Towards AKL with Intelligent Pruning
(extended abstract) [*]

Salvador Abreu *Luís Moniz Pereira*
Departamento de Informática
Universidade Nova de Lisboa
PORTUGAL
{spa,lmp}@fct.unl.pt

1 Introduction and Motivation

The Andorra Kernel Language (AKL) computational model is very different from Prolog's, ie. a computation does not explore the solution space by deploying a derivation tree but rather by a sequence of configurations (see [**?**]).

AKL (and the Andorra family of and-or parallel Prolog systems) already makes good use of two search-space narrowing techniques: The Andorra Principle, also known as "Sidetracking", which consists in focussing attention first on goals known to be determinate, and Constraint propagation.

It may therefore be questionable whether it is worth adding more overhead to the runtime execution of these systems in order to obtain a shorter proof (in terms of number of inference steps). We argue that it is, because both search-space narrowing techniques built into the AKL model rely exclusively on forward execution, not on any knowledge of whatever occurred during backward execution, especially the information contained in the occurrence of failures. Our previous work in this direction (see [**?**]) shows that there is a non-negligible gain to be obtained from the failure information, even with "simple-minded" techniques.

Our present work explores techniques susceptible of pruning the search-space so as to obtain consistent gains for combinatorial problems, while retaining an execution efficiency close to that of the original system. To achieve this we propose to make use of binding dependency information to guide the unfolding of an AKL computation. One way this is accomplished is by improving the accuracy of the methods explored in [**?**].

The basic problem of intelligent backtracking (IB) in Prolog (see [**?**, **?**]) is to locate the culprit bindings responsible for the occurrence of a failure, and backtrack directly to their originator, so that unnecessary work (ie. irrelevant to the conflict) is eschewed. To do so, schemes to implement IB in Prolog (for example see [**?**, **?**, **?**]) always refer to the Prolog runtime data structures used to implement nondeterminism, namely choice-points; ie. a binding is said to depend on a given goal if it is associated with that goal's choice-point.

[*]An earlier version of this work was presented at the WELP'93 Workshop.

AKL provides no such concept, nondeterminism being handled differently, by the use of *choice-boxes* which are one of the constructors for configurations. Our approach will tie bindings susceptible of being revised to the entities which may provide alternative bindings: the choice-boxes, and also provides a few other structuring concepts lacking in the computational model. We call our system AKL/IP for "AKL with Intelligent Pruning".

2 AKL with Intelligent Pruning

In order to introduce dependency-based execution in AKL, we provide the following structural devices:

- **Alternative generators.**

  These associate to each binding susceptible of being revised a reference to its alternative generator. The latter may be either a choice-box or an and-continuation.

  The binding of a variable V to a value X in AKL has an associated *binding environment* designated as $\mathbf{env}(V)$, which is simply an and-box. In AKLIP variable bindings will have an extra component, the *alternative generator*, designated as $\mathbf{alt}(V)$ which is a choice-box or an and-continuation, the latter being the "residue" of a choice-box after a determinate promotion. AKLIP choice-boxes also have an **alt** component.

- **Shared binding information.**

  Shared bindings are the result of the choice split operation, and even though the current AKL implementation (AKL/PS 0.8) does not provide actual sharing, the information is needed in order to figure out where a given binding is being used, other than the current and-box.

  The point is to allow us to forcibly remove from the configuration all and-boxes that involve a binding that is known to be invalid. This occurs as the result of a conflict's culprit being a given binding; it is then important to be able to locate all instances of that particular binding in order to fail the and-boxes where they occur.

  For this purpose we define a new component for bindings, and call it **other**. $\mathbf{other}(V)$, where V is a bound variable, will be an and-box, which shares the binding of V with $V's$ environment (as a result of a choice-split operation).

The AKL configuration transitions are extended to deal with the new constructs:

In the forward execution configuration transitions (local forking, determinate promotion, choice split, cut and commit), these structures must be maintained.

During backward execution (environment synchronization, failure propagation and choice elimination), they're used to perform further modifications to the configuration such as:

- Reordering as in [?]. The most significant difference here is that we can now bring the re-incident instances of the failing goal to immediately after the producer of the conflicting binding, which is arguably a near-optimal location.

- Performing a choice split in the alternative generator of one of the bindings involved in the conflict. This has the effect of calling the AKL engine's attention to that particular goal.

As a first approach, the representation of bindings and configuration constructors (and-boxes, choice-boxes and and-continuations) are extended to include circular linked lists for the alternative generators and the shared bindings. Support for the re-incident instances of goals is already available, from the work described in [?].

We're now starting work on the implementation and so cannot at the time of this writing provide any experimental results. Results from [?] indicate that the re-incident instance relocation alone should provide significant speedups for highly combinatorial programs. It is our expectation that having a dependency-directed scheme will eliminate the apparently random effect that the methods described in [?] can have on some programs.

Acknowledgements

We would like to thank Philippe Codognet for various fruitful discussions, and JNICT Portugal for their support.

References

[1] Salvador Abreu, Luís Moniz Pereira, and Philippe Codognet. Improving backward execution in the andorra family of languages. In Apt [?], pages 384–398.

[2] Krzysztof Apt, editor. *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Washington, USA, 1992. The MIT Press.

[3] M. Bruynooghe and L. M. Pereira. Deduction revision by intelligent backtracking. In J.A. Campbell, editor, *implementations of Prolog*. Ellis-Horwood, 1984.

[4] C. Codognet and P. Codognet. Non-deterministic stream AND-Parallelism based on intelligent backtracking. In Levi and Martelli [?], pages 63–79.

[5] Christian Codognet, Philippe Codognet, and Gilberto File. Yet another intelligent backtracking method. In Kowalski and Bowen [?], pages 447–465.

[6] Philippe Codognet and Thierry Sola. Extending the WAM for intelligent backtracking. In Furukawa [?], pages 127–141.

[7] Koichi Furukawa, editor. *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France, 1991. The MIT Press.

[8] Seif Haridi and Sverker Janson. Kernel andorra Prolog and its computation model. In Warren and Szeredi [?], pages 31–46.

[9] Robert A. Kowalski and Kenneth A. Bowen, editors. *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, Seatle, 1988. ALP, IEEE, The MIT Press.

[10] V. Kumar and Y.-J. Lin. An intelligent backtracking scheme for Prolog. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 406–414, San Francisco, August - September 1987. IEEE, Computer Society Press.

[11] Giorgio Levi and Maurizio Martelli, editors. *Proceedings of the Sixth International Conference on Logic Programming*, Lisbon, 1989. The MIT Press.

[12] Ehud Shapiro, editor. *Proceedings of the Third International Conference on Logic Programming*, Lecture Notes in Computer Science, London, 1986. Springer-Verlag.

[13] David H. D. Warren and Peter Szeredi, editors. *Proceedings of the Seventh International Conference on Logic Programming*, Jerusalem, 1990. The MIT Press.