One variant of assertional programming

Rukhaia Khimuri

Institute of Applied Mathematics,
Tbilisi State University
Georgia
rukhaia@viam.hepi.edu.ge

Draft

1 Introduction

In 1879 Gottlob Frege created predicate calculus, he considered predicate calculus as a universal language. For development of predicate calculus was necessary to be created an algorithms which describes deduction processes, These processes has to be mathematically precise.

In 1930 Herbrand, Goedel and Skolem independently showed that Frege's calculus works well with true sentences, moreover Herbrand gave proof search procedure. In 1936 Church and Turing showed, that predicate calculus is undecidable.

Beth and Gilmore implemented Herbrand's algorithm, they showed that Herbrand's algorithm was not effective enough. After Robinson developed resolution calculus and using this calculus Herbrand's algorithm become more effective.

In 1971 Alain Colmerauer studied all results mentioned above and developed system Prolog. In 1975 Robinson implemented LogLisp, which is a Prolog implementation in Lisp and allows Prolog programs to call Lisp and vice versa. The main goal of this system was to put together logical and functional programming concepts. Researchers considered LogLisp as a good example of assertional programming.

assertional programming is a programming type, where we declare some sentences as true and then prove some goals as logical consequence of these sentences.

2 "my family" program

Definition 1. *level of symbols*

1. *0-level symbol of τSR-logic is a fundamental symbol of τSR-logic*
2. *n-level ($n \in 1, ..., n$) symbol of τSR-logic is constructed by set of symbols, where each symbols level is less then n and contain at least one symbol that has level $n-1$.*

Definition 2. *we say the form is n-level form of τSR-logic, if it contains at least one n-level symbol and any other symbols has level less or equal to n.*

Let consider program "My family". This program contains following facts:

$$mother(lali, \_).$$
$$parent(mamuka, maka).$$
$$parent(mamuka, eka).$$
$$parent(mamuka, irakli).$$
$$parent(lali, maka).$$
$$parent(lali, irakli).$$
$$parent(nino, nana).$$
$$female(maka).$$
$$female(eka).$$
$$man(mamuka).$$
$$man(irakli).$$
$$distinct(eka, mamuka).$$

And following rules:

(1) $\text{child}(X, Y) : -\text{parent}(Y, X)$.
(2) $\text{mother}(X, Y) : -\text{parent}(X, Y) \wedge \text{female}(X)$.
(3) $\text{father}(X, Y) : -\text{parent}(X, Y) \wedge \text{man}(X)$.
(4) $\text{sister}(X, Y) : -\text{parent}(Z, X) \wedge \text{parent}(Z, Y) \wedge \text{female}(X) \wedge \text{distinct}(X, Y)$.
(5) $\text{brother}(X, Y) : -\text{parent}(Z, X) \wedge \text{parent}(Z, Y) \wedge \text{man}(X) \wedge \text{difference}(X, Y)$.
(6) $\text{aunt}(X, Y) : -\text{father}(Z, Y) \wedge \text{sister}(X, Z)$.
(7) $\text{cousin}(X, Y) : -\text{child}(X, Z) \wedge \text{aunt}(Z, Y)$.

If we ask following question to our program

?-female(lali).

Our goal will not successes and we will get answer "No".

Let extend our program by following rule:

$$(8)\ \text{female}(X) : -\text{mother}(X, \_).$$

and ask again same question to our extended program

?-female(lali).

Now our program can give us positive answer.

To observe our program we can discover in the rules (2) and (8) predicate *mother* is defined by predicate *female* and vice versa, so it gives higher risk of non termination computation.

3 Alternative of "my family" program

Let now consider the alternative program of the "my family" program.

This alternative program like "my family" program contain following facts:

$$
\begin{array}{l}
mother(lali, \_).\\
parent(mamuka, maka).\\
parent(mamuka, eka).\\
parent(mamuka, irakli).\\
parent(lali, maka).\\
parent(lali, irakli).\\
parent(nino, nana).\\
female(maka).\\
female(eka).\\
man(mamuka).\\
man(irakli).\\
distinct(eka, mamuka).
\end{array}
$$

Rules in our alternative program are defined following way:

$A \leftrightarrow B_1 \wedge ... \wedge B_n$

where $B_1 \wedge ... \wedge B_n$ contains only fundamental or predefined predicates and A is defined by this definition. A has an one level higher then highest levels predicates in $B_1 \wedge ... \wedge B_n$.

under $\sigma^{[m,n]}[\sigma_1, ..., \sigma_k]$ we mean an expression, where $\sigma_1, ..., \sigma_k$ are all pairwise distinct operators from $B_1, ..., B_n$ and m, n shows type and level of this expression respectively.

