遅延型論理演算に基づく制約型プログラミング

4M-1

向井国昭

(財)新世代コンピュータ技術開発機構

ABSTRACT

A simple constraint language is designed including equality and basic arithmetic relations. Constraints can be defined in the language by the user. An interpretation is given to the language based on the famous "freeze" control predicate. The basic idea for construction of the interpreter is to define the set of standard logical connectives "and", "or", "not", and so on in lazy evaluation principle by using the freeze.

INTRODUCTION

Complex structure and constraints form a very basic framework for designing complex problem areas like natural language processing. Various instances of this paradigm can be easily seen in recent computational linguistics and semantics, for example.

A brief explanation is given here about how to extend Prolog to have a more flexible power in describing structure and constraint. Our solution to the representation part is partial specified term, which is a kind of extension to the usual term. The other solution to the"constraint" part is "lazy evaluation" based on freeze(Colmerauer 82). Though our main intention is mixed use of both constraints and structures, we must concentrate here on how to build such a constraint system on top of Prolog.

What is constraint? For our purpose, we give it a restricted characterization as follows: 1) It has a declarative semantics. 2) Positve and negative constraint are treated symmetrically. 3) Interpretation of the constraint is done without deep backtracking.

The proposed constraint sublanguage of an extended Prolog is so simple and natural, and has a so clear declarative semantics that it will be a useful set of constraints.

It should be noted, however, that "freeze" based system can not derive the contradiction from such like conjunction x=y and x≠y unless both x and y become to be ground. This suggest that the sublanguage would be a middle level set of constraints.

FREEZE(BIND_HOOK)

The goal of the form "freeze(X, p(X))" means that the embedded goal p(X) is suspended while X is unbound, where X is a variable and p(X) is an executable goal. It is easy to see that the predicate can be used as the primitive for both stream and conroutine conroutine programming.

CONSTRAINT INTERPRETATION

"Constr" is the main predicate for constraint interpretation. For a constraint expression α, boolean expressions β, γ and δ, the meaning of of the goal of the form of constr(α,β,γ,δ) is explained as follows.

α :a given constraint expressin,

β,γ,δ :true,fase,undermined.

The operational meaning, on the otherhand, is explained by listing a portion of top level part of the actual definition in DEC-10 Prolog.

γ =true if α is proved to true.

γ =false if α is proved to be fasle.

γ =undetermined: otherewise.

Both of β and δ are used as control information.

```
?-constr(1=2, true, _, _) => 《fail》
?-constr(X=1, true, _, _)  => X=1
?-constr(X=1, false, _, _) => X is still unbound
?-constr(X=1, _, _, _)     => X is still unbound
?-constr(X\==2, false, U, _). => U = false, X = 2
?-constr(1+X=:=Y,_,_,_), Y=4 => X = 3, Y = 4
?-constr(1+X=\=Y), X=3, Y=4 => 《fail》
```

A PORTION OF THE DEFINITION.

```
constr(X,P,Q,R):-bound(R),!.
constr(X,P,Q,R):-unbound(X),!, X=P, X=Q.
constr(true,P,Q,R):-!,P=true,Q=true.
constr(false,P,Q,R):-!,P=true,Q=false.
constr(not(A),P,Q,R):-!,
        not(V,P),
        not(U,Q),
        constr(A,V,U,R).
constr((A,B),P,Q,R):-!,
        and(X,Y,P),
        and(Z,U,Q),
        constr(A,X,Z,R),
        constr(B,Y,U,R).
constr(and(A,B),P,Q,R):-!, % Sequential "and"
```

