The essence of the Oz higher-order procedure definitions and applications can be modelled in AKL as discussed in Section 3.4.1. (Oz, however, makes different instances of abstractions unequal using its ability to generate unique names.) In a language based on higher-orderness such as Oz, abstractions play a different rôle, however. For example, the extensive support for object-oriented programming offered by Oz relies on it. There is no need for a separate concept of modules, since variable hiding works for procedure names. The view of a program is more flexible.

The put and get operations of Oz constraint communication can be modelled by ports in the same way as the put and take operations of Id M-structures in Section 7.7.4.

The Oz process is a way of assigning a “virtual processor” to an expression. Each expression belongs to a process, and each process is given an opportunity to make progress in a round-robin fashion, much the same way as processes in an operating system. AKL as presented has no processes. A similar effect could be achieved by stipulating that the execution model should be fair, but this would not be enough for practical purposes (as discussed in Section 5.8).

It does not seem altogether unlikely that AKL, in the near future, will evolve in the direction of Oz, and support records, higher-orderness, and processes.

AD LIBITUM[1]

In the following, AKL is assessed in the form of a dialogue between researchers who represent quite different approaches to research in programming language design and implementation, as will be seen. Any resemblance between these fictional researchers and real persons is purely coincidental.

Cast:

S. Worker (a lowly graduate student)

Prof. H. A. Riddle (his supervisor)

Sir Cheswhat (a crusader against *ad hoc*'ery)

Prof. Warden (a guardian of relics)

S. Worker is giving a presentation of AKL, likely to continue for hours due to frequent interruptions by Prof. Riddle, clarifying various points of interest. He has just commented upon the language design, presented early in the talk. We enter the ensuing discussion …

Sir Cheswhat: … but what is your design methodology?

S. Worker: Our starting points were the languages Prolog and GHC, and the conceptual framework provided by concurrent constraint programming.

We tried to combine their essence, operationally, in a manner that allowed using the programming paradigms of both as well as exploiting the potential for parallelism of both. As usual we wanted the result to be simple, efficient, and expressive – you know, the usual stuff …

[1] **ad libitum** (L.) at pleasure

Sir Cheswhat: Yes, I know, but this makes your design entirely *ad hoc*[1]!

S. Worker: Indeed! AKL was designed *for this* very purpose. The AKL computation model was designed to make the synthesis of Prolog and GHC capabilities as smooth as possible. This could not have been achieved by using another, otherwise more general, formalism.

Prof. Warden: You claim that AKL is simple, but I just cannot agree. Comparing with Prolog, you introduce notions such as guard operators, suspension, and not least concurrency. This can hardly be called simple!

S. Worker: AKL is based on comparatively few notions. Based on these, we can model most of the functionality of modern dialects of Prolog. They are not simple. They provide different forms of side effects. They provide cut, inside disjunction and inside call. Many also provide suspension, which interacts strangely with other language features. Compared with this, I call AKL simple.

Sir Cheswhat: I don’t understand this preoccupation with Prolog and GHC. Why not simply recognise that they are both deficient languages of questionable parentage and that it is high time to move to a new beginning?

S. Worker: We are interested in innovation but also in consolidation. With Prolog, commercially available since many years and with a large community of users, and GHC, the basis of the fifth generation project in Japan, are associated many well-developed programming techniques, as well as a large number of applications. AKL encompasses both, while making a number of improvements. One could call it backward compatibility.

Sir Cheswhat: But is this science?

S. Worker: The premise that two given languages are to be synthesised in this manner does not make the problem easier nor less interesting. The underlying principles and implementation technology needed to support such a combination are quite novel, and of general interest for related languages. As regards the scientific method, the corroboration of these ideas involves making them suitably precise, and, in particular, to verify that they meet pragmatic requirements in terms of expressiveness and implementation efficiency.

Prof. Warden: Can you actually run all Prolog programs in AKL?

S. Worker: Not in the way I believe you mean. Whereas GHC is harboured within AKL as a syntactic subset, the exact behaviour of the execution-order sensitive side effect and metalogical operations of Prolog is not faithfully replicated.

[1] **ad hoc** (L.) for this

This doesn't mean that corresponding programs can't be written. It simply means that they have to be written manually, whereas the majority of Prolog programs can be translated into AKL automatically, with the aid of data-flow analysis techniques.

