My Personal Scientific Outlook

Hassan Aït-Kaci

HassanAitKaci@gmail.com

hak@acm.org

December 2011 — Present

Last modified on Sun Apr 19 10:21:18 2015 by hak

Table of Contents

- Overview
- Why? – How? – What?
- Background and Perspective
- Wherein Lies the Knowledge?
- Technical Overview
  - The Web of LIFE
  - LIFE's Experience
  - Formal Work
  - Implementation Work
  - References
- Selected Links, Publications, and Talks

Overview

This document gives the author's personal outlook for the next few years in terms of opportunities offered by applications of Artificial Intelligence (AI) technology to automatic decision-making in the real world; namely, Knowledge Representation (KR) and Automated Reasoning put to profit-making work by exploiting knowledge implicit in data. It gives a summary of why, how, and what technology may be put to effective use in the emerging context of networked intelligence.

Near the end of 2009, as I was idling through potentially interesting TED Talks, I just happened on this Why? – How? – What? analysis of one's organization of one's work that made great sense to me. Since then, I too strive to abide by this rhetorical strategy whenever I should (thank you kindly Mr Sinek). Simple is beautiful, as always. I use it because, if it works, it helps me focus and be efficient, as well as convincing (esp., *self*-convincing). If it fails, or is nonsensical, it still helps me find out *why*, think about *how* to improve it by understanding *what* makes it lame; or, or junk the whole thing as gobbledygook (or stack it away for later, if and when I ever become smarter and/or have time to waste—no need for apnea however).

Why? – How? – What?

- **Why?**—I believe that information and services available on the World Wide Web can be made vastly more intelligent than they are today. I believe that the web can be made into a self-adapting distributed medium truly capable of reasoning, learning, and evolving.

- **How?**—I propose a knowledge-representation and processing technology of disconcerting simplicity using the emerging W3C standards (*viz.*, RDF, Linked Data) as the perfect support to achieve what we seek simply by interpreting such information structures as collections of elementary *graph-based constraints*.

- **What?**—It is possible to enable ***knowledge***-oriented processing orders of magnitude more efficient, and more scalable, than is available today given the *de facto* approach followed by the ***data***-oriented technology we have inherited.

Even if you don't give a hoot, please check this out for a starter.

Background and Perspective

My qualifications and career have been in scientific research. My principal interests are in Artificial Intelligence (AI)—formalisation, application, and practice. My area of expertise is in: Programming and Natural Language Processing (NL) (effective and efficient operational semantics), and Data & Knowledge Base systems (incl. reasoning under uncertainty, knowledge acquisition, use, and maintenance, data aggregation and mining).

My work has now become relevant to the emerging context of superabundant data and the need to extract, represent, and use knowledge from raw data. I believe my ideas have direct practical, marketable, money-making potential for applications in very "hot" areas.

The simplest high-altitude perspective of what I propose is to identify, implement, test, adapt, and exploit with the latest technology, the various ideas in computer-based Knowledge Representation and Automated Reasoning that I and others have conceived and elaborated over the past several years in the light of arising opportunities. The rest of this document is to explain this claim.

Wherein Lies the Knowledge?

The next wave of information processing must adapt to a radical change of reality—namely, the enormous quantity of available data and the rate at which it accumulates. Implicit in this data hides a *wealth* of information—literally!

A recent article illustrates this by reporting the noticeable prediction success of a small data analysis company called Recorded Future whose main office is located in Gothenburg, Sweden. Such is this company's rate of success in predicting world's events and situations before anyone else, that most major world players (including Google and the CIA!) line up as its customers. How they do it is their trade secret, of course—but, put simply, *they find all they need in publicly available data*.

Yet, as blatantly successful as his company may be, Recorded Future's co-founder and CEO Christopher Ahlberg makes the following statement:

> *"... to develop a tool that could create predictions for any input, from finance to terrorism, would be much harder. [One] would not only have to index the internet, but also understand and interpret it."*
>
> —Christopher Ahlberg as quoted by Tom Cheshire in *Wired*–November 10, 2011

Indeed, by this statement, Recorded Future's boon may only be the tip of an iceberg. As explained next, this is where what I discuss below becomes relevant—how to extract and use knowledge hidden but implicit in public data. And we're talking about **Big Data**!

