*Supported by the
National Science Foundation
Engineering Research Center
Program (NSFD CD 8803012),
Industry and the University*

Knowledge Representation in PARKA - Part 2: Experiments, Analysis, and Enhancements

by L. Spector, B. Andersen, J. Hendler, B. Kettler, E. Schwartzman, C. Woods, and M. Evett

TR 92-10

Knowledge Representation in PARKA – Part 2: Experiments, Analysis, and Enhancements

Lee Spector, Bill Andersen, James Hendler, Brian Kettler
Eugene Schwartzman, Cynthia Woods, and Matthew Evett[1]

Department of Computer Science
Institute for Advanced Computer Studies
Systems Research Center
University of Maryland
College Park, MD 20742
email: spector@cs.umd.edu

1/28/92

Abstract

Our research group has designed and implemented a symbolic knowledge representation system called PARKA which runs on the Connection Machine, a massively parallel SIMD computer [9]. The semantics of this system are discussed in [11]. The details of the Connection Machine implementation and discussions of performance considerations can be found in [3],[4][5][6] and [7]. In the past year the PARKA project has made significant advances along several fronts of both theoretical and practical significance. This paper summarizes some of this work and outlines directions for further research.

1. Introduction

PARKA is a frame-based knowledge representation system which was designed to run on the Connection Machine (CM), a massively parallel SIMD computer with up to 64,000 processors [9]. The first phase of the PARKA project involved the initial coding of frame-system algorithms for the Connection Machine [6], and the development of a coherent high-level language based upon the perceived strengths and weaknesses of these algorithms [11]. The semantics of the system are discussed in [11], and discussions of implementation and performance considerations can be found in [3], [6] and [7].

Recent work in the PARKA project has proceeded along two fronts. One group has been extending and refining the Connection Machine implementation and analyzing its performance (see, e.g., [8]). The other group has been using PARKA to encode real-world knowledge in order to assess the semantic and practical adequacy of PARKA as a knowledge representation language. It is the work of this latter group that is summarized in the following pages.

PARKA represents the references of category and individual terms by specifying descriptions of the category members and individuals to which such terms apply. PARKA is a *terminological* language with no *assertional* component, in the sense of [1]. Although the descriptive facilities are

[1]Supported in part by ONR grant N-00014-91-J-1451 and NSF grant IRI-8907890. J. Hendler and L. Spector are also affiliated with the UM Systems Research Center (NSF EDR 8803012).

not so elaborate as those of several previous systems (e.g., KL-ONE and related systems [16]), great care was taken in the design of those constructs that *are* provided. For example, PARKA provides a principled approach to multiple inheritance based on the work of Touretzky [12], and a new mechanism for the representation of part/whole relations. While the availability of 64,000 processors can reduce drastically the run-times of certain operations, the expression of symbolic reasoning algorithms in a SIMD framework is non-trivial. Hence all of the representation mechanisms included in PARKA were designed to have reasonably simple SIMD algorithms, as well as to provide adequate expressive capabilities. Although we have always felt that PARKA's facilities would be useful for a wide range of AI applications, we previously had little data on which base such an assertion. The work described in this paper was undertaken to establish more firmly the utility of PARKA as a knowledge representation language.

In the remainder of this section we provide a review of PARKA's main representational conventions. In Section 2 we describe a set of knowledge bases that we have built in order to experiment with PARKA's capabilities. In 2.1 we describe and analyze a fairly large (over 1500 frames) knowledge base of biological and geographical knowledge. In 2.2 we briefly discuss the use of PARKA in case-based planning research. In Section 3 we reflect on the principles of knowledge representation inherent in PARKA's design, and we discuss difficulties that we have encountered in our continuing use of PARKA. In Section 4 we describe enhancements to the PARKA user interface, including a new pattern-based retrieval mechanism. Section 5 indicates the directions in which work on PARKA is currently moving. Full details of PARKA's representational conventions, and a synopsis of PARKA's syntax, can be found in [11].

PARKA's speed derives from its ability to perform certain types of inferences using parallel message-passing techniques [3]. The price for the improved performance is that most structural information must be represented explicitly. This requirement precludes uses of indirection, implicit knowledge generated at query time, and procedural attachment found in other systems. We achieve similar effects, however, through the use of *topology constraints* enforced at knowledge base update time. A PARKA topology constraint interacts with the system's update procedures in order to make some given relationship explicit in the structure of the knowledge base. The existence of a constraint can cause the system to perform integrity checks (and to reject an update if a check is not satisfied) and to add or delete links, create new frames, etc. As a result, many of our knowledge base update procedures are relatively expensive. This is a trade-off we are willing to make for the sake of high efficiency queries, particularly since we envision the system's applications as being query-intensive and update-rare.

The primitive representations in a PARKA knowledge base are *frames*, each of which consists of a set of named, typed pointers called *slots*. The target of a slot's pointer is referred to as that slot's *value,* and the value is said to *fill* the slot. In the original version of PARKA a slot could be filled with any single object (symbol, string, frame, etc.) but we now require slot fillers (except for the *name* slot) to be frames. This restriction discourages the use of "unstructured" components of representations and aids in the construction of knowledge base analysis tools.

Each frame has a distinguished *name* slot which must be filled with a unique symbol. A frame represents a description of the referent of the term (the symbol) that fills its name slot. Frames come in two varieties: *categories* and *individuals*. Category frames describe the referents of category terms such as **dog**, **color**, or **philosopher**. Individual frames describe the referents of names used to refer to specific individuals in the world, such as **Fido**, **cyan**, or **Aristotle**. Although a slot can only have one filler, an implicit universal quantifier is assumed by convention when the filler is a category. That is, we think of the slot as filled by *all* members of the provided category (unless the slot is of type RESTRICTION – see below).

We view the filling of a category's slot as the addition of a property attribution to a description of the referent(s) of a term. Categories are connected to one another by *ISA links* which encode category membership (individual $\rightarrow$ category) and subcategory (category $\rightarrow$ category) relations. The set of ISA links in any PARKA knowledge base must form a directed acyclic graph (i.e., multiple parents are permissible but ISA cycles are not), and slot values are *inherited* through the ISA hierarchy using Touretzky's "inferential distance ordering" [12].

It should be noted that our notion of a *category* is different from that of a *set*, a *class*, or a KL-ONE *Generic Concept* [2]. The slot values of a category and its position in the ISA hierarchy can be thought of as describing a "typical" member of the given category, but since most slot values can be overridden the descriptions are not normally necessary conditions for category membership. In addition, category descriptions are not assumed to be complete, so they are also not *sufficient* conditions for category membership. Necessary conditions on category membership *can* be expressed, though they are the exception rather than the rule. If a slot is created with the type DEFINITIONAL then its filler cannot be overridden; it is a necessary attribute of all subcategories and individual members of the category in question. The RESTRICTION slot type imposes a weaker constraint on the fillers of the given slot for descendants of the given category: they may be either proper subcategories or members of the restriction category[2]. Note that the implicit quantifier for category-valued slots is *existential* for restrictions, though it is universal in all other cases; the restriction mandates that values of the slot for subcategories/members be filled by *some* subcategory/member of the provided value, and *not* that it must be filled by *all* of the members (in which case a category-valued DEFINITIONAL slot would be appropriate).