Prof. Warden: If backward compatibility is a concern, shouldn't you give Prolog at least the same amount of attention as GHC? After all, Prolog is by far the more wide-spread language.

S. Worker: The decision was a compromise between innovation and consolidation. It would be possible to augment the AKL computation model with notions of side effects and metalogical operations, and their sequential execution, and thereby achieve better compatibility, but such augmentations would stand out clearly as foreign to the spirit of the model, and would also complicate implementations.

Instead, the augmentation with the process-oriented paradigm provides much better ways of doing things like input/output and manipulation of state. In particular, it does so in a way that interacts with nondeterminism in a meaningful way, something which can't be said about side effect operations.

Prof. Warden: But control in AKL is so explicit; this makes programming less declarative. I would prefer implicit control which exploits, intelligently, the logical structure of the problem representation.

S. Worker: AKL provides a strong logical reading for a large class of programs. To understand what such a program (potentially) computes, its declarative reading is sufficient. To understand how it is done, to understand the algorithmic behaviour of a program, an operational understanding is necessary, and there are simple computation rules to follow to get this understanding.

If the program text is changed, the behaviour is more or less changed, even if the "logic" remains the same. The behaviour of a program is not given by its declarative semantics. An "implicit" control regime may have weak points, where small changes lead to notable differences. That a programmer is not expected to understand the details of the computation model makes the problem worse. In AKL, a highly predictable control regime has been chosen, which the programmer can understand and control.

Prof. Warden: But even so, the control for a particular program is not what I want. For example, can you get these behaviours in AKL?

(He writes down a few small don't know nondeterministic logic programs and an outline of their desired behaviours on a piece of paper.)

(Some time for deliberation passes.)

S. Worker: Well, it seems that you can, for these examples, but in some cases my formulation is somewhat distorted, due to the need to express the particular control desired.

In fact, I wouldn't be surprised if it occasionally turned out to be very difficult to program a particular behaviour, in the worst case at the cost of working entirely at the metalevel, on a representation of the original logic program.

In many ways, the functionality you are looking for is that of an intelligent problem solver. AKL is a programming language, which supports the programming of problem solving programs.

(S. then continues his talk, hoping to get home in time for dinner.)

* * *

S. Worker has managed to fend off most interruptions, and has covered a major section of his talk, the description of the various programming paradigms provided by AKL, when Prof. Riddle cannot hold back his enthusiasm any longer …

Prof. Riddle (spreading his arms in a wide gesture): You see? What did I tell you? You can do anything in AKL!

Sir Cheswhat: In fact, I would have been more surprised if you couldn't write all programs in AKL, considering that it is likely to be Turing complete…

S. Worker: This is true, but the point made is that some languages that provide certain features, e.g., parallelism or don't know nondeterminism, lack the ability to use them to their full potential.

First, not all languages intended for parallel implementation allow arbitrary parallel random access algorithms to be expressed with appropriate efficiency. Prolog is not intended as a general purpose language for parallel machines. GHC is, but lacks the ability to simulate a PRAM efficiently, as do "declarative" languages in general. AKL solves the problem through *ports*, an extension in the spirit of the rest of the language, which can be added to GHC as well.

Second, not all languages with don't know nondeterminism have means to *encapsulate* nondeterminism within a suitable part of a program. In Prolog, nondeterminism is global. For example, the state of a user-interface to a nondeterministic program has to be stored in the database. Backtracking will erase any other state the program might have. AKL provides single and all solutions encapsulation in the form of *deep guards* and bagof, and more controllable encapsulation in the form of *engines*.

Sir Cheswhat: What are these deep guards anyway? Why do you think you need them? Wouldn't in fact engines suffice for encapsulation purposes?

S. Worker: Engines are very powerful, and can be added to any language which can model a state, even to Prolog with freeze, but they certainly do not replace deep guards. Guards provide user-definable conditions and negation. They also allow us to use don’t know nondeterminism at a much smaller scale than that of program modules.

Prof. Warden: I always thought that deep guards were inefficient, and that this was the reason for moving, for example, from *full* GHC to *flat* GHC?

S. Worker: There is an overhead, but it is very small. It is comparable to the overhead of supporting backtracking in Prolog. I believe that no scheme for encapsulating nondeterminism could be made cheaper. Since we provide encapsulated nondeterminism, we can also provide deep guards.