Technical Overview

The Web of LIFE

> The study of pattern is crucial to the understanding of living systems because systemic properties (...) arise from a configuration of orderered relationships.
>
> **Fritjof Capra—*The Web of Life***

In simple terms, my contribution deals with making it possible to reason with labelled graph objects and concepts. As it turns out, emerging standards for the nascent "Semantic Web" such as RDF and Linked Data rely precisely on such a representation format. One of the objectives of such a format is to allow processing of knowledge in the same efficient and scalable manner that has been possible for data. However, knowledge being more abstract than data, such processing must rely on adequate formal computational technology. One such technology is Constraint Processing. What is a "constraint" then? Simply, a constraint is a relational expression for which we know an efficient algorithm to normalize it into a canonical form—some of such canonical forms being solved forms. The essence of our technology is that of seeing labelled graphs as efficiently processable constraints.

How can a useful and usable knowledge representation and processing system be both formally simple and uniform, yet sufficiently effectively powerful, to accommodate the latest promising technology?

The essential idea that we advocate addresses the above simple question based on:

- viewing everything as a labelled graph;
- interpreting a labelled graph as a constraint;
- relying on operationally efficient graph constraint-solving;
- using it in the context of the Semantic Web.

Thus, this summary is an overview of the author's vision and perspectives based on some of the latest research for the development of an operational platform for Semantic Web (SW) applications. For effective, efficient, and scalable knowledge representation and use, and its acquisition from natural language sources, such knowledge must be encoded somehow. A key factor in what is proposed is the timely pervasive adoption of emerging standards such as the W3C's Resource Description Framework (RDF) [3] that represents all SW data and knowledge as (labelled) graphs —*viz.*, Linked Data [4].

As it turns out, our most essential technical insight is that such RDF graphs may be viewed as formal constraints, thus enabling efficient reasoning operationally. Inference in labelled graphs amounts to (graph) matching and (graph) unification, which respectively implement (logical) implication and (logical) conjunction of the formulas denoted by the graphs (since constraints have a simple logical semantics). Thus, our approach seems a perfect fit for processing knowldege represented as constrained graphs universally represented as RDF triples and exchanged in XML syntax. For processing such triple-based data, we may now define an inference engine based on well-understood compiling techniques from Constraint Logic Programming (CLP). Emphatically, we advocate a constraint-based approach because it is both formal and practically efficient. In addition, it is the key for accommodating necessary real-life compromises such as incomplete and approximate reasoning, mixing symbolic (*e.g.*, logical, functional, order-theoretic, *etc.*) and numerical (*e.g.*, probabilistic, fuzzy, rough-set, *etc.*) formalisms and algorithms. This may thus simply be construed as the timely taking advantage of this perfect fit, exploiting a well-understood computational system that is: (1) ***formal***, (2) ***operational***, (3) ***efficient***, (4) ***scalable***, and (5) ***extensible***.

LIFE's Experience

During the heydays of research in "symbolic programming" languages at the end of the 20th century, the author contributed to the design and implementation of one particular approach—the LIFE system, a programming language based on **L**ogic, **I**nheritance, **F**unctions, and **E**quations. In several ways, LIFE's ideas anticipated the latter evolution that followed of symbolic programming into constraint-based programming, where all data is formalized and implemented as constraints. It was indeed realized that Constraint Programming (CP) subsumes both Functional Programming (FP) and Logical Programming (LP) in the sense that FP and LP may be viewed as working on the particular constraint systems of First-Order Term (FOT) matching and unification. LIFE simply extends the data constraint language to be more expressive using Order-Sorted Featured (OSF) terms, which are much more expressive than FOTs. Indeed, ***the essence of LIFE is that all data is a computationally enforceable labelled-graph constraint. Rules, whether functional or logical,***

acting on data so-abstracted as labelled-graph constraints, can then be used in turn to specify arbitrarily more complex knowledge via executable constraints.

This is indeed a unique advantage that comes for free from the OSF constraint system underlying LIFE [1, 2]. It is now of direct and mature relevance to the nascent adoption of such representation standards as RDF [3] and Linked Data [4]. Since then, and especially over the past couple of decades, there have been several technological innovations that have come to maturity in the context of the Semantic Web. The latter has now provided for greater challenges to CP than just self-contained constraint-based programming—however useful or efficient. Those challenges offering the most intriguing prospects in the line of the ideas behind LIFE are (see references [4, 3, 5, 6, 7, 8, 9]):

- Linked Data (or more generally the RDF triple-based representation of all Semantic Web data);
- Data mining and rule learning—especially "non-crisp" rules (*e.g.*, fuzzy [9], rough sets, Bayesian technology, *etc.*) accommodating approximate reasoning in harmony with symbolic logic;
- Natural Language interfacing to knowledge repositories;
- Management and maintenance of evolving knowledge bases.

