知識工学と人工知能 59－6
（1988. 7. 7）

制約付き性質継承に関する並列アルゴリズム

毛受　哲、伊藤英則、森田幸伯

(財)新世代コンピュータ技術開発機構

例外のある多重継承に関して今まで色々な研究がされてきたが、それらはリンクがあれば無条件に推論できるものと考えていた。しかし、動的な情報や時制を表現しようとするには、それでは不十分である。これに対し、制約の概念を意味ネットワークに導入することは有効である。また、知識は表現するだけではなく、効率良く扱えることも必要である。

本論文では継承階層ネットワーク及び制約付き継承階層ネットワークについて述べた後、後者に対する並列推論アルゴリズムを示し、並列論理型言語GHCによる実現について述べる。これは、Touretzkyらの定義する信心的な解の1つを求めるものの拡張である。

A Parallel Algorithm for Inheritance Hierarchies with Constraints

Satoshi Menju, Hidenori Itoh and Yukihiro Morita

ICOT Research Center
21F, Mita Kokusai Bldg.,
1-4-28, Mita, Minato-ku, Tokyo, 108, Japan

This paper introduces the concept of constraints in multiple inheritance hierarchies with exceptions. We add constraints to the links of inheritance networks, then consider links with satisfied constraints only, ignoring links with unsatisfied constraints. This method can increase the expressive power.

This paper also describes a parallel algorithm for inheritance hierarchies with constraints. It terminates in O(n)-time, where n is the length of the longest path of an inheritance network including constraints. The algorithm obtains one of the solutions produced by the credulous reasoners of Touretzky and Etherington. We also implement the algorithm in a parallel logic programming language, Guarded Horn Clauses (GHC).

1. Introduction

Algorithms to process multiple inheritance with exceptions have been discussed in [1,2,3,4]. In these algorithms, the property inheritance between two objects is represented statically by a link. This paper introduces the idea of constraint attachment to properties in the representation of more complex multiple inheritance problems. The idea of constraints has been argued in logic programming languages [5,6] to solve more complex logical problems. This also seems to be a kind of knowledge representation language using constraints. From the point of view of constraints and their processing in parallel, the parallel processing algorithm presented in this paper is more powerful than the previous ones. It is expected to make a new paradigm of knowledge representation or knowledge management language.

In this research, not only representation power but also processing efficiency are important. This paper first explains the inheritance network model whose link contains added constraint information, and gives efficient parallel algorithms to process it. These algorithms are written in Guarded Horn Clauses (GHC) [7], a parallel logic programming language defined as the kernel language of the Fifth Generation Computer System.

2. Inheritance Network

This section lists some basic preparations for the following discussion. First, a multiple inheritance network is defined. An inheritance network is composed of a set of individuals and predicates and a set of links.

$< p, +q >$ and $< p, -q >$ are reasoning links (or, more simply, links), where p is an individual or a predicate and q is a predicate. $< p, +q >$ is called an *is-a* link and $< p, -q >$ is called an *is-not-a* link. The $< p, +q >$ link exists in an inheritance network iff p is reasoned to be q, and the $< p, -q >$ link exists iff p is reasoned to be not q. When a link exists from p to q, it is said that p is a parent of q and q is a child of p.

Graphically, an individual is denoted by a white node, a predicate is denoted by a black node, an *is-a* link is denoted by an arrow and an *is-not-a* link is denoted by a crosshatched arrow.

A reasoning path, $P(p_1, p_n)$, from node p_1 to node p_n is composed of links $< p_i, \alpha p_{i+1} >$, where $1 \leq i \leq n-1$ and α is $+$ or $-$. The logical length of path $P(p_1, p_n)$ is $n-1$.

An *is-a* link is a positive path, and an *is-not-a* link is a negative path. If a positive path, $P(x_1, x_n)$, and an *is-a* link, $< x_n, +y >$, exist, then path $P(x_1, y)$ is a positive path. If a positive path, $P(x_1, x_n)$, and an *is-not-a* link, $< x_n, -y >$, exist, then path $P(x_1, y)$ is a negative path. All paths which do not fulfil these conditions are called meaningless paths. For example, a path, $(< a, +b >, < b, -c >, < c, +d >)$, is a meaningless path.

