Knowledge and Information Systems (2002) 4: 305–328
Ownership and Copyright
Springer-Verlag London Ltd. © 2002

Knowledge and Information Systems

Efficient Transitive Closure Reasoning in a Combined Class/Part/Containment Hierarchy

Yugyung Lee[1] and James Geller[2]

[1]Computer Science Telecommunications, University of Missouri at Kansas City, MO, USA
[2]Department of Computer and Information Sciences, New Jersey Institute of Technology, Newark, NJ, USA

Abstract. Class hierarchies form the backbone of many implemented knowledge representation and reasoning systems. They are used for inheritance, classification and transitive closure reasoning. Part hierarchies are also important in artificial intelligence. Other hierarchies, e.g. containment hierarchies, have received less attention in artificial intelligence. This paper presents an architecture and an implementation of a hierarchy reasoner that integrates a class hierarchy, a part hierarchy, and a containment hierarchy into one structure. In order to make an implemented reasoner useful, it needs to operate at least at speeds comparable to human reasoning. As real-world hierarchies are always large, special techniques need to be used to achieve this. We have developed a set of parallel algorithms and a data representation called *maximally reduced tree cover* for that purpose. The maximally reduced tree cover is an improvement of a *materialized transitive closure* representation which has appeared in the literature. Our experiments with a medical vocabulary show that transitive closure reasoning for combined class/part/containment hierarchies in near constant time is possible for a fixed hardware configuration.

Keywords: IS-A hierarchies; Materialized transitive closure; Parallel artificial intelligence; Parallel reasoning; Part hierarchies; Transitive closure reasoning

1. Introduction

One original goal of artificial intelligence (AI) has been to provide computers with human-like capabilities in a number of areas such as natural language understanding, reasoning, learning, natural language generation, and image understanding. The Turing test (Turing, 1950) has been considered as a measure of our success in areas such as reasoning and natural language processing. As AI developed, it became clear that this goal was, and still is, elusive, and to date no

Received 10 January 2000
Revised 25 November 2000
Accepted 9 February 2001

general version of the Turing test has been passed by any program. It has also become clear that the Turing test itself is not sufficient to test many human abilities that seem rooted in factors such as perceptual processing. For that purpose, some researchers have argued that the Turing test should be replaced by a more general total Turing test (Harnad, 1989).

Even if we limit ourselves to a subfield of AI, namely reasoning, it is necessary to relax some requirements that are implied by a comparison of AI reasoning with human reasoning, in order to do successful research. Let us first make a list of which requirements are implicit in the phrase 'provide computers with human-like reasoning capabilities':

1. A reasoning algorithm needs to be implemented.
2. The algorithm needs to cover a combination of different forms of reasoning that humans are able to perform.
3. The algorithm needs to produce reasoning results that are either provably correct or comparable to human reasoning in its correctness. There is deep disagreement in the field which one of these two requirements needs to hold.
4. The algorithm needs to run at a speed comparable to human reasoning.
5. The algorithm needs to operate on large knowledge bases, comparable in size to human knowledge, without any performance degradation.

It is possible to classify all current reasoning research paradigms based on which of the above conditions are observed, and which are relaxed or ignored. In our research, we have been insisting on large knowledge bases and fast responses, while relaxing the requirement for universality of the reasoning algorithm. Indeed, we are limiting ourselves to *transitive closure reasoning* in class hierarchies. Most previous work on transitive closure reasoning is limited to a single hierarchy, typically the IS-A hierarchy. We present algorithms for transitive closure reasoning in a hierarchy that combines IS-A, PART-OF and Contained-in relations. The combination of these different kinds of relations into one hierarchy creates new difficulties for which we will present solutions.

Reasoning in class hierarchies is important for modeling human reasoning and has also been used in a number of application areas. For instance, class hierarchies have been used in data mining and real-world classification applications such as classifying documents into subject categories under the Library of Congress scheme, or classifying World Wide Web documents into topic hierarchies. The deeper logical interrelations between the concepts in a taxonomy, as well as an extensive classification, have been improved by introducing efficient hierarchy encodings in digital library applications (Schwartz, 1999; Geffner et al., 1999).

