# Parallel Prolog with Uncertainty Handling

Katalin Molnár

IQSOFT Intelligent Software Ltd.,
Teleki Blanka u 15-17, H-1142 Budapest, Hungary
*molnar_k@iqsoft.hu*

CUBIQ is a toolset that is built on top of Aurora [Carl 92], an OR-parallel implementation of Prolog for shared memory multiprocessors that provides support for the full Prolog language. CUBIQ introduces frames, blackboard handling, functional notation and uncertainty handling incrementally and independently of each other. As part of the CUBIQ project we investigated how parallelism can be exploited in deduction in uncertain knowledge bases.

Several models of uncertainty are suitable for parallel execution [Pear 88], [Ram 90], [Roj 93]. Logic programming languages have been extended to deal with support logic [Bal 87b], certainty functions [Nara 86] and fuzzy sets [Geig 94]. Meta-interpreters to handle uncertain information in Prolog have been designed by [Ster 86], [Yalç 90].

We used a compiled version of Yalçinalp and Sterling's layered interpreter [Yalç 90] in the CUBIQ toolset. The uncertainty is modelled after support logic programming as described in [Bal 87b] that represents Dempster-Shafer-style uncertainty and incorporates representation methods for fuzzy information and probabilistic rules.

## 1 Uncertain knowledge representation

Support logic programming assigns necessary support and possible support as a pair of values *[Sn,Sp]* to every clause. The rules of the knowledge base are represented by Prolog clauses that are extended with support pairs.

*(Head:- Body) : [Sn,Sp].*

Uncertainty can be assigned not only to whole clauses, but to individual arguments as well (so called fuzzy arguments). Following [Bal 87b] we use semantic unification instead of the Prolog unification for fuzzy arguments. The first use of a fuzzy argument has to be preceded by a CUBIQ declaration of the form:

*:- fuzzy(Name,[(Value:Membership_grade),...]*

There is a smooth transition between logic and uncertain knowledge inside the same knowledge base. One can invoke logical rules from uncertain ones and vice versa through CUBIQ built-in predicates:

*unc(:Call_to_uncertain_knowledge, -Unc)*

*logic(:Call_to_logic_knowledge, +Unc)*

Prolog built-in predicates are handled as having either [1,1] or [0,0] outcome (true or false). Meta-predicates (findall, if-then-else, call, etc) and cut and commit operators are not yet supported.

## 2 Parallel aspects of uncertainty handling

In this project, we focused on those aspects of uncertain knowledge bases in which the scope for exploiting OR-parallelism was most significant.

When evaluating logic knowledge it may be interesting to find either *one* or more solutions. One solution generally means the first evaluation path that leads to a solution. In an uncertain knowledge base, we have to explore *all* paths leading to the same solution in order to get the combined uncertainty that is calculated from the uncertainties of the different paths leading to that solution. The main problem is to decide which of the different solutions is considered the answer. In the best case there is one solution that has a significantly higher degree of certainty (less uncertainty) than all the other solutions. Searching for one solution in a logic knowledge base thus becomes searching for all solutions in an uncertain knowledge base. Searching for all solutions is obviously more effective when done in parallel than sequentially. Note that negated goals (negation by failure in Aurora) are evaluated through full search both in logic and in uncertain knowledge—thus in the case of negated goals, parallel execution in logic knowledge is exploited as much as in uncertain knowledge.

## 3 Implementation

Each predicate with uncertainty is extended with an additional argument, in which its uncertainty is returned. During the execution of the body of a goal the constituent subgoals are evaluated and their uncertainty is combined according to the support logic rules for the AND (Prolog ','), OR (Prolog ';') and NOT (Prolog '\+') operators [Bal 87a]. We have to preserve the uncertainty calculated for one evaluation path until all the other paths leading to the same solution are evaluated. Following the layered interpreter approach [Yalç 90] we use the Prolog dynamic database for asserting the solutions of a goal. Every solution is stored together with a unique identifier and its uncertainty measure. After all solutions of a goal have been found, the uncertainties of unifiable solutions are combined to give the general uncertainty of the solution.

As a first step we implemented a straightforward, unsophisticated Prolog interpreter without considering parallelism. We interpreted a knowledge base using multiple workers and found that the parallel evaluation takes more time than the sequential evaluation. The reason for this slowdown was that a worker performing a dynamic database update will suspend its branch, unless it is leftmost in the OR-tree (to preserve Prolog semantics). As the order of the solutions is not important, we could use the asynchronous database predicates of Aurora, and so avoid unnecessary suspensions.

Another source of the parallelization bottleneck was the improper use of cuts. In many cases the asynchronous database handling predicates had been suspended because they were in the scope of a cut [Haus 90]. This was due to bad programming style of placing cuts at the end of clauses. We restructured the CUBIQ interpreter by placing cuts as early as possible in the alternatives. This helped to get rid of most suspensions.