The program that this entails is therefore ambitious. The initial focus of our effort rests on the exploitation of emerging standards from the W3C for representing Internet-linked data; namely, Linked Data and its supporting RDF technology.

The main technical facets of my work and that of others that are relevant to enabling effective and efficient KR technology are overviewed below.

Formal Work

- Over the several past decades, a simple and general formal data structure has proven to be pervasively adopted in almost all venues of Computer Science—namely, *fielded classes and objects* populating them. Such data structures have also turned out to be most adequate in AI for Knowledge Representation and Natural Language Processing. Contributions in AI and DB languages have formalized such structures as labelled graphs. Such graphs are simply a "compiled" form of any variations on the Entity-Attribute-Value (E-A-V) Model.

  One particular approach that we advocate is to see such graphs as very simple and easily enforceable *constraints*, for practical reasons. It allows representing and manipulating graph-based objects (*e.g.*, record types) as Order-Sorted Feature (OSF) objects, that is simple, efficient, and practical. This formalism is a basic mathematical rendition of the essential informal insights underlying Semantics Networks of the 80's and 90's. The most interesting aspect of it is that it offers a direct interpretation of labelled graphs representing structural knowledge as efficiently solvable constraints.

  Reasoning with large and complex structures is done by interpreting such graphs as conjunctive or disjunctive sets of elementary constraints. Moreover, it turns out that these elementary graph constraints map naturally into a triple-based representation such as offered by RDF, the universal Internet data representation language and heirs such as LinkedData (see also Linked Data on wikipedia).

In simple terms, OSF technology provides a set of formal and practical tools appropriate for the Semantic Web.

- As it evolved out of unification, the OSF approach lends itself also to efficient implementation as it can be compiled into machine-level instructions. Moreover, these basic instructions are seen themselves as elementary constraints. This renders possible a generic abstract run-time machinery by simple instantiation of independent constraint solvers over constrained logical variables.

  The idea is to exploit years of research into such execution models (*e.g.*, LP, CLP, LIFE, Oz, *etc.*) using and adapting the techniques to work with current standards where appropriate. The objective is *not* to rebuild the universe, only better. Rather, it is to take advantage of a few techniques that were invented, prototyped, and implemented mostly in the context of CLP programming languages and systems.

  The somewhat technical issue explained in the next item's two paragraphs is summed up in one simple picture:

Innovation takes courage. . . (from Martin Wildberger's "Smarter Planet" Keynote, CASCON 2009)
If I'd asked my customers what they wanted, they'd have said a faster horse!—Henry Ford
hak
OSF Constraint Solver
No thanks!
We're busy
Semantic Web
W3C
OWL, Tableaux, DL-based reasoning, etc.

- With the advent of the Semantic Web and the W3C commitment to make it a reality, the predominantly cited software enabling actual KR reasoning has been the OWL family of languages. They are based on a set of techniques that are variations of bottom-up finite-model building. These methods are variations on tableau-based reasoning techniques and make the operational basis for Description Logics, or DLs (*i.e.*, OWLs and related formal languages are all DL-based). Technically, tableau-based reasoning works by building the *extension* of a DL-sentence—its model—following a least fixed-point semantics. It is precisely because OWL reasoning works by actually populating a sentence's model that it is limited to finite-domain formulas.

Independently, another approach to KR evolved out of unification-based computing—the order-sorted feature graph constraint formalism. It too has a logical semantics—but that can be formalized as a *constraint*-based logic. Thus, it may be realized using *lazy* proof methods. Unlike tableau-based techniques, these methods do not prove a descriptive sentence by building its model, but by reducing its syntax into normal form which can be one of three kinds: (1) solved, (3) inconsistent, or (3) undecided—hence the "laziness". Formally, these techniques follow a greatest fixed-point semantics. Contrary to DL-based reasoning, it does not prove a formula by building its model; it simply keeps formulas in normal (not necessarily solved) form. Such normal forms that are recognized as solved forms denote all the solutions in *intension*. In other words, there is no need to enumerate the elements populating a solution. This allows formulas denoting infinite models.

