follows.

```
        ?- ghc waltz1.          % Simple Cube

        ?- ghc waltz2.

        ?- ghc waltz3.

        ?- ghc waltz4.
```

If you want to try another line drawing, use waltz/3.
See waltz1 or others.

(11) Evaluation data: Not recorded

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

waltz1 :- true | waltz([p(a,1,7,2,1),p(l,3,0,2,2),p(a,3,8,4,3),p(f,7,9,8,4),
                  p(l,5,0,4,5),p(a,5,9,6,6),p(l,1,0,6,7)],
                 [1,2,3,4,5,6],
                 Result),
        wr(Result).
waltz2 :- true |
        waltz([p(a,1,9,2,1),p(l,3,0,2,2),p(t,4,3,14,3),
             p(l,5,0,4,4),p(a,5,15,6,5),p(l,7,0,6,6),
             p(a,7,17,8,7),p(l,1,0,8,8),p(f,9,17,10,9),
             p(a,16,11,10,10),p(l,12,0,11,11),p(a,12,13,14,12),
             p(f,13,16,15,13)],
        [1,2,3,4,5,6,7,8],
            Result),
        wr(Result).
waltz3 :- true |
        waltz([p(a,6,7,1,1),p(l,2,0,1,2),p(a,2,8,3,3),
             p(l,4,0,3,4),p(a,4,9,5,5),p(l,6,0,5,6),
             p(f,7,15,10,7),p(a,10,16,11,8),p(f,11,12,8,9),
             p(a,12,17,13,10),p(f,14,13,9,11),p(a,14,18,15,12),
             p(f,18,17,16,13)],
        [1,2,3,4,5,6],
            Result),
        wr(Result).
waltz4 :- true |
        waltz([p(a,8,9,1,1),p(l,2,0,1,2),p(a,2,12,3,3),p(l,4,0,3,4),
                p(a,4,13,5,5),p(f,5,14,6,6),p(a,6,15,7,7),p(l,8,0,7,8),
                p(f,9,15,10,9),p(a,10,14,11,10),p(f,11,13,12,11)],
                [1,2,3,4,5,6,7,8],
                Result),
        wr(Result).

waltz1(Result) :- true |
        waltz([p(a,1,7,2,1),p(l,3,0,2,2),p(a,3,8,4,3),p(f,7,9,8,4),
                 p(l,5,0,4,5),p(a,5,9,6,6),p(l,1,0,6,7)],
                [1,2,3,4,5,6],
                Result).
waltz2(Result) :- true |
        waltz([p(a,1,9,2,1),p(l,3,0,2,2),p(t,4,3,14,3),
             p(l,5,0,4,4),p(a,5,15,6,5),p(l,7,0,6,6),
             p(a,7,17,8,7),p(l,1,0,8,8),p(f,9,17,10,9),
             p(a,16,11,10,10),p(l,12,0,11,11),p(a,12,13,14,12),
             p(f,13,16,15,13)],
        [1,2,3,4,5,6,7,8],
            Result).
waltz3(Result) :- true |
        waltz([p(a,6,7,1,1),p(l,2,0,1,2),p(a,2,8,3,3),
             p(l,4,0,3,4),p(a,4,9,5,5),p(l,6,0,5,6),
             p(f,7,15,10,7),p(a,10,16,11,8),p(f,11,12,8,9),
             p(a,12,17,13,10),p(f,14,13,9,11),p(a,14,18,15,12),
             p(f,18,17,16,13)],
        [1,2,3,4,5,6],
            Result).

ana([p(Type,L,D,R,N)|Rest],Result,Kyoukai,Vars,NVar) :- true |
        conv(L,LL,Kyoukai,Vars,V0),
        conv(D,DD,Kyoukai,V0,V1),
        conv(R,RR,Kyoukai,V1,V2),
        Result = [p(Type,LL,DD,RR,N) | Resultt],
        ana(Rest,Resultt,Kyoukai,V2,NVar).
ana([],Result,Kyoukai,Vars,V0) :- true | Result =[],V0=[].

conv(0,V,K,L,NV) :- true | V = _|_,NV=L.
```

```
conv(N,V,K,[N-Var|L],NV) :- N > 0 | V=Var,NV=L.
conv(N,V,K,[M-Var|L],NV) :- N > 0,N =\= M |
                conv(N,V,K,L,NV1),NV=[M-Var|NV1].
conv(N,V,K,[],NV) :- N > 0 |
                conv1(K,N,V,NV).

conv1([A|B],A,V,NV) :- true | V = [[in,out]|X]-Y,NV=[A-(Y-X)].
conv1([A|B],C,V,NV) :- A =\= C | conv1(B,C,V,NV).
conv1([],C,V,NV) :- true | V = A-B,NV=[C-(B-A)].

wr([]) :- true | true.
wr([A-T-X|B]) :- wait(X),prolog((write(A-T-X),nl)) | wr(B).

waltz(List,K,Result) :- true |
                ana(List,L,K,[],_),
                waltz_go(L,Result).

waltz_go([ p(Type,L,D,R,N) | Rest],Out) :- true |
                junctions(Type,Types),
                p1(L,D,R,Types,Result),
                Out = [N-Type-Result | Ot],
                waltz_go(Rest,Ot).
waltz_go([],O) :- true | O=[].


p(L,D,R,[],Type) :- true |  Type =[(?,?,?)].
p(L-LL,D-DD,R-RR,[(L1,D1,R1)],Type) :- true |
                Type=[(L1,D1,R1)],LL=[L1],DD=[D1],RR=[R1].
p(L,D,[R|Rt]-RR,[T1,T2|T3],Type) :- prolog(atom(R)) |
                filter(r,[T1,T2|T3],R,New),
                ack(r,[T1,T2|T3],New,RR,RRt),
                p(L,D,Rt-RRt,New,Type).
p([L|Lt]-LL,D,R,[T1,T2|T3],Type) :- prolog(atom(L)) |
                filter(l,[T1,T2|T3],L,New),
                ack(l,[T1,T2|T3],New,LL,LLt),
                p(Lt-LLt,D,R,New,Type).
p(L,[D|Dt]-DD,R,[T1,T2|T3],Type) :- prolog(atom(D)) |
                filter(d,[T1,T2|T3],D,New),
                ack(d,[T1,T2|T3],New,DD,DDt),
                p(L,Dt-DDt,R,New,Type).

p([[L1|L2]|Lt]-LL,D,R,Types,Type) :- true |
                filter(l,Types,[L1|L2],New),
                ack(l,Types,New,LL,LLt),
                p(Lt-LLt,D,R,New,Type).
p(L,[[D1|D2]|Dt]-DD,R,Types,Type) :- true |
                filter(d,Types,[D1|D2],New),
                ack(d,Types,New,DD,DDt),
                p(L,Dt-DDt,R,New,Type).
p(L,D,[[R1|R2]|Rt]-RR,Types,Type) :- true |
                filter(r,Types,[R1|R2],New),
                ack(r,Types,New,RR,RRt),
                p(L,D,Rt-RRt,New,Type).

p1(L,D,R,(Left,Down,Right),Type) :- true |
                send(Left,L,LL),
                send(Down,D,DD),
                send(Right,R,RR),
                p(LL,DD,RR,(Left,Down,Right),Type).

send(Hand,X-To,New) :- true | send1(Hand,To1),To=[To1|T],New = X-T.
send1([[P|List]|B],To) :- true | To=[P|T],send1(B,T).
send1([],To) :- true | To = [].

ack(P,Old,List,Send,St) :- true |
                same(Old,List,X),ack_s(X,P,List,Send,St).
```

```
ack_s(y,P,List,Send,St) :- true | Send=[L|St],ack1(P,List,L,[]).
ack_s(n,P,List,Send,St) :- true | Send = St.

same([A|B],[AA|BB],X) :- true  | same(B,BB,X).
same([],[],X) :- true | X=y.
same([],A,X) :- A \= [] | X=n.
same(A,[],X) :- A \= [] | X=n.
same(A,B,X) :- A \= [_|_] | X = n.
same(B,A,X) :- A \= [_|_] | X = n.

ack1(l,[(L1,_,_)|LL],Send,M) :- true |
        filtack(L1,M,M1,Send,S1),ack1(l,LL,S1,M1).
ack1(d,[(_,L1,_)|LL],Send,M) :- true |
        filtack(L1,M,M1,Send,S1),ack1(d,LL,S1,M1).
ack1(r,[(_,_,L1)|LL],Send,M) :- true |
        filtack(L1,M,M1,Send,S1),ack1(r,LL,S1,M1).
ack1(_,[],Se,_) :- true | Se=[].

filtack(A,[A|B],X,Send,St) :- true | Send = St,X=[A|B].
filtack(A,[C|B],X,Send,St) :- A \= C | filtack(A,B,X1,Send,St),X=[C|X1].
filtack(A,[],X,Send,St) :- true | Send=[A|St],X=[A].


junctions(l,Candidates) :-
        L1=(out,nil,in),L2=(in,nil,out),L3=(p,nil,out),
        L4=(in,nil,p),L5=(m,nil,in),L6=(out,nil,m) |
        Candidates =
                ([[out,[L1,L6]],[in,[L2,L4]],[p,[L3]],[m,[L5]]],
                 [],
                 [[in,[L1,L5]],[out,[L2,L3]],[p,[L4]],[m,[L6]]]).
junctions(f,Candidates) :-
        F1=(p,p,p),F2=(m,m,m),F3=(in,m,out),F4=(m,out,in),F5=(out,in,m) |
        Candidates =
                ([[p,[F1]],[m,[F2,F4]],[in,[F3]],[out,[F5]]],
                 [[p,[F1]],[m,[F2,F3]],[out,[F4]],[in,[F5]]],
                 [[p,[F1]],[m,[F2,F5]],[out,[F3]],[in,[F4]]]).
junctions(t,Candidates) :-
        T1=(out,p,in),T2=(out,m,in),T3=(out,in,in),T4=(out,out,in) |
        Candidates =
                ([[out,[T1,T2,T3,T4]]],
                 [[p,[T1]],[m,[T2]],[in,[T3]],[out,[T4]]],
                 [[in,[T1,T2,T3,T4]]]).
junctions(a,Candidates) :-
        A1=(in,p,out),A2=(m,p,m),A3=(p,m,p) |
        Candidates =
                ([[in,[A1]],[m,[A2]],[p,[A3]]],
                 [[p,[A1,A2]],[m,[A3]]],
                 [[out,[A1]],[m,[A2]],[p,[A3]]]).

% filter([l,d,r],Input,[in,out,p,m),Out)

filter(l,(L,_,_),T,F) :- true | filter1(T,L,F).
filter(d,(_,D,_),T,F) :- true | filter1(T,D,F).
filter(r,(_,_,R),T,F) :- true | filter1(T,R,F).
filter(P,[A|B],T,F) :- true | filter(P,T,A,F,Ft),filter(P,B,T,Ft).
filter(P,[],T,F) :- true | F=[].

% filter([l,d,r],[in,out,p,m],Type,Out,Out_tail)
% filter( List  ,     ..     , .. , . ,  ..    )

filter(l,T,(L,D,R),F,Ft) :- prolog(atom(T)) | filter2(T,L,(L,D,R),F,Ft).
filter(d,T,(L,D,R),F,Ft) :- prolog(atom(T)) | filter2(T,D,(L,D,R),F,Ft).
filter(r,T,(L,D,R),F,Ft) :- prolog(atom(T)) | filter2(T,R,(L,D,R),F,Ft).
filter(P,[A],Type,F,Ft) :- true | filter(P,A,Type,F,Ft).
filter(P,[A,B|C],Type,F,Ft) :- true |
                filter(P,A,Type,F,Ft1),
```

```
                filter(P,[B|C],Type,Ft1,Ft).

filter1(p  ,[[p  ,L]|_],F) :- true | F=L.
filter1(m  ,[[m  ,L]|_],F) :- true | F=L.
filter1(in ,[[out,L]|_],F) :- true | F=L.
filter1(out,[[in ,L]|_],F) :- true | F=L.
filter1(p  ,[[T|_]|Rest],F) :- T \= p | filter1(p,Rest,F).
filter1(m  ,[[T|_]|Rest],F) :- T \= m | filter1(m,Rest,F).
filter1(in ,[[T|_]|Rest],F) :- T \= out | filter1(in,Rest,F).
filter1(out,[[T|_]|Rest],F) :- T \= in  | filter1(out,Rest,F).
filter1(_,  [],          F) :- true | F=[].
filter1([A|B],Types,F) :- true | filter1(A,Types,F1),filter1(B,Types,F2),
                                 append(F1,F2,F).
filter1([],_,F) :- true | F=[].

append([],X,Y) :- true | X=Y.
append([X|Y],L,Z) :- true | Z = [X|ZZ],append(Y,L,ZZ).

