INDRODUCTION

1 SYNTAX FOR EXPRESSION

2 EXECUTION TIME REPRESENTATION

3.SEQUENCING WITH NON ARITHMETIC

EXPRESSION

Syntax For Expression:

For syntax purpose of expression , we need three notation are:

- **Prefix (Polish Notation)**
- **infix**
- **Postfix (Suffix or Reverse Notation)**

Semantics With Expression:

Semantics determine the order of the expression in which they are evaluated

- **Prefix Evaluation**
- **Postfix Evaluation**
- **Infix Evaluation**

*****Algorithms same that have done in Data Structure**

Precedence: defines the which operator have what priority

Execution Time Representation

Because of the difficulty of decoding expressions in their original infix form it is common to translate expression in to executable code that may be easily decoded during execution. The following are most important alternatives:

1. **Machine code sequence:**

The expression is translated in to actual m/c code in one step. The ordering of instruction reflects the sequence-control structure of the original expression.

2.Tree structures:

Express may be executed directly in their natural tree structure notation(stage 1) using s/w interpreter. Execution(stage 2) may performed by a simple tree traversal.

Syntax For Expression:

3.Prefix or postfix forms:

Expression in both form prefix as well as postfix may executed in a single step by implementation of stacks or queue or pointer .

2.Sequencing with Non Arithmetic Expression

Some definitions are:

General Unification:

As given above, to unify two expressions U & V , find substitutions for the variables occurring in U & V that make two expressions identical.

- **Resolution :**

It is the basic inference rule that is used in logic programming. The term resolution refers to a mechanical method for providing the statements in first order logic. It is applied to two clauses in a sentence , and by unification it eliminates a literals that occurs positive in one clause and negative in second clause.

- **Substitution:**

Substitute the result accordingly the unification as well as Resolution.

2.Sequencing with Non Arithmetic Expression

Non arithmetic are used in logical language. There are several algorithm are used:

1.Pattern Matching:

A crucial operation in language like SNOBOL4 ,PRPLOG and ML is pattern matching. In this case an operation succeeds by matching and assigning a set of variables to a predefined template. For example the following grammar recognizes odd length palindromes over the alphabets 0 & 1 :

$$A \rightarrow 0A0 \mid 1A1 \mid 0 \mid 1$$

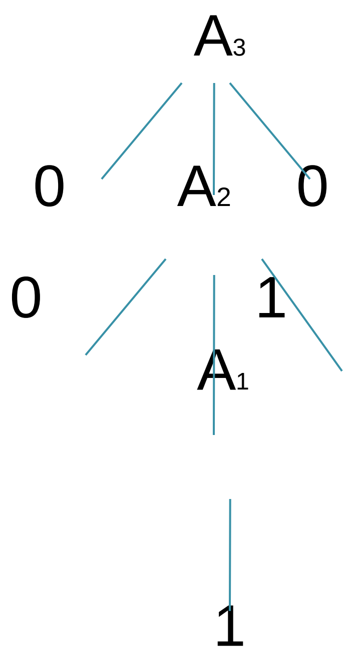
Recognition of the valid string 00100 proceeds as :

A1 matches the center 1

A2 matches the 0$A_1$0

A3 matches the 0$A_2$0

2.Sequencing with Non Arithmetic Expression



We can generate the string 001 with following derivations:

$A \Rightarrow 0B$ using rule $A \rightarrow 0B$

$0B \Rightarrow 00A$ using rule $B \rightarrow 0A$

$00A \Rightarrow 001$ using rule $A \rightarrow 1$

2.Sequencing with Non Arithmetic Expression

2.Unification

Any substitution that makes two or more expressions equal is called a unifier for the expression. Given that two expression that are unifiable ,such as expression C1 and C2 with a unifier "b" or we can state that C1b=C2 where b is the unifier.

The unifier algorithm:

1. Compare first element of the two statement. If they not the same then there is no way to unify them so exit with failure
2. If the first elements are same then we have to unify the rest of the literal moving from second to the third and so on

Unification is simple just we have to calling the algorithm recursively. The matching rules are as follows:

a) Different constant , function and procedure can't be match only identical can match
b) A variable can match other another variable , any constant ,function with restriction that function must not contain the any instance of variables being matched
c) Once we make a substitution we need to apply it to the remaining literals before calling the unification algorithm again

Unification algorithm is given below:

Unify (L1,L2)

1. If L1 or L2 is a variable or constant then

a) If L1 and L2 are identical then return NIL
b) Else L1 is variable then if L1 occurs in L2 then return Fail else return (L2/L1)

c)Else L2 is variable then if L2 occurs in L1 then return Fail else return (L1/L2)

d)Else return Fail

2.) I the initial predicate symbols in L1 and L2 are not identical then return Fail

3)If L1 & L2 have different number of arguments then return Fail

4)Set SUBST to NIL

5) For i=1 to the numbers of arguments in L1

a) Call Unify with the ith argument of L1 and the ith argument L2 putting result in S

b) If S not = NIL then

i)Apply S to the remainder of the both L1 and L2

ii) SUBST := Append(S,SUBST)

6)Return SUBST

2.Sequencing with Non Arithmetic Expression

3.Backtracking:

It is general programming technique that is available in any programming language that creates tree structures. We can build backtracking algorithms in all languages like LISP where tree is considered as a bulit-in data type so in language the implementation is quite easy.

In PROLOG simply uses the order in which the facts are entered in to data base. At any time we can try to match a sub goal. If the match is succeed , then one of possible goals will be succeed. If we start with a incorrect goal that will results in a incorrect goal. in this case we will try for another goal. If we reach at last sub goal and its too fail we say that the current sub goal is fails. Since we have stacked the set of sub goals we are searching for, we back up to the previous sub goal that matched ,

2.Sequencing with Non Arithmetic Expression

We call this is a general backtracking algorithm as follows:

1. **For sub goal (i,j) set k=1 as a new goal**
2. **Successively try to match goal(k) for k=1..M and return either success or fail depending upon if any goal(k) succeeds or all fails**
3. **If goal(k) succeeds then sub goal (i,j) succeeds. Save k and match sub goal i, j+1**
4. **If goal(k)fails for all k then sub goal(i,j) fails. Back up to sub goal(I,j-1) and try next possible goals k+1 for that sub goal**
5. **If sub goal (i,n) succeeds, then returns success as the result of the parent goal(k) that was being searched for.**
6. **If sub goal(i,j) fail for all j , then this goals fails and return fail as parent's search for goal(k)**