Non-Sequential Prolog

Héctor Jiménez Salazar
Fac. de Ciencias Físico Matemáticas, UAP.
hjimenez@fcfm.buap.mx

Mayo, 1997

Abstract

In this brief note we present the main features of the language Non-Sequential Prolog (NSP), its differences with other non sequential logic languages: Concurrent Prolog, Parlog, GHC, and the syntax and semantics that clarify the functioning of NSP language.

1 Parallel logic languages

The role of parallel logic languages has been determinant on the design of new machine architectures. We based our work on a concurrent-prolog like version named non-sequential Prolog. Like Parlog, GHC and concurrent Prolog we have clauses. Nevertheless, there exists an essential difference among this languages, this is the kind of synchronization mechanism: a variable is constrained to be consulted only when it has been instantiated, forcing to the process suspend until the variable is instantiated. We will make a brief analysis on the logical variable access mode (Takeuchi). Commonly in parallel logic languages the synchronization and communication mechanism is realized by constraining the access to logical variables. We will see each one of these approaches.

Concurrent Prolog Is a language whose access restriction to variables is represented at data level because of when a process tries to read a uninstantiated variable in read only mode then the process is suspended. It follows a program in concurrent Prolog that merges two lists, which shows the read only mode variable (through the use of ?, e.g. `?X`):

```
merge([A|X],Y,[A|Z]):- true|merge(X?,Y,Z).
merge(X,[A|Y],[A|Z]):- true|merge(X,Y?,Z).
merge([],Y,Y):- true|true.
merge(X,[],X):- true|true.
```

Parlog *Parallel Logic.* Offers two modes to declare each variable of the procedure head in a program. Variables may be of input (declared with ? symbol) or output type (using the ^ symbol). An input variable cannot be instantiated and, if it is not instantiated at match time between a goal and the head of a clause or the guard solution then the process representing the goal is suspended. Follows the same example, now in Parlog:

```
mode merge(?,?,^).
merge([A|X],Y,[A|Z]):- true|merge(X,Y,Z).
merge(X,[A|Y],[A|Z]):- true|merge(X,Y,Z).
merge([],Y,Y):- true|true.
merge(X,[],X):- true|true.
```

GHC *Guarded Horn Clauses.* The concept of protected input is utilized in GHC to means that any tray to instantiate a goal variable while the clause election process (this is the match between a clause head and the goal or the guard solution) is ought to suspend. An example may clear this idea:

```
merge([A|X],Y,OZ):- true|OZ=[A|Z],merge(X,Y,Z).
merge(X,[A|Y],OZ):- true|OZ=[A|Z],merge(X,Y,Z).
merge([],Y,OZ):- true|OZ=Y.
merge(X,[],OZ):- true|OZ=X.
```

We remark that the `OZ` variable is not defined at the head, like in the others examples.

NSP *Non-Sequential Prolog.* There exists the data level synchronization too. But there is not, differently to concurrent prolog, a reactivation of the non commited choices when a fail occurs. We are referring to a uniprocessor implementation. The program for a merge it would be identical to the concurrent prolog program.

In the following section we resume the main features of NSP.

2 Syntax

Constants: Integers, strings and lists of constants.

Variables: Each variable may have two modes: *Read-Only Mode (ROM)* or *general mode (GM).*

$$\forall X \in \mathit{Variables} : \begin{cases} ?X & \text{denotes } X \text{ in ROM} \\ X & \text{denotes } X \text{ in GM} \end{cases}$$

And 4 states: GMI=(GM, Instantiated), GMN=(GM, Non-instantiated), ROMI=(ROM, Instantiated), ROMN=(ROM, Non-instantiated).

Atoms: Lists whose first element is a predicate name and whose tail consists of variables, in either mode, and constants. The length of the tail is the *arity* of the atom.

Guards: Specific tests on the universe of discourse. Tests related to the current mode of a given variable.

Clauses: Lists of atoms, with length at least 2. The head of the clause is its *consequence*. The second element is its *guard*. The remaining part of the list form its *antecedent*.

Goals: Atom lists.

Processes: Atoms.

3 Semantics

Switching variable mode: It is possible to instantiate a variable just once. A variable may be instantiated only by processes containing the variable in GM.

Unification: Two atoms are *unified* in the usual Prolog sense.

Goal reduction: Given a goal $g = (M_1 \dots M_n)$, if $c = (M_1' G' A_0 \dots A_m)$ is a clause whose head unifies with the first goal M_1, by the unifier ϕ, and $G(\phi)$ is satisfied, then substitute g by $(A_0(\phi) \dots A_m(\phi)) * (M_2 \dots M_n)$.

Waiting processes: Given a goal $g = (M_1 M_2 \dots M_n)$ if in M_1 there appears at least one variable in state ROMN then put M_1 into the *wait list* and substitute g by $g' = (M_2 \dots M_n)$.

End condition: The whole procedure is finished when both the goal and the wait list are empty.

Swapping waiting processes: If the goal is empty and the wait list is not, then consider the wait list as the current goal and make empty the wait list.

If the same situation repats at the above **Swapping waiting processes** action, then a condition of *deadlock* can be recognized.