It is possible to virtually eliminate even this overhead for code executing outside any encapsulation, at the price of possibly having two compiled versions of agents which are used inside and outside encapsulation, respectively. The gain would probably be negligible.

Prof. Warden: You say that Prolog is essentially provided, but isn’t a program with deep conditional guards more speculative than the corresponding Prolog program with cut?

S. Worker: It is possible, but only in contrived programs, and then only if the implementation is in conflict with our recommendations.

Sir Cheswhat: I can't give my favourite *xyz* semantics to AKL!

S. Worker: We can give different semantics, or readings, to different subsets of AKL. If a particular reading is important in a certain context, then stick to the corresponding subset. AKL’s lack of an *xyz* semantics for the whole language does not in any way prevent, nor does it seem to make more difficult, formal manipulations such as data flow analyses and program transformations. If and when languages with *xyz* semantics prove their overall advantages, we will have to reconsider our design.

Sir Cheswhat: I still believe one could do much better. There are a number of unexplored possibilities, which could give us the expressiveness you are looking for while resting upon a firm semantical foundation.

S. Worker: I hope you are right! But we are not there yet. Meanwhile, AKL is definitely a step forward compared to Prolog, GHC, and many other languages.

Sir Cheswhat: Well, it's all there in my *Tractatus* anyway…

Prof. Warden: I still don’t understand what’s wrong with Prolog…

(Here the exhausted researchers break for the evening…)

BIBLIOGRAPHY

Abreu, Salvador, Luís Moniz Pereira, and Philippe Codognet [1992]. Improving backward execution in the Andorra family of languages. In *Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming*. The MIT Press.

Agha, Gul [1986]. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press.

Agha, Gul and Carl Hewitt [1987]. Actors: a conceptual foundation for concurrent object-oriented programming. In [Shriver and Wegner 1987].

Aït-Kaci, Hassan [1991]. *Warren's Abstract Machine: A Tutorial Reconstruction*. The MIT Press.

Aït-Kaci, Hassan, Andreas Podelski, and Gert Smolka [1992]. A feature-based constraint system for logic programming with entailment. In *Fifth Generation Computer Systems 1992*. IOS Press.

Ali, Khayri M. and Roland Karlsson [1990]. The Muse approach to or-parallel Prolog. *International Journal of Parallel Programming* **19**(2):129-162. (Also in [Karlsson 1992].)

Armstrong, Joe L., S. Robert Virding, and Mike C. Williams [1991]. Erlang user's guide & reference manual, version 3.1. Computer Science Laboratory, Ellemtel Telecommunications Systems Laboratories, Älvsjö, Sweden.

Armstrong, Joe L., Mike C. Williams, and S. Robert Virding [1993]. Concurrent Programming in Erlang. Prentice Hall.

Bahgat, Reem and Steve Gregory [1989]. Pandora: non-deterministic parallel logic programming. In *Logic Programming: Proceedings of the Sixth International Conference*. The MIT Press.

Bahgat, Reem [1991]. Pandora: non-deterministic parallel logic programming, Ph.D. diss., Department of Computing, Imperial College of Science and Technology, London.

Barth, Paul S., Rishiyur S. Nikhil, and Arvind [1991]. M-structures: extending a parallel, non-strict, functional language with state. In *Functional Programming and Computer Architecture '91*.

Brand, Per [1994]. Decision graph compilation, Draft SICS Research Report, Swedish Institute of Computer Science.

Bueno, Francisco and Manuel Hermenegildo [1992]. An automatic translation scheme from Prolog to the Andorra Kernel Language. In *Fifth Generation Computer Systems 1992*. IOS Press.

Carlson, Björn, Mats Carlsson, and Daniel Diaz [1994]. Entailment of finite domain constraints. In *Logic Programming: Proceedings of the Eleventh International Conference*. The MIT Press. (To appear.)

Carlson, Björn, Seif Haridi, and Sverker Janson [1994]. AKL(FD)—A concurrent language for FD programming. Unpublished manuscript. Swedish Institute of Computer Science.

Carlsson, Mats [1990]. Design and implementation of an or-parallel Prolog engine. Ph.D. diss., RIT(KTH) TRITA-CS-9003, Department of Telecommunication and Computer Systems, The Royal Institute of Technology, Stockholm, and SICS Dissertation Series 02, Swedish Institute of Computer Science.