Now we assume a few restrictions on our model as follows.

(1) Every pair of nodes has at most one link, and
(2) no paths make a loop.

Here, a positive node set, a negative node set and a non-conclusion node set are defined. A positive node set of the starting node, s, is a set of nodes which can be reached through the positive path from s. A negative node set of s is a set of nodes which can be reached through the negative path from s. If a node has both positive and negative paths from s, then the node should be decided by the path length and starting node priority rules, which are given in section 4, showing whether it belongs to the positive or negative node set. A non-conclusion node set of s is a set of nodes which can be reached through a meaningless path from s.

A triplet of these sets is a resolution of the inheritance network. There are two ways of finding the resolutions. One is credulous reasoning, and the other is sceptical reasoning, whose definitions are given in [1,2].

Credulous reasoning gathers all nodes which can be drawn through positive and/or negative paths in an inheritance network. Sceptical reasoning does not gather nodes which can be drawn through both positive and

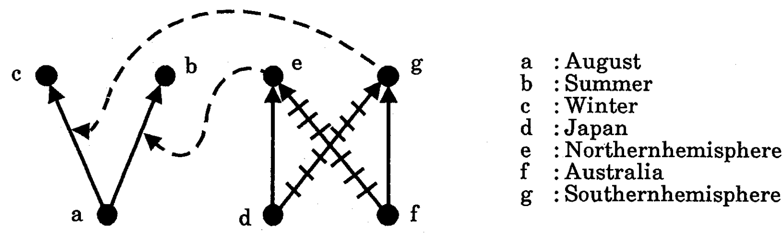


Figure 1 Simple example

negative paths in an inheritance network. It gathers only nodes which can be drawn exclusively through positive or negative paths.

We have already shown a parallel reasoning algorithm to find one of the resolutions using credulous reasoning [1], and have also shown that the processing time of order $O(n)$ is $C \times n$, that is, linearly, where n is the length of the longest path in the multiple inheritance network [4].

The following sections expand the parallel reasoning algorithm for the inheritance network whose link has constraints.

3. Inheritance Network with Constraints

This section discusses an inheritance network with constraints.

A link with an added constraint is defined as follows. $< p, \alpha q, C >$ is a link with an added constraint, C, where α is $+$ or $-$. If α is $+$, then it is called an *is-a* link with constraint C, and if α is $-$, then it is called an *is-not-a* link with constraint C. When C fulfils all conditions, p can be reasoned αq, and when C does not fulfil at least one of them, p cannot be reasoned αq. C is a set of conditions, that is, $C = \{\alpha s(x_i) | 1 \leq i \leq n\}$. Each node, x_i, has a condition αs.

Now, constraint $C = \{\alpha s(x)\}$ and link $< p, \alpha q >$ are given, where link $< p, \alpha q >$ is connected by a positive constraint link (denoted $---$ >) from node x, αs is $+s$, and condition $+s$ is true iff C acts on link $< p, \alpha q >$ as a positive constraint. Moreover, constraint $C = \{\alpha s(x)\}$ and link $< p, \alpha q >$ are given, where link $< p, \alpha q >$ is connected by a negative constraint link (denoted $+++$ >) from node x, αs is $-s$, and condition $-s$ is true iff C acts on link $< p, \alpha q >$ as a negative constraint. In these two cases only, constraint C is said to fulfil the conditions.

A condition, αs, at node x is given from outside initially or is an intermediate result of its reasoning.

Briefly, a link, $< p, \alpha q >$, with constraint C exists, C fulfils the positive or negative condition and link $< p, \alpha q >$ is connected by the positive or negative constraint link iff p is reasoned to be αq under C.

Here, a simple example is shown.