- Once understood, the formal relation between DL and OSF proof techniques makes it clear why the former faces formidable challenges in order to scale up to sentences denoting large or infinite models, while the latter does not since it relies on constraint normalization—*i.e.*, syntax simplification rules. The price to pay for such efficiency is some mild form of incompleteness as reduction to normal-form may yield inconclusive forms.

- Like data, knowledge is *not* static. Conjugating the time dimension into the structural ones, it is possible to make graph-based constraints controlable by temporal event-triggered agents. Interestingly, the reasoning power needed for the realistic orchestration and choreography of such events is also formalizable using constraint-based reasoning (most notably, soft temporal constraint-solving). Dynamic Schema evolution must always be kept faithful with the actually encoded knowledge. Although it is expected that most DB maintenance and operation technology can and will be reusable, or easily adaptable, to the KB needs, there will also be notable differences.

- Last, but not least, OSF graph-constraint technology has been at work with great success in two essential areas of AI: NLP and Machine Learning:

    - Interestingly, though not surprisingly, the formal approach we advocate for expressive knowledge representation and efficient implementation thereof (using OSF constraints) based on "feature-object with type inheritance" has been a major paradigm in the field of NLP for a long time [15]—so-called "Head-driven Phrase Stucture Grammar" (HPSG) and Unification Grammar technology. This is indeed not surprising given the ease with which feature structure unification enables combining both syntactic and semantic information in a clean, declarative, and efficient way.

    - Similarly, while most of the attention in the OSF literature has been devoted to *unification*, an operation that basically computes the most general OSF term subsumed by two given OSF terms, the dual operation—namely, *generalization*—is just as simple to use, which computes the most specific OSF term that subsumes two given terms (see also this). This operation is central in Machine Learning and with it, OSF technology lends itself to be combined with popular Data Mining techniques such as Support Vector Machines using frequency or probabilistic information (see [16], *e.g*).

Implementation Work

- Implementation of OSF constraint technolgy has the advantage of being one based on compilation to lower level basic constraint-solving abstract machines. For the past 15 years, the author has developed a Java-based meta-compiler system with built-in XML serializing capabilities—the **Jacc** system (**J**ust **a**nother **c**ompiler **c**ompiler). The essential motivation for this work has been to provide the tooling needed for experimentation with the convenience of easy computer-based language parser generation—syntax *and* semantics. One of Jacc's most appreciated feature has been how it simplifies and facilitates the "XML-ification" of any syntax with minimal grammar annotation.

  The recent proposal to use easier-to-read formats than XML for RDF (namely, JSON and JSON-LD), if generated by Jacc (as an optional alternative to RDF in XML), should make Jacc a handy tool for experimenting with diverse potential specifications interpreting KBs made of sets of RDF-triples as order-sorted graph constraints. Below is a high-altitude overview what Jacc is already useful (and used) for.

- At first sight, Jacc may be seen as a "100% Pure Java" implementation of an LALR(1) parser generator [10, 2] in the fashion of the well known UNIX tool known as "Yacc"—"Yet another compiler compiler" [13]. However, Jacc is in fact much more than... *just another compiler compiler!* It extends Yacc to enable the generation of flexible and efficient Java-based parsers and provides enhanced functionality rarely available in other similar systems.

- The fact that Jacc uses Yacc's meta-syntax makes it readily usable on most Yacc grammars. Other Java-based parser generators all depart from Yacc's format, requiring nontrivial meta-syntactic preprocessing to be used on existing Yacc grammars—which abound in the world, Yacc being by far the most popular tool for parser generation. Importantly, Jacc is programmed in pure Java—this makes it fully portable to all existing platforms, and immediately exploitable for web-based software applications. Jacc further stands out among other known parser generators, whether Java-based or not, thanks to several additional features. The most notable are:

  - Jacc uses the most efficient algorithm known to date for its most critical computation (*viz.*, the propagation of LALR(1) lookahead sets). Traditional Yacc implementations use the method originally developed by DeRemer and Penello [12]. Jacc uses an improved method due to Park, Choe, and Chang [14], which drastically ameliorates the method of by DeRemer and Penello. To this author's best knowledge, no other (available) Java-based meta-compiler system implements the Park, Choe, and Chang method [11].