Additional mechanisms provide support for sufficient conditions, enabling automatic categorization in some cases. We have developed two such mechanisms to date: *set-constructor* categories and *aggregations*. Set-constructor categories (union, intersection and set-difference) are defined as set-theoretic combinations of their specified constituent categories, and are maintained

[2]If the restricted slot in the descendant is also of type RESTRICTION then the filler need not be a *proper* subcategory — in this case it is also permissable for the filler to be the restriction category itself.

across updates. For example, if **toy-elephant** is defined as the intersection of **toy** and **elephant,** then **Clyde,** who was specified to be both a **toy** and an **elephant,** will be an ISA descendant of **toy-elephant.** However, when we discover that **Clyde** is really a poorly formed toy Aardvark we can break the ISA link to **elephant** and the link to **toy-elephant** will be broken automatically. Aggregations provide a similar kind of ISA link maintenance in the service of part-whole relations. Part-whole relations and ISA relations have transitivity properties (see [14]) that can be captured easily with aggregations. For example, we can ensure that **idle-adjust-screw,** which is part of a **carburetor,** inherits attributes from **vehicle-part** even though no direct ISA connection was specified by the knowledge engineer (see [11] for a detailed look at this example).

2. Experiments: Knowledge Base Construction

PARKA's representational mechanisms were designed, like those of most other knowledge representation systems, to strike a balance between considerations of efficiency and considerations of the *perceived* need for expressive power. PARKA's massive parallelism changed the character of these decisions to some extent, but the question of how a system meets *actual* representational needs is as relevant for PARKA as it is for any other representation system. This question can only be answered by empirical investigation – that is, by constructing knowledge bases and by using them in applications.

For this reason we have encoded a corpus of common sense knowledge in the PARKA system, and we have begun to use the resulting knowledge bases for case-based planning. Our ideas about the virtues of various PARKA mechanisms, and about PARKA's "explicit" style of representation were impacted by these experiments, and we discuss these issues in a later section.

The construction of common-sense knowledge bases serves another important purpose within the PARKA project. The parallel implementation has been developed and tested using large (up to 256k frames) randomly generated knowledge bases. The knowledge base generator is parameterized to allow for the generation of various topologies – this feature has been used extensively to test the performance of the parallel implementation [8]. Given a reasonably large non-random knowledge base, developed by a group of programmers to encode a corpus of real-world knowledge, we can extract characteristics (depth, topology, etc.) which may be hypothesized as occurring in other real-world PARKA systems. This will allow us to generate more "realistic" large random knowledge bases for further testing.

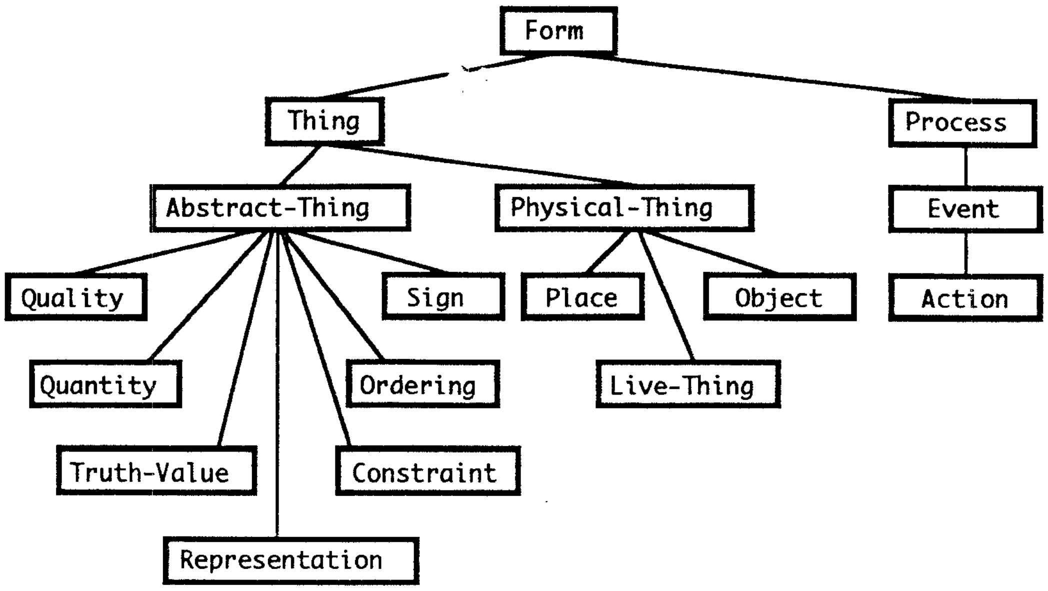

Figure 1. The top of the ISA hierarchy. The full "top of net" skeletal network has 160 frames; we show here only the top-most 17.

2.1. Biological/Geographical Knowledge Base

We began by encoding a skeletal network of high-level categories (such as quantity, quality, physical-thing, process, etc. – see Figure 1) and then filled in detail to various depths. The ubiquity of animal-related examples in the knowledge representation literature led us to the representation of animal-related knowledge as our first extension to the high-level ontology. This led to the representation of habitats, climates, and geographic knowledge in general. A typical frame and a small segment of the ISA hierarchy is shown in Figure 2. We restricted ourselves to North America, but represented a wide range of knowledge within this domain. The knowledge base contains information about countries, states, borders, bodies of water, climates, physical characteristics of animals, food requirements, habitats, etc. In the following subsections we summarize various characteristics of the knowledge base.

Size

The knowledge base currently contains 1690 frames of which 1150 are categories (68%) and 540 are individuals (32%). 362 of the categories (31.5%) are aggregations. Set-constructor categories are used sparsely, as unanticipated interactions between set-constructors made their use somewhat confusing (see Section 3).