So, rules in our alternative program are:

$D_1[I, I]$ child(X, Y) $\leftrightarrow$ parent(Y, X).
$D_2[I, I]$ mother(X, Y) $\leftrightarrow$ parent(X, Y) $\wedge$ female(X).
$D_3[I, I]$ father(X, Y) $\leftrightarrow$ parent(X, Y) $\wedge$ man(X).
$D_4[II, I]$ sister(X, Y) $\leftrightarrow$ parent(Z, X) $\wedge$ parent(Z, Y) $\wedge$ female(X) $\wedge$ distinct(X, Y).
$D_5[II, I]$ brother(X, Y) $\leftrightarrow$ parent(Z, X) $\wedge$ parent(Z, Y) $\wedge$ man(X) $\wedge$ difference(X, Y).
$D_6[II, II]$ aunt(X, Y) $\leftrightarrow$ father(Z, Y) $\wedge$ sister(X, Z).
$D_7[II, III]$ cousin(X, Y) $\leftrightarrow$ child(X, Z) $\wedge$ aunt(Z, Y).

The predicates defined by $D_1 - D_5$, D_6 and D_7 rules are one, two, three level predicates respectively. We can rewrite rule D_1 as.

$\text{child}^{[I,I]}[\text{parent}]$

so, predicate *child* is defined by predicate *parent*. We can similarly rewrite $D_2 - D_7$ rules.

to satisfy our goal an effective way we modify our program as is shown in table 1, table 2.

0-level predicates	I-level predicates	III-level predicates	IV-level predicates
parent	child[parent]	aunt[father,sister]	cousin(child,aunt)
female	mother[parent,female]		
distinct	father[parent,man]		
man	sister[parent,female,distinct]		
	brother[parent,man,distinct]		

table 1.

Fundamental facts	Derivative facts		
1-level	2-level	3-level	4-level
parent(mamuka,maka)	child(maka,mamuka)	aunt(nino,maka)	cousin(nino,nana)
parent(mamuka,eka)	sister(maka,eka)		
parent(mamuka,irakli)	mother(lali,eka)		
parent(lali,maka)	mother(lali,maka)		
parent(lali,irakli)	sister(nino,mamuka)		
parent(nino,nana)			
female(maka)			
man(mamuka)			
man(irakli)			
female(eka)			
distinct(eka,mamuka)			

table 2.

Here we give computational procedure:

INPUT:

a program P and goal G.

OUTPUT

A substitution θ, such that the $G\theta$ is deduced from P, or failure if failure has occurred

$Subst$ is an empty substitution ε and $Vars$ is a set of variables of G. While G is not empty do

1) Choose a subgoal S from G.

2)If S is of n-level, try to find a renamed copy R of an n-level clause in P such that S and head of R unifies with an mgu θ.

3)If 2) succeeds remove S from the goal and add the body of R to G.

4) if 2) fails then try to find a renamed copy of a k-level clause (k¿n) R in P whose body contains a subgoal that is unifiable with S by an mgu θ, remove S from G, and add the head of R to G

5) if 4) fails, exit the while loop

6) $G := G\theta$,$Subst := Subst\theta$

7) if G is not empty, return the restriction of $Subst$ to $Vars$, failure otherwise

using this procedure our program can answer now goal "?-female(lali)" and we get answer "yes".

4 Future work

We are going to develop and implement the method mentioned above.

5 references

1) Matthias Baaz, Andrei Voronkov: Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002, Tbilisi, Georgia, October 14-18, 2002, Proceedings Springer 2002

2) BRATKO, Ivan. Prolog Programming for Artificial Intelligence; Third edition. Pearson Education, Addison-Wesley, 2001. Second edition, Addison-Wesley 1990. First edition, Addison-Wesley 1986.

3) Clocksin, W.F. and Mellish, C.S., 1987. Programming in Prolog. Third Edition. Springer Verlag, Berlin.

4) Sh. Pkhakadze, Some problems of the notation theory(Tbilisi University Press, Tbilisi, 1977).(in Russian)

5)On One Variant of τ Theory Extended With Reducing Symbols, Wissenschaftliche Beitrage der Friedrich Schiller University,Jena,1979,pp.365-381(In Russian).

6) Rukhaia Kh, τSR Logic-Foundation of Assertional programming.

7)K Rukhaia, L Tibua, ONE VERSION OF PROGRAMMING ACCORDING TO LOGIC τSR-language;Bulletin of TIGMI; vol. 4; 2000