<August, +Summer, +Northernhemisphere> means that it is summer in August in the northern hemisphere, and *<August, +Winter, +Southernhemisphere>* means that it is winter in August in the southern hemisphere. In this example, *+Northernhemisphere* and *+Southernhemisphere* are the constraints of each link. Fig. 1 shows this.

Next, more examples are given.

(1) Priority

A link, $< a, +c, -b(b) >$, where constraint $-b(b)$ is predicate b is not true at node b, means that if a is reasoned not to be b, then a is reasoned to be c. This is shown in Fig. 2.

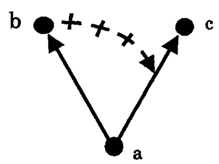


Figure 2 Priority

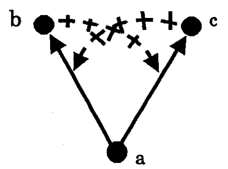


Figure 3 Exclusive

(2) Exclusive

$< a, +b, -c(c) >$ and $< a, +c, -b(b) >$ mean that a can be reasoned $+b$ or a can be reasoned $+c$ exclusively. This is also shown in Fig. 3.

(3) Default (Etherington and Reiter [3])

The example of Etherington and Reiter [3] is written in the following links with added constraints (Fig. 4).

$$< a, +d, >, < a, +b, >, < b, +c, >,$$
$$< b, -d, -a(a) >, < c, +d, -b(b) >$$

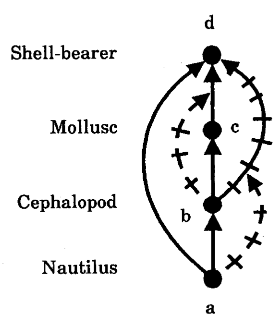


Figure 4 Default

4. Parallel Algorithm

This section shows a parallel algorithm to find one of the resolutions by credulous reasoning. Before showing it, we impose the following restriction on the inheritance network.

No cycle of the reasoning paths through the constraint links may exist in an inheritance network. The example in Fig. 3 has a cycle of reasoning paths through constraint links. Our parallel algorithm treats only acyclic inheritance networks.

Definitions are given.

n is a node and s is a starting node in an inheritance network. A triplet, $(mark(s), i(s), p(s))$, on n is defined as follows. $mark(s)$ is tm (truth mark), fm (false mark) or mm (meaningless mark). One of these marks is propagated from starting node s. $i(s)$ is the logical path length from s to n. $p(s)$ is the priority of starting node s.

Next, the mark propagation rule is defined as follows. Node b is assumed to have a triplet, $(tm(s), i(s), p(s))$, and node c is connected with b by a reasoning link. If a constraint, C, of link $< b, +c >$ / $< b, -c >$ fulfils the conditions, node c receives the $(tm(s), i + 1(s), p(s))$ / $(fm(s), i + 1(s), p(s))$ triplet from node b, and if a constraint, C, of link $< b, \alpha c >$ does not fulfil the conditions, node c receives the $(mm(s), \infty, \infty)$ triplet from node b.

Here, a constraint, C, from node x fulfils the conditions iff node x has a tm and link $< b, \alpha c >$ is connected by a positive constraint link from node x (Fig. 5 (a)), or node x has fm or mm and link $< b, \alpha c >$ is connected by a negative constraint link from node x (Fig. 5 (b)). C does not fulfil the conditions in any other case.

Node c always receives the $(mm(s), \infty, \infty)$ triplet from node b, whose mark is fm or mm. In this way, these triplets are propagated to every child of node b in parallel. Note that if a triplet has an mm, then both the logical length and the priority are ∞.

Next, if node c receives many triplets, $(mark, L, P)$, from parent nodes, then one triplet of node c from them must be defined by using the logical path length,

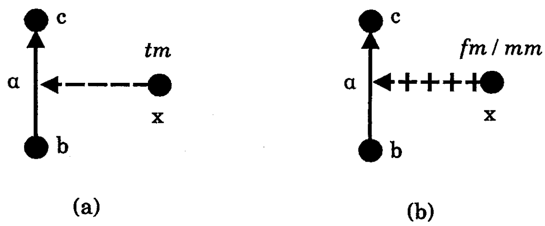


Figure 5 Satisfiability of constraints

L, from the starting node and the priority, P, of the starting node.