Classification algorithms have been developed in order to extract interesting information (Taylor et al., 1997; Zupan et al., 1997), to improve prediction accuracy (Greiner et al., 1996), to discover meta-rules as a guidance for finding multiple-level association rules in databases (Han et al., 1996; Srikant and Agrawal, 1997), to deal with semantic heterogeneity problems in cooperative information systems (Han et al., 1998), and to mine sequential patterns for discovering concepts across all levels of a taxonomy (Srikant and Agrawal, 1996). All these studies have in common that effective classification rules covering generalization and specialization in determining the range of queries, pruning data or mining data can be identified using multiple levels of abstraction. As realistic hierarchies and taxonomies are large, any such systems stand to gain from improvements in hierarchy representations.

The limitation to special-purpose reasoning is not sufficient to produce fast reasoning with large knowledge bases. Two additional ingredients are necessary, and they are high-performance computing and good data organization. We will first discuss the use of powerful high-performance computing, specifically, parallel computing. Several researchers have used the combination of special-purpose reasoning and parallelism. The approach of (Shastri, 1988) is connectionist. Other approaches have been based on symbolic AI (Hillis, 1985; Waltz and Stanfill, 1988; Waltz, 1990; Kitano and Higuchi, 1991; Evett et al., 1993). Barnden and Srinivas (1992) have combined symbolic and connectionist techniques in order to reap the benefits of both approaches. We will give extensive justifications for the need of high-performance computing in Section 2.1.

Besides special-purpose reasoning and high-performance computing, another necessary ingredient of our research is organizational. We need to use good data representations for the hierarchies that we want to reason with. We are using a representation called *materialized transitive closure* (Agrawal et al., 1989, 1990). We have adapted materialized transitive closures to high-performance computing. In Section 3 we will describe our data representation in more detail. To summarize, this paper reports on our work on representations and algorithms for transitive closure reasoning over large knowledge bases, containing combined relational hierarchies, on high-performance hardware.

This paper is organized as follows. Section 2 describes our motivations. Section 3 summarizes our previous work on massively parallel reasoning (Lee and Geller, 1993, 1995, 1996a, 1996b; Lee, 1997). In Sections 4 and 5 we concentrate on the definitions and algorithms necessary for intuitively correct reasoning with combined relations and for implementing this kind of reasoning within our massively parallel paradigm. Section 6 describes experimental results of our work with the InterMED. Section 7 contains our conclusions. A preliminary version of this paper appeared in Lee and Geller (1996b).

2. Research Motivation

2.1. Why High-Performance Computing?

It is not immediately intuitive to everybody why high-performance computing should be needed to achieve fast reasoning. A number of previous publications have addressed this question in the context of AI (Hillis, 1985, 1988; Shastri, 1988; Waltz, 1988, 1990). However, we will briefly summarize arguments why high-performance computing appears necessary, both for modeling human reasoning and in several related application areas.

1. *Intuition of scalability:* People are amazingly fast at many tasks, e.g. face recognition. While we don't know how this is achieved, we propose the following strawman algorithm. Every known face is stored as a feature vector on its own processor. A test face, which needs to be recognized, is translated into a feature vector and broadcast to all processors. The test feature vector is compared in parallel to all the stored feature vectors. Every processor reports a similarity value and the result is the feature vector with the largest similarity value. Intuitively, this recognition method will scale up well as long as there are more parallel processors available than feature vectors.[1] Since scalability has been a major problem

[1] Increasing the number of processors will lead to increased broadcast times.

in knowledge representation, any methodology that appears to be scalable should receive consideration.

2. *Evidence of parallelism in the human brain:* The human brain operates in parallel, and it seems like a good idea to imitate this feature, regardless of the fact that airplanes don't fly like birds. How do we even know that the human brain operates in parallel? Feldman (1982) has argued for a *hundred-step rule* of mental processing in the following way. Neurons are very slow processors. Many human tasks, such as face recognition, can be performed in about 500 ms. If the cycle time of a neuron is about 5 ms, then one neuron can perform at most 100 operations for a typical everyday problem. As we don't assume that a few neurons can perform face recognition in 100 processing cycles, it follows that large amounts of neurons need to be active at the same time.

In newer research, Thorpe et al. (1996) demonstrated that certain decisions, such as whether an image depicts an animal or not, are made in about 150 ms. That means that the brain needs only about 30 cycles to solve a problem that is still beyond the reach of implemented AI. Since the human brain is still the only cognitive processor that has passed the Turing test, it appears beneficial to incorporate parallelism into reasoning systems.