filter2(p,p,Type,F,Ft) :- true | F = [Type|Ft].
filter2(m,m,Type,F,Ft) :- true | F = [Type|Ft].
filter2(in,out,Type,F,Ft) :- true | F = [Type|Ft].
filter2(out,in,Type,F,Ft) :- true | F = [Type|Ft].
filter2(p,X,Type,F,Ft) :- X \= p | F = Ft.
filter2(m,X,Type,F,Ft) :- X \= m | F = Ft.
filter2(in,X,Type,F,Ft) :- X \= out | F = Ft.
filter2(out,X,Type,F,Ft) :- X \= in | F = Ft.
```

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%%%%%%%%%%%%%%%%%%% GHC Compiler System for SICSTUS Prolog %%%%%%%%%%%%%%%%%%
%                                                                              %
%                        Version 1.9  (November 18, 1987)                      %
%                                                                              %
%                        Kazunori Ueda                                         %
%             Institute for New Generation Computer Technology                 %
%              4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan                  %
%                                                                              %
%               phone: +81-3-456-2514    telex: ICOT J 32964                   %
%               csnet: ueda%icot.jp@relay.cs.net                               %
%               uucp:  {enea,inria,kddlab,mit-eddie,ukc}!icot!ueda             %
%                                                                              %
%       Modification for outside ICOT by S. Takagi, 4-Jan-88                   %
%       This Program is derived from DEC-10 Prolgo version.                    %
%       This is tested on SICSTUS Prolog running on SUN-3/260HM                %
%       OS 3.3.                                                                %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(1150,fx, (ghc)).
:- op(1150,fx, (ghcspy)).
:- op(1150,fx, (ghcnospy)).
:- op(700, xfx,:=).              % becomes
:- op(700, xfx,\=).              % dif
:- op(50,  xf, ?).               % for specifying 'trace-mode'

%%%% SAVING EXECUTABLE IMAGE %%%%
:- public saveghc/1.
saveghc(F) :- save(F), ghcbacktrace, header.
:- public header/0.
header :-
    write('*** GHC System Vers. 1.9B on SICSTUS (1988-1-4)'),
    nl,
    write('***  by Takashi Chikayama and Shigeyuki Takagi, ICOT'),
    nl,
    write('*** based on DEC-10 Prolog Vers. 1.9 (1987-11-18)  '),
    write('by Kazunori Ueda, ICOT'),
    nl.

%%%% RUN-TIME SUPPORT %%%%

%%% TOP LEVEL %%%

:- public (ghc)/1, (ghc)/2, (ghc)/3.
ghc(Goals)             :- ghc(Goals, 100,   _).
ghc(Goals, Bound)      :- ghc(Goals, Bound, _).
ghc(_,     Bound, _) :- illegal_bound(Bound), !,
    display('Illegal bound value: '), display(Bound), ttynl, fail.
ghc(Goals, Bound, T) :-
    save_standard_io, reset_ioflag,
    solve(Goals, Bound), !,
    statistics(runtime,[_,T]), restore_standard_io,
  ( recorded('$GHCBACKTRACE',_,_), !, display(T), display(' msec.'), ttynl;
    true ).
ghc(_,     _,     _) :- /* failed in solving Goals */
    restore_standard_io,
    recorded('$GHCBACKTRACE',_,_), !,
  ( recorded('$GHCBACKTRACE_END',_,Ref), !, erase(Ref);
    recorded('$GHCUNIFYFAIL',    _,Ref), !, erase(Ref);
    display('Body unification failure or call of an undefined predicate.'),
    ttynl ),
    fail.
```

```
:- mode illegal_bound(?).
illegal_bound(Bound) :- integer(Bound), Bound>0, !, fail.
illegal_bound(_    ).

:- mode solve(?, +).
solve(Goals,  _    ) :- var(Goals), !, fail.
solve(Goals?, Bound) :- !,
    prepare_goal_queue(Goals, Bound, Ch,Ct, Nextid),
    statistics(runtime,_),
    incore('$ENDSPY'(0, Ch,Ct, nd, Nextid)).
solve(Goals,  Bound) :- /* not_functor(Goals, ?, _), !, */
    prepare_goal_queue(Goals, Bound, Ch,Ct, Nextid),
    statistics(runtime,_),
    incore('$END'(Ch,Ct, nd, Nextid)).

:- mode prepare_goal_queue(?, +, -,-, -).
prepare_goal_queue(Goals, Bound, Ch,Ct, Nextid) :-
    put_queue(          % The definition of 'put_queue' is in the compiler.
        Goals,          % Conjunctive goals to be put in the queue
        Ch,Ct,          % The queue (d-list) of goals
        Bound,Bound,    % Current and initial values of the reduction counter
        Id_calc,true,   % D-conjunction of arithmetic goals to calculate the
                        % IDs of Goals to be displayed on tracing
        1,              % Initial value of the goal IDs
        0,Id1,          % Initial and final values of the offsets of goal IDs
        '$TOP',         % Predicate that called Goals
        [],_,           % Old and new predicate dictionaries (for compiling
                        % ':-' goals that are called not at the top level)
        debug           % Debugging mode (debug/nodebug)
        ),
    Nextid is 1+Id1,    % Compute the ID of the next goal to be generated
    call(Id_calc), !.   % Provide the top-level goals with their IDs

%%% UNIFICATION %%%

% Unification in clause bodies:
:- mode ubody(?,?).
ubody(X,X) :- !.
ubody(X,Y) :- /* X and Y are ununifiable */
    recorded('$GHCBACKTRACE',_,_),
    recorda('$GHCUNIFYFAIL',_,_),
    display('You tried to unify '), writeuser(X), display(' with '),
    writeuser(Y), ttynl, fail.

:- mode 'ubody?'(?,?, +).
'ubody?'(X,Y, Myid) :- ghcspied('=',2), trace_call(Myid,X=Y), fail.
'ubody?'(X,Y, _   ) :- ubody(X,Y).

% Ununifiability
% ghcdif(X,Y) succeeds if and when X and Y proved to be ununifiable.
:- mode ghcdif(?,?).
ghcdif(X,_) :- var(X), !, fail.
ghcdif(_,Y) :- var(Y), !, fail.
% Here, 1st & 2nd args are non-variables.
ghcdif(X,Y) :- functor(X,F,A), functor(Y,F,A), !, ghcdif_args(A,X,Y).
ghcdif(_,_).  % succeeds when the function symbols or the arities mismatch

:- mode ghcdif_args(+, +,+).
ghcdif_args(0, _,_) :- !, fail.
ghcdif_args(N, X,Y) :- /* N>0, */
    arg(N,X,Xn), arg(N,Y,Yn), ghcdif(Xn,Yn), !.
ghcdif_args(N, X,Y) :- /* N>0, not yet dif(Xn,Yn) */
    N1 is N-1, ghcdif_args(N1, X,Y).

%%% SYSTEM PREDICATES %%%
```

```
% System predicates are executed even if the reduction counter indicates 0.

%% Input and Output %%
% Instream and outstream can be called only once for each.

:- public instream/8.
:- mode instream(?, +,+,-,+,+,+,+).
instream(S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    '$START_IO'(instream, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- public outstream/8.
:- mode outstream(?, +,+,-,+,+,+,+).
outstream(S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    '$START_IO'(outstream, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- public '$START_IO'/9.
:- mode '$START_IO'(+, ?, +,+,-,+,+,+,+).
'$START_IO'(IO, _, _, _, _, _, _, _, _) :-
    recorded('$GHCIOUSING',IO,_), !,
    display('Error: Instream/outstream cannot be called twice.'),
    ttynl, restore_standard_io, abort.
'$START_IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    /* not_recorded('$GHCIOUSING',IO,_), !, */
    recorda('$GHCIOUSING',IO,_),
    '$IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- public '$IO'/9.
:- mode '$IO'(+, ?, +,+,-,+,+,+,+).
'$IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    ghcspied(IO,1),
    functor(G,IO,1), arg(1,G,S), trace_call(Myid, G), !,
    '$IO?'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).
'$IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    '$IO!'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- mode '$IO?'(+, ?, +,+,-,+,+,+,+).
'$IO?'(IO, S, RC,Ch,Ct,_, RCmax,_, Nextid) :-
    nonvar(S), S=[H|T], nonvar(H), do_io(H,IO), !,
    Nextid1 is Nextid+1,
    '$IO'(IO, T, RC,Ch,Ct,nd,RCmax,Nextid,Nextid1).
'$IO?'(_, S, _ ,Ch,Ct,_, _, _, Nextid) :-
    S==[], !, do_next(Ch,Ct, Nextid).
'$IO?'(IO, S, RC,[$(Gx, Ch,Ct,Dnd,Nextid)|Ch],
                    [$('$IO'(IO,S, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                        Ch2,Ct2,Dnd2,          Nextid2)
                    |Ct],
                       Dnd,RCmax,Myid,Nextid) :-
    trace_suspension(RC, Myid), !, incore(Gx).

:- public '$IO!'/9.
:- mode '$IO!'(+, ?, +,+,-,+,+,+,+).
'$IO!'(IO, S, RC,Ch,Ct,_, RCmax,_, Nextid) :-
    nonvar(S), S=[H|T], nonvar(H), do_io(H,IO), !,
    Nextid1 is Nextid+1,
    '$IO!'(IO, T, RC,Ch,Ct,nd,RCmax,Nextid,Nextid1).
'$IO!'(_, S, _ ,Ch,Ct,_, _, _, Nextid) :-
    S==[], !, do_next(Ch,Ct, Nextid).
'$IO!'(IO, S, _, [$(Gx, Ch,Ct,Dnd,Nextid)|Ch],
                    [$('$IO'(IO,S, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                        Ch2,Ct2,Dnd2,          Nextid2)
                    |Ct],
                       Dnd,RCmax,Myid,Nextid) :- incore(Gx).

:- mode do_io(+,+).
do_io(see(F),    instream) :- !, nonvar(F), see(F).
do_io(seeing(F), instream) :- !, seeing(F).
```

```
do_io(seen,       instream) :- !, seen.
do_io(tell(F),    _       ) :- !, nonvar(F), tell(F).
do_io(telling(F),_        ) :- !, telling(F).
do_io(told,       _       ) :- !, told.

do_io(close(F),    _      ) :- !, nonvar(F), my_close(F).
do_io(fileerrors,  _      ) :- !, fileerrors.
do_io(nofileerrors,_      ) :- !, nofileerrors.

do_io(read(X),    instream) :- !, my_read(X).
do_io(write(X),   _       ) :- !, ground(X), write(X).
do_io(writeq(X),  _       ) :- !, ground(X), writeq(X).
do_io(print(X),   _       ) :- !, ground(X), print(X).
do_io(nl,         _       ) :- !, nl.
do_io(get0(C),    instream) :- !, my_get0(C).
do_io(get(C),     instream) :- !, my_get(C).
do_io(skip(C),    instream) :- !, ground(C), skip(C).
do_io(put(C),     _       ) :- !, ground(C), put(C).
do_io(tab(N),     _       ) :- !, ground(N), tab(N).

do_io(display(X),_        ) :- !, ground(X), display(X).
do_io(ttynl,      _       ) :- !, ttynl.
do_io(ttyflush,   _       ) :- !, ttyflush.
do_io(ttyget0(C),instream) :- !, my_ttyget0(C).
do_io(ttyget(C), instream) :- !, my_ttyget(C).
do_io(ttyskip(C),instream) :- !, ground(C), ttyskip(C).
do_io(ttyput(C), _       ) :- !, ground(C), ttyput(C).
do_io(ttytab(N), _       ) :- !, ground(N), ttytab(N).

do_io(prompt(O,N),instream):- !, atom(N), prompt(O,N).
do_io(G,          _       ) :-
    display('Error: Illegal request in instream/outstream:  '),
    writeuser(G), ttynl, restore_standard_io, abort.

% If nofileerrors has been executed, input goals can be called arbitrarily
% many times.
:- mode my_get0(?), my_ttyget0(?), my_get(?), my_ttyget(?), my_read(?).
my_get0(C )     :- get0(C), !.
my_get0(26).
my_ttyget0(C ) :- ttyget0(C), !.
my_ttyget0(26).
my_get(C )      :- get(C), !.
my_get(26).
my_ttyget(C )  :- ttyget(C), !.
my_ttyget(26).
my_read(X)      :- read(X), !, numbervars(X, 0,_).     % An input term is
                                                       % made ground.

% the following is for discarding an alternative created by the system
% predicate close/1 when the file specified by its argument is currently
% open.
:- mode my_close(+).
my_close(F) :- close(F), !.


%% Other System Predicates Callable from Bodies %%

% The following routine is used only when ':='/2 is called directly from
% the top level; otherwise ':='/2 is compiled into a spcialized predicate.
:- public (:=)/9.
:- mode :=(?,?, +,+,-,+,+,+,+).
:=(X,Y, _,  _, _, _,  _,    Myid,_     ) :-
    ghcspied(':=',2), trace_call(Myid, X:=Y), fail.
:=(X,Y, _,  Ch,Ct,_,  _,    _,   Nextid) :-
    numbervars(Y,0,0), call(X2 is Y), !,
    ubody(X2,X), do_next(Ch,Ct,Nextid).
```

```
:-(_,_, RC0,_, _, _,  _,    Myid,_     ) :-
    ghcspied(':-',2), trace_suspension(RC0,Myid), fail.
:-(X,Y, _,  [$(Gx, Ch,Ct,Dnd,Nextid) | Ch],
              [$(:-(X,Y, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                Ch2,Ct2,Dnd2,          Nextid2) | Ct],
                Dnd,RCmax,Myid,Nextid) :- incore(Gx).

% The following routine is used only when '='/2 is called directly from
% the top level; otherwise '='/2 is compiled into 'ubody/2'.
:- public (=)/9.
:- mode =(?,?, +,+,-,+,+,+,+).
=(X,Y, _,_, _, _,_,Myid,_     ) :-
    ghcspied('=',2), trace_call(Myid, X=Y), fail.
=(X,Y, _,Ch,Ct,_,_,_,   Nextid) :- ubody(X,Y), do_next(Ch,Ct, Nextid).

:- public prolog/8.
:- mode prolog(?, +,+,-,+,+,+,+).
prolog(X, _,  _, _, _,  _,    Myid,_     ) :-
    ghcspied(prolog,1), trace_call(Myid, prolog(X)), fail.
prolog(X, _,  Ch,Ct,_,  _,    _,   Nextid) :-  call(X), !,
    do_next(Ch,Ct,Nextid).
prolog(_, RC0,_, _, _,  _,    Myid,_     ) :-
    ghcspied(prolog,1), trace_suspension(RC0, Myid), fail.
prolog(X, _,  [$(Gx, Ch,Ct,Dnd,Nextid) | Ch],
                [$(prolog(X, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                Ch2,Ct2,Dnd2,          Nextid2) | Ct],