If $(mark(s_k), i(s_k), p(s_k))$s are received triplets at node c, $1 \leq k \leq l$ and l is the number of parents of node c, then triplet (M, L, P) of node c is defined as follows.

$P = p'(s_k)$, where $p'(s_k)$ is a minimum value among every priority, $p(s_k)$.

$L = i'(s_k)$, where $i'(s_k)$ is a maximum value of $i(s_k)$ whose priority is $p'(s_k)$.

$M = mark\ (s_k)$, where the $(mark(s_k)\,,\ i''(s_k)\,,\ p'(s_k))$ triplet exists, and $i''(s_k)$ is a minimum value of $i(s_k)$ whose priority is $p'(s_k)$. $mark(s_k)$ is propagated along the path of the shortest logical path length.

Note that M cannot decide the *tm* or *fm* definitely. In credulous reasoning, both *tm* and *fm* are resolutions. However, for the simplicity of the algorithm and processing efficiency, the user decides at the beginning of processing which mark, *tm* or *fm*, is to be used, according to the property of the given problem. Then our parallel algorithm is defined for one of the resolutions in credulous reasoning.

Because our inheritance network is restricted to be acyclic and the number of nodes is finite, the termination and soundness of this parallel algorithm are clear. This parallel algorithm stops at a time in the linear order of n, where n is the maximum length of the reasoning paths connected by constraint links which do not form any cycle.

Initially, a question node, a, is assigned and some constraint nodes, $c(i)$, of a are chosen, then these nodes work as the starting nodes under the above parallel algorithm. Generally, the highest priority is assigned to a question node a, that is to say, $p(a) = 0$.

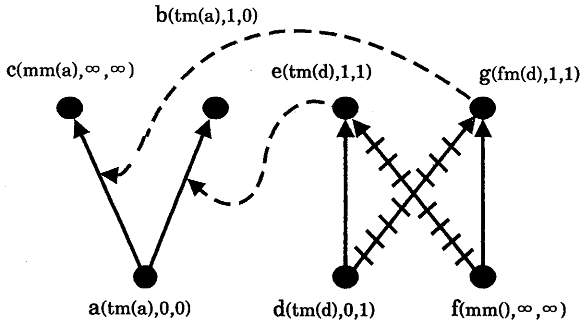
When this algorithm stops, then the attached nodes, $tm(a)$, are collected as the answers of the question to the inheritance network with constraints.

The simple example explained in section 3 is shown in Fig. 6 again. Here, we want to find an answer to the following question. Is it summer in August in Japan? In this example, ***August*** is assigned as a question node and ***Japan*** is assigned as a constraint node. Initially, the ***August*** node has a triplet, $(tm(August), 0, 0)$, and ***Japan*** has also a triplet, $(tm(Japan), 0, 1)$. Using the parallel algorithm, we can find the answer node, ***Summer***, with $tm(August)$ attached.

5. Programs in Parallel Logic Programming Language GHC

This section shows programs in parallel logic programming language Guarded Horn Clauses (GHC) [7] to find the resolution of a given inheritance network with constraints according to the parallel algorithm given in the previous section.