3. *Benefits of raw computing power:* Estimates show that the brain seems to be of enormous memory and computational power, containing about 100 billion neurons. Many of these neurons have direct connections to between 1000 and 10,000 other neurons (Schwartz, 1988). The memory capacity of the brain has been estimated at 10,000 trillion bytes and its processing speed at 1,000,000 trillion arithmetic operations per second, give or take a factor of 100 (Schwartz, 1988).

Having raw computational power available seems at least helpful in achieving the appearance of intelligence. In a historic event, Deep Blue II defeated chess world champion G. Kasparov in May 1997 3 1/2 to 2 1/2. Nobody, not even Deep Blue's designers, claimed that the machine has any form of human-like intelligence. Chess is considered an easy AI task, because it is accessible, deterministic, semi-static, and discrete, following the analysis of Russel and Norvig (1995). The real world tends to be inaccessible, non-deterministic, non-episodic, dynamic, and continuous. Maybe with more power, more difficult tasks will become solvable.

We are firm believers that a quantitative improvement becomes at a certain point a qualitative improvement (Geller, 1993). Interestingly, Kasparov (1997) himself has used just this phrase:

From a certain point on, it seems, at least in chess, that enormous quantity turns into quality. [Translated from German]

There are numerous other examples in technological history, where an improvement in speed of one or two orders of magnitude has led to complete changes in culture and even history (cars, airplanes, microprocessors, computer networks). Having seen examples where raw power led to remarkable *qualitative* improvements in system behavior, there is justification for investigating high-performance computing approaches to AI.

4. *Successful parallel applications:* Even in fairly applied research areas such as data mining, high-performance computing has been a useful tool (John and Langley, 1996; Carino and Jahnke, 1998). Parallel processing allows one to answer complex queries in very large knowledge bases in fractions of a second (Stoffel et al., 1997). Parallel high-dimensional proximity joins (Shafer and Agrawal, 1997)

and parallel searching for text inference in natural language applications such as information extraction, text understanding, and machine translation (Harabagiu and Moldovan, 1998) have been studied. Parallelism was adopted in collective search for economic dispatch problems in the cooperative agent domain (Song et al., 1999). Parallel architectures have been designed for high-speed image recognition (Farroha and Deshmukh, 1995). Recently, *cellular computing* has been introduced (Sipper, 1998, 1999) as an extension of massive parallelism for more efficient computation, in terms of speed, cost, power dissipation, information storage, solution quality and potential for addressing much larger problem instances.

2.2. Why Transitive Closure Reasoning?

AI has defined many different tasks that require reasoning. Let us choose two paradigmatic examples: transitive closure reasoning and theorem proving. The second requirement in the list of requirements in Section 1 says that the algorithm needs to cover all kinds of reasoning that *humans* are able to perform. Transitive closure reasoning is routinely performed by humans. Inheritance reasoning, which is closely related to transitive closure reasoning, is a mental operation that humans perform daily.

Inheritance reasoning has been heavily investigated in AI. IS-A hierarchies have been a main staple in knowledge representation and reasoning since Quillian (1968). The number of papers on other hierarchies is, comparatively, much smaller. However, PART-OF has received some attention (Schubert, 1979; Schubert et al., 1983, 1987). Specifically, in Schubert et al. (1987) the importance of transitive closure reasoning in IS-A and PART-OF hierarchies has been well explained, so that we don't need to elaborate on it in this paper. We have previously done research on PART-OF hierarchies, especially on inheritance reasoning (Halper et al., 1992, 1993, 1994, 1998).

Transitive closure reasoning is not limited to hierarchical relations such as IS-A and PART-OF. It pervades human reasoning in other forms also. For instance, a human can answer 'Is an elephant bigger than a can opener'? if he knows that an elephant is bigger than a person, and a person is bigger than a can opener.