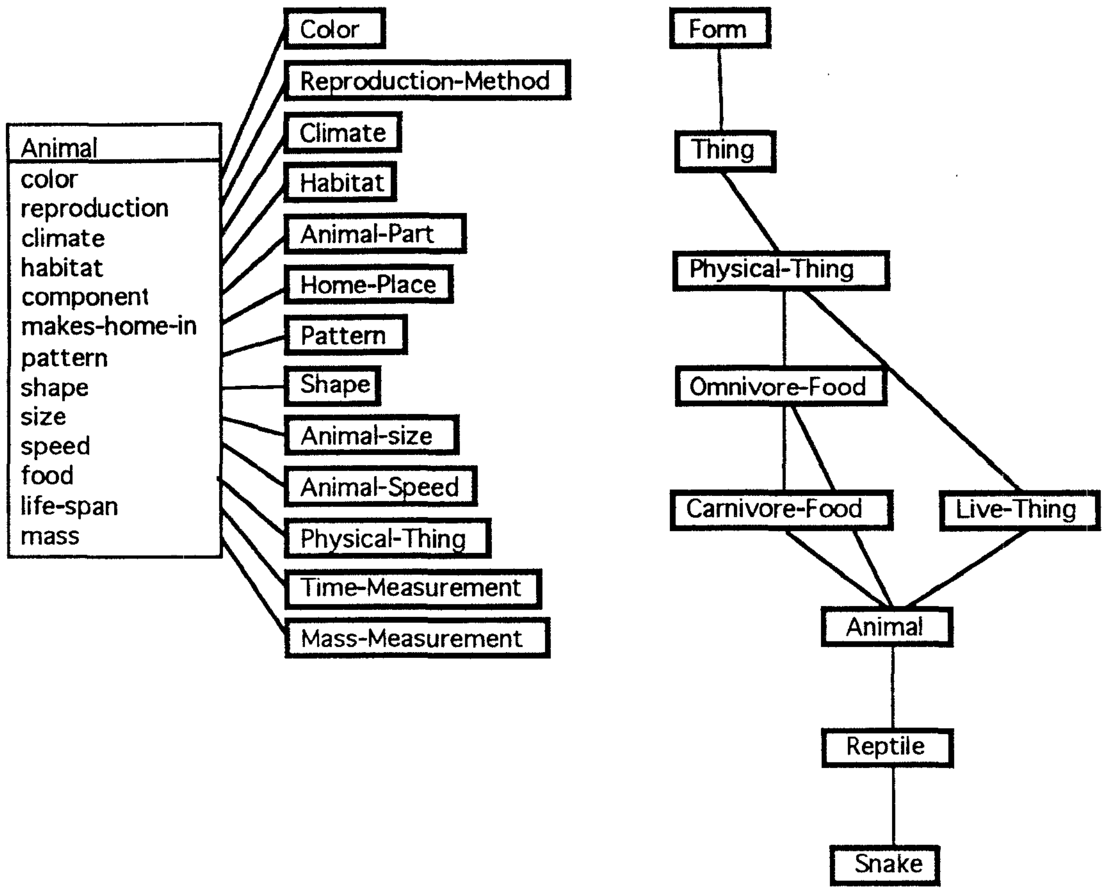

Figure 2. Segments of the Animals knowledge base.
All of the slots shown for the [animal] frame are restrictions.

Branching

There are 2819 ISA links, with a maximum fan-out (to children) of 93 and a maximum fan-in (from parents) of 82. This 82 ISA-parent maximum will look surprising to those familiar with other knowledge representation schemes. It arises from the integration of the part-whole and ISA hierarchies in order to properly support part/ISA transitivities. Where others might fill a *part* slot with a list of parts, we fill it with a single aggregation that has the parts as descendants. This means that things that are parts of lots of things end up being children of lots of these aggregation categories. (In a different system they'd just be values of lots of slots.) In the current system, for example, an individual like *Mississippi-River* may show up as a part of a the border of a number of US States and regions, and thus would have this large number of ISA parents.

The branching factor for categories is 2.45, but there is quite a bit of variability in branching across the knowledge base:

	Mean	Median	Std. Dev.
Parents/Category	1.40	1.00	3.45
Children/Category	2.45	1.00	5.70
Parents/Individual	2.24	1.00	2.77

Although cycles in the ISA network are prohibited, many cycles of various lengths exist in the slot-value network.

Depth

The maximum depth (distance from the root) in the entire knowledge base is 10, and the minimum depth for any individual is 4. For individuals the average depth is 6.17, with a median of 6.00 and a standard deviation of 1.00. It is significant that these depth figures are so low and so uniform, since most of the parallel PARKA algorithms have run-times proportional to the depth (and not the overall size) of the knowledge base.

Slot Profiles

The following tables profile the slots of the average category and the average individual in our knowledge base. We show how many slots the average frames have, broken down by slot type and by filler type. Note that each of these values must be an integer for any *particular* frame; the statement that the average individual has 0.24 DEFINITIONAL slots means that there's about 1 definitional slot for every 4 individuals.

Slot Profile of the Average Category

	Mean	Median	Std. Dev.
Total Number of Slots	2.33	1.00	3.96
EXPLICIT (not inherited)	0.43	0.00	1.11
SIMPLE (defeasible)	0.43	0.00	1.11
DEFINITIONAL (necessary)	0.01	0.00	0.10
RESTRICTION	1.80	1.00	2.48
Filled w/INDIVIDUALS	0.12	0.00	0.38
Filled w/CATEGORIES	2.37	1.00	3.97
Filled w/AGGREGATIONS	0.09	0.00	0.28
Filled w/SET-CONSTRUCTORS*	0.02	0.00	0.14

Slot Profile of the Average Individual

	Mean	Median	Std. Dev.
Total Number of Slots	4.33	6.00	2.45
EXPLICIT (not inherited)	3.06	5.00	2.24
SIMPLE (defeasible)	3.06	5.00	2.24
DEFINITIONAL (necessary)	0.24	0.00	0.81
RESTRICTION	1.29	1.00	0.78
Filled w/INDIVIDUALS	2.37	3.00	1.60
Filled w/CATEGORIES	2.44	2.00	1.76
Filled w/AGGREGATIONS	0.70	1.00	0.58
Filled w/SET-CONSTRUCTORS*	0.00	0.00	0.00

* set-constructors were used sparsely – see text

2.2. Case-Based Planning

Thorough assessment of a knowledge representation system requires that the system be *used* in applications, and that the application programmers provide feedback to the designers of the representation system regarding the system's strengths and weaknesses. To this end we have been using PARKA as the representational basis for a project in case-based planning. The details of the planning project are not germane to the present paper, but the use of PARKA within the project bears on our discussion of PARKA's utility.

Case-based planning knowledge presents several challenges for a representation system. Our system includes case-retrieval mechanisms, traditional nonlinear planning mechanisms, and mechanisms for execution simulation; hence the representations must be general enough to support a wide variety of algorithms. In addition, both the case-base and the representation of the world must be *dynamic*, reflecting changes in the world and in the history of the agent. Many planning algorithms also require the representation of *hypothetical* world states. Hence it is imperative that the representation system be capable of representing multiple world states without confusion. Meta-knowledge (knowledge about the planning process itself) must also be representable. To date PARKA has provided the representational support that has been required in all of these areas. The category/individual distinction has turned out to be particularly useful in distinguishing hierarchical specifications of potential actions (represented as categories) from the episodic memories of actions performed in the past (represented as individuals). Figure 3 shows an excerpt from the case-based planning knowledge base in a cooking domain.