                  Dnd,RCmax,Myid,Nextid) :- incore(Gx).

%% Calling the Next Goal in the Queue %%

:- mode do_next(+,-, +).
do_next([$(Gx, Ch2,Ct,nd,Nextid)|Ch2],Ct, Nextid) :-  incore(Gx).

%%% CYCLE MARKERS %%%

%%  Standard Version %%

:- public '$END'/4.
:- mode '$END'(?,-,+,+).
'$END'([],_, _, _     ) :- !. % succeeds if the queue is empty, i.e.,
                              % the first arg is UNINSTANTIATED.
'$END'([$(Gx, Ch,Ct,d,Nextid)|Ch],
          [$('$END'(Ch2,Ct2,Dnd2,Nextid2),Ch2,Ct2,Dnd2,Nextid2)|Ct],
             nd,Nextid) :- !,
    incore(Gx).               % otherwise, if some goals have been reduced
                              % in the last 'cycle' (i.e., the deadlock
                              % flag is 'nd'), then try  another 'cycle'.
'$END'(Gs,_, _, _     ) :- backtrace(Gs), fail.

%% Cycle Marker with Spy Facilities %%

:- public '$ENDSPY'/5.
:- mode '$ENDSPY'(+, ?,-,+,+).
'$ENDSPY'(Endid, Ch,Ct,Dnd,Nextid) :-
    display('Cycle '), display(Endid),
    display('; function (h for help)? '), ttyflush, ttyget0(C),
  ( C=\=31/*newline*/, !, ttyskip(31); true ),
    endspy(C, Endid, Ch,Ct,Dnd,Nextid).

:- mode endspy(+, +, ?,-,+,+).
endspy(31 /*NL*/, Endid, Ch,Ct,Dnd,Nextid) :- !,          % CONTINUE
    endspy2(Endid, Ch,Ct,Dnd,Nextid).
endspy(119/*w*/,  Endid, Ch,Ct,Dnd,Nextid) :- !,          % WRITE QUEUE
    display_queue(Ch), '$ENDSPY'(Endid, Ch,Ct,Dnd,Nextid).
endspy(110/*n*/,  _    , Ch,Ct,Dnd,Nextid) :- !,          % NODEBUG MODE
    '$END'(Ch,Ct,Dnd,Nextid).
```

```
endspy(104/*h*/,  Endid, Ch,Ct,Dnd,Nextid) :- !,          % HELP
    display('<cr>: continue'), ttynl,
    display('   w: Write goals in the queue'), ttynl,
    display('   n: continue in Nodebug mode'), ttynl,
    display('   h: Help'), ttynl,
    display('   a: Abort current execution'), ttynl,
    display('   @: Accept a Prolog goal'), ttynl,
    '$ENDSPY'(Endid,Ch,Ct,Dnd,Nextid).
endspy(97 /*a*/,  _,      _, _, _,  _      ) :- !,          % ABORT
    restore_standard_io, abort.
endspy(64 /*@*/,  Endid, Ch,Ct,Dnd,Nextid) :- !,          % ACCEPT COMMAND
    display('| :- '), ttyflush, seeing(F), see(user), read(G), see(F),
  ( call(G), !, display('yes'); display('no') ), ttynl,
    '$ENDSPY'(Endid, Ch,Ct,Dnd,Nextid).
endspy(_,         Endid, Ch,Ct,Dnd,Nextid) :- /* !, */ % Other chars are
    endspy(104, Endid, Ch,Ct,Dnd,Nextid).              % interpreted as 'h'

:- mode endspy2(+, ?,-,+,+).
endspy2(_,      [],_, _, _      ) :- !.
endspy2(Endid, [$(Gx, Ch,Ct,d,Nextid)|Ch],
                  [$('$ENDSPY'(Endid2, Ch2,Ct2,Dnd2,Nextid2),
                                       Ch2,Ct2,Dnd2,Nextid2)|Ct],
                     nd,Nextid) :- Endid2 is Endid+1, !, incore(Gx).
endspy2(_,      Gs,_, _, _      ) :- backtrace(Gs), fail.

%%% TRACING AND BACKTRACING %%%

:- public
    (ghcspy)/0, (ghcspy)/1, (ghcnospy)/0, (ghcnospy)/1, ghcspying/0,
    ghcbacktrace/0, ghcnobacktrace/0.

%% Tracing %%

% Setting, Resetting and Showing Spy Points

ghcspy :-
  ( recorded('$GHCSPY',X,_), var(X), !,
      display('Spy points have already been set on all goals.'), ttynl;
    ( erasel, fail; recorda('$GHCSPY',_,_) ) ).
:- mode ghcspy(?).
ghcspy(X) :- atom(X), !,
  ( recorded('$GHCSPY',X/V,_), var(V), !, complain_double_spy(X);
    ( erasel(X), fail; recorda('$GHCSPY',X/_,_) ) ).
ghcspy(X) :- nonvar(X), X=P/A, atom(P), integer(A), !,
  ( recorded('$GHCSPY',X,_), !, complain_double_spy(X);
    recorda('$GHCSPY',X,_) ).
ghcspy(X) :- nonvar(X), X=(X1,X2), !, ghcspy(X1), ghcspy(X2).
ghcspy(X) :- complain_predicate_spec(X).

ghcnospy :-
  ( erasel, !, ( erasel, fail; true ); complain_nothing_to_reset ).
:- mode ghcnospy(?).
ghcnospy(X) :- atom(X), !,
  ( erasel(X), !, ( erasel(X), fail; true );
    recorded('$GHCSPY',Y,_), var(Y), !, complain_too_specific(X);
    complain_nothing_to_reset(X) ).
ghcnospy(X) :- nonvar(X), X=P/A, atom(P), integer(A), !,
  ( erasel(P,A), !;
    recorded('$GHCSPY',Y,_), Y=P/V, var(V), !,
      complain_too_specific(X);
    complain_nothing_to_reset(X) ).
ghcnospy(X) :- nonvar(X), X=(X1,X2), !, ghcnospy(X1), ghcnospy(X2).
ghcnospy(X) :- complain_predicate_spec(X).

erasel :- recorded('$GHCSPY',_,R), erase(R).
:- mode erasel(+).
```

```
erasel(P) :- recorded('$GHCSPY',X,R), nonvar(X), X=P/_,  erase(R).
:- mode erasel(+,+).
erasel(P,A) :- recorded('$GHCSPY',X,R), nonvar(X), X=P/B, A==B, erase(R).

:- mode complain_double_spy(+).
complain_double_spy(X) :-
    display('Spy point on '), writeuser(X),
    display(' has already been set.'), ttynl.
:- mode complain_predicate_spec(?).
complain_predicate_spec(X) :-
    display('Illegal predicate specification:  '), writeuser(X), ttynl.
complain_nothing_to_reset :-
    display('No GHC goals are spied.'), ttynl.
:- mode complain_nothing_to_reset(+).
complain_nothing_to_reset(X) :-
    display('Spy point has not been set on '),
    writeuser(X), display('.'), ttynl.
:- mode complain_too_specific(+).
complain_too_specific(X) :-
    display('Spy point cannot be reset selectively on '),
    writeuser(X), display('.'), ttynl.

ghcspying :- setof($(X), R^recorded('$GHCSPY', X, R), S), !,
  ( S=[$(Y)|_], var(Y), !, display('All GHC goals are spied.');
    display('GHC spy-points have been set on '), write_spied(S) ), ttynl.
ghcspying :- display('No GHC goals are spied.'), ttynl.

:- mode write_spied(+).
write_spied([$(P/A)|Ps]) :- display(P),
  ( nonvar(A), !, ttyput("/"), display(A); true ),
  ( Ps\==[], !, display(', '), write_spied(Ps); ttyput(".") ).

% Runtime Support for Tracing

:- mode ghcspied(+,+).
ghcspied(P,A) :-  recorded('$GHCSPY',P/A,_), !.

:- mode trace_call(+, +).
trace_call(Id, G) :-
    numbervars(G,0,_), display_id(Id), display(': Call '),
    writeuser(G), ttynl, fail.
trace_call(_,  _).

:- mode display_id(+).
display_id(0 ) :- !.
display_id(Id) :- /* Id>0, !, */ display(Id).

:- mode trace_suspension(+, +).
trace_suspension(0, Id) :- !,
    display_id(Id), display(': Swapped out'), ttynl.
trace_suspension(N, Id) :- /* N>0, !, */
    display_id(Id), display(': Suspended'), ttynl.

%% Backtracing %%

% Setting and Resetting
ghcbacktrace   :- recorda('$GHCBACKTRACE',_,_).
ghcnobacktrace :- recorded('$GHCBACKTRACE',_,R), erase(R), fail.
ghcnobacktrace.

% Runtime Support
:- mode backtrace(+).
backtrace(Gs) :-
    recorded('$GHCBACKTRACE',_,_),
    display('Deadlock detected.  Suspended goals are:'), ttynl,
    display_queue(Gs), recorda('$GHCBACKTRACE_END',_,_).
```

```
%% Runtime Support for Displaying the Goal Queue %%

:- mode display_queue(+).
display_queue(Gs) :- numbervars(Gs,0,_), dq2(Gs), fail.
display_queue(_ ).

:- mode dq2(+).
dq2([G|Gs]) :- !, display_goal(G), dq2(Gs).
dq2(_).

:- mode display_goal(+).
display_goal($(Gx, _,_,_,_)) :-
    functor(Gx,P,A7), A is A7-7, functor(G,P,A),
    copy_args(A, Gx,G),
    A6 is A+6, arg(A6,Gx,Id), display_id(Id),
    display(': '), writeuser(G), ttynl.

%%% MISCELLANEOUS %%%

:- mode ground(?).
ground(X) :- var(X), !, fail.
ground(X) :- /* !, */ functor(X,_,Arity), ground_args(Arity, X).

:- mode ground_args(+, +).
ground_args(0, _) :- !.
ground_args(N, X) :- /* !, */
    arg(N,X,T), ground(T), N1 is N-1, ground_args(N1, X).

reset_ioflag :- recorded('$GHCIOUSING',_,R), erase(R), fail.
reset_ioflag.

save_standard_io :-
    seeing(In),   recorda('$GHCSTDIN', In, _),
    telling(Out), recorda('$GHCSTDOUT',Out,_).

restore standard io :- fileerrors,
    recorded('$GHC$TDIN', In, Ref1), erase(Ref1), see(In),
    recorded('$GHCSTDOUT',Out,Ref2), erase(Ref2), tell(Out).
```

```
%%%% COMPILATION %%%%

%%% READING PROGRAMS %%%

:- public ghccompile/1, ghccompile/2, ghcdcompile/1, ghcdcompile/2.
:- mode ghccompile(?), ghccompile(?,?), ghcdcompile(?), ghcdcompile(?,?).
ghccompile(S) :- /* !, */
    plsys(mktemp('/tmp/ghcXXXXXX', TempFile)),
    ghccompile(S, TempFile, nodepth), compile(TempFile).
ghccompile(S,O) :- /* !, */ ghccompile(S,O, nodepth).
ghcdcompile(S) :- /* !, */
    plsys(mktemp('/tmp/ghcXXXXXX', TempFile)),
    ghccompile(S, TempFile, depth),   compile(TempFile).
ghcdcompile(S,O) :- /* !, */ ghccompile(S,O, depth).

ghccompile(S,O, Depth) :-
    telling(Old), nofileerrors, tell(O), !,
    ghccomp(S, [],Dict, Depth),
    write_last_clause_group(Dict, Depth),
    told, tell(Old), display('END.'), ttynl.
ghccompile(_,O, _    ) :- /* failed to open O */
    complain_unable_to_open(O).

% ghccomp(Files, Current_dict,New_dict, Depth_first?).
% Each member of the predicate dictionary has the form
% $$(Functor/Arity, Name_of_internal_subpredicate,
%    Mode_decl, Debug_mode, #_of_body_assignment+"@").
:- mode ghccomp(?, +,-, +).
ghccomp(F,     Dict0,Dict1, _    ) :- var(F), !,
    Dict0=Dict1, complain_unable_to_open(F).
ghccomp(F,     Dict0,Dict1, Depth) :-
    atom(F), seeing(Old), nofileerrors, see(F), !,
    read(X), ghccomp1(X, Dict0,Dict1, nodebug, Depth), seen, see(Old).
ghccomp(F?,    Dict0,Dict1, Depth) :-
    atom(F), seeing(Old), nofileerrors, see(F), !,
    read(X), ghccomp1(X, Dict0,Dict1, debug,   Depth), seen, see(Old).
ghccomp([H|T], Dict0,Dict2, Depth) :- !,
    ghccomp(H, Dict0,Dict1, Depth), ghccomp(T, Dict1,Dict2, Depth).
ghccomp([],    Dict0,Dict1, _    ) :- !, Dict0=Dict1.
ghccomp(F,     Dict0,Dict1, _    ) :- /* failed to open F */
    Dict0=Dict1, complain_unable_to_open(F).

:- mode ghccomp1(?, +,-, +, +).
ghccomp1(X,               Dict0,Dict1, Debug, Depth) :- var(X), !,
    display('Uninstantiated clause found'), ttynl,
    read(Next), ghccomp1(Next, Dict0,Dict1, Debug, Depth).
ghccomp1(end_of_file,     Dict0,Dict1, _,     _    ) :- !, Dict0=Dict1.
ghccomp1((:- mode Mdecl), Dict0,Dict2, Debug, Depth) :- nonvar(Mdecl), !,
    functor(Mdecl, P, A),
  ( lookup_dict(P/A, Dict0, _,_,_), !,
      errmsg('Duplicate or misplaced mode declaration: ', Mdecl),
      Dict1=Dict0;
    notify_compilation(P,A), write_mode_decl(P,A),
  ( check_mode(Mdecl), !,
      write_transmit_clause(Mdecl), make_internal_name(P,IP),
      write_mode_decl(IP,A);
    IP=P ),
    write_callmsg_clause(IP,A,P, Debug),
    Dict1=[$$(P/A, IP,Mdecl,Debug,64/*@*/)|Dict0] ),
    read(Next), ghccomp1(Next, Dict1,Dict2, Debug, Depth).
ghccomp1((:- op(Priority,Type,Op)), Dict0,Dict1, Debug, Depth) :-
    op(Priority,Type,Op), !,                  % Operator decl must be exe-
    write_clause((:- op(Priority,Type,Op))), % cuted and also copied.
    read(Next), ghccomp1(Next, Dict0,Dict1, Debug, Depth).
ghccomp1((:- X),      Dict0,Dict1, Debug, Depth) :-
 /* not_functor(X, mode, 1), not_functor(X, op, 3) */ !,
```

```
    errmsg('Illegal directive: ', (:- X)),
    read(Next), ghccompl(Next, Dict0,Dict1, Debug, Depth).
ghccompl(X,           Dict0,Dict3, Debug, Depth) :-
 /* clause_with_positive_atom(X), !, */
  ( X=(Head:-_), !; X=Head ),
    functor(Head,P,A),
  ( lookup_dict(P/A, Dict0, _,_,_), !, Dict1=Dict0;
    notify_compilation(P,A),
    write_mode_decl(P,A), write_callmsg_clause(P,A,P, Debug),
    Dict1=[$$(P/A, P,'$NOMODE',Debug,64/*@*/)|Dict0] ),
  ( c_clause(X, C, Dict1,Dict2, Debug, Depth), !, write_clause(C);
    errmsg('Cannot compile: ', X), Dict2=Dict1 ),
    read(Next), ghccompl(Next, Dict2,Dict3, Debug, Depth).

:- mode complain_unable_to_open(?).
complain_unable_to_open(F) :- /* !, */
    fileerrors, display('Cannot open file: '), writeuser(F), ttynl.

%%% WRITING OBJECT CLAUSES %%%

:- mode write_clause(?).
write_clause(X) :-
    make_writable(X,Y, 0,_), writeq(Y), put("."), nl, fail.
write_clause(_).

:- mode make_writable(?,-, +,-).
make_writable(X,          Y, V0,V1) :- var(X), !,
    X='$VAR'(V0), Y=X, V1 is V0+1.
make_writable('$VAR'(X), Y, V0,V1) :- !, Y='$VAR'(X), V0=V1.
                              % Variable already made ground ('numbervar'ed)
make_writable('$REF'(X), Y, V0,V1) :- !, make_writable(X,Y, V0,V1).
make_writable('$WAIT'(X),Y, V0,V1) :- !, make_writable(X,Y, V0,V1).
make_writable('$INT'(X), Y, V0,V1) :- !, make_writable(X,Y, V0,V1).
make_writable(X,          Y, V0,V1) :- /* nonvariable(X), !, */
    functor(X,F,A), functor(Y,F,A), make_writable_args(0,A, X,Y, V0,V1).

:- mode make_writable_args(+,+, +,+, +,-).
make_writable_args(K,N, X,Y, V0,V2) :- K<N, !,
    K1 is K+1, arg(K1,X,Xk), make_writable(Xk,Yk, V0,V1), arg(K1,Y,Yk),
    make_writable_args(K1,N, X,Y, V1,V2).
make_writable_args(N,N, _,_, V0,V1) :- /* !, */ V0=V1.

%%% MAKING ADDITIONAL CLAUSES %%%

:- mode write_mode_decl(+,+).
write_mode_decl(P,A) :- /* !, */
    A7 is A+7, write_clause((:- public P/A7)),