Transitive closure reasoning also plays an important practical role. For instance, it is used in aggregating objects, referring to their parts without specifying a level of decomposition (Horrocks and Sattler, 1999) and combining results of individual passes of data-pruning processes in data-cleansing tasks (Hernandez and Stolfo, 1998). Plexousakis (1993) proposed a representation language, called Telos, to represent structural, temporal and assertional knowledge in the form of deductive rules and integrity constraints and also for the incremental maintenance of the computed transitive closure for knowledge bases. In Horrocks and Sattler (1999), an extensive study of role hierarchies, including part–whole, transitive, and inverse relations, was performed. Joslyn and Rocha (1998) have shown how to formally combine various uncertainty representations in a taxonomic structure, capturing both syntactic and semantic generalization.

Transitive closure reasoning that combines IS-A relations with other relations, such as PART-OF, has received little attention. An exception is a seminal paper by Winston et al. (1987) that combines different binary transitive relations into a single reasoning process. However, not every conclusion that can be drawn in such a case is correct. Winston et al. describe a condition when such reasoning is correct. For example, with the premises:

(1) Wings are parts of birds.
(2) Birds are creatures.

we can consider the following two conclusions:

(3) Wings are parts of creatures.
(4) * Wings are creatures.

We have obtained a reasonable conclusion (3) while (4) is an invalid conclusion. Winston et al. (1987) introduced a hierarchical priority ordering among the hierarchical relations, such that combined inclusion relation syllogisms are valid if and only if the conclusion expresses the lower relational priority appearing in the premises. In this paper, we are addressing combined relational transitive closure reasoning, following Winston et al.'s paradigm. For an alternative approach to this problem see Cohen and Loiselle (1988).

2.3. Why Good Data Structures?

The need for good data structures is normally not questioned. However, it is always helpful to have *better* data structures, and Schubert has introduced a particularly good data structure, which represents the materialized transitive closure of an IS-A hierarchy. His representation permits transitive closure reasoning in constant time, practically independent of the height of the knowledge base. In our previous work we have developed techniques and algorithms for dealing with two weaknesses of Schubert's approach: (1) Schubert did not present (incremental) update algorithms; and (2) Schubert's original approach is limited to trees. We have dealt with problem (1) by using massive parallelism (Lee, 1997), and with problem (2) by adapting a representation of Agrawal et al. (1989, 1990), that permits closures of directed acyclic graphs (DAGs), to parallel processing.

Currently, there is great interest in directed acyclic inheritance graphs for dynamic and complex objects. Van Bommel and Beck (1999) show experimental results from comparisons of several incremental update methods for multiple inheritance hierarchies. Among their many interesting results, we find that while a top-down incremental update algorithm minimizes the storage needs for hierarchies of less than 300 nodes, for larger hierarchies Agrawal's approach (1989) is better. Our recent study (Han and Lee, 2000) analyzed the time and space complexity for several well-known encodings including Boolean matrix (Warshall, 1962), matrix multiplication (Munro, 1971; Fischer and Meyer, 1971), improved matrix encodings such as Ait-Kaci et al.'s (1989) and Ganguly et al.'s (1994) and also Agrawal et al.'s (1989). During the analysis, we noticed that the matrix-based encodings (Warshall, 1962; Munro, 1971; Fischer and Meyer, 1971) require constant storage space, i.e. $O(n^2)$, and excessive encoding construction time, i.e. $O(n^3 \log n)$. Ait-Kaci et al.'s (1989) and Ganguly et al.'s (1994) constant time verification results appear to be underestimates. In Han and Lee (2000), we showed that the actual estimates for their representations should be $O(n/\log n)$. We also derived (Han and Lee, 2000) that Agrawal et al.'s encoding requires less space for real-world problems and less construction time compared with the encodings of Warshall (1962), Munro (1971), Fischer and Meyer (1971), Ait-Kaci et al. (1989), and Ganguly et al. (1994). Several transitive closure studies (Ebert, 1981; Ioannidis et al., 1993; Schmitz, 1983; Nuutila,

1994) are based on strong component computing of Tarjan (1972). Specifically, a reachability-based transitive encoding (Nuutila, 1994) called the compact successor set is based on Tarjan's depth-first spanning tree algorithm and Agrawal et al.'s encoding. The primary concern of this encoding seems to be fast transitive closure construction time. However, since the spanning tree construction is a part of preprocessing in our approach, we are more interested in optimization of encoding storage than in tree construction time. Agrawal et al. (1989) proved that the spanning tree constructed by their approach is optimal in terms of space requirements for storing the encoding. Furthermore, it is not clear whether the encoding of Nuutila (1994) is appropriate for our reasoning model, which requires a reasoning process combining several different relations. In the future, we will investigate whether or not our approach can be combined with this encoding.