We currently represent goals, plans, actions, world objects, episodic knowledge (planning experiences in the form of cases), failures, and repairs. Episodes include representations of goals, the plans that were generated, any failures that were encountered, and any repairs that were performed. Episodes are represented as categories with links to context frames (e.g., place, time, etc.) that can guide retrieval. In a ldition, the components of an episode form a richly interconnected structure. For example, plans are associated with the goals that they were intended to solve and the failures that they engendered, and failures are linked to possible explanations and to repairs that were attempted. This structure allows for a variety of pattern-based retrieval strategies, in contrast to traditional case-based systems which typically retrieve cases only through properties designated as indices when the case is stored. Pattern-based retrieval mechanisms can be extended to incorporate pattern-relaxation and a variety of partial match criteria, and hence our pattern-based approach is quite flexible. The plan representations are hierarchical, and subplans inherit context from their parents. This allows subplans to be retrieved and adapted independently from their parents.

An unexpected characteristic of our planning representations was their size – a typical case-memory in our initial system uses on the order of 50 frames, while cases described in the literature appear to be significantly smaller. In retrospect this seems quite logical, and it seems reasonable to expect that future work in knowledge-rich case-based planning will require progressively larger structures. This provides a further argument for the use of PARKA for work in case-based planning – the need for rapid matching and manipulation of large structures was one of the key motivations for the design of PARKA and its Connection Machine implementation.

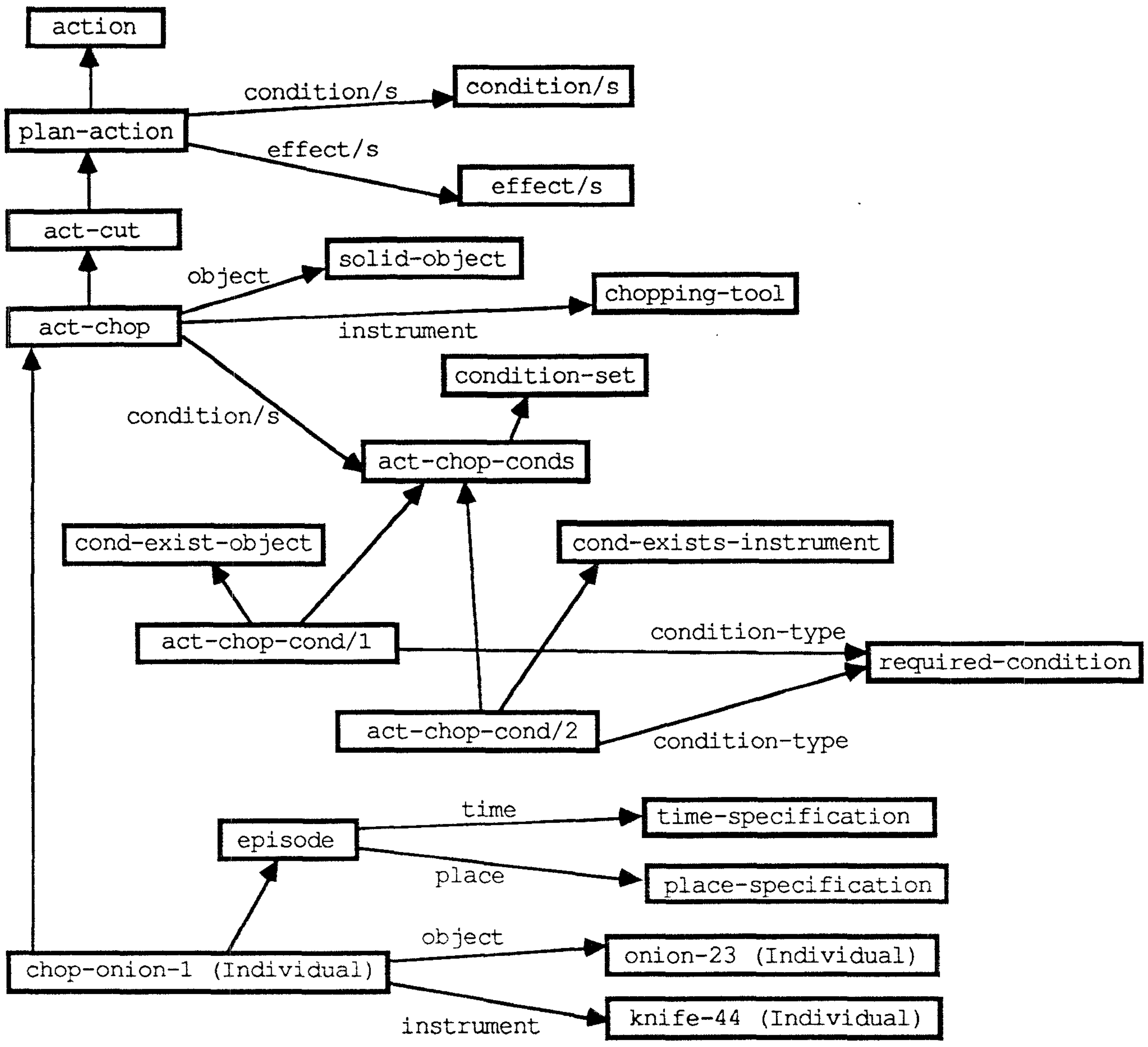

Figure 3. Part of the representation of an onion-chopping action. Arrows without labels represent ISA links. Slot types are not shown, and many slots and links have been omitted for readability.

3. Analysis: Knowledge Representation Issues

A principal requirement of PARKA's massively parallel activation wave propagation algorithms is that many of the relevant relations between concepts must be present in the network *explicitly*, i.e. as pointers or as chains of pointers, at query time. As mentioned in the Introduction, this requirement forced us to move large components of our inferential mechanisms from the query processing algorithms into the update algorithms; at query time the important relations are represented explicitly as pointer chains, and hence our fast wave propagation algorithms can be used.

We found that the move to an emphasis on update-time computation had beneficial effects which were independent of SIMD parallelizability. In particular, we found that significant efficiency improvements resulted from computing certain relations at update time and storing the results (in the form of explicit pointers), even when serial query algorithms were used. In the absence of such pre-computing and caching of relations, some relations may be recomputed several times over a set of queries. Worse, in the absence of such mechanisms it is possible for a single query to trigger a cascade of recomputations. So long as the relations in question are reasonably well bounded (admittedly this may not always be the case) the principle of early computation and caching makes good sense. The semanticist's goal then becomes one of devising a set of relations which provides adequate semantic coverage while meeting a restriction of "non-excessive proliferation."

PARKA's current semantics, as described in [11], are the result of our first pass at achieving these goals. At its lowest level of abstraction a PARKA representation consists of category and individual descriptions connected to one another by explicit subsumption (ISA) relations and by three types of property attribution (defeasible, definitional and restrictive). Additional layers of descriptive capability are added by attaching "topology constraints" to segments of the representation. A PARKA topology constraint interacts with the system's knowledge base update procedures in order to make some given relationship explicit and to maintain the validity and explicitness of the relationship across future updates. The existence of such a constraint can cause the system to perform integrity checks and to perform arbitrary modifications of the knowledge-base structure at update time. As described above, the current set of topology constraints includes facilities for maintaining descriptions of set-theoretic entities (composed of unions, intersections, and set-differences) and an "aggregation" mechanism which can be used to ensure that the proper transitivity relations hold in part/whole hierarchies.