    functor(Mode,P,A), fill_questions(A, Mode),
    extend_atom(Mode,Modex, +,+,-,+,+,+,+), write_clause((:- mode Modex)).

:- mode fill_questions(+, +).
fill_questions(0, _) :- !.
fill_questions(K, X) :- /* K>0, !, */
    arg(K,X,?), K1 is K-1, fill_questions(K1, X).

:- mode write_callmsg_clause(+,+,+, +).
write_callmsg_clause(P,A,Pspy, debug) :- !,
    functor(H,P,A), extend_atom(H,Headx, _,_,_,_,_,Myid,_),
    functor(Hspy,Pspy,A), copy_args(A, H,Hspy),
    write_clause((Headx :- ghcspied(Pspy,A), trace_call(Myid,Hspy), fail)).
write_callmsg_clause(_,_,_,     _     ) /* :- ! */.

:- mode write_last_clause_group(+, +).
write_last_clause_group([$$(P/A, P, _,Debug,_)|L], Depth) :- !,
    write_suspmsg_clause(P, A, P, Debug, Depth),
    write_susp_clause(P,A),
```

```
    write_last_clause_group(L, Depth).
write_last_clause_group([$$(P/A, IP,_,Debug,_)|L], Depth) :- /*P\==IP,*/ !,
    write_callsuspmsg_clause(P,A,P, Debug, Depth),
    write_susp_clause(P,A),
    write_suspmsg_clause(IP,A,P, Debug, Depth),
    write_susp_clause(IP,A),
    write_last_clause_group(L, Depth).
write_last_clause_group([],                          _    ) /* :- ! */.

:- mode write_suspmsg_clause(+,+,+, +, +).
write_suspmsg_clause(P,A,Pspy, debug, Depth) :- !,
    functor(H,P,A), extend_atom(H,Headx, RC0,_,_,_,_,Myid,_),
  ( Depth==nodepth, !, RC=RC0; RC=1 ),
    write_clause((Headx :- ghcspied(Pspy,A),
                           trace_suspension(RC,Myid), fail)).
write_suspmsg_clause(_,_,_,    _,     _    ) /* :- ! */.

:- mode write_callsuspmsg_clause(+,+,+, +, +).
write_callsuspmsg_clause(P,A,Pspy, debug, Depth) :- !,
    functor(H,P,A), extend_atom(H,Headx, RC0,_,_,_,_,Myid,_),
    functor(Hspy,Pspy,A), copy_args(A, H,Hspy),
  ( Depth==nodepth, !, RC=RC0; RC=1 ),
    write_clause((Headx :- ghcspied(Pspy,A), trace_call(Myid,Hspy),
                           trace_suspension(RC,Myid), fail)).
write_callsuspmsg_clause(_,_,_,    _,     _    ) /* :- ! */.

:- mode write_susp_clause(+,+).
write_susp_clause(P,A) :- /* !, */
    functor(H,P,A),
    extend_atom(H,Goalx, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
    extend_atom(H,Headx, _,    [$(Gx, Ch,Ct,Dnd,Nextid)|Ch],
                                   [$(Goalx, Ch2,Ct2,Dnd2,Nextid2)|Ct],
                                       Dnd, RCmax,Myid,Nextid ),
    write_clause((Headx :- incore(Gx))).

:- mode write_transmit_clause(+).
write_transmit_clause(Mdecl) :- /* !, */
    functor(Mdecl,P,A), make_internal_name(P,IP), A7 is A+7,
    functor(Headx,P,A7), functor(Goalx,IP,A7),
    transmit_args(0,A7, Mdecl, Headx, Goalx,Body),
    write_clause((Headx :- Body)).

% transmit_args(Arg_position,Arity, Mode, Head, Subgoal,Body)
% Body = suquence of calls to 'nonvar/1' followed by Subgoal.
:- mode transmit_args(+,+, +, +, +,-).
transmit_args(K,N, Mdecl, Headx, Goalx,Body0) :- K<N, !,
    K1 is K+1, arg(K1,Headx,Xk1), arg(K1,Goalx,Xk1),
  ( arg(K1,Mdecl,+), !, Body0=(nonvar(Xk1), Body1);
    Body0=Body1 ),                    % Mode is not '+' or K1>arity(Mdecl)
    transmit_args(K1,N, Mdecl, Headx, Goalx,Body1).
transmit_args(N,N, _,     Headx, Goalx,Body ) :- /* !, */
    N6 is N-6, arg(N6,Headx,Xn6), Body=(Xn6>0, !, Goalx).

:- mode make_internal_name(+,-).
make_internal_name(P,IP) :- /* !, */
    name(P,Pl), name(IP,[36,36,36/*$$$*/|Pl]).

%%% COMPILING CLAUSES %%%

%% Decomposing and Composing Clauses %%

% c_clause(GHC_clause, Prolog_clause,
%          Current_dict, New_dict, Debug_mode?, Depth_first?).
% c_clause may fail if an uninstantiated atom appears.
:- mode c_clause(+, -, +,-, +, +).
c_clause((Head:-_),    _, _,    _,     _,     _    ) :- var(Head), !,
```

```
    fail.
c_clause((Head:-X),    C, Dict0,Dict1, Debug, Depth) :-
    nonvar(X), X=(Guard|Body), !,
    c_head_guard(Head, Guard, PH, PB0,(!,PB1), Dict0, RC0, Depth),
    extend_atom(PH,PHx, RC0,Ch,Ct,_,RCmax,_,Nextid),
    separate_unify(Body, Bnounify, Bunify),
    c_body_unify(Bunify, Nextid,0,Id1, PB1,PB2, Debug),
    c_body_arith(Bnounify,Bnounify_rest, PB2,PB3, Debug),
    put_queue(Bnounify_rest, Q0,Q1, RC1,RCmax,
              PB3,PB4, Nextid,Id1,Id2, Head, Dict0,Dict1, Debug),
    c_update_nextid(Debug, Nextid,Id2,Newid, PB4,PB5),
    c_make_tail(Depth, Q0,Q1, RC0,RC1, Ch,Ct, Newid, PB5),
    C=(PHx:-PB0).
c_clause((Head:-Body), C, Dict0,Dict1, Debug, Depth) :-
 /* not_functor(Body, '|', 2) */ !,
    errmsg('Warning: No commitment operator:  ', (Head :- Body)),
    c_clause((Head :- true|Body), C, Dict0,Dict1, Debug, Depth).
c_clause(Head,          C, Dict0,Dict1, Debug, Depth) :-
 /* not_functor(Head, ':-', 2) */ !,
    errmsg('Warning: No commitment operator:  ', Head),
    c_clause((Head :- true|true), C, Dict0,Dict1, Debug, Depth).

% separate_unify(Goals, Nounify, Unify).
% Nounify and Unify become LINEAR conjunction of goals.
:- mode separate_unify(?, -, -).
separate_unify(G, NU, U) :- separate_unify(G, NU,true, U,true).

:- mode separate_unify(?, -,?, -,?).
separate_unify(X,       _,  _,    _, _ ) :- variable(X), !, fail.
separate_unify((X,Y), NU0,NU2, U0,U2) :- !,
    separate_unify(X, NU0,NU1, U0,U1), separate_unify(Y, NU1,NU2, U1,U2).
separate_unify(X=Y,   NU0,NU1, U0,U1) :- !, NU0=NU1, U0=(X=Y,U1).
separate_unify(G,     NU0,NU1, U0,U1) :- /* not_unification(G), !, */
    NU0=(G,NU1), U0=U1.

%% Compiling Heads and Guards %%

% c_head_guard(GHC_head, GHC_guard,
%     Prolog_head, Prolog_body, Prolog_body_tail, Current_dict,
%     Current_reduction_count, Depth_first?)
:- mode c_head_guard(+, ?, -, -,+, +, -, +).
c_head_guard(Head, Guard, PH, PB0,PB4, Dict, RC, Depth) :-
    functor(Head,P,A), lookup_dict(P/A, Dict, IP,Mode,_),
    functor(PH,IP,A),
  ( separate_unify(Guard, Gnounify, Gunify), !,
    ( call(Gunify), !,
        c_args(0,A, Head, PH, PB0,PB1, Mode),
      ( c_guard(Gnounify, PB1,PB2, PB3,PB4), !,
        ( P==IP, Depth==nodepth, !,
            PB2=(RC>0,PB3);
          PB2=PB3 );
        errmsg('Guards can call predefined predicates only:  ', Guard),
        PB1=(fail,PB4) );
      errmsg('Warning: Unsucceedable guard:  ', Guard),
      PB0=(fail,PB4) );
    errmsg('Uninstantiated variables in guard:  ', Guard),
    PB0=(fail,PB4) ).

:- mode c_args(+,+, +, -, -,?, +).
c_args(K,N, Head, PH, PB0,PB2, M) :- K<N, !,
    K1 is K+1, arg(K1,Head,Ak), arg(K1,PH,PHk),
  ( M=='$NOMODE', !, Mk=(?);  arg(K1,M,Mk) ),
    c_unify(Ak, PHk, PB0,PB1, Mk), c_args(K1,N, Head, PH, PB1,PB2, M).
c_args(N,N, _,    _,  PB0,PB1, _) :- /* !, */ PB0=PB1.

% c_unify(Original_argument, Generated_argument,
```

```
%         Unify_code, Unify_code_tail, Mode).
% Note that f(X,X) must be compiled into f(X,Y) :- X==Y.
% This is because X and Y must be unified with no substitutions.
% For integer comparison, =:= and =\= are faster than = and \=.
:- mode c_unify(?, -, -,?, +).
c_unify(V,          X, G0,G1, M) :- var(V), !,
    X=V, G0=G1, V='$REF'(Y), mark(Y, M).
c_unify('$REF'(V), X, G0,G1, M) :- !, G0=(V==X,G1), mark(X, M).
c_unify([],         X, G0,G1, M) :- !,
  ( M=='+', !, X=[], G0=G1; G0=(X==[],G1) ).
c_unify(A,          X, G0,G1, M) :- atomic(A), !,
  ( M=='+', !, X=A, G0=G1; G0=(X==A,G1) ).
c_unify([H|T],      X, G0,G3, M) :- !,
  ( M=='+', !, X=[H0|T0], G0=G1; G0=(nonvar(X),X=[H0|T0],G1) ),
    c_unify(H, H0, G1,G2, ?), c_unify(T, T0, G2,G3, ?).
c_unify(S,          X, G0,G2, M) :- /* other_structure(S), !, */
    functor(S,F,A), functor(S0,F,A),
  ( M=='+', !, X=S0, G0=G1; G0=(nonvar(X),X=S0,G1) ),
    c_unify_args(0,A, S, S0, G1,G2).

:- mode c_unify_args(+,+, +, +, -,?).
c_unify_args(K,N, S, X, G0,G2) :- K<N, !,
    K1 is K+1, arg(K1,S,Sk), arg(K1,X,Xk),
    c_unify(Sk, Xk, G0,G1, ?), c_unify_args(K1,N, S, X, G1,G2).
c_unify_args(N,N, _, _, G0,G1) :- /* !, */ G0=G1.

% c_guard(Original_goals, Before_goals, Before_goals_tail,
%         After_goals, After_goals_tail)
% Before_goals are called before checking the reduction counter, while
% After_goals are not.
:- mode c_guard(+, -,?, -,?).
c_guard(true,       Gb0,Gb1, Ga0,Ga1) :- !, Gb0=Gb1, Ga0=Ga1.
c_guard((X,Y),      Gb0,Gb2, Ga0,Ga2) :- !,
    c_guard(X, Gb0,Gb1, Ga0,Ga1), c_guard(Y, Gb1,Gb2, Ga1,Ga2).
c_guard(prolog(X), Gb0,Gb1, Ga0,Ga1) :- !, Gb0=Gb1, Ga0=(X,Ga1).
                    % prolog(X) must be executed after checking the
                    % reduction counter, since it may cause side effects.
c_guard(X,          Gb0,Gb1, Ga0,Ga1) :- /* other_form(X), !, */
    c_guard_2(X, Gb0,Gb1), Ga0=Ga1.

% c_guard_2 fails if the first arg is not a system predicate.
:- mode c_guard_2(+, -,?).
c_guard_2(wait(X), Gb0,Gb1) :- !, c_w(X, Gb0,Gb1).
c_guard_2(X\=Y,    Gb0,Gb1) :- atomic(X), !, c_w(Y, Gb0,(X\==Y,Gb1)).
c_guard_2(X\=Y,    Gb0,Gb1) :- atomic(Y), !, c_w(X, Gb0,(X\==Y,Gb1)).
c_guard_2(X\=Y,    Gb0,Gb1) :- /* non_atomic(X), non_atomic(Y) */ !,
    Gb0=(ghcdif(X,Y),Gb1).
c_guard_2(X:=Y,    Gb0,Gb2) :- appeared_in_head(X), !,
    c_iw(X, Gb0,Gb1), c_iw_all(Y, Gb1,(X is Y,Gb2)).
c_guard_2(X:=Y,    Gb0,Gb1) :- /* not_appeared_in_head(X) */ !,
    c_iw_all(Y, Gb0,(X is Y,Gb1)).
c_guard_2(X<Y,     Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X<Y,  Gb1)).
c_guard_2(X>Y,     Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X>Y,  Gb1)).
c_guard_2(X=<Y,    Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X=<Y, Gb1)).
c_guard_2(X>=Y,    Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X>=Y, Gb1)).
c_guard_2(X=:=Y,   Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X=:=Y,Gb1)).
c_guard_2(X=\=Y,   Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X=\=Y,Gb1)).

:- mode c_w(?, -,?).
c_w(X,          G0,G1) :- var(X), !, G0=(nonvar(X),G1), X='$WAIT'(_).
c_w('$REF'(X), G0,G1) :- !, c_w(X, G0,G1).
c_w(X,          G0,G1) :-
 /* ( X='$WAIT'(_); X='$INT'(_); not_variable(X) ), !, */ G0=G1.

:- mode c_iw(?, -,?).
c_iw(X,           G0,G1) :- var(X), !, G0=(integer(X),G1), X='$INT'(_).
```

```
c_iw('$REF'(X),  G0,G1) :- !, c_iw(X, G0,G1).
c_iw('$WAIT'(X), G0,G1) :- !, c_iw(X, G0,G1).
c_iw(_,          G0,G1) :- /* ( X='$INT'(_); not_variable(X) ), ! */ G0=G1.

:- mode c_iw_2(?,?, -,?).
c_iw_2(X,Y, G0,G2) :- /* !, */ c_iw_all(X, G0,G1), c_iw_all(Y, G1,G2).

:- mode c_iw_all(?, -,?).
c_iw_all(X,          G0,G1) :- var(X), !, G0=(integer(X),G1), X='$INT'(_).
c_iw_all('$REF'(X),  G0,G1) :- !, c_iw_all(X, G0,G1).
c_iw_all('$WAIT'(X), G0,G1) :- !, c_iw_all(X, G0,G1).
c_iw_all('$INT'(_),  G0,G1) :- !, G0=G1.
c_iw_all(X,          G0,G1) :- /* structure(X), !, */
    functor(X,_,A), c_iw_args(0,A, X, G0,G1).

:- mode c_iw_args(+,+, ?, -,?).
c_iw_args(K,N, X, G0,G2) :- K<N, !,
    K1 is K+1, arg(K1,X,Xk), c_iw_all(Xk, G0,G1),
    c_iw_args(K1,N, X, G1,G2).
c_iw_args(N,N, _, G0,G1) :- /* !, */ G0=G1.

%% Compiling Bodies %%

:- mode c_body_unify(+, -,+,-, -,?, +).
c_body_unify((X=Y,Bunify), Id,Id0,Id2, PB0,PB3, Debug) :- !,
    c_id_setup(Debug, PB0,PB1, Id,Id0,Id1, Myid),
  ( variable(X), !, R=X, L=Y;  % for efficiency on DEC-10 Prolog
    L=X, R=Y ),
  ( Debug==nodebug, !, PB1=(L=R, PB2);
    PB1=('ubody?'(L,R,Myid), PB2) ),
    c_body_unify(Bunify, Id,Id1,Id2, PB2,PB3, Debug).
c_body_unify(true,          _, Id0,Id1, PB0,PB1, _    ) :- /* !, */
    Id0=Id1, PB0=PB1.

:- mode c_id_setup(+, -,?, ?,+,-, -).
c_id_setup(nodebug, Ig0,Ig1, _, Id0,Id1, Myid) :- !,
    Myid=0, Ig0=Ig1, Id1 is Id0 .
c_id_setup(debug,   Ig0,Ig1, Id,Id0,Id1, Myid) :- /* !, */
    ( Id0=:=0, !, Myid=Id, Ig0=Ig1; Ig0=(Myid is Id+Id0, Ig1) ),
    Id1 is Id0+1.

% Initial sequence of assignments that need no waiting is processed on the
% spot.
:- mode c_body_arith(+,-, -,?, +).
c_body_arith((V:=E,Body0),Body1, PB0,PB2, nodebug) :-
    c_iw_all(E, W,true), W=true, !,  % check if waiting is needed for E
  ( integer(E), !, PB0=(ubody(E,V), PB1);
    PB0=(V2 is E, ubody(V2,V), PB1) ),
    c_iw(V, _,_),                    % mark V as 'having an integer value'
    c_body_arith(Body0,Body1, PB1,PB2, nodebug).
c_body_arith(Body0,       Body1, PB0,PB1, _      ) :-
 /* top element of Body0 is not a non-suspending assignment, !, */
    Body0=Body1, PB0=PB1.

% Put_queue is called also at run time to make a goal queue.  It fails if
% there exists an uninstantiated goal.
% put_queue(Body_goals, Goal_queue_head, Goal_queue_tail,
%           New_reduction_count, Max_reductions,
%           Id_calc_goal_head, Id_calc_goal_tail,
%           Next_id, Initial_id_offset, Final_id_offset,
%           Head, Dict_old, Dict_new, Debug_mode?)
:- mode put_queue(?, -,-, ?,?, -,?, ?,+,-, +, +,-, +).