2.4. The MED Medical Vocabulary as Application Area

Research on reasoning with large knowledge bases is often performed with random graphs as simulated knowledge bases, because few real knowledge DAGs are widely available. One knowledge base, a DAG that contains IS-A and PART-OF relations, is the Medical Entities Dictionary (MED) (Cimino et al., 1994). The MED is a very large vocabulary, incorporating over 60,000 concepts and over 84,677 hierarchical links. The MED is not publicly available, but a subset, the InterMED, is accessible over the Web and was used in our research.

An example of a combined relational query which can be answered with the MED is the following. *Is it the case that Aspirin may be coated?* Aspirin itself is represented in the MED as a concept. This concept might have several descendants according to different common preparations, such as pills, drops, or capsules. Capsules consist of two parts: the active ingredient and the coating. Thus, the answer is 'Yes, Aspirin can be coated.'

Other researchers have combined medicine and knowledge representation. Because a common language is a prerequisite for integrating disparate applications in health care, formal representations based on description logics are increasingly used for medical knowledge. Recent interest in medical reasoning and strong motivation to use shared medical conceptualizations have led to the implementation of large-scale formal reasoning in medical ontologies (Rector et al., 1998).

GALEN's experience (Rector et al., 1997) raises two interesting issues: how to make an ontology based on description logics easy to use and understand, and what features are required for an effective ontology and description logic. In addition, transitive closure reasoning in part–whole hierarchies as well as concept specialization and role propagation have been investigated. Integrating knowledge base technology with database systems provides flexible answering of queries (Chu et al., 1996). Rector et al. (1998) have pointed out that combined relational hierarchies are extremely common in existing medical classifications but there are major limitations of their use for formal reasoning (e.g., how to differentiate 'kind-of' and 'PART-OF'). The transitive relations need to be expressively formalized despite the computational cost, and that computational cost is acceptable in practice (Rector et al., 1998). In dealing with large amounts of medical data, PARKA-DB (Hendler and Stoffel, 1999) presented scalable and high-performance computing using an ontology and relational database techniques.

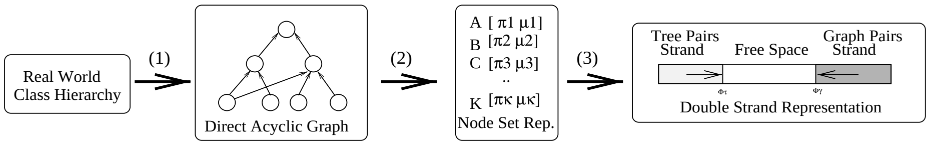

Fig. 1. Three-step mapping.

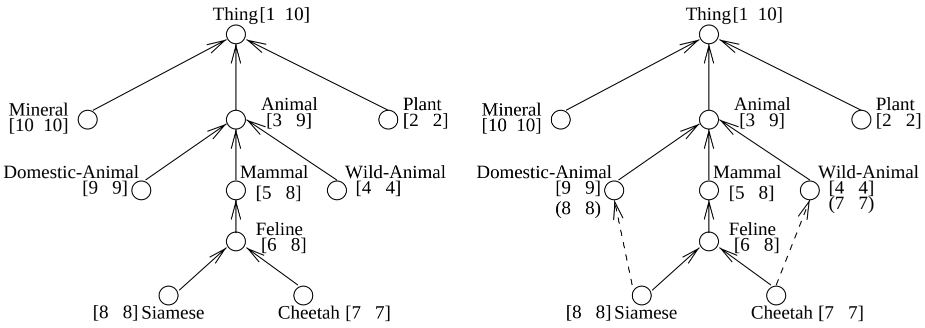

Fig. 2. (a) IS-A hierarchy; (b) directed acyclic graph (DAG) of class hierarchy.

3. Efficient Representations for Class Hierarchies

This research has focused on building an efficient transitive closure reasoner, called Hydra, that can dynamically assimilate any transitive, binary relation and efficiently answer queries using the transitive closure of all those relations (Lee, 1997). Hydra can dynamically insert new concepts or new links into its knowledge base.