The power of such "highly explicit" representations is not well understood. We know that we can implement such systems, and that the performance of such systems (especially on parallel hardware) is impressive. Our success in representing a corpus of common sense knowledge in PARKA lends credence to the idea that this style of representation will be useful for a wide range of AI applications. However, our knowledge base construction experiments also highlighted some of the difficulties with this style of representation (more on this below).

In designing "highly explicit" representation systems one must decide *which* representational mechanisms to precompute and to store explicitly. If *all* representations are cached in this manner then it is likely that the storage space requirements and the run-times for *all* updates would become prohibitively large. It is not unusual for a classification-based representation system to "flatten" the ISA hierarchy – that is, to precompute property inheritance much as we precompute set-theoretic relations. In the case of PARKA we chose to leave property inheritance implicit, and to make all *other* structural relations explicit. This is because we knew that our parallel implementation would deliver prop·rty inheritance almost "for free." When ·nheritance is implicit, property updates have a minimal impact on the structure and on the size of the knowledge base. On the other hand, the set of topology constraints supported by PARKA is open-ended, and there may not be an obvious way to enforce a given constraint at query-time with the SIMD query algorithms. It therefore makes sense to "compile" all such constraints into the explicit structure of the knowledge base at update-time.

The major difficulty encountered by the PARKA knowledge base constructors was unanticipated interactions between the effects of sets of topology constraints. These interactions were of two kinds: procedural artifacts and genuine interactions. We describe each of these in turn.

1) Procedural Artifacts: Some sets of constraints that *can* be mutually satisfied may nonetheless cause problems for naive implementations of the constraint validating functions. In particular, our initial algorithms for maintaining set-constructors would sometimes get caught in cycles when working on knowledge-base segments with many, closely connected set constructors. Procedural artifacts present no theoretical problems, but finding and removing them can sometimes be difficult. For example, consider the following parka definitions, diagrammed in Figure 4:

```
(make-category 'animal)
(make-category 'snake)
(isa [snake] [animal])
(make-category 'rattle-snake)
(isa [rattle-snake] [snake])
(make-category 'constrictor)
(isa [constrictor] [snake])
(make-category 'poisonous-thing)
(make-intersection 'poisonous-animal [poisonous-thing] [animal])
(make-intersection 'poisonous-snake [poisonous-thing] [snake])
(make-union 'deadly-snake [poisonous-snake] [constrictor])
```

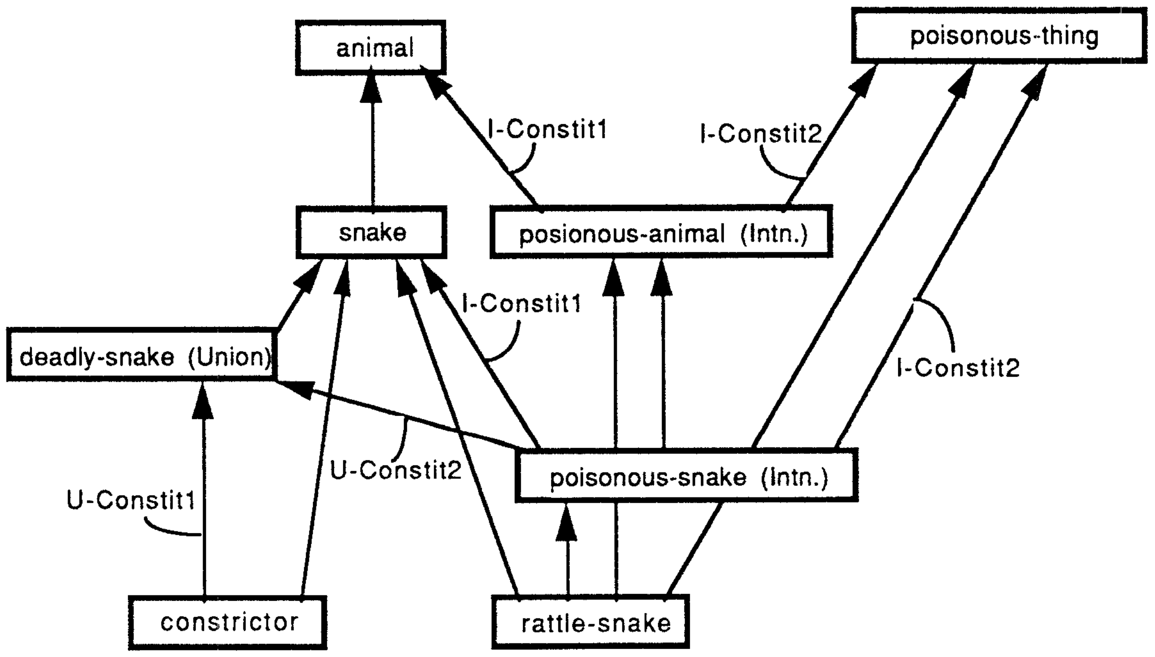

Figure 4. A problematic case for a naive intersection-maintenance algorithm.

Our initial set-constructor maintenance algorithm caused an ISA link to be added from the intersection [poisonous-snake] to itself, resulting in a control stack overflow in Lisp during subsequent processing of that intersection. The following is a simplified version of the original algorithm:

```
Algorithm: Intersection-Constraint (int)
    1. for each constituent $c_i$, in int do
            delete the ISA link from int to $c_i$
    2. Let I = descendants($c_1$) $\cap$ descendants($c_2$) $\cap$ ... $\cap$ descendants($c_n$)
    3. Let U = { i $\in$ I : i is an upper bound on <I, ISA> }
    4. for each frame $u_i \in$ U do
            add an ISA link from $u_i$ to int
    5. for each constituent $c_i$, of int do
            replace the ISA link from int to $c_i$
```

The intention is to calculate the set-theoretic intersection of the descendants of each constituent (I) of the intersection, and then to direct ISA links from the elements in I to the intersection. However, the algorithm does not take into account the cases in which there is another ISA path from the intersection to a constituent. Such a path may have been created by the constraints of another set-constructor. In the present case such a path exists: from [poisonous-snake] to [deadly-snake] to [snake]. The value of I at step 2, then, is {[poisonous-snake], [rattle-snake]}. Since [rattle-snake] ISA [poisonous-snake], the set U in step 3 is {[poisonous–snake]}. Finally, in step

4, the algorithm creates an ISA link from [poisonous-snake] to itself; this causes subsequent constraint checks to enter an infinite loop.