The logical structure of Fig. 1 is written as shown in Fig. 7 (b) in GHC. For comparison, a program with-



a : August
b : Summer
c : Winter
d : Japan
e : Northernhemisphere
f : Australia
g : Southernhemisphere

Figure 6 Example of a parallel algorithm

```
gen(Ms,N) :- true |
        node(a,Ms,Na,[]          ,TMa,_  ),
        node(b,Ms,Nb,[TMa]       ,_  ,_  ),
        node(c,Ms,Nc,[TMa]       ,_  ,_  ),
        node(d,Ms,Nd,[]          ,TMd,FMd),
        node(e,Ms,Ne,[TMd,FMf],_  ,_  ),
        node(f,Ms,Nf,[]          ,TMf,FMf),
        node(g,Ms,Ng,[TMf,FMd],_  ,_  ),
        N=[Na,Nb,Nc,Nd,Ne,Nf,Ng].
```

(a) Program for Fig. 1 without constraints

```
gen(Ms,N) :- true |
        node(a,Ms,Na,Pa,[]                        ,TMa,_  ),
        node(b,Ms,Nb,Pb,[(TMa,[(+,Ne)])]          ,_  ,_  ),
        node(c,Ms,Nc,Pc,[(TMa,[(+,Ng)])]          ,_  ,_  ),
        node(d,Ms,Nd,Pd,[]                        ,TMd,FMd),
        node(e,Ms,Ne,Pe,[(TMd,[]),(FMf,[])],_  ,_  ),
        node(f,Ms,Nf,Pf,[]                        ,TMf,FMf),
        node(g,Ms,Ng,Pg,[(TMf,[]),(FMd,[])],_  ,_  ),
        N=[(Na,Pa),(Nb,Pb),(Nc,Pc),(Nd,Pd),(Ne,Pe),(Nf,Pf),(Ng,Pg)].
```

(b) Program for Fig. 1

Figure 7 Sample program in GHC

out both constraints and priorities for the structure is shown in Fig. 7 (a). It cannot find a correct solution in this case. *gen* and *node* are processes defined in GHC. *Ms* is the input of the *gen* process, and $N = [(N\text{-}node, P\text{-}node)|node = a, b, c, d, e, f, g]$ is the output of the *gen* process. Here, *Ms* is a list of the triplets and $(N\text{-}node, P\text{-}node)$ is a pair of the mark and priority of each node.

The first argument of the *node* process is the name of the node in Fig. 1. The second argument is an input list of the *gen* process. The third argument is the mark given at the node. The fourth argument is the priority given at the node. The fifth argument is marks and constraints at the node. The sixth argument is a common variable defined in GHC, and it is used for the communication with the nodes connected by the *is-a* link from its node. The seventh argument is also a common variable and it is used for communication with the nodes connected by the *is-not-a* link from its node.

Each process node receives *Ms* from the process node connected with the common variable, calculates the mark, the logical length and the priority of its node, and then sends them through the common variable in the sixth and the seventh arguments.

The appendix gives the whole program in GHC.

6. Conclusion

This paper introduced the idea of inheritance network with constraints and a method of representation. This method of representation is expected to become a useful knowledge representation language, because, in many cases, the relations among objects represent constraints.

A parallel algorithm for inheritance networks with constraints is also shown. This algorithm seems to be an expansion of Touretzky's algorithm for static multiple inheritance without constraints.

This algorithm terminates at time $O(n)$ in linear order where n is the maximum length of the reasoning paths connected by constraint links, because our inheritance network has no cycle of reasoning paths through the constraint links.

The discussions in this paper dealt with the model and approaches to parallel processing, co-operative problem solving and logic programming with constraints. The feasibility and flexibility of the parallel algorithm must be verified by applying it to the larger real applications.

References

[1] Touretzky, D. S., *The Mathematics of Inheritance Systems*, Morgan Kaufmann Publishers, Los Altos, CA, 1986.

[2] Horty, J. F., Thomason, R. H. and Touretzky, D. S., A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks, *Proceedings of AAAI-87*, 1987, pp.358-363.

[3] Etherington, D. W. and Reiter, R., On Inheritance Hierarchies With Exceptions, *Proceedings of AAAI-83*, 1983, pp.104-108.

[4] Menju, S., Morita, Y. and Itoh, H., A Parallel Algorithm for Inheritance Network with Exceptions, *Proceedings of 36th IPSJ Conference 6P-8*, 1988, pp.1491-1492 (in Japanese).