An important problem we solved was how to map a given class hierarchy onto a space of parallel processors. The most obvious intuitive choice is to assign every class of the hierarchy to a single processor. However, this intuitive choice does not carry over to the links between classes (Lee and Geller, 1993). We have developed a three-step mapping (Fig. 1) to deal with this problem. In this section we will only describe the representation of the IS-A backbone of the class hierarchy. The combined relational hierarchy will be described in Section 4.

Step 1: We are modeling a microworld by a class hierarchy. Concepts of the world are represented by classes that are connected by IS-A links. In the simplest possible case this hierarchy is a tree. In Fig. 2(a), an example of such an IS-A hierarchy is shown. The IS-A relation is transitive. The node Feline is connected by one IS-A arc to the node Mammal, which means that every Feline is a Mammal. Due to the transitivity of the IS-A relation, every Siamese is also a Mammal, etc. More interesting than trees are directed acyclic graphs with multiple inheritance. We are limiting ourselves to singly rooted DAGs by introducing an artificial root 'Thing' whenever necessary (Fig. 2(b)). DAGs will be discussed in Step 2(c) below.

Step 2: The *hierarchy* of nodes is mapped onto a *set* of those nodes, so that every node is annotated with one or more number pairs. This is called the node set representation. The number pair assignment was developed based on Schubert et al.'s (1983, 1987) special-purpose reasoner, which permits transitive closure reasoning in constant time (in a tree), practically independent of the size of the knowledge base.

Step 2(a): A spanning tree of the DAG is constructed. (If the knowledge base is a tree, this step is omitted.) Many efficient spanning tree construction algorithms are available. Tarjan's (1972) depth-first approach, in particular, is well known and widely used in various applications, including compiler construction. There are several reasons we employed Agrawal et al.'s approach instead. First, since the spanning tree construction is a part of preprocessing in our approach, we are more interested in optimization of encoding storage than in tree construction time. Agrawal et al. (1989) proved that the spanning tree constructed by their approach is optimal in terms of space requirements for storing the encoding. Second, our reasoning model and our node set representation were built based on Agrawal et al.'s encoding. Therefore, their spanning tree algorithm was chosen, as will be further explained below.

Step 2(b): The nodes in the (spanning) tree are annotated with number pairs. The first number in every number pair indicates after how many steps this node was reached in a preorder traversal of the spanning tree. This is called the preorder number. (For historical reasons this traversal is done from right to left.) The second number in a number pair of a node A is the maximum of all the preorder numbers that appear under A. This is called the maximum number. We call the number pairs obtained in this way *tree pairs* and use the notation $[\pi \;\; \mu]$ for them.

These number pairs can be used for verifying a transitive relation between any pair of nodes in the spanning tree *without actually traversing the path from one node to the other node*. For instance, in Fig. 2(a), we may conclude that a Cheetah is an Animal, because [7 7] is contained in [3 9].

Step 2(c): For trees every node needs only one number pair. However, for DAGs, a node might need several number pairs. Let us assume that a node A in the DAG is reachable through a link, which is not part of the spanning tree, from a node B (B below A). If B has a number pair P that is not contained in (or equal to) any number pair at A, then P needs to be 'propagated' upward from B to A. We call the number pairs obtained by propagation *graph pairs* and use the notation $(\pi \;\; \mu)$ for them. Agrawal's spanning tree algorithm is optimal, because it minimizes the number of graph pairs. However, we will explain below how we can improve the representation for a parallel environment.

Now all transitive relations can be verified by comparing number pairs. However, as some of the nodes have several pairs, this is not a constant time operation anymore. For example, (Fig. 2(b)), Siamese is a Domestic Animal because [8 8] of Siamese is equal to (8 8) of Domestic Animal. (Equality is a special case of containment.)

Step 2(d): Now links are no longer necessary and may be omitted. The annotated nodes are placed in a set, the resulting node set representation. All relational information is contained in the number pairs. The node set representation simplifies the parallel update operations necessary to maintain a class hierarchy. More details can be found in Lee and Geller (1993) and Lee (1997).

