PRISM: A Parallel Inference System for Problem Solving

Simon Kasif Madhur Kohli Jack Minker

Department of Computer Science, University of Maryland, College Park, MD 2074

1. Introduction

PRISM (a Parallel Inference System), for parallel execution of problems, has been implemented on ZMOB (Rieger [1980]). PRISM is an experimental tool for the development of distributed AI problem solvers and is based on logic programming. Familiarity with logic programming (Kowalski [1979]) is assumed.

In conventional programming systems the logic and control of an algorithm are combined making it difficult to separate or to modify control without affecting the logic. Logic as the specification language, is neutral with respect to control and specifies only the problem semantics. The method of how the problem is to be solved is external to the logic specification. Thus, a primary issue in achieving a parallel system is developing an effective control specification that exploits parallelism. PRISM permits us to specify the problem independently of the control and allows us to experiment with alternative control possibilities for the same problem.

The basic tasks identified in the execution of a problem are: management of the search space; management and retrieval of procedures (deductive axioms); management and retrieval of assertions (facts, or knowledge); and interaction with the problem and control specifier (user).

PRISM consists of three functionally distinct sets of machines. The Intensional database machines (IDB) unifies a goal with all procedure heads with the same name. The extensional database machines (EDB) function as a distributed relational database machine and provide fast access to function-free ground assertions. The problem solving machines (PSM) manage the search space. The PSM is the major portion of PRISM and is where the control specification is primarily interpreted. In this paper we describe the PSM and its support of the control facilities provided in PRISM. (See Kasif [1983] for further details and references on PRISM).

2. Control Issues

2.1. Goal tree and Control Issues

A goal tree is generated in the problem solving process. The tree consists of a set of nodes, where each node consists of a set of goals. Subgoals in a node may be characterized as dependent or independent of one another, A subgoal is dependent if its execution must await the successful execution of another subgoal in the same node. It is independent otherwise. An acyclic partial order expresses such a relationship among subgoals. At any stage of the execution of a node, all independent subgoals may be executed asynchronously. However, goals which are candidates for simultaneous execution must be treated specially if they share unbound variables. There may be several assertion/procedure heads which match a selected goal. Any procedure head which matches a goal can potentially lead to-solving that goal, independent of other matching procedure heads. All matching procedure heads are therefore candidates for asynchronous execution. Thus one may specify that certain alternatives need be explored only if other alternatives have failed or that some alternatives are more likely to succeed than others. All possible asynchronous operations may be executed on autonomous machines.

2.2. PRISM Control Facilities and Language

PRISM provides the ability to specify partial order for the execution of every goal and procedure body. The partial order on goals expresses dependencies among subgoals within a goal. The order is specified by a notation as illustrated by:

$$P \leftarrow\!\!- (G_1,[G_2,(G_3,G_4),G_5],[G_6,G_7]).$$

The procedure head is on the left of the arrow, while the body is on the right. The body consists of a set of goals, separated by commas and formed into groups by properly nested pairs of parentheses and brackets. All groups of goals enclosed by parentheses, must be executed in a left-to-right sequence. Groups of goals enclosed in brackets may be executed independently of other groups in the same set of brackets.

PRISM provides a notation for specifying both a recommended and a forced ordering of procedures. Consider the example:

$$\begin{array}{rl} 1: & P \leftarrow\!\!- G_1,G_2 \\ 1: & P \leftarrow\!\!- G_3 \\ 2: & P \leftarrow\!\!- G_4,G_5,G_6 \\ *3: & P \leftarrow\!\!- G_7 \\ 4: & P \leftarrow\!\!- G_8,G_9 \end{array}$$

The integers represent a recommended order of execution. The asterisked integers represent a foroed ordering. The first two procedures (priority=1) may be executed simultaneously. The third procedure (priority=2) is less likely to succeed but may also be executed in parallel with the first two, or before them if the problem solver so decides. However, the fourth procedure (priority=*3) cannot be executed unless the preceding procedures have been completely executed. A default ordering is provided by PRISM when the procedures are not numbered.

