An expert system for geometric constructions

Pascal Schreck

Université de Strasbourg - LSIIT, UMR CNRS 7005

July 2012

Geometric constraint solving: several domains, several contexts

- Education: Statement $\rightarrow$ program of construction;
- Technical drawing: sketch $\rightarrow$ precise drawing;
- Architecture, photogrammetry (projections $\rightarrow$ 3D-objects);
- molecule problem, robotic (distance geometry) ...

We focus here on education.

Program of construction

What a student should do

- Reasoning gives intermediate objects.
- Necessary conditions give locii.
- Discussions give the non-degeneracy conditions.
- Discussions give the number of solutions (and all the constructions).

$\longrightarrow$ a notion of literal statement is needed
$\longrightarrow$ a notion of program of construction is needed
$\longrightarrow$ Progé mimics a student way of constructing figures

Geometric Universe

For some reasons, we dispatch the geometric knowledge into the different concepts of our based knowledge system.

- some knowledge is attached to sorts, functional symbol and predicative symbols
- high level knowledge is expressed through "production" rules

this allows to consider a geometric universe as a parameter of the system while keeping efficient mechanisms (for instance, unification modulo).

Geometric sorts

```
dmax(droite,2).
autom(droite,D,
      [dir :: Di nomme dird(D)], [dpdir(nul, Di)]).
princip(droite,[dro,dpdir]).
rep_par(droite,d(_, _, _)).
dessinable(droite).
dessine(_, d(A,B,C)) :-
     point_bas(A,B,C,Xb,Yb),
     point_haut(A,B,C,Xh,Yh),
     tbw_draw_line(0,Xb,Yb,Xh,Yh,1,0).
saisie(droite, Nom, d(A,B,C)) :- ...
```

Functional symbols


```
profil(intercc, cercle x droite >> point).
rmult(intercd).
decomp(intercd,[cercle/incid, droite/incid]).
intercd(D1,D2) equiv interdc(D2,D1).
deg_titre(incid, 1).

'maj:' M eg mil(A,B) ==> [M est_sur dro(A,B)] :- !.
```

Predicative symbols

```
pprofil('=p=', point x point).
pprofil(ortho, droite x droite).
pprofil(est_sur, point x lieu).

ptitre(est_sur, incid).

constructif(est_sur, incident, 1).
D ortho DD pequiv DD ortho D.
```

Rules (1)

```
2 # si [ did(A,D) '=l=' H]
     et
      [connu D, connu H, pas_connu A]
   alors
     [ A est_sur dpd(D,H) : 1].
```

Rules (1)

```
3 # si [ did(A, dro(B,C)) '=l=' L]
      et
      [connu A, connu B, connu L,
       differents [B, prj(A, dro(B,C))],
       pas_connu C]
   alors
      [
       H nomme prj(A, dro(B,C)),
       H est_sur cdiam(A,B),
       H est_sur ccr(A,L) : 1
      ].
```

Rules (1)

```
7 # si  [did(A,D1) '=l=' did(A,D2)]
     et
        [differents [D1,D2], connu D1,
         connu D2, pas_connu A]
    alors
      soit [dird(D1) diff dird(D2)]
        et [ A est_sur bis(D1,D2) : 1]
      ou
       soit [dird(D1) eg dird(D2), D1 diff D2]
        et [A est_sur dmd(D1,D2) : 1]
      ou
       soit [D1 eg D2] et [].
```

Formal figure

```
is_fe(fe(_,_,_,_,_,_)).
fe_nom(fe(Nom,_,_,_,_,_),Nom).
fe_type(fe(_,Typ,_,_,_,_), Typ).
deg_lib(fe(_,_,D,_,_,_), D).
fe_def(fe(_,_,_,Def,_,_), Def).
fe_reprs(fe(_,_,_,_,Lr,_), Lr).
fe_parts(fe(_,_,_,_,_,Lp), Lp).
```

Formal figure

```
ajoute_rep(Terme, FeE, FeS) :-
    not(decomposable(Terme)), !,
    ajoute_repr(Terme, FeE, FeS).
ajoute_rep(Terme, FeE, FeS) :-
    decomposable(Terme, Ltit),
    ajoute_parts(Ltit, FeE, FeS).
```

Formal figure

```
ajoute_repr(Terme,
            fe(Nom, Ty, D, Def, Reps, Parts),
            Fe) :-
         appartient_rep(Terme, Reps), !,
         Fe = fe(Nom, Ty, D, Def, Reps, Parts)
     ;
         D == 0, !,
         Fe = fe(Nom, Ty, D, Def, [Terme|Reps], Parts)
     ;
         construit(Terme), !,
         Fe = fe(Nom, Ty, 0, Terme, [Terme|Reps], Parts)
     ;
         Fe = fe(Nom, Ty, D, Def, [Terme|Reps], Parts)
```

atomization

Each object has a name in Progé.