put_queue(X,      _, _,  _,_,_,_,_, _,  _,   _,    _,     _    ) :-
    variable(X), !, fail.
put_queue((X,Y), Q0,Q2, RC1,RCmax,
                        Ig0,Ig2, Id,Id0,Id2, Head, Dict0,Dict2, Debug) :- !,
```

```
    put_queue(X, Q0,Q1, RC1,RCmax,
                        Ig0,Ig1, Id,Id0,Id1, Head, Dict0,Dict1, Debug),
    put_queue(Y, Q1,Q2, RC1,RCmax,
                        Ig1,Ig2, Id,Id1,Id2, Head, Dict1,Dict2, Debug).
put_queue(true,  Q0,Q1, _, _,
                        Ig0,Ig1, _, Id0,Id1, _,    Dict0,Dict1, _    ) :- !,
    Q0=Q1, Ig0=Ig1, Id0=Id1, Dict0=Dict1.
put_queue(V:=E,  Q0,Q1, RC1,RCmax,
                        Ig0,Ig1, Id,Id0,Id1, Head, Dict0,Dict1, Debug) :-
    Head\=='$TOP', !,
    make_arith_head(Head, V,E, AH,AP,AA, Dict0,Dict1),
    write_mode_decl(AP,AA),
    extend_atom(AH,AHx, RC1,Ch,Ct,Dnd,RCmax,Myid,Nextid),
  ( Debug==debug, !,
      write_clause((AHx :- ghcspied(':=',2), trace_call(Myid,V:=E), fail));
    true ),
    c_iw_all(E, G,(V2 is E, !, ubody(V2,V), do_next(Ch,Ct,Nextid))),
    write_clause((AHx :- G)),
  ( Debug==debug, !,
      write_clause((AHx :- ghcspied(':=',2),
                           trace_suspension(1,Myid), fail));
    true ),
    write_susp_clause(AP,AA),
    Q0=[$(AHx, Ch,Ct,Dnd,Nextid) | Q1],
    c_id_setup(Debug, Ig0,Ig1, Id,Id0,Id1, Myid).
put_queue(X,     Q0,Q1, RC1,RCmax,
                        Ig0,Ig1, Id,Id0,Id1, _,    Dict0,Dict1, Debug) :-
 /* user_defined_goal(X), !, */
    extend_atom(X,Xx, RC1,Ch,Ct,Dnd,RCmax,Myid,Nextid),
    Q0=[$(Xx, Ch,Ct,Dnd,Nextid) | Q1],
    c_id_setup(Debug, Ig0,Ig1, Id,Id0,Id1, Myid),
    Dict0=Dict1.

:- mode make_arith_head(+, ?,?, -,-,-, +,-).
make_arith_head(Head, V,E, AH,AP,AA, Dict0,Dict1) :-
    functor(Head,P,A), inc_arith(P/A, Dict0,Dict1, Arithid),
    name(P,Pl), name(AP,[36,36,36,Arithid|Pl]),
    analyze_exp(E, L,[]),
    AH =.. [AP,V|L], functor(AH,_,AA).

:- mode analyze_exp(?, -,?).
analyze_exp(E, L0,L1) :- variable(E), !, L0=[E|L1].
analyze_exp(E, L0,L1) :- /* not_variable(E), !, */
    functor(E, _, A), analyze_exp_args(0,A, E, L0,L1).

:- mode analyze_exp_args(+,+, +, -,?).
analyze_exp_args(K,N, E, L0,L2) :- K<N, !,
    K1 is K+1, arg(K1,E,Ek), analyze_exp(Ek, L0,L1),
    analyze_exp_args(K1,N, E, L1,L2).
analyze_exp_args(N,N, _, L0,L1)  :- /* !, */ L0=L1.

:- mode c_update_nextid(+, -,+,-, -,-).
c_update_nextid(debug, Nextid,Id1,Newid, PB4,PB5) :- Id1>0, !,
    PB4=(Newid is Nextid+Id1, PB5).
c_update_nextid(Debug, Nextid,Id1,Newid, PB4,PB5) :-
 /* ( Debug==nodebug; Id1=<0 ), !, */
    Newid=Nextid, PB4=PB5.

% c_make_tail(Depth_first?, Body_queue_head,Body_queue_tail,
%             Reduction_count_old,Reduction_count_new,
%             Cont_h,Cont_t, New_goal_id, Prolog_body).
:- mode c_make_tail(+, ?, -,-,-,-,-,-,-).
c_make_tail(_,     H,          T, _,  _,    Ch,Ct, Newid, PB5) :- H==T, !,
    PB5=(Ch=[$(Gx, Ch1,Ct,nd,Newid)|Ch1], incore(Gx)).
c_make_tail(Depth, [Qitem|H1],T, RC0,RC1, Ch,Ct, Newid, PB5) :- /* !, */
    c_update_rc(Depth, RC0,RC1, PB5,PB6),
```

```
    Qitem= $(PB6, H1,Ct,nd,Newid),
    T=Ch.                    % The d-list [Qitem|H1]-T is connected to Ch-Ct

:- mode c_update_rc(+, -,-, -,-).
c_update_rc(nodepth, RC0,RC1, PB5,PB6) :- !,
    PB5=(RC1 is RC0-1, PB6).
c_update_rc(depth,   RC0,RC1, PB5,PB6) :- /* !, */
    PB5=PB6, RC1=RC0.

%%% MISCELLANEOUS ROUTINES %%%

%% Predicate Dictionary Management %%

:- mode lookup_dict(+, +, -,-,-).
lookup_dict(Pred, [$$(Pred, IP,Mode,Debug,_)|_], IP,Mode,Debug) :- !.
lookup_dict(Pred, [Item|Dict],                   IP,Mode,Debug) :-
 /* Item=$$(Pred2, _,_,_,_), Pred\==Pred2, !, */
    lookup_dict(Pred, Dict, IP,Mode,Debug).

:- mode inc_arith(+, +,-, -).
inc_arith(Pred, Dict0,Dict1, Arithid) :- /* !, */
    find_item(Pred, Dict0,Dict_rest, Item),
    Item= $$(Pred, IP,Mode,Debug,Arithid0), Arithid is Arithid0+1,
    Dict1=[ $$(Pred, IP,Mode,Debug,Arithid)|Dict_rest].

:- mode find_item(+, +,-, -).
find_item(Pred, [I|Dict],Dict_rest, Item) :- arg(1,I,Pred), !,
    Dict_rest=Dict, Item=I.
find_item(Pred, [I|Dict], Dict_rest, Item) :- /* not_arg(1,I,Pred), !, */
    Dict_rest=[I|Dict_rest1], find_item(Pred, Dict,Dict_rest1, Item).

%% Constructing Atomic Formulas %%

:- mode extend_atom(+, -, ?,?,?,?,?,?,?).
extend_atom(X, Xx, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :- /* !, */
    functor(X,F,A),
    A1 is A+1, A2 is A+2, A3 is A+3, A4 is A+4,
    A5 is A+5, A6 is A+6, A7 is A+7,
    functor(Xx,F,A7), copy_args(A,X,Xx),
    arg(A1,Xx,RC),    arg(A2,Xx,Ch),   arg(A3,Xx,Ct),     arg(A4,Xx,Dnd),
    arg(A5,Xx,RCmax), arg(A6,Xx,Myid), arg(A7,Xx,Nextid).

:- mode copy_args(+, +,+).
copy_args(0, _,_ ) :- !.
copy_args(K, X,Xx) :- /*K>0, !,*/
    arg(K,X,Xk), arg(K,Xx,Xk), K1 is K-1, copy_args(K1, X,Xx).


%% Analyzing Mode Declaration %%

% check_mode(Mdecl) succeeds if Mdecl contains the '+' mode, fails
% otherwise.  Also, it complains of modes other than '+' and '?'.
:- mode check_mode(+).
check_mode(Mdecl) :- /* !, */
    functor(Mdecl,_,A), check_mode_args(0,A, Mdecl, no).
:- mode check_mode_args(+,+, +, +).
check_mode_args(K,N, Mdecl, YN0) :- K<N, !,
    K1 is K+1, arg(K1,Mdecl,M), check_mode_arg(M, YN0,YN1),
    check_mode_args(K1, N, Mdecl, YN1).
check_mode_args(N,N, _,     yes) /* :- ! */.  % fails if 4th arg is 'no'.

:- mode check_mode_arg(?, +,-).
check_mode_arg(+, _,  YN1) :- !, YN1=yes.
check_mode_arg(?, YN0,YN1) :- !, YN1=YN0.
check_mode_arg(M, YN0,YN1) :- /* !, */
    display('Illegal mode specifier: '), display(M), ttynl, YN1=YN0.
```

```
%% Analyzing Variables in Clauses %%

:- mode variable(?).
variable(X          ) :- var(X), !.
variable('$REF'(_) ) :- !.
variable('$WAIT'(_)) :- !.
variable('$INT'(_) ) /* :- ! */.

:- mode appeared_in_head(?).
appeared_in_head(X          ) :- var(X), !, fail.
appeared_in_head('$REF'(_)) /* :- ! */.

:- mode mark(-, +).
mark(X, +) :- !, X='$WAIT'(_).
mark(_, M) /* :- M\=='+', ! */.

%% Displaying Messages %%

:- mode notify_compilation(+,+).
notify_compilation(P,A) :-
    display(P), ttyput("/"), display(A), display(', '), ttyflush.

:- mode errmsg(+, ?).
errmsg(Msg, Clause) :- /* !, */
    telling(File), tell(user), nl,
    write(Msg), write_clause(Clause), nl, tell(File).

:- mode writeuser(?).
writeuser(T) :- /* !, */ telling(File), tell(user), write(T), tell(File).
```

```
/* Copyright (c) 1987, 1988, Institute for New Generation Computer Technology.
   All rights reserved.
   Permission to use this program is granted for academic use only. */

%%%%%%%%%%%%%%%%%% GHC Compiler System for DEC-10 Prolog %%%%%%%%%%%%%%%%%%%%
%                                                                             %
%                      Version 1.9  (November 18, 1987)                       %
%                                                                             %
%                        Kazunori Ueda                                        %
%             Institute for New Generation Computer Technology                %
%              4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan                 %
%                                                                             %
%               phone: +81-3-456-2514    telex: ICOT J 32964                 %
%               csnet: ueda%icot.jp@relay.cs.net                             %
%               uucp:  {enea,inria,kddlab,mit-eddie,ukc}!icot!ueda           %
%                                                                             %
%       Modification for outside ICOT by S. Takagi, 4-Jan-88                  %
%       This program is designed and tested on DEC-10 Prolog ver. 3.52        %
%       running on DEC 2065 TOPS-20 ver. 6.1.                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(1150,fx, (ghc)).
:- op(1150,fx, (ghcspy)).
:- op(1150,fx, (ghcnospy)).
:- op(700, xfx,:=).             % becomes
:- op(700, xfx,\=).             % dif
:- op(50,  xf, ?).              % for specifying 'trace-mode'

%%%% SAVING EXECUTABLE IMAGE %%%%
:- public saveghc/0.
saveghc :- plsys(core_image), nolog, ghcbacktrace, header.
header :-
    write('*** GHC System Vers. 1.9 (1987-11-18)  by Kazunori Ueda, ICOT'), nl.

%%%% RUN TIME SUPPORT %%%%

%%% TOP LEVEL %%%

:- public (ghc)/1, (ghc)/2, (ghc)/3.
ghc(Goals)            :- ghc(Goals, 100,    _).
ghc(Goals, Bound)     :- ghc(Goals, Bound, _).
ghc(_,     Bound, _) :- illegal_bound(Bound), !,
    display('Illegal bound value: '), display(Bound), ttynl, fail.
ghc(Goals, Bound, T) :-
    save_standard_io, reset_ioflag,
    solve(Goals, Bound), !,
    statistics(runtime,[_,T]), restore_standard_io,
  ( recorded('$GHCBACKTRACE',_,_), !, display(T), display(' msec.'), ttynl;
    true ).
ghc(_,     _,     _) :- /* failed in solving Goals */
    restore_standard_io,
    recorded('$GHCBACKTRACE',_,_), !,
  ( recorded('$GHCBACKTRACE_END',_,Ref), !, erase(Ref);
    recorded('$GHCUNIFYFAIL',    _,Ref), !, erase(Ref);
    display('You called an undefined predicate.'), ttynl ), fail.

:- mode illegal_bound(?).
illegal_bound(Bound) :- integer(Bound), Bound>0, !, fail.
illegal_bound(_    ).

:- mode solve(?, +).
solve(Goals,  _    ) :- var(Goals), !, fail.
solve(Goals?, Bound) :- !,
    prepare_goal_queue(Goals, Bound, Ch,Ct, Nextid),
    statistics(runtime,_),
    incore('$ENDSPY'(0, Ch,Ct, nd, Nextid)).
```

```
solve(Goals,  Bound) :- /* not_functor(Goals, ?, _), !, */
    prepare_goal_queue(Goals, Bound, Ch,Ct, Nextid),
    statistics(runtime,_),
    incore('$END'(Ch,Ct, nd, Nextid)).

:- mode prepare_goal_queue(?, +, -,-, -).
prepare_goal_queue(Goals, Bound, Ch,Ct, Nextid) :-
    put_queue(          % The definition of 'put_queue' is in the compiler.
        Goals,          % Conjunctive goals to be put in the queue
        Ch,Ct,          % The queue (d-list) of goals
        Bound,Bound,    % Current and initial values of the reduction counter
        Id_calc,true,   % D-conjunction of arithmetic goals to calculate the
                        % IDs of Goals to be displayed on tracing
        1,              % Initial value of the goal IDs
        0,Id1,          % Initial and final values of the offsets of goal IDs
        '$TOP',         % Predicate that called Goals
        [],_,           % Old and new predicate dictionaries (for compiling
                        % ':=' goals that are called not at the top level)
        debug           % Debugging mode (debug/nodebug)
        ),
    Nextid is 1+Id1,    % Compute the ID of the next goal to be generated
    call(Id_calc), !. % Provide the top-level goals with their IDs

%%% UNIFICATION %%%
% The following unification predicates fail on suspension.

% Compound term and constant:  uskel(X,Skeleton)
:- mode uskel(+,+).
uskel(X,X).

% Nil:  unil(X) (Hacked version of uskel)
:- mode unil(+).
unil([]).

% List:  ulist(X,Car,Cdr) (Hacked version of uskel)
:- mode ulist(+,-,-).
ulist([H|T],H,T).

% Unification in clause bodies:
:- mode ubody(?,?).
ubody(X,X) :- !.
ubody(X,Y) :- /* X and Y are ununifiable */
    recorded('$GHCBACKTRACE',_,_),
    recorda('$GHCUNIFYFAIL',_,_),
    display('You tried to unify '), writeuser(X), display(' with '),
    writeuser(Y), ttynl, fail.

:- mode 'ubody?'(?,?, +).
'ubody?'(X,Y, Myid) :- ghcspied('=',2), trace_call(Myid,X=Y), fail.
'ubody?'(X,Y, _   ) :- ubody(X,Y).

% Ununifiability
% ghcdif(X,Y) succeeds if and when X and Y proved to be ununifiable.
:- mode ghcdif(?,?).
ghcdif(X,_) :- var(X), !, fail.
ghcdif(_,Y) :- var(Y), !, fail.
% Here, 1st & 2nd args are non-variables.
ghcdif(X,Y) :- functor(X,F,A), functor(Y,F,A), !, ghcdif_args(A,X,Y).
ghcdif(_,_).  % succeeds when the function symbols or the arities mismatch

:- mode ghcdif_args(+, +,+).
ghcdif_args(0, _,_) :- !, fail.
ghcdif_args(N, X,Y) :- /* N>0, */
    arg(N,X,Xn), arg(N,Y,Yn), ghcdif(Xn,Yn), !.
ghcdif_args(N, X,Y) :- /* N>0, not yet dif(Xn,Yn) */
    N1 is N-1, ghcdif_args(N1, X,Y).
```

```
%%% SYSTEM PREDICATES %%%
% System predicates are executed even if the reduction counter indicates 0.

%% Input and Output %%
% Instream and outstream can be called only once for each.

:- public instream/8.
:- mode instream(?, +,+,-,+,+,+,+).
instream(S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    '$START_IO'(instream, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- public outstream/8.
:- mode outstream(?, +,+,-,+,+,+,+).