Finally, we substituted the interpreter with a compiler. The compiled code runs about three times faster than the interpreted code.

We used the ship knowledge base described in [Bal 87a] to evaluate the performance of an uncertain knowledge base. This is a classification problem about recognising warships and ordinary ships on the base of visual, acoustic and radar information. The knowledge base contains 112 clauses.

The table below shows performance results for the ship program in various stages of our implementation. We present execution times in milliseconds for 1, 3 and 4 workers on a SEQUENT/DYNIX system with 4 of 50Mhz Intel486 processors.

| | 1 worker | 3 workers(speedup) | 4 workers(speedup) |
|---|---|---|---|
| simple interpreter | 1526 | 2020 (0.75) | 2178 (0.71) |
| interpreter | 1591 | 638 (2.49) | 515 (3.12) |
| compiler | 593 | 241 (2.46) | 193 (3.07) |

## 4 Future work

The main problem is the representation of the uncertain knowledge in the Prolog language in such a way that all the language structures of Prolog are allowed. The current model of uncertainty in CUBIQ does not fulfill this requirement. The uncertainty of built-in procedures and especially meta-predicates (findall, if-then-else, etc.) should still be investigated.

We devoted some effort to exploring possible interpretations for the cut operator in uncertain knowledge bases. In Prolog we use cut operators when we want to prune the evaluation of other alternatives after some subgoals in an alternative have been found true. We plan to introduce an uncertain cut operator with the following meaning: the cut operator is executed when the preceding subgoals are evaluated with high support (near to true) and ignored when the support for the preceding subgoals is low (near to false). The applicability of the cut operator may also depend on the general support of the containing clause.

A further area to explore is the uncertainty in frames with multiple inheritance [Itz 94]. In this model the ancestor relationship is extended to allow uncertain parent(s). In the CUBIQ tool-set the knowledge engineer is allowed to provide his/her own definition of ancestorship. One may consider using this mechanism for experimenting with such extended forms of inheritance.This may also lead to a search mechanism that is highly parallel.

## 5 Conclusion

Aurora Prolog has been extended with uncertainty handling and experiments show a reasonable speedup on multiple workers. We have found that there is a bigger scope for exploiting parallelism in an uncertain knowledge base than in an ordinary logic knowledge base because of the nature of calculating uncertainty.

Our experiments contributed to expanding our knowledge on applicability of parallel Prolog. We have found the asynchronous database handling predicates of Aurora useful for implementing efficient information transfer between the parallel branches of the search tree. We have found that in a parallel environment one has to pay attention to careful use of the non-logical features of Prolog (such as the cut operator), as improper use may lead to serious performance degradation.

## 6 Acknowledgement

This work has been supported by the European Union in the framework of Co-operation in Science and Technology with Central and Eastern European Countries (CUBIQ — PECO Project 10979).

The author is much indebted to Péter Szeredi for his valuable comments and encouragement.

## References

[Bal 87a] Baldwin, J. F., Martin, T. P., Pilsworth, B. W.: FRIL Programming Language Reference Manual.

[Bal 87b] Baldwin, J. F.: Evidential Support Logic Programming in Fuzzy Sets and Systems **24**, (1987) 1–26

[Carl 92] Carlsson, M., et al.: Aurora Prolog User's Manual.

[Geig 94] Geiger, C., Lehrenfeld, G.: The Application of Concurrent Fuzzy Prolog in the Field of Modelling Flexible Manufacturing Systems, in Proc. of 2nd Int. Conf. on the Practical Applications of Prolog, London, (1994) 233–251

[Haus 90] Hausman, B.: Pruning and Speculative Work in OR-Parallel Prolog, Dissertation, Stockholm, Sweden, (1990)

[Itz 94] I. Itzkovich, I., Hawkes, L. W.: Fuzzy extension of inheritance hierarchies, in Fuzzy Sets and Systems **62**, (1994) 143–153

[Nara 86] Narain, S.: MYCIN: The Expert System and Its Implementation in LogLisp, in van Caneghem, M., Warren D. H.: Logic Programming and its Applications in Ablex Series in AI, (1986) 161–174

[Pear 88] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. USA, (1988)

[Ram 90] Ramsey, C. L., Booker, L. B.: A Parallel Implementation of a Belief Maintenance System, in Proc. The Fifth Ann. AI Systems in Gov. Conf., (1990) 180–186

[Roj 93] Rojas-Guzman, C., Kramer, M. A.: Comparison of Belief Networks and Rule-Based Expert Systems for Fault Diagnosis of Chemical Processes, in Engineering Appl. of AI, Vol. 6, Iss 3, (1993) 191–202

[Ster 86] Sterling, L., Shapiro, E. H.: The Art of Prolog. MIT Press, (1988)

[Yalç 90] Yalçinalp, L. Ü., Sterling, L.: Building Embedded Languages and Expert System Shells in Prolog, in Proc. of 2nd Int. IEEE Conf. on Tools for AI, (1990) 56–62