TP2 - Programmation logique inductive

Paul Chaignon - Ulysse Goarant

13 février 2014

1 Prise en main - Les trains de Michalsky

Ci-dessous le prédicat calculé à la fin de *sat*(1).

```
eastbound(A) :-
    has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
    short(E), short(C), closed(C), long(D),
    long(B), open_car(E), open_car(D), open_car(B),
    shape(E,rectangle), shape(D,rectangle),
    shape(C,rectangle), shape(B,rectangle),
    wheels(E,2), wheels(D,3), wheels(C,2), wheels(B,2),
    load(E,circle,1), load(D,hexagon,1), load(C,triangle,1),
    load(B,rectangle,3).
```

Ci-dessous le prédicat calculé à la fin de *reduce*.

```
eastbound(A) :-
    has_car(A,B), short(B), closed(B).
```

L'opérateur de raffinement d'Aleph a une approche bottom-up. En effet, le premier prédicat calculé suite à *sat*(1). est beaucoup plus spécifique que le second. La second étape *reduce*. permet de généraliser ce prédicat.

2 Une affaire de famille

Nos exemples positifs sont les suivants.

```
fille_de(mary, ann).
fille_de(rosy, mary).
fille_de(eve, tom).
fille_de(lisa, tom).
```

Nos exemples négatifs sont les suivants. Nous avons choisi de les indiquer intégralement.

```
fille_de(ann, mary).
fille_de(ann, ann).
fille_de(ann, tom).
fille_de(ann, lisa).
fille_de(ann, rosy).
```

```
fille_de(ann, eve).
fille_de(ann, bob).

fille_de(mary, mary).
fille_de(mary, tom).
fille_de(mary, lisa).
fille_de(mary, rosy).
fille_de(mary, eve).
fille_de(mary, bob).

fille_de(tom, mary).
fille_de(tom, ann).
fille_de(tom, tom).
fille_de(tom, lisa).
fille_de(tom, rosy).
fille_de(tom, eve).
fille_de(tom, bob).

fille_de(rosy, ann).
fille_de(rosy, tom).
fille_de(rosy, lisa).
fille_de(rosy, rosy).
fille_de(rosy, eve).
fille_de(rosy, bob).

fille_de(eve, mary).
fille_de(eve, ann).
fille_de(eve, lisa).
fille_de(eve, rosy).
fille_de(eve, eve).
fille_de(eve, bob).

fille_de(lisa, mary).
fille_de(lisa, ann).
fille_de(lisa, lisa).
fille_de(lisa, rosy).
fille_de(lisa, eve).
fille_de(lisa, bob).

fille_de(bob, mary).
fille_de(bob, ann).
fille_de(bob, tom).
fille_de(bob, lisa).
fille_de(bob, rosy).
fille_de(bob, eve).
fille_de(bob, bob).
```

Le *background knowledge* est défini de la manière suivante.

```
:- modeb(*, pere(+, -)).
:- modeb(1, pere(-, +)).
:- modeb(*, mere(+, -)).
:- modeb(1, mere(-, +)).
:- modeb(*, fille(-)).

:- determination(fille_de/2, pere/2).
:- determination(fille_de/2, mere/2).
:- determination(fille_de/2, fille/1).

pere(tom, eve).
pere(tom, lisa).
pere(tom, bob).
mere(ann, mary).
mere(mary, rosy).
fille(mary).
fille(eve).
fille(ann).
fille(rosy).
fille(lisa).
```

Ci-dessous se trouve la réponse fournie par Aleph. Il a en fait créé des règles spécifiques à chaque exemple positif et n'a donc pas réussi à généraliser le concept de *fille de*.

```
[Rule 1] [Pos cover = 1 Neg cover = 0]
fille_de(mary,ann).

[Rule 2] [Pos cover = 1 Neg cover = 0]
fille_de(rosy,mary).

[Rule 3] [Pos cover = 1 Neg cover = 0]
fille_de(eve,tom).

[Rule 4] [Pos cover = 1 Neg cover = 0]
fille_de(lisa,tom).
```

3 Les figures du Poker

Les ensembles des exemples positifs et négatifs sont générés à l'aide du script Perl utilitaire à partir du fichier de données fourni. Toutes les figures ne sont pas sélectionnées afin de ne pas rendre le processus de déduction trop long.

```
while(<IN>) {
    if(/^([1-4]),([0-9]+),([1-4]),([0-9]+),([1-4]),([0-9]+)
        ,([1-4]),([0-9]+),([1-4]),([0-9]+),([0-9])/) {
        my @couleurs = (Trans_couleur($1), Trans_couleur($3),
            Trans_couleur($5), Trans_couleur($7), Trans_couleur($9)
            );
```

```
my @valeurs = (Trans_valeur($2), Trans_valeur($4),
   Trans_valeur($6), Trans_valeur($8), Trans_valeur($10));
my $figure = Trans_main($11);

$nb_ex++;
$ident = 'main'.$nb_ex;
my $bool_desc;
my $carteIdent;

if($figure eq 'carre') {
   print POS "carre($ident).\n";
   $bool_desc = 1;
} elsif($figure eq 'rien') {
   if(int rand 100 == 0) {
      print NEG "carre($ident).\n";
      print NEG "paire($ident).\n";
      print NEG "suite($ident).\n";
      print NEG "brelan($ident).\n";
      $bool_desc = 1;
      }
} elsif($figure eq 'paire') {
   if(int rand 100 == 0) {
      print POS "paire($ident).\n";
      $bool_desc = 1;
   }
} elsif($figure eq 'suite') {
   print POS "suite($ident).\n";
   $bool_desc = 1;
} elsif($figure eq 'brelan') {
   if(int rand 2 == 0) {
      #print POS "brelan($ident).\n";
      #$bool_desc = 1;
   }
} elsif(int rand 50 == 0) {
   print NEG "carre($ident).\n";
   print NEG "paire($ident).\n";
   print NEG "suite($ident).\n";
   print NEG "brelan($ident).\n";
   $bool_desc = 1;
}

if($bool_desc) {
   foreach my $i (0..4) {
      $carteIdent = "carte_${nb_ex}_$i";
      print BK "carte($carteIdent).\n";
      print BK "a_carte($ident,␣$carteIdent).\n";
      print BK "valeur($carteIdent,␣$valeurs[$i]).\n";
      print BK "couleur($carteIdent,␣$couleurs[$i]).\n";
```

```
            }
        }
    } else {
        print STDERR "$prg: pb ligne au format inconnu :\n$_\n\n";
    }
}
```

La règle obtenue pour la détermination du carré est la suivante.

```
[Rule 1] [Pos cover = 236 Neg cover = 0]
carre(A) :-
    cartes(A,main(B,C,D,E,F)), valeur(E,G), valeur(D,G), valeur(C
        ,G),
    valeur(B,G).
```

Même en augmentant le nombre d'exemples positifs et négatifs Aleph n'a pas réussi à déterminer une règle pour paire, brelan ou suite. Ces figures sont plus difficiles à faire apprendre car le point commun des exemples positifs est plus complexe.

La suite impose d'avoir conscience de la relation d'ordre sur les valeurs que Aleph ne connait pas. Les figures paire et brelan ne peuvent être des carrés. Il faut donc comprendre que les paires, par exemple, ont uniquement deux valeurs identiques et pas plus.