1 Notes - TDDD08

Unification in Prolog

1. term1 and term2 are constants then they unifiy if they are the same constant or number
2. term1 is a variable and term2 is any type of term term1 unify with term2. same other way around and if both are variables the substitutes to each other and term1 = term2
3. If both are complex terms they unify if
   (a) They have the same functor and arrity,
   (b) All their arguments unify,
   (c) The variable instantiations are compatible.

Prolog cheats and doesn't use a full unification algorithm for example : $a(X) = X$ unifies because of the second clause but does no in theory. To combat this one can specify occur checks in prolog.

Program specification

Example:

$$S = split(l, l_1, l_2)\epsilon\mathbf{B}_A|l_1, l_2 \textit{ are lists}, |l_1| - |l_2|\epsilon 0, 1 \tag{1}$$

Proving S: for every ground instance of a clause, if each body atom is in S then the head must be in S.

$$split([h|t], [h|t_2], t_1) \leftarrow split(t, t1, t2) \tag{2}$$

assume $split(t, t_1, t_2)\epsilon S$ means $|l_1| - |l_2|\epsilon\{0, 1\}$ meaning $|[h|t_2] - |t_1| = 1 + |t_2| - |t_1|\epsilon\{1 - 0, 1 - 1\}$

SLDNF - forest Do regular SLD - tree but break into a new tree for each negation.

Finitely failed means a finite tree with no answer substitutions Floundering means anything that doesn't terminate.

Program completion Convert clauses and add qualifiers

- $s(L, M) \leftrightarrow\sim ns(L, M)$
- $ns(L, M) \leftrightarrow \exists_X m(X, L) \sim m(X, M)))$
- $m(X, L^{'}) \leftrightarrow \exists_L(L^{'} = [X|L]) \vee \exists_{Y,L}(L^{'} = [Y|L]\, m(X, L))))$

Insert the queries to test and check if the logic holds true. if it's true comp(program) $\models query$.

Herbrand interpretation

- Universe: U_A is collection of all constants and function symbols.
- Base: $B_A = \{predicate(p)|p\epsilon U_A\} \cup \{predicate2(p)|p2\epsilon U_A\}$
- Least Model: $M_P = \{A\epsilon B_A|P \models A\}$

PTR

- $PTR_0 = \{gt(t)|tisaterm\}$
- $PTR_1 = \{\textit{all terms proven with } PTR_0\}UPTR_0$
- General proof is done with inference, prove PTR_0, PTR_1, etc, until general pattern, then prove PTR_K and then PTR_{K+1}. Remember to specify bounds of K.

Extra bits

- Soundes means all computed answers are logical consequences of a program.
- Completenes means that all logical consequences of a program are is a computed answer.