outstream(S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    '$START_IO'(outstream, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- public '$START_IO'/9.
:- mode '$START_IO'(+, ?, +,+,-,+,+,+,+).
'$START_IO'(IO,  ,  _,  _,  _,  _,  _,    _,    _     ) :-
    recorded('$GHCIOUSING',IO,_), !,
    display('Error: Instream/outstream cannot be called twice.'),
    ttynl, restore_standard_io, abort.
'$START_IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    /* not_recorded('$GHCIOUSING',IO,_), !, */
    recorda('$GHCIOUSING',IO,_),
    '$IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- public '$IO'/9.
:- mode '$IO'(+, ?, +,+,-,+,+,+,+).
'$IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    ghcspied(IO,1),
    functor(G,IO,1), arg(1,G,S), trace_call(Myid, G), !,
    '$IO?'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).
'$IO'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :-
    '$IO!'(IO, S, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid).

:- mode '$IO?'(+, ?, +,+,-,+,+,+,+).
'$IO?'(IO, S, RC,Ch,Ct,_,  RCmax,_,   Nextid) :-
    nonvar(S), ulist(S,H,T), nonvar(H), do_io(H,IO), !,
    Nextid1 is Nextid+1,
    '$IO'(IO, T, RC,Ch,Ct,nd,RCmax,Nextid,Nextid1).
'$IO?'(_,  S, _ ,Ch,Ct,_,  _,    _,   Nextid) :-
    nonvar(S), unil(S), !, do_next(Ch,Ct, Nextid).
'$IO?'(IO, S, RC,[$(Gx, Ch,Ct,Dnd,Nextid)|Ch],
                    [$('$IO'(IO,S, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                        Ch2,Ct2,Dnd2,          Nextid2)
                    |Ct],
                       Dnd,RCmax,Myid,Nextid) :-
    trace_suspension(RC, Myid), !, incore(Gx).

:- public '$IO!'/9.
:- mode '$IO!'(+, ?, +,+,-,+,+,+,+).
'$IO!'(IO, S, RC,Ch,Ct,_,  RCmax,_,   Nextid) :-
    nonvar(S), ulist(S,H,T), nonvar(H), do_io(H,IO), !,
    Nextid1 is Nextid+1,
    '$IO!'(IO, T, RC,Ch,Ct,nd,RCmax,Nextid,Nextid1).
'$IO!'(_,  S, _ ,Ch,Ct,_,  _,    _,   Nextid) :-
    nonvar(S), unil(S), !, do_next(Ch,Ct, Nextid).
'$IO!'(IO, S, _, [$(Gx, Ch,Ct,Dnd,Nextid)|Ch],
                    [$('$IO'(IO,S, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                        Ch2,Ct2,Dnd2,          Nextid2)
                    |Ct],
                       Dnd,RCmax,Myid,Nextid) :- incore(Gx).

:- mode do_io(+,+).
```

```
do_io(see(F),     instream) :- !, nonvar(F), see(F).
do_io(seeing(F), instream) :- !, seeing(F).
do_io(seen,       instream) :- !, seen.
do_io(tell(F),    _       ) :- !, nonvar(F), tell(F).
do_io(telling(F),_        ) :- !, telling(F).
do_io(told,       _       ) :- !, told.

do_io(close(F),     _      ) :- !, nonvar(F), my_close(F).
do_io(fileerrors,  _      ) :- !, fileerrors.
do_io(nofileerrors,_      ) :- !, nofileerrors.

do_io(read(X),    instream) :- !, my_read(X).
do_io(write(X),   _       ) :- !, ground(X), write(X).
do_io(writeq(X), _        ) :- !, ground(X), writeq(X).
do_io(print(X),   _       ) :- !, ground(X), print(X).
do_io(nl,         _       ) :- !, nl.
do_io(get0(C),    instream) :- !, my_get0(C).
do_io(get(C),     instream) :- !, my_get(C).
do_io(skip(C),    instream) :- !, ground(C), skip(C).
do_io(put(C),     _       ) :- !, ground(C), put(C).
do_io(tab(N),     _       ) :- !, ground(N), tab(N).

do_io(display(X),_        ) :- !, ground(X), display(X).
do_io(ttynl,      _       ) :- !, ttynl.
do_io(ttyflush,  _        ) :- !, ttyflush.
do_io(ttyget0(C),instream) :- !, my_ttyget0(C).
do_io(ttyget(C), instream) :- !, my_ttyget(C).
do_io(ttyskip(C),instream) :- !, ground(C), ttyskip(C).
do_io(ttyput(C), _        ) :- !, ground(C), ttyput(C).
do_io(ttytab(N), _        ) :- !, ground(N), ttytab(N).

do_io(prompt(O,N),instream):- !, atom(N), prompt(O,N).
do_io(G,          _       ) :-
    display('Error: Illegal request in instream/outstream:  '),
    writeuser(G), ttynl, restore_standard_io, abort.

% If nofileerrors has been executed, input goals can be called arbitrarily
% many times.
:- mode my_get0(?), my_ttyget0(?), my_get(?), my_ttyget(?), my_read(?).
my_get0(C )     :- get0(C), !.
my_get0(26).
my_ttyget0(C ) :- ttyget0(C), !.
my_ttyget0(26).
my_get(C )      :- get(C), !.
my_get(26).
my_ttyget(C )  :- ttyget(C), !.
my_ttyget(26).
my_read(X)      :- read(X), !, numbervars(X, 0,_).     % An input term is
                                                       % made ground.

% the following is for discarding an alternative created by the system
% predicate close/1 when the file specified by its argument is currently
% open.
:- mode my_close(+).
my_close(F) :- close(F), !.


%% Other System Predicates Callable from Bodies %%

% The following routine is used only when ':='/2 is called directly from
% the top level; otherwise ':='/2 is compiled into a specialized predicate.
:- public (:=)/9.
:- mode :=(?,?, +,+,-,+,+,+,+).
:=(X,Y, _,  _, _,  _,    Myid,_     ) :-
    ghcspied(':=',2), trace_call(Myid, X:=Y), fail.
:=(X,Y, _,  Ch,Ct,_,  _,    _,    Nextid) :-
```

```
    numbervars(Y,0,0), call(X2 is Y), !,
    ubody(X2,X), do_next(Ch,Ct,Nextid).
:=(_,_, RC0,_, _, _,  _,    Myid,_     ) :-
    ghcspied(':=',2), trace_suspension(RC0,Myid), fail.
:=(X,Y, _,  [$(Gx, Ch,Ct,Dnd,Nextid) | Ch],
              [$(:=(X,Y, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                Ch2,Ct2,Dnd2,          Nextid2) | Ct],
                 Dnd,RCmax,Myid,Nextid) :- incore(Gx).

% The following routine is used only when '='/2 is called directly from
% the top level; otherwise '='/2 is compiled into 'ubody/2'.
:- public (=)/9.
:- mode =(?,?, +,+,-,+,+,+,+).
=(X,Y, _,_, _, _,_,Myid,_     ) :-
    ghcspied('=',2), trace_call(Myid, X=Y), fail.
=(X,Y, _,Ch,Ct,_,_,_,   Nextid) :- ubody(X,Y), do_next(Ch,Ct, Nextid).

:- public prolog/8.
:- mode prolog(?, +,+,-,+,+,+,+).
prolog(X, _,  _, _, _,  _,    Myid,_     ) :-
    ghcspied(prolog,1), trace_call(Myid, prolog(X)), fail.
prolog(X, _,  Ch,Ct,_,  _,    _,   Nextid) :-  call(X), !,
    do_next(Ch,Ct,Nextid).
prolog(_, RC0,_, _, _,  _,    Myid,_     ) :-
    ghcspied(prolog,1), trace_suspension(RC0, Myid), fail.
prolog(X, _,  [$(Gx, Ch,Ct,Dnd,Nextid) | Ch],
               [$(prolog(X, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
                                Ch2,Ct2,Dnd2,          Nextid2) | Ct],
                  Dnd,RCmax,Myid,Nextid) :- incore(Gx).

%% Calling the Next Goal in the Queue %%

:- mode do_next(+,-, +).
do_next([$(Gx, Ch2,Ct,nd,Nextid)|Ch2],Ct, Nextid) :-  incore(Gx).

%%% CYCLE MARKERS %%%

%%  Standard Version %%

:- public '$END'/4.
:- mode '$END'(?,-,+,+).
'$END'([],_, _, _     ) :- !. % succeeds if the queue is empty, i.e.,
                              % the first arg is UNINSTANTIATED.
'$END'([$(Gx, Ch,Ct,d,Nextid)|Ch],
          [$('$END'(Ch2,Ct2,Dnd2,Nextid2),Ch2,Ct2,Dnd2,Nextid2)|Ct],
             nd,Nextid) :- !,
    incore(Gx).               % otherwise, if some goals have been reduced
                              % in the last 'cycle' (i.e., the deadlock
                              % flag is 'nd'), then try  another 'cycle'.
'$END'(Gs,_, _, _     ) :- backtrace(Gs), fail.

%% Cycle Marker with Spy Facilities %%

:- public '$ENDSPY'/5.
:- mode '$ENDSPY'(+, ?,-,+,+).
'$ENDSPY'(Endid, Ch,Ct,Dnd,Nextid) :-
    display('Cycle '), display(Endid),
    display('; function (h for help)? '), ttyflush, ttyget0(C),
  ( C=\=31/*newline*/, !, ttyskip(31); true ),
    endspy(C, Endid, Ch,Ct,Dnd,Nextid).

:- mode endspy(+, +, ?,-,+,+).
endspy(31 /*NL*/, Endid, Ch,Ct,Dnd,Nextid) :- !,          % CONTINUE
    endspy2(Endid, Ch,Ct,Dnd,Nextid).
endspy(119/*w*/,  Endid, Ch,Ct,Dnd,Nextid) :- !,          % WRITE QUEUE
    display_queue(Ch), '$ENDSPY'(Endid, Ch,Ct,Dnd,Nextid).
```

```
endspy(110/*n*/,  _     , Ch,Ct,Dnd,Nextid) :- !,          % NODEBUG MODE
    '$END'(Ch,Ct,Dnd,Nextid).
endspy(104/*h*/,  Endid, Ch,Ct,Dnd,Nextid) :- !,          % HELP
    display('<cr>: continue'), ttynl,
    display('   w: Write goals in the queue'), ttynl,
    display('   n: continue in Nodebug mode'), ttynl,
    display('   h: Help'), ttynl,
    display('   a: Abort current execution'), ttynl,
    display('   @: Accept a Prolog goal'), ttynl,
    '$ENDSPY'(Endid,Ch,Ct,Dnd,Nextid).
endspy(97 /*a*/,  _,      _, _, _,  _      ) :- !,          % ABORT
    restore_standard_io, abort.
endspy(64 /*@*/,  Endid, Ch,Ct,Dnd,Nextid) :- !,          % ACCEPT COMMAND
    display('| :- '), ttyflush, seeing(F), see(user), read(G), see(F),
  ( call(G), !, display('yes'); display('no') ), ttynl,
    '$ENDSPY'(Endid, Ch,Ct,Dnd,Nextid).
endspy(_,         Endid, Ch,Ct,Dnd,Nextid) :- /* !, */ % Other chars are
    endspy(104, Endid, Ch,Ct,Dnd,Nextid).              % interpreted as 'h'

:- mode endspy2(+, ?,-,+,+).
endspy2(_,     [],_, _, _      ) :- !.
endspy2(Endid, [$(Gx, Ch,Ct,d,Nextid)|Ch],
                  [$('$ENDSPY'(Endid2, Ch2,Ct2,Dnd2,Nextid2),
                                       Ch2,Ct2,Dnd2,Nextid2)|Ct],
                    nd,Nextid) :- Endid2 is Endid+1, !, incore(Gx).
endspy2(_,     Gs,_, _, _      ) :- backtrace(Gs), fail.

%%% TRACING AND BACKTRACING %%%

:- public
    (ghcspy)/0, (ghcspy)/1, (ghcnospy)/0, (ghcnospy)/1, ghcspying/0,
    ghcbacktrace/0, ghcnobacktrace/0.

%% Tracing %%

% Setting, Resetting and Showing Spy Points

ghcspy :-
  ( recorded('$GHCSPY',X,_), var(X), !,
      display('Spy points have already been set on all goals.'), ttynl;
    ( erasel, fail; recorda('$GHCSPY',_,_) ) ).
:- mode ghcspy(?).
ghcspy(X) :- atom(X), !,
  ( recorded('$GHCSPY',X/V,_), var(V), !, complain_double_spy(X);
    ( erasel(X), fail; recorda('$GHCSPY',X/_,_) ) ).
ghcspy(X) :- nonvar(X), X=P/A, atom(P), integer(A), !,
  ( recorded('$GHCSPY',X,_), !, complain_double_spy(X);
    recorda('$GHCSPY',X,_) ).
ghcspy(X) :- nonvar(X), X=(X1,X2), !, ghcspy(X1), ghcspy(X2).
ghcspy(X) :- complain_predicate_spec(X).

ghcnospy :-
  ( erasel, !, ( erasel, fail; true ); complain_nothing_to_reset ).
:- mode ghcnospy(?).
ghcnospy(X) :- atom(X), !,
  ( erasel(X), !, ( erasel(X), fail; true );
    recorded('$GHCSPY',Y,_), var(Y), !, complain_too_specific(X);
    complain_nothing_to_reset(X) ).
ghcnospy(X) :- nonvar(X), X=P/A, atom(P), integer(A), !,
  ( erasel(P,A), !;
    recorded('$GHCSPY',Y,_), Y=P/V, var(V), !,
      complain_too_specific(X);
    complain_nothing_to_reset(X) ).
ghcnospy(X) :- nonvar(X), X=(X1,X2), !, ghcnospy(X1), ghcnospy(X2).
ghcnospy(X) :- complain_predicate_spec(X).
```

```
erase1 :- recorded('$GHCSPY',_,R), erase(R).
:- mode erase1(+).
erase1(P) :- recorded('$GHCSPY',X,R), nonvar(X), X=P/_,  erase(R).
:- mode erase1(+,+).
erase1(P,A) :- recorded('$GHCSPY',X,R), nonvar(X), X=P/B, A==B, erase(R).

:- mode complain_double_spy(+).
complain_double_spy(X) :-
    display('Spy point on '), writeuser(X),
    display(' has already been set.'), ttynl.
:- mode complain_predicate_spec(?).
complain_predicate_spec(X) :-
    display('Illegal predicate specification:  '), writeuser(X), ttynl.
complain_nothing_to_reset :-
    display('No GHC goals are spied.'), ttynl.
:- mode complain_nothing_to_reset(+).
complain_nothing_to_reset(X) :-
    display('Spy point has not been set on '),
    writeuser(X), display('.'), ttynl.
:- mode complain_too_specific(+).
complain_too_specific(X) :-
    display('Spy point cannot be reset selectively on '),
    writeuser(X), display('.'), ttynl.

ghcspying :- setof($(X), R^recorded('$GHCSPY', X, R), S), !,
  ( S=[$(Y)|_], var(Y), !, display('All GHC goals are spied.');
    display('GHC spy-points have been set on '), write_spied(S) ), ttynl.
ghcspying :- display('No GHC goals are spied.'), ttynl.

:- mode write_spied(+).
write_spied([$(P/A)|Ps]) :- display(P),
  ( nonvar(A), !, ttyput("/"), display(A); true ),
  ( Ps\==[], !, display(', '), write_spied(Ps); ttyput(".") ).

% Runtime Support for Tracing

:- mode ghcspied(+,+).
ghcspied(P,A) :-  recorded('$GHCSPY',P/A,_), !.

:- mode trace_call(+, +).
trace_call(Id, G) :-
    display_id(Id), display(': Call '), writeuser(G), ttynl.

:- mode display_id(+).
display_id(0 ) :- !.
display_id(Id) :- /* Id>0, !, */ display(Id).

:- mode trace_suspension(+, +).
trace_suspension(0, Id) :- !,
    display_id(Id), display(': Swapped out'), ttynl.
trace_suspension(N, Id) :- /* N>0, !, */
    display_id(Id), display(': Suspended'), ttynl.

%% Backtracing %%

% Setting and Resetting
ghcbacktrace   :- recorda('$GHCBACKTRACE',_,_).
ghcnobacktrace :- recorded('$GHCBACKTRACE',_,R), erase(R), fail.
ghcnobacktrace.

% Runtime Support
:- mode backtrace(+).
backtrace(Gs) :-
    recorded('$GHCBACKTRACE',_,_),