Step 3: The node set and the associated number pairs are mapped onto the processor space of a high-performance computer. We have developed and implemented a method for mapping this set-based representation onto a Connection Machine (CM-5), called the *Double Strand Representation.* The Double Strand Representation consists of two linear strands of processors (Fig. 1). In the *tree* *pairs strand*, every tree pair of a node is represented by one processor. In the *graph pairs strand*, pairs of nodes represent 'propagations'. One node in each pair represents a propagation source, and the other node represents a propagation target. (Do not confuse the pairs of processors with number pairs!)

In brief, our three-step mapping (Fig. 1) can be summarized as follows: *class hierarchy → directed acyclic graph → node set + number pairs → processor space.* As a result of this mapping we get nearly constant time responses for transitive closure queries (Lee and Geller, 1995). For example, it takes as much time to verify that a Cheetah is an Animal as to verify that a Cheetah is a Feline (Fig. 2(b)). We are using the node set representation in a parallel computing environment. However, because it is a pointerless representation, it may also be used in distributed environments where large sub-hierarchies are represented on networked workstations. We have described the construction of our representation assuming a static world. However, most real-world reasoning situations require evolving knowledge. It has been a major point in our previous research (Lee, 1997) to develop and implement parallel algorithms for updating a class hierarchy represented by the Double Strand Representation.

4. Extension to Combined Relational Hierarchy

A *combined relational hierarchy* allows multiple relations to coexist in one knowledge base. This permits transitivity through several different relations. However, this leads to difficult tasks, both during construction of the representation and during transitive closure reasoning. Below we explain how we have added combined relational hierarchies to our representation of Section 3.

First, we are modeling a microworld by a combined relational hierarchy. This hierarchy is then mapped onto a singly rooted DAG of classes with the following constraints. The subgraph consisting of all the classes and all the IS-A links is a connected DAG. To this DAG are added PART-OF and Contained-in relations with the limitation that no cycles may be introduced by this process. Thus, the final hierarchy is a DAG. In our models we have found that this representation is sufficiently powerful to be of real interest. A more general approach would be to allow for a union of three independent hierarchies $I \cup P \cup C$, where I is an IS-A hierarchy, P is a PART-OF hierarchy, and C is a Contained-in hierarchy. An analysis of this approach, which might introduce cycles, is beyond the scope of this paper. Then we construct a spanning tree as before, limited to the IS-A relations. This spanning tree becomes the backbone of the structure while the remaining IS-A links and the other hierarchical relations form its branches. Figure 3(a) shows the backbone and Fig. 3(b) shows the backbone and the branches of an example.

Next, we add number pairs as we did for pure IS-A hierarchies. However, the following questions arise: (1) How do we distinguish one relation from the others? (2) How do we combine relations when required? Our representation, extending what was described in Section 3, is generated as follows:

1. *Construct spanning tree:* We construct an optimal spanning tree of a given combined relational DAG. Here we are using a new technique, different from the one introduced by Agrawal et al. (1989). At every node with multiple parents, we include the link to the parent with the maximum number of *weak predecessors* in the spanning tree. Informally, a weak predecessor B of a node A is a predecessor