- Jacc allows the user to define a complete class hierarchy of parse node classes (the objects pushed on the parse stack and that make up the parse tree: nonterminal and terminal symbols), along with any Java attributes to be used in semantic actions annotating grammar rules. All these attributes are accessible directly on any pseudo-variable associated with a grammar rule constituents (*i.e.*, $$, $1, $2, *etc.*).
- Jacc makes use of all the well-known conveniences defining precedences and associativity associated to some terminal symbols for resolving parser conflicts that may arise. While such conflicts may in theory be eliminated for any LALR(1) grammar, such a grammar is rarely completely obtainable. In that case, Yacc technology falls short of providing a safe parser for non-LALR grammar. Yet, *Jacc can accommodate any such eventual unresolved conflict using non-deterministic parse actions that may be tried and undone.*
- Further still, Jacc can also tolerate *non-deterministic tokens*. In other words, the same token may be categorized as several distinct lexical units to be tried in turn. This allows, for example, parsing languages that use no reserved keywords (or more precisely, whose keywords may also be tokenized as identifier).
- Better yet, it accommodates run-time grammar symbol precedence specification: Jacc allows dynamically (re-) definable operators in the style of the Prolog language (*i.e.*, at parse-time). This offers great flexibility for on-the-fly syntax customization, as well as a much greater recognition power, even where operator symbols may be overloaded (*i.e.*, specified to have several precedences and/or associativity for different arities). In fact, this feature essentially makes Jacc an LALR(k) parser generator, for any number k ($k > 0$) of look-ahead symbols.
- Jacc supports partial parsing. In other words, in a grammar, one may indicate any nonterminal as a parse root. Then, constructs from the corresponding sublanguage may be parsed independently from a reader stream or a string.
- Jacc automatically generates a full HTML documentation of a grammar as a set of interlinked files from annotated `/**...*/` javadoc-style comments in the grammar file, including a navigatable pure grammar in "Yacc form," obtained after removing all semantic and serialization annotations, leaving only the bare syntactic rules.
- Jacc may be directed to build a parse-tree automatically (for the concrete syntax, but also for a more implicit form which rids a concrete syntax tree of most of its useless information). By contrast, regular Yacc necessitates that a programmer add explicit semantic actions for this purpose.
- Jacc supports a simple annotational scheme for automatic XML serialization of complex Abstract Syntax Trees (AST's). Grammar rules and non-punctuation terminal symbols (*i.e.*, any meaning-carrying tokens such as, *e.g.*, identifiers, numbers, *etc.*) may be annotated with simple XML templates expressing their XML forms. Jacc may then use these templates to transform the Concrete Parse Tree (CST) into an AST of radically different structure, constructed as a JDOM XML document. This yields a convenient declarative specification of a tree transduction process guided by just a few simple annotations, where Jacc's "sensible" behavior on unannotated rules and terminals works "as expected." This greatly eases the task of retargeting the serialization of a language

depending on variable or evolving XML vocabularies.

- The next step is to extend Jacc by providing alternative structure-generating options besides XML, such as the JavaScript Object Notation (JSON) and its version for Linked Data (JSON-LD, or in fact any Entity-Attribute-Value (E-A-V) Model data structure description formalism such as, *e.g.*, Pivot and Virtuoso). With this tool, it will then be easier to experiment using Jacc to generate RDF-triples (or variations thereof) as compilation schemes from high-level (*i.e.*, more legible and user-friendly) KR languages (such as, *e.g.*, OSF or LIFE syntax—or even higher level *e.g.*, NL dialects).

Keywords: Constraint-Logic Programming; Functional Programming; Rule-Based Computing; Object-Based Computing; Ontological Reasoning; Approximate Reasoning; Knowledge Acquisition; Rule Learning; Language Parsing; Meta-compilation; XML; RDF; Linked Data; JSON; JSON-LD; Entity-Attribute-Value Model.