    display('Deadlock detected.  Suspended goals are:'), ttynl,
    display_queue(Gs), recorda('$GHCBACKTRACE_END',_,_).
```

```
%% Runtime Support for Displaying the Goal Queue %%

:- mode display_queue(+).
display_queue(Gs) :- numbervars(Gs,0,_), dq2(Gs), fail.
display_queue(_ ).

:- mode dq2(+).
dq2([G|Gs]) :- !, display_goal(G), dq2(Gs).
dq2(_).

:- mode display_goal(+).
display_goal($(Gx, _,_,_,_)) :-
    functor(Gx,P,A7), A is A7-7, functor(G,P,A),
    copy_args(A, Gx,G),
    A6 is A+6, arg(A6,Gx,Id), display_id(Id),
    display(': '), writeuser(G), ttynl.

%%% MISCELLANEOUS %%%

:- mode ground(?).
ground(X) :- var(X), !, fail.
ground(X) :- /* !, */ functor(X,_,Arity), ground_args(Arity, X).

:- mode ground_args(+, +).
ground_args(0, _) :- !.
ground_args(N, X) :- /* !, */
    arg(N,X,T), ground(T), N1 is N-1, ground_args(N1, X).

reset_ioflag :- recorded('$GHCIOUSING',_,R), erase(R), fail.
reset_ioflag.

save_standard_io :-
    seeing(In),   recorda('$GHCSTDIN', In, _),
    telling(Out), recorda('$GHCSTDOUT',Out,_).

restore_standard_io :- fileerrors,
    recorded('$GHCSTDIN', In, Ref1), erase(Ref1), see(In),
    recorded('$GHCSTDOUT',Out,Ref2), erase(Ref2), tell(Out).
```

```
%%%% COMPILATION %%%%

%%% READING PROGRAMS %%%

:- public ghccompile/1, ghccompile/2, ghcdcompile/1, ghcdcompile/2.
:- mode ghccompile(?), ghccompile(?,?), ghcdcompile(?), ghcdcompile(?,?).
ghccompile(S) :- /* !, */
    ghccompile(S, '-temp-.ghc', nodepth), compile('-temp-.ghc').
ghccompile(S,O) :- /* !, */ ghccompile(S,O, nodepth).
ghcdcompile(S) :- /* !, */
    ghccompile(S, '-temp-.ghc', depth),   compile('-temp-.ghc').
ghcdcompile(S,O) :- /* !, */ ghccompile(S,O, depth).

ghccompile(S,O, Depth) :-
    telling(Old), nofileerrors, tell(O), !,
    ghccomp(S, [],Dict, Depth),
    write_last_clause_group(Dict, Depth),
    told, tell(Old), display('END.'), ttynl.
ghccompile(_,O, _     ) :- /* failed to open O */
    complain_unable_to_open(O).

% ghccomp(Files, Current_dict,New_dict, Depth_first?).
% Each member of the predicate dictionary has the form
% $$(Functor/Arity, Name_of_internal_subpredicate,
%    Mode_decl, Debug_mode, #_of_body_assignment+*@").
:- mode ghccomp(?, +,-, +).
ghccomp(F,     Dict0,Dict1, _     ) :- var(F), !,
    Dict0=Dict1, complain_unable_to_open(F).
ghccomp(F,     Dict0,Dict1, Depth) :-
    atom(F), seeing(Old), nofileerrors, see(F), !,
    read(X), ghccomp1(X, Dict0,Dict1, nodebug, Depth), seen, see(Old).
ghccomp(F?,    Dict0,Dict1, Depth) :
    atom(F), seeing(Old), nofileerrors, see(F), !,
    read(X), ghccomp1(X, Dict0,Dict1, debug,   Depth), seen, see(Old).
ghccomp([H|T], Dict0,Dict2, Depth) :- !,
    ghccomp(H, Dict0,Dict1, Depth), ghccomp(T, Dict1,Dict2, Depth).
ghccomp([],    Dict0,Dict1, _     ) :- !, Dict0=Dict1.
ghccomp(F,     Dict0,Dict1, _     ) :- /* failed to open F */
    Dict0=Dict1, complain_unable_to_open(F).

:- mode ghccomp1(?, +,-, +, +).
ghccomp1(X,               Dict0,Dict1, Debug, Depth) :- var(X), !,
    display('Uninstantiated clause found'), ttynl,
    read(Next), ghccomp1(Next, Dict0,Dict1, Debug, Depth).
ghccomp1(end_of_file,     Dict0,Dict1, _,     _     ) :- !, Dict0=Dict1.
ghccomp1((:- mode Mdecl), Dict0,Dict2, Debug, Depth) :- nonvar(Mdecl), !,
    functor(Mdecl, P, A),
  ( lookup_dict(P/A, Dict0, _,_,_), !,
      errmsg('Duplicate or misplaced mode declaration: ', Mdecl),
      Dict1=Dict0;
    notify_compilation(P,A), write_mode_decl(P,A),
  ( check_mode(Mdecl), !,
      write_transmit_clause(Mdecl), make_internal_name(P,IP),
      write_mode_decl(IP,A);
    IP=P ),
    write_callmsg_clause(IP,A,P, Debug),
    Dict1=[$$(P/A, IP,Mdecl,Debug,64/*@*/)|Dict0] ),
    read(Next), ghccomp1(Next, Dict1,Dict2, Debug, Depth).
ghccomp1((:- op(Priority,Type,Op)), Dict0,Dict1, Debug, Depth) :-
    op(Priority,Type,Op), !,                  % Operator decl must be exe-
    write_clause((:- op(Priority,Type,Op))), % cuted and also copied.
    read(Next), ghccomp1(Next, Dict0,Dict1, Debug, Depth).
ghccomp1((:- X),      Dict0,Dict1, Debug, Depth) :-
 /* not_functor(X, mode, 1), not_functor(X, op, 3) */ !,
    errmsg('Illegal directive: ', (:- X)),
    read(Next), ghccomp1(Next, Dict0,Dict1, Debug, Depth).
```

```
ghccompl(X,           Dict0,Dict3, Debug, Depth) :-
 /* clause_with_positive_atom(X), !, */
  ( X=(Head:-_), !; X=Head ),
    functor(Head,P,A),
  ( lookup_dict(P/A, Dict0, _,_,_), !, Dict1=Dict0;
    notify_compilation(P,A),
    write_mode_decl(P,A), write_callmsg_clause(P,A,P, Debug),
    Dict1=[$$(P/A, P,'$NOMODE',Debug,64/*@*/)|Dict0] ),
  ( c_clause(X, C, Dict1,Dict2, Debug, Depth), !, write_clause(C);
    errmsg('Cannot compile: ', X), Dict2=Dict1 ),
    read(Next), ghccompl(Next, Dict2,Dict3, Debug, Depth).

:- mode complain_unable_to_open(?).
complain_unable_to_open(F) :- /* !, */
    fileerrors, display('Cannot open file: '), writeuser(F), ttynl.

%%% WRITING OBJECT CLAUSES %%%

:- mode write_clause(?).
write_clause(X) :-
    make_writable(X,Y, 0,_), writeq(Y), put("."), nl, fail.
write_clause(_).

:- mode make_writable(?,-, +,-).
make_writable(X,          Y, V0,V1) :- var(X), !,
    X='$VAR'(V0), Y=X, V1 is V0+1.
make_writable('$VAR'(X), Y, V0,V1) :- !, Y='$VAR'(X), V0=V1.
                              % Variable already made ground ('numbervar'ed)
make_writable('$REF'(X), Y, V0,V1) :- !, make_writable(X,Y, V0,V1).
make_writable('$WAIT'(X),Y, V0,V1) :- !, make_writable(X,Y, V0,V1).
make_writable('$INT'(X), Y, V0,V1) :- !, make_writable(X,Y, V0,V1).
make_writable(X,          Y, V0,V1) :- /* nonvariable(X), !, */
    functor(X,F,A), functor(Y,F,A), make_writable_args(0,A, X,Y, V0,V1).

:- mode make_writable_args(+,+, +,+, +,-).
make_writable_args(K,N, X,Y, V0,V2) :- K<N, !,
    K1 is K+1, arg(K1,X,Xk), make_writable(Xk,Yk, V0,V1), arg(K1,Y,Yk),
    make_writable_args(K1,N, X,Y, V1,V2).
make_writable_args(N,N, _,_, V0,V1) :- /* !, */ V0=V1.

%%% MAKING ADDITIONAL CLAUSES %%%

:- mode write_mode_decl(+,+).
write_mode_decl(P,A) :- /* !, */
    A7 is A+7, write_clause((:- public P/A7)),
    functor(Mode,P,A), fill_questions(A, Mode),
    extend_atom(Mode,Modex, +,+,-,+,+,+,+), write_clause((:- mode Modex)).

:- mode fill_questions(+, +).
fill_questions(0, _) :- !.
fill_questions(K, X) :- /* K>0, !, */
    arg(K,X,?), K1 is K-1, fill_questions(K1, X).

:- mode write_callmsg_clause(+,+,+, +).
write_callmsg_clause(P,A,Pspy, debug) :- !,
    functor(H,P,A), extend_atom(H,Headx, _,_,_,_,_,Myid,_),
    functor(Hspy,Pspy,A), copy_args(A, H,Hspy),
    write_clause((Headx :- ghcspied(Pspy,A), trace_call(Myid,Hspy), fail)).
write_callmsg_clause(_,_,_,     _     ) /* :- ! */.

:- mode write_last_clause_group(+, +).
write_last_clause_group([$$(P/A, P, _,Debug,_)|L], Depth) :- !,
    write_suspmsg_clause(P, A, P, Debug, Depth),
    write_susp_clause(P,A),
    write_last_clause_group(L, Depth).
write_last_clause_group([$$(P/A, IP,_,Debug,_)|L], Depth) :- /*P\==IP,*/ !,
```

```
    write_callsuspmsg_clause(P,A,P, Debug, Depth),
    write_susp_clause(P,A),
    write_suspmsg_clause(IP,A,P, Debug, Depth),
    write_susp_clause(IP,A),
    write_last_clause_group(L, Depth).
write_last_clause_group([],                          _    ) /* :- ! */.

:- mode write_suspmsg_clause(+,+,+, +, +).
write_suspmsg_clause(P,A,Pspy, debug, Depth) :- !,
    functor(H,P,A), extend_atom(H,Headx, RC0,_,_,_,_,Myid,_),
  ( Depth==nodepth, !, RC=RC0; RC=1 ),
    write_clause((Headx :- ghcspied(Pspy,A),
                           trace_suspension(RC,Myid), fail)).
write_suspmsg_clause(_,_,_,    _,     _    ) /* :- ! */.

:- mode write_callsuspmsg_clause(+,+,+, +, +).
write_callsuspmsg_clause(P,A,Pspy, debug, Depth) :- !,
    functor(H,P,A), extend_atom(H,Headx, RC0,_,_,_,_,Myid,_),
    functor(Hspy,Pspy,A), copy_args(A, H,Hspy),
  ( Depth==nodepth, !, RC=RC0; RC=1 ),
    write_clause((Headx :- ghcspied(Pspy,A), trace_call(Myid,Hspy),
                           trace_suspension(RC,Myid), fail)).
write_callsuspmsg_clause(_,_,_,    _,     _    ) /* :- ! */.

:- mode write_susp_clause(+,+).
write_susp_clause(P,A) :- /* !, */
    functor(H,P,A),
    extend_atom(H,Goalx, RCmax,Ch2,Ct2,Dnd2,RCmax,Myid,Nextid2),
    extend_atom(H,Headx, _,    [$(Gx, Ch,Ct,Dnd,Nextid)|Ch],
                                   [$(Goalx, Ch2,Ct2,Dnd2,Nextid2)|Ct],
                                       Dnd, RCmax,Myid,Nextid ),
    write_clause((Headx :- incore(Gx))).

:- mode write_transmit_clause(+).
write_transmit_clause(Mdecl) :- /* !, */
    functor(Mdecl,P,A), make_internal_name(P,IP), A7 is A+7,
    functor(Headx,P,A7), functor(Goalx,IP,A7),
    transmit_args(0,A7, Mdecl, Headx, Goalx,Body),
    write_clause((Headx :- Body)).

% transmit_args(Arg_position,Arity, Mode, Head, Subgoal,Body)
% Body = suquence of calls to 'nonvar/1' followed by Subgoal.
:- mode transmit_args(+,+, +, +, +,-).
transmit_args(K,N, Mdecl, Headx, Goalx,Body0) :- K<N, !,
    K1 is K+1, arg(K1,Headx,Xk1), arg(K1,Goalx,Xk1),
  ( arg(K1,Mdecl,+), !, Body0=(nonvar(Xk1), Body1);
    Body0=Body1 ),                    % Mode is not '+' or K1>arity(Mdecl)
    transmit_args(K1,N, Mdecl, Headx, Goalx,Body1).
transmit_args(N,N, _,     Headx, Goalx,Body ) :- /* !, */
    N6 is N-6, arg(N6,Headx,Xn6), Body=(Xn6>0, !, Goalx).

:- mode make_internal_name(+,-).
make_internal_name(P,IP) :- /* !, */
    name(P,Pl), name(IP,[36,36,36/*$$$*/|Pl]).

%%% COMPILING CLAUSES %%%

%% Decomposing and Composing Clauses %%

% c_clause(GHC_clause, Prolog_clause,
%          Current_dict, New_dict, Debug_mode?, Depth_first?).
% c_clause may fail if an uninstantiated atom appears.
:- mode c_clause(+, -, +,-, +, +).
c_clause((Head:-_),    _, _,    _,     _,     _    ) :- var(Head), !,
    fail.
c_clause((Head:-X),    C, Dict0,Dict1, Debug, Depth) :-
```

```
      nonvar(X), X=(Guard|Body), !,
      c_head_guard(Head, Guard, PH, PB0,(!,PB1), Dict0, RC0, Depth),
      extend_atom(PH,PHx, RC0,Ch,Ct,_,RCmax,_,Nextid),
      separate_unify(Body, Bnounify, Bunify),
      c_body_unify(Bunify, Nextid,0,Id1, PB1,PB2, Debug),
      c_body_arith(Bnounify,Bnounify_rest, PB2,PB3, Debug),
      put_queue(Bnounify_rest, Q0,Q1, RC1,RCmax,
                PB3,PB4, Nextid,Id1,Id2, Head, Dict0,Dict1, Debug),
      c_update_nextid(Debug, Nextid,Id2,Newid, PB4,PB5),
      c_make_tail(Depth, Q0,Q1, RC0,RC1, Ch,Ct, Newid, PB5),
      C=(PHx:-PB0).
c_clause((Head:-Body), C, Dict0,Dict1, Debug, Depth) :-
 /* not_functor(Body, '|', 2) */ !,
     errmsg('Warning: No commitment operator:  ', (Head :- Body)),
     c_clause((Head :- true|Body), C, Dict0,Dict1, Debug, Depth).
c_clause(Head,          C, Dict0,Dict1, Debug, Depth) :-
 /* not_functor(Head, ':-', 2) */ !,
     errmsg('Warning: No commitment operator:  ', Head),
     c_clause((Head :- true|true), C, Dict0,Dict1, Debug, Depth).

% separate_unify(Goals, Nounify, Unify).
% Nounify and Unify become LINEAR conjunction of goals.
:- mode separate_unify(?, -, -).
separate_unify(G, NU, U) :- separate_unify(G, NU,true, U,true).

:- mode separate_unify(?, -,?, -,?).
separate_unify(X,       _,  _,   _, _ ) :- variable(X), !, fail.
separate_unify((X,Y), NU0,NU2, U0,U2) :- !,
    separate_unify(X, NU0,NU1, U0,U1), separate_unify(Y, NU1,NU2, U1,U2).
separate_unify(X=Y,    NU0,NU1, U0,U1) :- !, NU0=NU1, U0=(X=Y,U1).
separate_unify(G,      NU0,NU1, U0,U1) :- /* not_unification(G), !, */
    NU0=(G,NU1), U0=U1.

%% Compiling Heads and Guards %%

% c_head_guard(GHC_head, GHC_guard,
%     Prolog_head, Prolog_body, Prolog_body_tail, Current_dict,
%     Current_reduction_count, Depth_first?)
:- mode c_head_guard(+, ?, -, -,+, +, -, +).
c_head_guard(Head, Guard, PH, PB0,PB4, Dict, RC, Depth) :-
    functor(Head,P,A), lookup_dict(P/A, Dict, IP,Mode,_),
    functor(PH,IP,A),
  ( separate_unify(Guard, Gnounify, Gunify), !,
    ( call(Gunify), !,
        c_args(0,A, Head, PH, PB0,PB1, Mode),
      ( c_guard(Gnounify, PB1,PB2, PB3,PB4), !,
        ( P==IP, Depth==nodepth, !,
            PB2=(RC>0,PB3);
          PB2=PB3 );
        errmsg('Guards can call predefined predicates only:  ', Guard),
        PB1=(fail,PB4) );
      errmsg('Warning: Unsucceedable guard:  ', Guard),
      PB0=(fail,PB4) );
    errmsg('Uninstantiated variables in guard:  ', Guard),
    PB0=(fail,PB4) ).

:- mode c_args(+,+, +, -, -,?, +).
c_args(K,N, Head, PH, PB0,PB2, M) :- K<N, !,
    K1 is K+1, arg(K1,Head,Ak), arg(K1,PH,PHk),
  ( M=='$NOMODE', !, Mk=(?);  arg(K1,M,Mk) ),