[5] Colmerauer, A., Opening the Prolog III Universe: A New Generation of Prolog Promises Some Powerful Capabilities, *BYTE*, August 1987, pp.177-182.

[6] Jaffar, J. and Lassez, J-L., Constraint Logic Programming, In *4th IEEE Symposium on Logic Programming*, 1987.

[7] Ueda, K., Guarded Horn Clauses, *Proceedings of Logic Programming '85*, Lecture Notes in Computer Science 221, Springer-Verlag, Berlin Heidelberg, 1986, pp.168-179.

APPENDICES

A. GHC Program

The GHC program of our algorithm is represented below. The initial goal of the program is *go*(*Ms*), where *Ms* is a stream variable of lists of triplets, (*NodeName, Mark, Priority*). In this program, the priority of the *mm* is 0 for convenience.

```
% gen(Ms,N) represents the structure of a network.
% Ms is the input, a list of triplets, (NodeName, Mark, Priority).
% N is the output, a list of pairs, (Mark, Priority), of each node.

gen(Ms,N) :- true |
        node(a,Ms,Na,Pa,[]                       ,TMa,_  ),
        node(b,Ms,Nb,Pb,[(TMa,[(+,Ne)])]         ,_  ,_  ),
        node(c,Ms,Nc,Pc,[(TMa,[(+,Ng)])]         ,_  ,_  ),
        node(d,Ms,Nd,Pd,[]                       ,TMd,FMd),
        node(e,Ms,Ne,Pe,[(TMd,[]),(FMf,[])],_    ,_  ),
        node(f,Ms,Nf,Pf,[]                       ,TMf,FMf),
        node(g,Ms,Ng,Pg,[(TMf,[]),(FMd,[])],_    ,_  ),
        N=[(Na,Pa),(Nb,Pb),(Nc,Pc),(Nd,Pd),(Ne,Pe),(Nf,Pf),(Ng,Pg)].

% go(Ms) is the initial goal of the program.
% Ms is the input, a stream of lists of triplets, (NodeName, Mark, Priority).

go(Ms) :- true | go1(Ms,Os) , outstream(Os).
go1([M1|Ms],Os) :- true | gen(M1,N) , Os=[nl,write(M1),nl,write(N),nl|Os1],
    go1(Ms,Os1).
go1([]     ,Os) :- true | Os=[write('terminated.'),nl].

% node(Name,Ms,N,P,Xs,TM,FM) computes a mark, N, and a priority, P, of a
% node corresponding to Name, and marks propagated from the node, using
% input Ms, which is a list of triplets, (NodeName, Mark, Priority), and
% Xs, which is a list of links to the node. TM (FM) is a mark to be
% propagated through is-a links (is-not-a links, respectively) from the node.

node(Name,[(N1,_,_)|Cs],N,P,Xs,TM,FM) :- N1\=Name | node(Name,Cs,N,P,Xs,TM,FM).
node(Name,[],N,P,Xs,TM,FM) :- true |
    search(mm,0,1,1,Xs,N,P,MaxC) , prop(N,TM1,FM1) , C:=MaxC+1 , TM=(TM1,P,C),
    FM=(FM1,P,C).
node(Name,[(Name,M,P)|_],N,P1,Xs,TM,FM) :- true | search(M,P,0,0,Xs,N,P1,_),
    prop(N,TM1,FM1) , TM=(TM1,P1,0) , FM=(FM1,P1,0).

% prop(M,TM,FM) chooses the types of marks which a node with mark M
% propagates.

prop(tm,TM,FM) :- true  | TM=tm , FM=fm.
prop(M ,TM,FM) :- M\=tm | TM=mm , FM=mm.
```

```
% search(TM,TP,TMax,TMin,Xs,N,P,C) computes a mark, N, a priority, P,
% and a logical path length, C. TM, TP, TMax and TMin are a temporary
% mark, a temporary priority, a temporary maximum logical path length, and
% a temporary minimum logical path length, respectively. Xs is a list of