For instance, if you consider `prj(A, dro(B,C))]`, Progé creates a line with name `droite05` containing points `A` and `B`, and a point, say `point12`, with the equality `point12=prj(A,droite06)`

If there is a point, say `p1` which is already equal to `point12=prj(A,droite06)` no new point is created, instead, point `p1` is updated.

As a consequence, internally, the terms have at most depth 1.

fusion

- Usually, objects with different names are considered different.
- But, when you consider degeneracy, you must consider equalities between points, lines, etc.
- In Progé, when objects are declared as equal, they are fusioned ... which is a heavy operation.

alias

When 2 objects becomes equal, we have two names (at least for the same object).

We use a tree to deal with this equivalence relation (*designing the same object*) through a predicate named `alias/2`

The root of the tree is the privileged name for the object.

Unification

The unification process must take *all* the information contained in the figure into account.

For instance, if you have to unify `mil(a,b)` with `interdd(l1, d)`, the system looks for names to `mil(a,b)` and to `interdd(l1, d)` into the figure.

Now, if you have to unify `m` with `mil(X, interdd(d1,Y))` the system will try all the objects which are equal to `mil(X1,X2)` and then try to unify `X2` with `interdd(d1,Y)`.

The atomization operation, the alias predicate and the argument permutations are widely used in the process.

Graph of reasoning

In order to prevent infinite loops, Progé keeps a trace of the applied rules together with the facts used to apply them.

Each discovered comes from an atomization process! For instance, `dist(a,b) = l1` becomes `l1 = l1` and the *figure* contains the information `dist(a,b) = l1`

The history is a graph, where the nodes are the atomized facts and the edges the rules used to derived them. Plus some control informations.

Graph of reasoning

```
sommet_num(sommet(Num, _, _, _), Num).
sommet_prop(sommet(_, Prop, _, _), Prop).
sommet_nba(sommet(_, _, Nba, _), Nba).
sommet_nbp(sommet(_,_,_, Nbp), Nbp).

deriv_num(deriv(Num, _, _), Num).
deriv_clause(deriv(_, Clause, _), Clause).
deriv_n_regle(deriv(_, _, Nregle), Nregle).
```

Rule application

There are several strategies to choose a pair (rule, list of facts).

When a rule is chosen, the system try to instantiate it by searching in the graph reasoning *and* the figure.
For instance, in the parallelogram exercise, the statement contains the fact `dist(a,x) + dist(b,y) = l` which is atomized in `l = l` with the informations `long01 = dist(a,x)` and `long02 = dist(b, y)` and `l = long01 + long02` stored in the formal figure.

Reasoning (continued)

Considering the rule:

```
300 # si [dist(A,M) + dist(B,N) '=l=' L]
    et
       [connu dro(A, M) ]
    alors
    [
     X nomme interdc(dro(A,M), ccr(A,L)),
        A est_sur ccr(X,L),
     dist(X,M) '=l=' dist(B,N),
     tinhibe(300)
    ].
```

the system has to search for a fact using a distance equality and then to unify: `dist(A,M) + dist(B,N)'=l=' L` with `l '=l=' l`

Reasoning (continued)

Unification is done mostly by using the figure (and possibly permutation rules)

Then the system tests if `connu dro(A, M)`

if all succeeds, the rule is *possibly* applied. In fact, in order to have a fair use of the deduced facts, we implement a kind of breadth-first search by using nba and nbp numbers.

Link(s) with the figure

When a rule succeeds, new facts and new objects.

If an object o is constructed (that is, its degree of freedom becomes 0), a definition o=t is inserted in the construction plane,

But … things are more complicated when the function symbol is interpreted by a multifunction (giving a foreach … in … do structure).

Link(s) with the figure (continued)

Things are even more complicated when there exists preconditions for applying the function related to the functional symbol:

- the system tries to proof that the precondition(s) holds, if it it is ok, then ... it is ok;
- if it fails, it tries to proof the negation of the precondition, if it succeeds, the negation of the precondition is added to the facts (usually more specific knowledge is associated);
- if both proofs fail, then a if ... then ... else structure is added in the program and a context is pushed on a stack of contexts.

Links with the application of disjunctive rules

This is more or less the same mechanism when the system applies a disjunctive rule:

- if there are additional conditions, the system try to prove them ...if it fails, it puts a particular structure (foreach case in a list of cases). Note that the additional conditions are not forcefully testable as they are interpreted during the interpretation of the program construction.
- if there are no additional conditions, the system put a if... then ... else structure in the program.

Conclusion

Need for a new fresh programing instead of dirty prototyping in Prolog ...