2) Genuine Interactions: Some sets of constraints are simply inconsistent and cannot be mutually satisfied. Although the topology constraints were designed to minimize the occurrence of such interactions, they cannot be ruled out. Genuine interactions are handled poorly in the current version of PARKA. At a minimum, the system designers should attempt to characterize the situations in which such interactions occur, so that knowledge base programmers will not be baffled by them. In addition, such interactions should be detected and reported when possible; the current system will sometimes cycle indefinitely in a futile attempt to satisfy all constraints. For example consider the following code, diagrammed in Figure 5:

```
(make-category 'a)
(make-category 'b)
(make-set-difference 'diff [a] [b])
(make-intersection 'int [b] [diff])
```

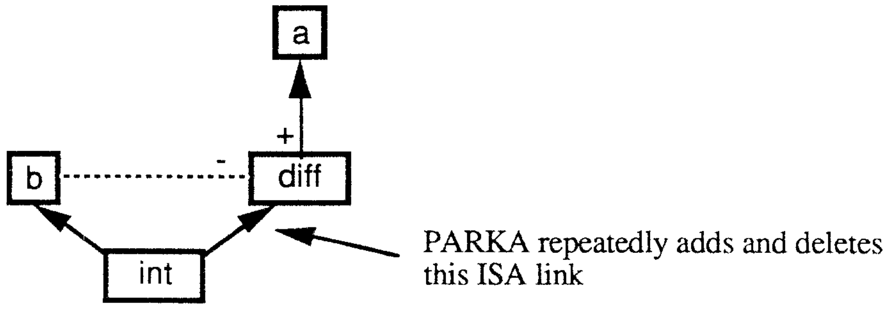

Figure 5. An inherently inconsistent PARKA specification.

This configuration of set-constructors currently causes PARKA to go into an infinite loop trying to satisfy the constraints on the set-difference [diff] and on the intersection [int]. Note that no constraint procedure can satisfy this configuration of constraints; the constraint on [int] requires that there be a link from [int] to [diff] (an intersection is always a subset of its constituents) and the constraint on [diff] requires that there *not* be a link from [int] to [diff] (since [int] is a descendant of [b] and [diff]=[a]–[b] in the set-theoretic sense). One might claim that there is no inconsistency here – that [int] is simply the empty set ([diff]=[a]–[b], [int]=[b]$\cap$[diff], [int]=[b]$\cap$[a]–[b]=$\varnothing$). However, due to the manner in which category-theory and set-theory are (intentionally) conflated in PARKA, the problem is a real one, and the two constraints will continue their tug-of-war indefinitely. This sort of problem is not limited to cases in which two set constructors are immediate neighbors in the ISA hierarchy; Figure 6 shows a more complex example. The problem also points to some murkiness in the semantics of set constructors in general (i.e., if [int] were a regular category rather than an intersection, would we allow the constraint on [diff] to break the link from [int] to [c]?). Whether the best solution will involve a revision of the semantics (e.g., the inclusion of cancellation links) or a smarter update-checking algorithm (which would refuse to create such configurations) will be decided by future research.

```
(make-category 'a)
(make-category 'b)
(make-set-difference 'diff [a] [b])
(make-category 'c)
(isa [c] [a])
(make-intersection 'int [b] [c])
```

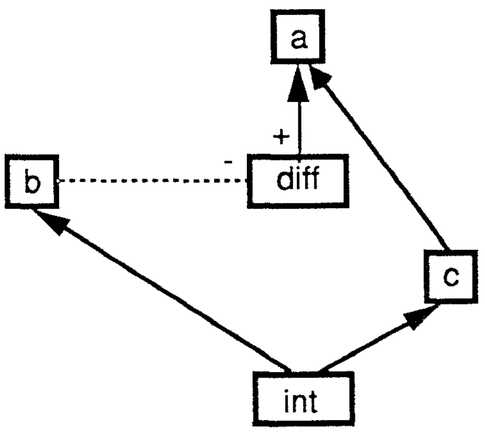

Figure 6. Another inconsistency: [c] should be an ISA descendant of [diff], but [int] should not.

4. Enhancements: Interacting with PARKA

Although the massively parallel implementation is PARKA's *raison d'être,* it is more practical to develop knowledge bases and to experiment with knowledge representation techniques with a serial version of the system running on less expensive hardware. Hence a version of the original PARKA language was implemented in standard, serial Common Lisp, which could be run on a wide range of platforms.[3]

[3]Observant Lisp-literate readers will have noticed that PARKA's use of "slot-value" is problematic in a Lisp system with CLOS. The current implementation runs only in pre-CLOS systems, and it is likely that we will change a few of the documented function names during our translation to CLOS.