Carlsson, Mats, Johan Andersson, Stefan Andersson, Kent Boortz, Hans Nilsson, Thomas Sjöland, and Johan Widén [1993]. SICStus Prolog user's manual, SICS Technical Report T93:01, Swedish Institute of Computer Science.

Carriero, Nicholas and David Gelernter [1989]. How to write parallel programs: a guide to the perplexed. *ACM Computing Surveys* **21**(3):323–357.

Cheng, M. H. M., M. H. van Emden, and B. E. Richards [1990]. On Warren's method for functional programming in logic. In *Logic Programming: Proceedings of the Seventh International Conference*. The MIT Press.

Chikayama, Takashi and Y. Kimura [1987]. Multiple reference management in Flat GHC. In *Logic Programming: Proceedings of the Fourth International Conference*. MIT Press.

Chikayama, Takashi [1992]. Operating system PIMOS and kernel language KL1. In *Fifth Generation Computer Systems 1992*. IOS Press.

Chikayama, Takashi, Tetsuro Fujise, and Hiroshi Yashiro [1993]. A portable and reasonably efficient implementation of KL1 (poster abstract). In *Logic Programming: Proceedings of the Tenth International Conference on Logic Programming*. The MIT Press.

Clark, Keith L. and Steve Gregory [1981]. A relational language for parallel programming. In *Proceedings of the ACM Conference on Functional Programming Languages and Computer Architecture*. ACM Press. (Reprinted in [Shapiro 1987].)

Clark, Keith L. and Steve Gregory [1986]. PARLOG: parallel programming in logic. *ACM Transactions on Programming Languages and Systems* **8**(1). (Revised version in [Shapiro 1987].)

Clinger, W. D. [1981]. Foundations of Actor semantics. AI-TR-633, MIT Artificial Intelligence Laboratory.

Clocksin, William F. and Christopher S. Mellish [1987]. *Programming in Prolog*. 3rd ed. Springer-Verlag.

Crammond, Jim [1990]. Scheduling and variable assignment in the parallel Parlog implementation. In *Logic Programming: Proceedings of the 1990 North American Conference*. The MIT Press.

Davison, Andrew [1989]. POLKA: A PARLOG object-oriented language, Ph.D. diss., Department of Computing, Imperial College, London.

DeGroot, Doug [1984]. Restricted and-parallelism. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1984*. Ohmsha, Ltd., Japan.

Foster, Ian and Stephen Taylor [1989]. Strand: a practical parallel programming language. In *Logic Programming: Proceedings of the North American Conference 1989*. MIT Press.

Foster, Ian and Stephen Taylor [1990]. *Strand, New Concepts in Parallel Programming*. Eaglewood Cliffs.

Foster, Ian and Steven Tuecke [1991]. Parallel programming with PCN, version 1.2. Technical Report ANL-91/32, Argonne National Laboratory, University of Chicago.

Foster, Ian and Will Winsborough [1991]. Copy avoidance through compile-time analysis and local reuse. In *Logic Programming: Proceedings of the 1991 International Symposium*. The MIT Press.

Franzén, Torkel [1991]. Logical aspects of the Andorra Kernel Language. SICS Research Report R91:12, Swedish Institute of Computer Science.

Franzén, Torkel [1994] Some formal aspects of AKL. Draft SICS Research Report, Swedish Institute of Computer Science.

Franzén, Torkel, Seif Haridi, and Sverker Janson [1992]. An overview of AKL. In *ELP'91 Extensions of Logic Programming*. LNAI (LNCS) 596, Springer-Verlag.

Gabriel, Richard P [1994]. Lisp: good news, bad news, how to win big. Lucid, Inc. Unpublished.

Gelernter, David, Nicholas Carriero, S. Chandran, and S. Chang [1985]. Parallel programming in Linda. In *Proceedings of the International Conference on Parallel Processing* (St. Charles, Ill., Aug.). IEEE Computer Society Press.

Girard, Jean-Yves [1987]. Linear logic. *Journal of Theoretical Computer Science* **50**(1).

Goldberg, Yaron, William Silverman, and Ehud Shapiro [1992]. Logic programs with inheritance. In *Fifth Generation Computer Systems 1992*. IOS Press.

Gregory, Steve [1987]. *Parallel Logic Programming in PARLOG*. Addison-Wesley.