Constraint Programming based on Lazy Boolean Predicates
Kuniaki Mukai
Institute for New Generation Computer Technology

```
        and(X,Y,P),
        and(Z,U,Q),
        constr(A,X,Z,R),
        freeze(Z, constr(B,Y,U,R)).
constr(assign(X,Y), P, Q, R):-!,
        assign(X,Y,Q,R),
        P=Q.
constr(X=Y,P,Q,R):-!, equal(X,Y,P,Q,R).
constr(X\==Y,P,Q,R):-!,constr(not(X=Y),P,Q,R).
constr(X=:=Y,P,Q,R):-!,
        and(P1,P2,P),
        and(Q1,Q2,Q),
        exp(X,T,P1,Q1,R),
        exp(Y,T,P2,Q2,R).
constr(X, P, Q, R):- %user defined
        defcon(X, Y),
        constr(Y, P, Q, R).
```

LAZY BOOLEAN PREDICATES

This secion is a technical core of the paper. Basic idea is explained with the following simple example: and(X, Y, Z):-Z= 《Y&Z》

```
?-   and(X, Y, Z), Z = true.
 => X = true, Y = true, Z = true
?- and(X, Y, false). =>X and Y are undetermined.
```

Note that in this case there is an ambiguity in the possible values of (X, Y) from the Boolean logical law. The following are sample definitions of these lazy Boolean predicates.

```
not(X,Y):-
        freeze(X,pv(F,if(X,Y=false,Y=true))),
        freeze(Y,pv(F,if(Y,X=false,X=true))).

and(X,Y,Z):-
        freeze(X,pv(F,if(X,Y=Z,Z=false))),
        freeze(Y,pv(F,if(Y,X=Z,Z=false))),
        freeze(Z,pv(F,if(Z,(X=true,Y=true),
                         andx(X,Y)))).
andx(X,Y):-
        freeze(X,if(X, Y=false)),
        freeze(Y,if(Y, X=false)).

if(true,X=Y,_):-!,X=Y.
if(_,_,X=Y):-X=Y.

pv(F,_):-bound(F),!.
pv(1,X=Y):-X=Y.
```

LAZY EQUALITY

The special case of the form constr(X=Y, Z, U, V) is reduced to equal(X, Y, Z, U, V). The following is its procedural interpretation: 1) Check equality between X and Y in lazy way, The result(true/false) will be returned to U. 2) When "true" is given to Z, X and Y are unified immediately. X and Y should be different from each other if "false" is given to Z. 3) Whenever V is bound, there is no need to continue this checking.

```
?- equal(X,Y,false,A,B), X=1,Y=4.
  =>   X = 1, Y = 4, A = false, B = _91
```

ASSIGNMENT

The predicate called ssignment is internal predicates for restricted uses. Argument passing is a main intention of its use. It lookes like an extension of the assignment in the programming languages FORTRAN, for instance. The unification works both ways as follows: unify(f(X,1), f(2,Y)) => X=2, Y=1. On the oter hand, assignment works only a specified way: assign(f(X,1), f(2,Y)) => Y=1 (X: unbound) Restricton:Each variable is not allowed to occur more than once in the second argument: the form "assign(f(A,g(B)), f(Z,Z))" is not allowed.

SEQUENTIAL CONTROL

We often need some sequential controls for constraint evaluation. For example, Let us consider the following definition of the "member".

```
member(X, [X|_]).
member(X, [_|Y]):-member(X, Y).
```

We can define a constraint version, say it "mem", of the predicate in our language as follows:

```
 defcon(mem(X,Y),
        and(assign(Y, [H|T]), (X=H; mem(X, T)))).
```

The body of this definition tells, by using the sequential "and", that "parameter passing" must be prior to the evaluation of the body.

```
?- constr(mem(X,[Y|Z]), true, A, _), Y=1, Z=[U, B],
X=2,B=3.  => U = 2, A = true, X = 2, B = 3, Y = 1,
Z = [2,3]

?- constr(mem(X,[Y|Z]), false, A), Y=1, Z=[U, B],
X=2,B=3. => X = 2, B = 3,Y = 1
```

REFERENCE

[Colmerauer 1982] A. Colmerauer: Prolog II: Reference Manual and Theoretical Model, Internal Report, Groupe Intelligence Artificielle, Universite d'Aix-Marseille II, 1982.