3. The Problem Solving Machine (PSM)

3-1- The Role of the PSM

The central task of the PSM is to manage the search space. The complete separation of logic (the problem specification) and control (the strategy of solving the problem) allows flexibility while executing the program. Not only can the search strategy be varied dynamically, but due to the inherently non-deterministic nature of logic programs, several mutually exclusive possibilities may be explored simultaneously. The PSMs permit this inherent parallelism to be exploited during problem solving.

Initially a goal, which represents the problem to be solved, is sent by the Host to ZMOB and is read by some PSM. This PSM places the goal as the root of a proof tree.

At any given instant in the problem solution process the search space administered by each PSM consists of a tree of goal nodes. The root of the tree is the original goal with which the PSM was initiated. The successor of any nodes in this tree is the resolvent obtained by resolving program clauses with some atom in the parent node. When an atom is expanded, several program clauses which represent alternative ways to solve the problem, may resolve with it. These alternatives lead to branching in the search tree (OR branches). Nodes in the tree can be in one of five states: the empty node; a failure node; an open node not yet selected for expansion; an active node selected for expansion, but not yet fully expanded; and a closed node which has been fully expanded.

At any stage the PSM must select an open node from the search tree, and then select one or more atoms from this node. This selected atom is sent to an IDB and/or an EDB for expansion. While the IDB and/or EDB are working on this atom, the PSM can transfer its attention to other nodes in the search tree. An atom sent to the IDB may unify with one or more procedure heads. All corresponding bodies are sent baok to the PSM which initiated the search, either one at a time or all at once. When more than one procedure body is returned for a given atom, several mutually exclusive subgoal nodes are generated. These mutually exclusive goals can then be solved independently in separate machines. Thus each PSM can dynamically initiate another PSM machine, if one is available. As with the goal transmitted by the VAX to a PSM, the goal transmitted from one PSM to another becomes a root of a goal tree in the new PSM whose parent is the sending PSM. Each PSM can independently develop and manage subtrees of the search spaoe generated by the goal node transmitted to the PSM. Each PSM is autonomous except for knowledge of the parent-child relationship. When a goal assigned to a PSM is completely solved it transmits the solution or failure to its parent PSM. The parent of the PSM that contains the original goal is the Host (VAX) machine.

3.2. Control in the PSM

3.2.1_. Control Specification Support - Selection Process

The selection procedure determines the control strategy of the system. The user is permitted to specify guidelines to direct the selection process. The selection procedure has four main functions: node (clause) selection, atom selection, procedure selection and PSM creation.

Node selection: Any node not fully expanded is a candidate for selection. A fully expanded node is a node whose selected atom has been expanded and all leaf nodes descended from the node are either failure nodes or null clauses. A non-fully expanded node may be either an active or an open node. An active node is one from which one or more atoms have been selected for expansion, but is not fully expanded. An open node is one from which no atom has yet been selected for expansion.

Atom selection is concerned with selecting a atom, for expansion, from a selected node in the search tree. There are several system and user defined constraints that affect atom selection.

As defined in section 2, the user can specify which atoms in a node may be executed in parallel and which must be done in sequence, i.e. a partial order on the execution of the atoms. These user specified constraints limit the atoms which can be selected at any stage. Only atoms which do not depend on any other atom or those for which the atoms they depend on have already been solved are candidates for selection.

In addition to user-defined orderings, certain orderings are implied by the node structure itself. Two basic ways in which the contents of a node dictate the ordering on atom selection are: dependent atoms, and special predicates. Two or more atoms in a node are said to be dependent when they share variables. In this case what is desired is the first (or all) binding(s) which cause the atoms to succeed. This can be accomplished either by processing the atoms in parallel and then intersecting the sets of bindings for the shared variables, or by finding a binding which satisfies one predicate and then substituting it in the others and determining if they succeed with that binding (nested loops method). In either method a special AND node is generated with the dependent atoms as its children and one of the above techniques applied. Special predicates are a set of language supplied predicates whose semantios dictate that certain other predicates in the clause must be fully solved before these system defined predicates may be invoked.