Gregory, Steve and Rong Yang [1992]. Parallel constraint solving in Andorra-I. In *Fifth Generation Computer Systems 1992*. IOS Press.

Gregory, Steve [1993]. Experiments with speculative parallelism in Parlog. In *Logic Programming: Proceedings of the 1993 International Symposium*. The MIT Press.

Gudeman, David, Koenraad De Bosschere, and Saumya K. Debray [1992]. jc: an efficient and portable sequential implementation of Janus. In *Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming*. The MIT Press.

Gupta, Gopal and Manuel Hermenegildo [1991]. ACE: and/or-parallel copying-based execution of logic programs, Technical Report TR-91-25, Department of Computer Science, University of Bristol.

Gupta, Gopal and Manuel Hermenegildo [1992]. Recomputation-based implementations of and/or-parallel Prolog. In *Fifth Generation Computer Systems 1992*. IOS Press.

Gupta, Gopal, Vitor Santos Costa, Rong Yang, and Manuel Hermenegildo [1991]. IDIOM: a model integrating dependent-, independent-, and or-parallelism. In *Logic Programming: Proceedings of the 1991 International Symposium*. The MIT Press.

Haridi, Seif and Per Brand [1988]. Andorra Prolog, an integration of Prolog and committed choice languages. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*. Ohmsha, Ltd., Japan.

Haridi, Seif [1990]. A logic programming language based on the Andorra model. *Journal of New Generation Computing* **8**(7):109–125. (Revised version of [Haridi and Brand 1988].)

Haridi, Seif and Sverker Janson [1990]. Kernel Andorra Prolog and its computation model. In *Logic Programming: Proceedings of the Seventh International Conference*. The MIT Press. (Revised version of SICS Research Report R90002.)

Haridi, Seif and Catuscia Palamidessi [1991]. Structural operational semantics for Kernel Andorra Prolog. In *Proceedings of PARLE'91*. LNCS, Springer-Verlag.

Haridi, Seif, Sverker Janson, and Catuscia Palamidessi [1992]. Structural operational semantics for AKL. *Journal of Future Generation Computer Systems* **8**(1992).

Hermenegildo, Manuel and K. Greene [1990]. &-Prolog and its performance: exploiting independent and-parallelism. In *Logic Programming: Proceedings of the Seventh International Conference*. The MIT Press.

Hewitt, Carl E. and Henry Baker [1977]. Actors and continuous functionals. In *Proceedings IFIP Working Conference on Formal Description of Programming Concepts*.

Hudak, Paul and Philip Wadler [1991], eds. Report on the programming language Haskell: a non-strict, purely functional language, version 1.0. Yale University and University of Glasgow. (haskell@cs.yale.edu, haskell@cs.glasgow.ac.uk)

Jaffar, Joxan and Jean-Louis Lassez [1987]. Constraint logic programming. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*. ACM Press.

Janson, Sverker and Seif Haridi [1991]. Programming paradigms of the Andorra Kernel Language. In *Logic Programming: Proceedings of the 1991 International Symposium*. The MIT Press. (Revised version of SICS Research Report R91:08.)

Janson, Sverker and Johan Montelius [1992]. The design of a prototype implementation of the Andorra Kernel Language. Public Deliverable Report, ESPRIT project PEPMA (EP 2471).

Janson, Sverker, Johan Montelius, and Seif Haridi [1993]. Ports for objects. In *Research Directions in Concurrent Object-Oriented Programming*, MIT Press.

Janson, Sverker and Seif Haridi [1994]. An introduction to AKL—a multiparadigm programming language. In *Constraint Programming*, NATO-ASI Series, Springer-Verlag.

Janson, Sverker, Johan Montelius, Kent Boortz, Per Brand, Björn Carlson, Ralph Clarke Haygood, Björn Danielsson, and Seif Haridi [1994]. AGENTS user manual, Draft SICS Research Report, Swedish Institute of Computer Science.

Kahn, Kenneth M., Eric Dean Tribble, Mark S. Miller, and Daniel G. Bobrow [1987]. Vulcan: logical concurrent objects. In [Shriver and Wegner 1987]. (Also in [Shapiro 1987].)

Kahn, Kenneth M. and Vijay A. Saraswat [1990]. Actors as a special case of concurrent constraint programming. In *OOPSLA/ECOOP '90 Conference Proceedings*. ACM Press.