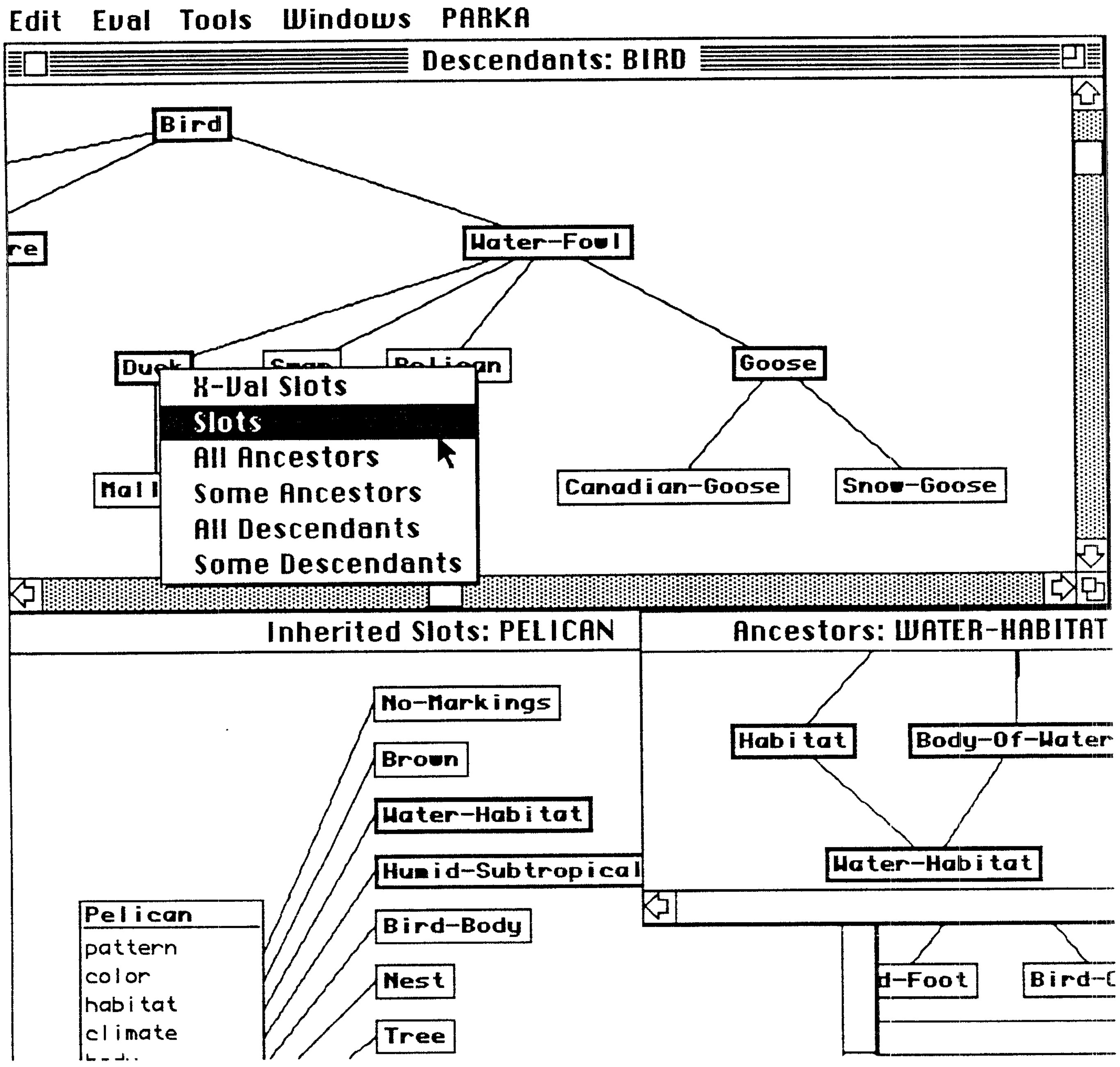

Figure 7. A snapshot of the new PARKA user interface.

This serial implementation, while sufficient for experiments with small knowledge bases, was not an appropriate instrument for the creation of large knowledge bases for several reasons. Since the original serial version was developed primarily to prototype the parallel system, little attention was paid to efficiency considerations in the serial algorithms. Predictably, these algorithms did not scale up well as the knowledge bases grew, and it was necessary to rewrite them. In addition, the original version of the system was equipped with a programmer's interface that was well defined but rather sparse. Since large knowledge bases are often created incrementally and by teams of programmers, we added a number of development tools, including a graphical browser, a menu interface, a knowledge base dumper, an apropos facility, a set of knowledge base analysis tools, and a pattern-based knowledge-structure retriever. The retriever is of particular interest because it was designed to have an efficient SIMD implementation — for this reason we intend to incorporate it into PARKA proper, and not just into the serial development toolkit. In the remainder of this section we describe the new retrieval facility; the serial system's recent efficiency improvements are summarized in an Appendix.

The new PARKA retriever was designed to allow a wide range of queries without making presumptions about how particular programmers would choose to use PARKA's various representational mechanisms. It was also designed to have reasonably efficient algorithms in *both* the serial and parallel versions of the system.

Retrieval probes are specified as graphs, with nodes consisting of frame variables and edges consisting of relations that must hold between frames bound to the variables. Given a probe graph, the retrieval algorithm searches for instances of the graph in the PARKA knowledge base, resulting in a set of variable bindings that satisfy the probe. The retrieval algorithm performs a simultaneous search of the probe graph and the PARKA knowledge base, accumulating bindings as the search proceeds. As an example, consider the PARKA knowledge base fragment in Figure 8, in which s is some particular slot. Suppose we wish to retrieve cases in which a frame overrides a default for the slot s. This probe pattern is illustrated in Figure 9.

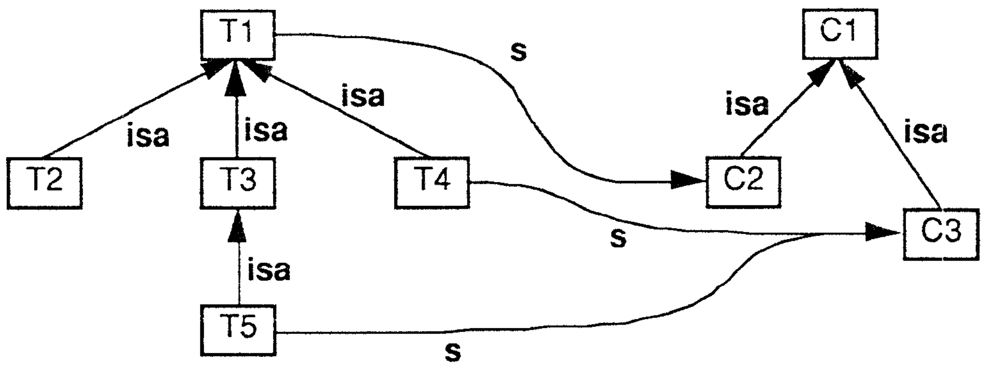

Figure 8. A knowledge base fragment.

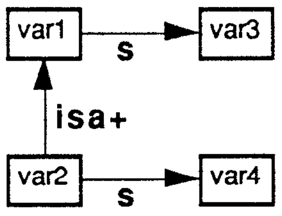

Figure 9. A retrieval probe.

The probe specifies that **var2** must be connected to **var1** by one or more ISA links, that **var1** must be connected to **var3** by slot s, and that **var2** must likewise be connected to **var4** by slot s. There are three sets of bindings that match the probe, and they are as follows (the actual retriever output is a list of Lisp lists):
{var1 = T1, var2 = T5, var3 = C2, var4 = C3}
{var1 = T1, var2 = T4, var3 = C2, var4 = C3}
{var1 = T3, var2 = T4, var3 = C2, var4 = C3}

The actual syntax for the probe in Figure 9 is as follows:

```
(parka-retrieve
   :frames      (var1)
                (var2)
                (var3)
                (var4)
   :relations   (var1 ancestor-of var2)
                (var1 p var3)
                (var2 p var4)
                (var3 neq var4))
```

The full syntax for retrieval probes (in extended BNF) is:

```
          <probe>   ::= (parka-retrieve <frames> <relations>)
         <frames>   ::= :frames (<var> [<var-specifier> <frame>])*
  <var-specifier>   ::= parent-of | child-of |
                        ancestor-of | descendant-of | =
      <relations>   ::= :relations (<var> <rel-specifier> <var>)+
  <rel-specifier>   ::= parent-of | child-of |
                        ancestor-of | descendant-of |
                        neq | <slot-name>
```

Restrictions on the bindings of frame variables may be included in the `:frames` clauses; this provides context for the probe and is usually more efficient than expressing the same restrictions in `:relations` clauses. For example, if we were only interested in descendants of **T1** that override the value of slot s provided by *other* descendants of **T1**, where both s values are descendants of **C1**, then we could express our probe as follows:

```
(parka-retrieve
   :frames     (var1 descendant-of [t1])
               (var2 descendant-of [t1])
               (var3 descendant-of [c1])
               (var4 descendant-of [c1])
   :relations  (var1 ancestor-of var2)
               (var1 p var3)
               (var2 p var4)
               (var3 neq var4))
```

5. Future Directions

The work outlined in this paper points to several directions for future research. These are all under currently under investigation by various members or our research group.