% ((mark, priority, logical path length),constraint).

search(TM,TP,TMax,TMin,[((mm,_,_),_)|Xs],N,P,C) :- true |
    search(TM,TP,TMax,TMin,Xs,N,P,C).
search(mm,_ ,_   ,_   ,[((M1,P1,C1),Cons)|Xs],N,P,C) :- true | con(Cons,CM),
    check(CM,mm,M1,NM,C1,C1,_,C1,C1,_,P1,P1,_) , search(NM,P1,C1,C1,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((_ ,P1,_),Cons)|Xs],N,P,C) :- TM\=mm,TP<P1 |
    search(TM,TP,TMax,TMin,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((M1,P1,C1),Cons)|Xs],N,P,C) :- M1\=mm,P1<TP |
    con(Cons,CM) , check(CM,TM,M1,NM,TMin,C1,NMin,TMax,C1,NMax,TP,P1,NP),
    search(NM,NP,NMax,NMin,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((M1,P1,C1),Cons)|Xs],N,P,C) :- TM\=mm,P1=TP,TMin=<C1 |
    max(TMax,C1,NMax) , search(TM,TP,NMax,TMin,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((M1,P1,C1),Cons)|Xs],N,P,C) :- M1\=mm,P1=TP,TMin>C1 |
    con(Cons,CM) , check(CM,TM,M1,NM,TMin,C1,NMin,TMax,C1,_,TP,P1,_),
    search(NM,TP,TMax,NMin,Xs,N,P,C).
search(TM,TP,TMax,_,[],N,P,C) :- true | C=TMax , N=TM , P=TP.

max(X,Y,Z) :- X>=Y | Z=X.
max(X,Y,Z) :- X<Y  | Z=Y.

% con(Cons,CM) checks whether the constraint, Cons, is satisfied.

con([]          ,CM) :- true  | CM=tm.
con([(+,N)|Xs],CM) :- N=tm  | con(Xs,CM).
con([(-,N)|Xs],CM) :- N\=tm | con(Xs,CM).
con([(+,N)|Xs],CM) :- N\=tm | CM=fm.
con([(-,N)|Xs],CM) :- N=tm  | CM=fm.

% check(CM,...) chooses a temporary mark, a temporary minimum logical path
% length, a temporary maximum logical path length and a temporary priority.

check(tm,_ ,M2,M3,_    ,MinC2,MinC3,_    ,MaxC2,MaxC3,_ ,P2,P3) :- true |
    M3=M2 , MinC3=MinC2 , MaxC3=MaxC2 , P3=P2.
check(fm,M1,_ ,M3,MinC1,_    ,MinC3,MaxC1,_    ,MaxC3,P1,_ ,P3) :- true |
    M3=M1 , MinC3=MinC1 , MaxC3=MaxC1 , P3=P1.
```

B. Example

Suppose that we want to know, first, whether August is summer or winter in Japan, and second, whether it is summer or winter in Australia. For the first question, we set a mark, *tm*, with priority 0 on node *a*, corresponding to August, a mark, *tm*, with priority 1 on node *d*, corresponding to Japan. For the second question, we set a mark, *tm*, with priority 0 on node *a*, a mark, *tm*, with priority 1 on node *f*, corresponding to Australia. Then our program runs as follows. Note that the two questions are also processed in parallel.

```
| ?- ghc go([[(a,tm,0),(d,tm,1)],[(a,tm,0),(f,tm,1)]]).

[(a,tm,0),(d,tm,1)]
[(tm,0),(tm,0),(mm,0),(tm,1),(tm,1),(mm,0),(fm,1)]

[(a,tm,0),(f,tm,1)]
[(tm,0),(mm,0),(tm,0),(mm,0),(fm,1),(tm,1),(tm,1)]

terminated.

yes
| ?-
```

Since, in the first solution list, the second pair is (*tm*,0) and the second node, *b*, means summer, August is summer in Japan. Similarly, since, in the second solution list, the third pair is (*tm*,0) and the third node, *c*, means winter, August is winter in Australia.