Onoe user and system defined constraints have been satisfied, a set of atoms whioh are candidates for selection will remain. The atom seleoted from this set will be seleoted based on user or system supplied heuristics.

Procedure selection is concerned with choosing which procedure body should be given the highest priority when several bodies match an atom. This decision is made exclusively by the IDB.

PSM selection, is concerned with the decision of when to initiate another PSM with a subproblem. Whenever a branching of the search tree is induced by either multiple alternate subproblems (0R-branches) or by independent conjunctive subproblems (AND-branches), this branch becomes a candidate for execution in another PSM machine. The actual process of determining when a new PSM is initiated is discussed below.

3.2.2. PSM Creation

The decision of when to initiate another PSM with a subproblem is not an easy one. If the subproblem is too small, a large amount of overhead would be incurred to solve it. If the subproblem is too large, the parent PSM may complete before the child and remain idle until its children complete. In general it is not possible to determine how complex a subproblem is. Thus no attempt is made to determine the complexity of a subproblem, in the initial system. Instead a new PSM is initiated every time a branching of the search tree takes place, and a PSM is available. At any given instant, there may be several active branches within a PSM, and thus several candidate nodes may be sent to other machines. In this case all or only some of these nodes may be shipped out. This is determined by the user or by system supplied heuristics.

In order to reduce idle time, machines which have completed their alloted task are permitted to accept fresh queries, as follows. If no further processing can be done then either all possible answers for the goals this PSM was invoked with have been found, or all paths resulted in failure, or all paths local to this PSM have been fully explored and there are some children of this PSM which have not yet completed their work. When all answers have been found or all paths have failed, this information is transmitted to the parent of this PSM, and the PSM state is restored to one in which a new query can be accepted. When all local paths have been explored and some chidren PSMs are still active, a data structure is constructed which contains enough information to reconstruct the complete answer from the information in this PSM and from the answers from the currently active children PSM. Once this data structure is constructed, the local proof tree is destroyed and the PSM state is restored to permit a new query to be accepted.

In this manner PSMs are not kept idle in case they complete before their children do. This also allows a PSM to be its own ancestor if so desired.

3.2.2. AND/OR Parallelism

The existence of multiple bodies that match a selected atom result in the formation of an OR-node with each of these bodies as a child. Since these children are independent of eaoh other they may be exeouted in separate machines.

AND-parallelism arises when a conjunction of two or more atoms appear in a node. Independent atoms that do not share variables are executed in parallel and result in the creation of an AND-branch.

3.2.4. Handling Negation

The NOT meta-predicate defined in most sequential logic programming is an implementation of Negation-by-Failure [Clark 1978]. This is not well defined when one or more of the arguments are unbound variables. The behaviour of the NOT meta-predicate can be anomalous in the case where all arguments are not constants. The semantics of negation is extended to handle atoms with variables as arguments by creating a set of bindings for which the atom fails and assuming the negation of the atom holds for precisely this set of bindings.

4. Acknowledgements

The work was Supported by AFOSR grant 82-0303 and NSF grant MCS-79-19418.

5. References

[1] Clark, K.L., "Negation as Failure'', in Logic and Databases, H. Gallaire and J. Minker (Eds.), Plenum Press, 1978, New York.

[2] Kasif, S., Kohli, M., Minker, J., "Control Facilities in PRISM: A Parallel Inference System for Problem Solving", TR, Dept. of Computer Sc, U of MD, College Park, MD.

[3] Kowalski, R.A., "Logic for Problem Solving", Elsevier North Holland Inc., 1979, New York.

[4] Rieger, C, Bane, J., Trigg, R., "ZMOB : A Highly Parallel Multiprocessor", TR-911, Dept. of Computer Sc, U of MD, May 1980, College Park, MD.