• Larger Knowledge Bases. We are continuing our investigation of the efficacy of PARKA for representing large amounts of "common-sense" knowledge. As well as a continued extension of the knowledge base described above, we have been negotiating with MCC for access to the knowledge ontology developed for CYC [10]. We hope to translate a large subset of this knowledge base into PARKA, and to use this for testing both the speed of the parallel algorithms and the power of the representation scheme.

• Applications. The development of knowledge bases is not sufficient to test the mettle of a knowledge representation system; it must be used in a reasonably large, real-world application. Our current work in this area involves applications in case-based planning.

• Analysis. The constraint mechanisms and their interactions must be studied further in order to provide programmers with better intuitions about the capabilities of the system. We are currently looking into the literature of constraint-based programming.

• Comparisons. We are examining the descriptive constructs of terminological languages in the tradition of KL-ONE (see [16]) in order to determine which of these can be represented in the highly explicit manner required by PARKA. This will allow us to compare PARKA's expressive power, and the expressive power of highly explicit representation systems in general, with that of several popular knowledge representation languages.

• Foundations. We believe that the notion of "highly explicit" representation bears comparison with the more traditional notion of *extensional* (as opposed to *intensional*) representation. The conventional wisdom in AI (at least since [15]) has been that intensions cannot be eliminated, but recent work by Wilensky [13] raises questions about the validity of this assumption. To the extent that an explicit/extensional analogy can be drawn, this work may serve to further establish the utility of highly explicit representations as required by PARKA.

6. Conclusions

The PARKA system has been enhanced with new programming tools and a sophisticated graphical user interface; this enabled a team of programmers to encode a realistic corpus of common-sense knowledge in the PARKA formalism. This effort has provided us with data about the characteristics of "realistic" knowledge bases which will be useful in further testing our parallel implementation. In addition, the knowledge base development effort provided valuable feedback on the utility of PARKA's various representation mechanisms. Several problems remain, and directions for further research have been indicated.

Our experience with PARKA shows that implementational concerns can play an important role in the development of knowledge representation theory. In particular, the constraints of our massively parallel implementation have lead us to a notion of explicit structure which may prove useful to theoreticians as well as to language implementors.

Appendix. Performance Enhancements to Serial PARKA

Part of the philosophy behind PARKA is the insistence that queries should be extremely fast relative to updates. This feature was driven by the commitment to explicit representation in PARKA as well as by computational constraints imposed by the Connection Machine's SIMD architecture [11]. While any serial implementation will, in general, be far slower than the parallel CM implementation, we have taken care to maintain the same principle of fast queries at the expense of update speed in the serial PARKA. There are three types of queries, in general, which can be applied to a PARKA knowledge base:

- ISA relation queries, e.g. "is Tweety a bird?"
- Bottom-up property inheritance queries, e.g. "what is Tweety's color?"
- Top-down property inheritance queries, e.g. "which things are yellow?"

The first type are handled by a depth-first search of upward ISA links. The choice of upward links is a pragmatic one, based on the fact that upward branching factors tend to be quite small compared to downward branching factors, making these queries basically logarithmic in the total number of nodes. For most knowledge bases, these queries can be performed almost as fast in serial PARKA as in parallel PARKA.

In order to compute property inheritance efficiently, we make extensive use of indexing. Our indexing algorithms are applied at update time to the portions of the knowledge base affected by a given update. The index structure that we found to be most useful is an *explicit-value ordering table*. This table contains, for each slot in the system, the partial ordering (by ISA links) of the frames in the system which are explicitly valued for that slot. The information in this table has many uses, but its primary use is for the rapid computation of top-down inheritance queries, subject to Touretzky's inferential distance ordering [12]. The use of this algorithm resulted in a speedup of approximately 40 fold over the original serial PARKA algorithm.

Other performance improvements increased the speed of PARKA's global topology checks by about 10 times over the previous serial version.

References

[1] Brachman, Ronald J., Fikes, Richard E. and Levesque, Hector J., "KRYPTON: a Functional Approach to Knowledge Representation," in Brachman, Ronald J. and Levesque, Hector J., Editors, *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Inc., May, 1983.

[2] Brachman, Ronald J., and Schmolze, James G., "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science* Volume 9, 1985, pp. 171-216.

[3] Evett, M.P., Spector, L. and Hendler, J.A., "Knowledge Representation on the Connection Machine," *Proceedings of Supercomputing '89*, (Reno, NV; Nov. 13-17, 1989), ACM, New York, NY, 1989.

[4] Evett, M. and Hendler, J., "Massively parallel SIMD algorithms for property inheritance," in *Proc. AAAI Workshop on Parallel Algorithms for Machine Intelligence, IJCAI-89*, August 1989.

[5] Evett, M.P. and Hendler, J.A., "PARKA: Frame-Based Property Inheritance on the Connection Machine," in *Parallel Computing '89; Proceedings of the Conference*, (Leiden, The Netherlands, Aug. 29-Sept. 1, 1989), Elsevier, Amsterdam, The Netherlands, 1989.

[6] Evett, Matthew, Hendler, James, and Spector, Lee, "PARKA: Parallel Knowledge Representation on the Connection Machine," University of Maryland Technical Report UMIACS-TR-90-22, CS-TR-2409, February 1990.

[7] Evett, M.P., Spector, L. and Hendler, J.A. "Property Inheritance in PARKA", to appear in *The Journal of Parallel and Distributed Processing*, (accepted, October, 1990).

[8] Evett, M. and Hendler, J., "Achieving Computationally Effective Knowledge Representation via Massively Parallel Lisp Implementation," in *Proc. High Performance and Parallel Computing in Lisp*, (Twickenham, London, UK; Nov. 12-13, 1990), EUROPAL, Dorking, UK, 1990.

[9] Hillis, W. D., *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[10] Lenat, Douglas B. and Guha, R. V., *Building Large Knowledge-Based Systems*, Addison-Wesley Publishing Company, Inc., 1990.

[11] Spector, Lee, Hendler, James A., and Evett, Matthew P., "Knowledge Representation in PARKA," University of Maryland Technical Report UMIACS-TR-90-23, CS-TR-2410, February 1990.

[12] Touretzky, D.S., The Mathematics of Inheritance Systems, Morgan Kaufmann Publishers, Inc., 1986.

[13] Wilensky, Robert, "Sentences, Situations and Propositions," Report No. UCB/CSD 90/585, Computer Science Division, University of California, Berkeley, August 1990.

[14] Winston, Morton E., Chaffin, Roger, and Herrmann, Douglas, "A Taxonomy of Part-Whole Relations," *Cognitive Science* Volume 11, 1987, pp. 417-444.

[15] Woods, William A., "What's in a link: Foundations for Semantic Networks," in Bobrow, D. G. and Collins, A. M., Editors, Representation and Understanding: Studies in Cognitive Science, Academic Press, New York, pp. 35–82, 1975.

[16] Woods, William A. and Schmolze, James G., "The KL-ONE Family," TR-20-90, Harvard University, Center for Research in Computing Technology, August 1990.