References

1. Hassan Aït-Kaci, "An introduction to LIFE—Programming with Logic, Inheritance, Functions, and Equations," in Dale Miller, editor, *Proceedings of the International Symposium on Logic Programming*, pp. 52–68, MIT Press, October 1993.
2. Hassan Aït-Kaci, "Data models as constraint systems—A key to the Semantic Web," *Constraint Processsing Letters*, 1(1):33–88, November 2007.
3. RDF Working Group, "Resource Description Framework (RDF)," *World-Wide Web Consortium*, February 10, 2004.
4. Christian Bizer, Tom Heath, and Tim Berners-Lee, "Linked Data—The Story So Far," in *International Journal on Semantic Web and Information Systems (IJSWIS)*, Special Issue on Linked Data, Tom Heath and Christian Bizer, Eds., 2009 (see also this).
5. Dan Brinkley and Ramanathan V. Guha, "RDF Vocabulary Description Language 1.0—RDF Schema," *World-Wide Web Consortium*, February 10, 2004.
6. Ben Adida and Mark Birbeck, "RDFa Primer—Bridging the human and data webs," *World-Wide Web Consortium*, October 14, 2008.
7. Alistair Miles and Sean Bechhofer, "SKOS—Simple Knowledge Organization System Reference," *World-Wide Web Consortium*, August 18, 2009.
8. Leora Morgenstern, Chris Welty, and Harold Boley, "RIF Primer," *World-Wide Web Consortium*, 2010.
9. Ermir Rogova, Panagiotis Chountas, and Krassimir Atanassov, "Imprecise Data and Knowledge Based OLAP," in *Proceedings of the 11th East-European Conference on Advances in Databases and Information Systems (ADBIS 2007)*, Varna, Bulgaria, Sept. 29–Oct. 03, 2007.
10. Alfred Aho, Ravi Sethi, and Jeffrey Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986. (This text is also familiarly known as *"The Dragon Book"*.)
11. Kwang-Moo Choe, Private email communication, Korean Advanced Institute of Science and Technology, Seoul, South Korea, December 2000.
12. Frank DeRemer and Thomas Penello, "Efficient computation of lookahead sets," *ACM Transactions of Programming Languages and Systems*, 4, 4 (October 1982), 615–749.
13. Stephen C. Johnson, "Yacc: Yet another compiler compiler," *Computer Science Technical Report 32*, AT&T Bell Labs, Murray Hill, NJ (USA), 1975. (Reprinted in the 4.3 BSD Unix Programmer's Manual, Supplementary Documents 1, PS1:15. UC Berkeley, 1986.)
14. Joseph Park, Kwang-Moo Choe, and C.H. Chang, "A new analysis of LALR formalisms," *ACM Transactions of Programming Languages and Systems* 7, 1 (January 1985), 159–175.
15. Hassan Aït-Kaci and Patrick Lincoln, "LIFE—A Natural Language for Natural Language," *T.A. Informations*, 30(1–2):37–67, 1989.
16. Hassan Aït-Kaci and Yutaka Sasaki, "An Axiomatic Approach to Feature Term Generalization," in *Proceedings of the 12th European Conference on Machine Learning* (ECML 2001), pp. 1–12, Freiburg, Germany, September 5–7, 2001.

Selected Links, Publications, and Talks

This section indicates a few links that represent the author's most relevant public documents. This is by no means a complete list of his published contributions. For an exhaustive and up-to-date list of publications and appearances, please click on the link for the detailed CV below.

- Short bio (English - Français)
- LinkedIn site: http://www.linkedin.com/in/hak2007
- Detailed CV
- Self-introduction to the W3C RIF WG
- Data models as constraint systems—A key to the Semantic Web
- Keynote speaker at WI-IAT 2011 (talk abstract, presentation slides)
- DL *vs.* OSF—a short (6 mins) video explaining the gist of of how Description Logic reasoning and Order-Sorted Feature constraint processing relate. This is a commented animation of my presentation at the 2007 International Workshop on Description Logics (DL2007) in Brixen-Bressanone, Italy, 8–10 June, 2007.
- "Efficient Implementation of Lattice Operations," Hassan Aït-Kaci *et al.*, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(1), January 1989. (The key to compiling partially-ordered knowledge that can scale up to very large ontologies.)
- An introduction to LIFE—Programming with Logic, Inheritance, Functions, and Equations
- "LIFE—A Natural Language for Natural Language," Hassan Aït-Kaci and Patrick Lincoln, *T.A. Informations*, 30(1-2), Association pour le Traitement Automatique des Langues, Paris, France, pp. 37–67, 1989.
- LIFE Su Doku—An interesting–and entertaining–illustration of how LIFE's added declarativeness enables constraint-activated daemons to keep a tight control on solution searching by detecting any violation at the earliest possible time, thus automatically yielding a surprisingly focused solving strategy.