    c_unify(Ak, PHk, PB0,PB1, Mk), c_args(K1,N, Head, PH, PB1,PB2, M).
c_args(N,N, _,     _,  PB0,PB1, _) :- /* !, */ PB0=PB1.

% c_unify(Original_argument, Generated_argument,
%         Unify_code, Unify_code_tail, Mode).
% Note that f(X,X) must be compiled into f(X,Y) :- X==Y.
```

```
% This is because X and Y must be unified with no substitutions.
% For integer comparison, =:= and =\= are faster than = and \=.
:- mode c_unify(?, -, -,?, +).
c_unify(V,          X, G0,G1, M) :- var(V), !,
    X=V, G0=G1, V='$REF'(Y), mark(Y, M).
c_unify('$REF'(V), X, G0,G1, M) :- !, G0=(V==X,G1), mark(X, M).
c_unify([],         X, G0,G1, M) :- !,
  ( M=='+', !, X=[], G0=G1; G0=(nonvar(X),unil(X),G1) ).
c_unify(A,          X, G0,G1, M) :- atomic(A), !,
  ( M=='+', !, X=A, G0=G1; G0=(nonvar(X),uskel(X,A),G1) ).
c_unify([H|T],      X, G0,G3, M) :- !,
  ( M=='+', !, X=[H0|T0], G0=G1; G0=(nonvar(X),ulist(X,H0,T0),G1) ),
    c_unify(H, H0, G1,G2, ?), c_unify(T, T0, G2,G3, ?).
c_unify(S,          X, G0,G2, M) :- /* other_structure(S), !, */
    functor(S,F,A), functor(S0,F,A),
  ( M=='+', !, X=S0, G0=G1; G0=(nonvar(X),uskel(X,S0),G1) ),
    c_unify_args(0,A, S, S0, G1,G2).

:- mode c_unify_args(+,+, +, +, -,?).
c_unify_args(K,N, S, X, G0,G2) :- K<N, !,
    K1 is K+1, arg(K1,S,Sk), arg(K1,X,Xk),
    c_unify(Sk, Xk, G0,G1, ?), c_unify_args(K1,N, S, X, G1,G2).
c_unify_args(N,N, _, _, G0,G1) :- /* !, */ G0=G1.

% c_guard(Original_goals, Before_goals, Before_goals_tail,
%         After_goals, After_goals_tail)
% Before_goals are called before checking the reduction counter, while
% After_goals are not.
:- mode c_guard(+, -,?, -,?).
c_guard(true,       Gb0,Gb1, Ga0,Ga1) :- !, Gb0=Gb1, Ga0=Ga1.
c_guard((X,Y),      Gb0,Gb2, Ga0,Ga2) :- !,
    c_guard(X, Gb0,Gb1, Ga0,Ga1), c_guard(Y, Gb1,Gb2, Ga1,Ga2).
c_guard(prolog(X), Gb0,Gb1, Ga0,Ga1) :- !, Gb0=Gb1, Ga0=(X,Ga1).
                    % prolog(X) must be executed after checking the
                    % reduction counter, since it may cause side effects.
c_guard(X,          Gb0,Gb1, Ga0,Ga1) :- /* other_form(X), !, */
    c_guard_2(X, Gb0,Gb1), Ga0=Ga1.

% c_guard_2 fails if the first arg is not a system predicate.
:- mode c_guard_2(+, -,?).
c_guard_2(wait(X), Gb0,Gb1) :- !, c_w(X, Gb0,Gb1).
c_guard_2(X\=Y,    Gb0,Gb1) :- atomic(X), !, c_w(Y, Gb0,(X\==Y,Gb1)).
c_guard_2(X\=Y,    Gb0,Gb1) :- atomic(Y), !, c_w(X, Gb0,(X\==Y,Gb1)).
c_guard_2(X\=Y,    Gb0,Gb1) :- /* non_atomic(X), non_atomic(Y) */ !,
    Gb0=(ghodif(X,Y),Gb1).
c_guard_2(X:=Y,    Gb0,Gb2) :- appeared_in_head(X), !,
    c_iw(X, Gb0,Gb1), c_iw_all(Y, Gb1,(X is Y,Gb2)).
c_guard_2(X:=Y,    Gb0,Gb1) :- /* not_appeared_in_head(X) */ !,
    c_iw_all(Y, Gb0,(X is Y,Gb1)).
c_guard_2(X<Y,     Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X<Y,  Gb1)).
c_guard_2(X>Y,     Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X>Y,  Gb1)).
c_guard_2(X=<Y,    Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X=<Y, Gb1)).
c_guard_2(X>=Y,    Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X>=Y, Gb1)).
c_guard_2(X=:=Y,   Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X=:=Y,Gb1)).
c_guard_2(X=\=Y,   Gb0,Gb1) :- !, c_iw_2(X, Y, Gb0,(X=\=Y,Gb1)).

:- mode c_w(?, -,?).
c_w(X,          G0,G1) :- var(X), !, G0=(nonvar(X),G1), X='$WAIT'(_).
c_w('$REF'(X), G0,G1) :- !, c_w(X, G0,G1).
c_w(X,          G0,G1) :-
 /* ( X='$WAIT'(_); X='$INT'(_); not_variable(X) ), !, */ G0=G1.

:- mode c_iw(?, -,?).
c_iw(X,          G0,G1) :- var(X), !, G0=(integer(X),G1), X='$INT'(_).
c_iw('$REF'(X),  G0,G1) :- !, c_iw(X, G0,G1).
c_iw('$WAIT'(X), G0,G1) :- !, c_iw(X, G0,G1).
```

```
c_iw(_,          G0,G1) :- /* ( X='$INT'(_); not_variable(X) ), ! */ G0=G1.

:- mode c_iw_2(?,?, -,?).
c_iw_2(X,Y, G0,G2) :- /* !, */ c_iw_all(X, G0,G1), c_iw_all(Y, G1,G2).

:- mode c_iw_all(?, -,?).
c_iw_all(X,          G0,G1) :- var(X), !, G0=(integer(X),G1), X='$INT'(_).
c_iw_all('$REF'(X),  G0,G1) :- !, c_iw_all(X, G0,G1).
c_iw_all('$WAIT'(X), G0,G1) :- !, c_iw_all(X, G0,G1).
c_iw_all('$INT'(_),  G0,G1) :- !, G0=G1.
c_iw_all(X,          G0,G1) :- /* structure(X), !, */
    functor(X,_,A), c_iw_args(0,A, X, G0,G1).

:- mode c_iw_args(+,+, ?, -,?).
c_iw_args(K,N, X, G0,G2) :- K<N, !,
    K1 is K+1, arg(K1,X,Xk), c_iw_all(Xk, G0,G1),
    c_iw_args(K1,N, X, G1,G2).
c_iw_args(N,N, _, G0,G1) :- /* !, */ G0=G1.

%% Compiling Bodies %%

:- mode c_body_unify(+, -,+,-, -,?, +).
c_body_unify((X=Y,Bunify), Id,Id0,Id2, PB0,PB3, Debug) :- !,
    c_id_setup(Debug, PB0,PB1, Id,Id0,Id1, Myid),
  ( variable(X), !, R=X, L=Y;  % for efficiency on DEC-10 Prolog
    L=X, R=Y ),
  ( Debug==nodebug, !, PB1=(ubody(L,R), PB2);
    PB1=('ubody?'(L,R,Myid), PB2) ),
    c_body_unify(Bunify, Id,Id1,Id2, PB2,PB3, Debug).
c_body_unify(true,          _, Id0,Id1, PB0,PB1, _    ) :- /* !, */
    Id0=Id1, PB0=PB1.

:- mode c_id_setup(+, -,?, ?,+,-, -).
c_id_setup(nodebug, Ig0,Ig1, _, Id0,Id1, Myid) :- !,
    Myid=0, Ig0=Ig1, Id1 is Id0 .
c_id_setup(debug,   Ig0,Ig1, Id,Id0,Id1, Myid) :- /* !, */
    ( Id0=:=0, !, Myid=Id, Ig0=Ig1; Ig0=(Myid is Id+Id0, Ig1) ),
    Id1 is Id0+1.

% Initial sequence of assignments that need no waiting is processed on the
% spot.
:- mode c_body_arith(+,-, -,?, +).
c_body_arith((V:=E,Body0),Body1, PB0,PB2, nodebug) :-
    c_iw_all(E, W,true), W=true, !,  % check if waiting is needed for E
  ( integer(E), !, PB0=(ubody(E,V), PB1);
    PB0=(V2 is E, ubody(V2,V), PB1) ),
    c_iw(V, _,_),                    % mark V as 'having an integer value'
    c_body_arith(Body0,Body1, PB1,PB2, nodebug).
c_body_arith(Body0,       Body1, PB0,PB1, _       ) :-
 /* top element of Body0 is not a non-suspending assignment, !, */
    Body0=Body1, PB0=PB1.

% Put_queue is called also at run time to make a goal queue.  It fails if
% there exists an uninstantiated goal.
% put_queue(Body_goals, Goal_queue_head, Goal_queue_tail,
%           New_reduction_count, Max_reductions,
%           Id_calc_goal_head, Id_calc_goal_tail,
%           Next_id, Initial_id_offset, Final_id_offset,
%           Head, Dict_old, Dict_new, Debug_mode?)
:- mode put_queue(?, -,-, ?,?, -,?, ?,+,-, +, +,-, +).
put_queue(X,      _, _,  _,_,_,_,_,_,  _,  _,   _,    _,    _,     _    ) :-
    variable(X), !, fail.
put_queue((X,Y), Q0,Q2, RC1,RCmax,
                        Ig0,Ig2, Id,Id0,Id2, Head, Dict0,Dict2, Debug) :- !,
    put_queue(X, Q0,Q1, RC1,RCmax,
                        Ig0,Ig1, Id,Id0,Id1, Head, Dict0,Dict1, Debug),
```

```
    put_queue(Y, Q1,Q2, RC1,RCmax,
                        Ig1,Ig2, Id,Id1,Id2, Head, Dict1,Dict2, Debug).
put_queue(true,  Q0,Q1, _, _,
                        Ig0,Ig1, _, Id0,Id1, _,    Dict0,Dict1, _    ) :- !,
    Q0=Q1, Ig0=Ig1, Id0=Id1, Dict0=Dict1.
put_queue(V:=E,  Q0,Q1, RC1,RCmax,
                        Ig0,Ig1, Id,Id0,Id1, Head, Dict0,Dict1, Debug) :-
    Head\=='$TOP', !,
    make_arith_head(Head, V,E, AH,AP,AA, Dict0,Dict1),
    write_mode_decl(AP,AA),
    extend_atom(AH,AHx, RC1,Ch,Ct,Dnd,RCmax,Myid,Nextid),
  ( Debug==debug, !,
    write_clause((AHx :- ghcspied(':=',2), trace_call(Myid,V:=E), fail));
    true ),
    c_iw_all(E, G,(V2 is E, !, ubody(V2,V), do_next(Ch,Ct,Nextid))),
    write_clause((AHx :- G)),
  ( Debug==debug, !,
    write_clause((AHx :- ghcspied(':=',2),
                         trace_suspension(1,Myid), fail));
    true ),
    write_susp_clause(AP,AA),
    Q0=[$(AHx, Ch,Ct,Dnd,Nextid) | Q1],
    c_id_setup(Debug, Ig0,Ig1, Id,Id0,Id1, Myid).
put_queue(X,     Q0,Q1, RC1,RCmax,
                        Ig0,Ig1, Id,Id0,Id1, _,    Dict0,Dict1, Debug) :-
 /* user_defined_goal(X), !, */
    extend_atom(X,Xx, RC1,Ch,Ct,Dnd,RCmax,Myid,Nextid),
    Q0=[$(Xx, Ch,Ct,Dnd,Nextid) | Q1],
    c_id_setup(Debug, Ig0,Ig1, Id,Id0,Id1, Myid),
    Dict0=Dict1.

:- mode make_arith_head(+, ?,?, -,-,-, +,-).
make_arith_head(Head, V,E, AH,AP,AA, Dict0,Dict1) :-
    functor(Head,P,A), inc_arith(P/A, Dict0,Dict1, Arithid),
    name(P,Pl), name(AP,[36,36,36,Arithid|Pl]),
    analyze_exp(E, L,[]),
    AH =.. [AP,V|L], functor(AH,_,AA).

:- mode analyze_exp(?, -,?).
analyze_exp(E, L0,L1) :- variable(E), !, L0=[E|L1].
analyze_exp(E, L0,L1) :- /* not_variable(E), !, */
    functor(E, _, A), analyze_exp_args(0,A, E, L0,L1).

:- mode analyze_exp_args(+,+, +, -,?).
analyze_exp_args(K,N, E, L0,L2) :- K<N, !,
    K1 is K+1, arg(K1,E,Ek), analyze_exp(Ek, L0,L1),
    analyze_exp_args(K1,N, E, L1,L2).
analyze_exp_args(N,N, _, L0,L1)  :- /* !, */ L0=L1.

:- mode c_update_nextid(+, -,+,-, -,-).
c_update_nextid(debug, Nextid,Id1,Newid, PB4,PB5) :- Id1>0, !,
    PB4=(Newid is Nextid+Id1, PB5).
c_update_nextid(Debug, Nextid,Id1,Newid, PB4,PB5) :-
 /* ( Debug==nodebug; Id1=<0 ), !, */
    Newid=Nextid, PB4=PB5.

% c_make_tail(Depth_first?, Body_queue_head,Body_queue_tail,
%             Reduction_count_old,Reduction_count_new,
%             Cont_h,Cont_t, New_goal_id, Prolog_body).
:- mode c_make_tail(+, ?, -,-,-,-,-,-,-).
c_make_tail(_,     H,          T, _,  _,   Ch,Ct, Newid, PB5) :- H==T, !,
    PB5=(Ch=[$(Gx, Ch1,Ct,nd,Newid)|Ch1], incore(Gx)).
c_make_tail(Depth, [Qitem|H1],T, RC0,RC1, Ch,Ct, Newid, PB5) :- /* !, */
    c_update_rc(Depth, RC0,RC1, PB5,PB6),
    Qitem= $(PB6, H1,Ct,nd,Newid),
    T=Ch.                   % The d-list [Qitem|H1]-T is connected to Ch-Ct
```

```
:- mode c_update_rc(+, -,-, -,-).
c_update_rc(nodepth, RC0,RC1, PB5,PB6) :- !,
    PB5=(RC1 is RC0-1, PB6).
c_update_rc(depth,   RC0,RC1, PB5,PB6) :- /* !, */
    PB5=PB6, RC1=RC0.

%%% MISCELLANEOUS ROUTINES %%%

%% Predicate Dictionary Management %%

:- mode lookup_dict(+, +, -,-,-).
lookup_dict(Pred, [$$(Pred, IP,Mode,Debug,_)|_], IP,Mode,Debug) :- !.
lookup_dict(Pred, [Item|Dict],                   IP,Mode,Debug) :-
 /* Item=$$(Pred2, _,_,_,_), Pred\==Pred2, !, */
    lookup_dict(Pred, Dict, IP,Mode,Debug).

:- mode inc_arith(+, +,-, -).
inc_arith(Pred, Dict0,Dict1, Arithid) :- /* !, */
    find_item(Pred, Dict0,Dict_rest, Item),
    Item= $$(Pred, IP,Mode,Debug,Arithid0), Arithid is Arithid0+1,
    Dict1=[ $$(Pred, IP,Mode,Debug,Arithid)|Dict_rest].

:- mode find_item(+, +,-, -).
find_item(Pred, [I|Dict],Dict_rest, Item) :- arg(1,I,Pred), !,
    Dict_rest=Dict, Item=I.
find_item(Pred, [I|Dict], Dict_rest, Item) :- /* not_arg(1,I,Pred), !, */
    Dict_rest=[I|Dict_rest1], find_item(Pred, Dict,Dict_rest1, Item).

%% Constructing Atomic Formulas %%

:- mode extend_atom(+, -, ?,?,?,?,?,?,?).
extend_atom(X, Xx, RC,Ch,Ct,Dnd,RCmax,Myid,Nextid) :- /* !, */
    functor(X,F,A),
    A1 is A+1, A2 is A+2, A3 is A+3, A4 is A+4,
    A5 is A+5, A6 is A+6, A7 is A+7,
    functor(Xx,F,A7), copy_args(A,X,Xx),
    arg(A1,Xx,RC),    arg(A2,Xx,Ch),   arg(A3,Xx,Ct),     arg(A4,Xx,Dnd),
    arg(A5,Xx,RCmax), arg(A6,Xx,Myid), arg(A7,Xx,Nextid).

:- mode copy_args(+, +,+).
copy_args(0, _,_ ) :- !.
copy_args(K, X,Xx) :- /*K>0, !,*/
    arg(K,X,Xk), arg(K,Xx,Xk), K1 is K-1, copy_args(K1, X,Xx).


%% Analyzing Mode Declaration %%

% check_mode(Mdecl) succeeds if Mdecl contains the '+' mode, fails
% otherwise.  Also, it complains of modes other than '+' and '?'.
:- mode check_mode(+).
check_mode(Mdecl) :- /* !, */
    functor(Mdecl,_,A), check_mode_args(0,A, Mdecl, no).
:- mode check_mode_args(+,+, +, +).
check_mode_args(K,N, Mdecl, YN0) :- K<N, !,
    K1 is K+1, arg(K1,Mdecl,M), check_mode_arg(M, YN0,YN1),
    check_mode_args(K1, N, Mdecl, YN1).
check_mode_args(N,N, _,     yes) /* :- ! */.  % fails if 4th arg is 'no'.

:- mode check_mode_arg(?, +,-).
check_mode_arg(+, _,  YN1) :- !, YN1=yes.
check_mode_arg(?, YN0,YN1) :- !, YN1=YN0.
check_mode_arg(M, YN0,YN1) :- /* !, */
    display('Illegal mode specifier: '), display(M), ttynl, YN1=YN0.

%% Analyzing Variables in Clauses %%
```

```
:- mode variable(?).
variable(X          ) :- var(X), !.
variable('$REF'(_) ) :- !.
variable('$WAIT'(_)) :- !.
variable('$INT'(_) ) /* :- ! */.

:- mode appeared_in_head(?).
appeared_in_head(X          ) :- var(X), !, fail.
appeared_in_head('$REF'(_)) /* :- ! */.

:- mode mark(-, +).
mark(X, +) :- !, X='$WAIT'(_).
mark(_, M) /* :- M\==+', ! */.

%% Displaying Messages %%

:- mode notify_compilation(+,+).
notify_compilation(P,A) :-
    display(P), ttyput("/"), display(A), display(', '), ttyflush.

:- mode errmsg(+, ?).
errmsg(Msg, Clause) :- /* !, */
    telling(File), tell(user), nl,
    write(Msg), write_clause(Clause), nl, tell(File).

:- mode writeuser(?).
writeuser(T) :- /* !, */ telling(File), tell(user), write(T), tell(File).
```