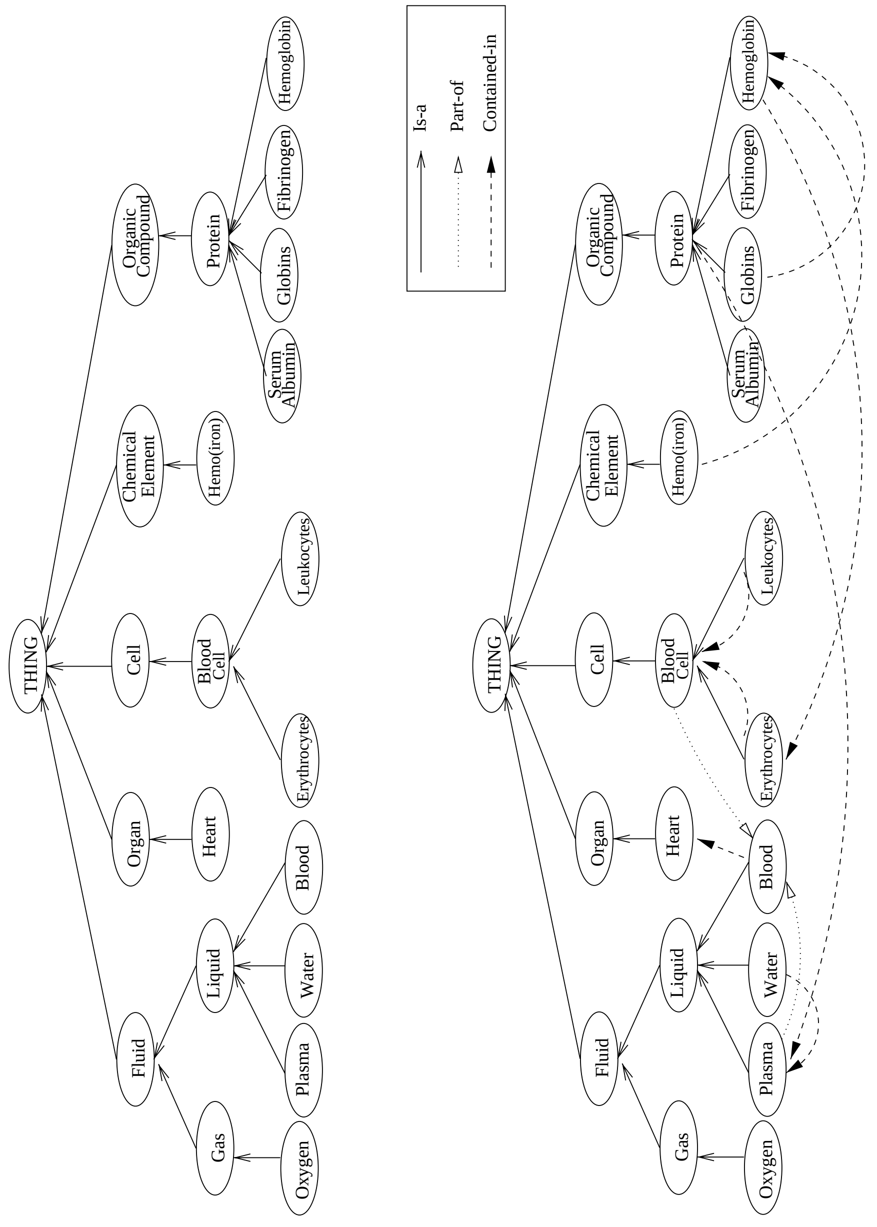

Fig. 3. (a) A backbone of a combined relational hierarchy; (b) an example of a combined relational hierarchy.

which is reachable from A by an upward path containing at least one graph arc and B is at the head of a graph arc. In order to formally define the notion of a weak predecessor we need several auxiliary definitions which are closely following the terminology of Agrawal et al. (1989). Alternative definitions have appeared in the literature, e.g. the use of forward and cross-links (Nuutila, 1994).

Definition 4.1. A *tree arc* from B to A is an arc from B to A that is part of the spanning tree. A *graph arc* from B to A is an arc from B to A that is not part of the spanning tree.

Definition 4.2. A *tree path* from B to A is a path from B to A that consists of tree arcs only. A *graph path* from B to A is a path from B to A that contains at least one graph arc.

Definition 4.3. The set of *predecessors* of a node A is the set of all nodes that are reachable from the node A by any arc or path.

Definition 4.4. The set of *successors* of a node A is the set of all nodes from which the node A is reachable by any arc or path.

Definition 4.5. A *tree predecessor* A of a node B is a predecessor of B such that A is reachable from B by a tree path.

Definition 4.6. A *tree successor* A of a node B is a successor of B such that B is reachable from A by a tree path.

Definition 4.7. A *weakly terminated path* is a path that consists of a tree path of length n, $n \geqslant 0$, followed by a single graph arc.

Definition 4.8. A *weak predecessor* A of a node B is a predecessor of B, such that

$$\left\{\begin{array}{ll} & A \text{ is at the end of a weakly terminated path from } B \text{ to } A \\ OR & \\ & \text{Some node } X \text{ exists, such that } X \text{ is a weak predecessor of } B, \\ & \text{and } A \text{ is at the end of a weakly terminated path from } X \text{ to } A. \end{array}\right.$$

Thus, we are constructing a spanning tree by always choosing the parent that has the maximum number of weak predecessors. If there is a node P which is a weak predecessor of C *and* also a tree predecessor of C, we count the weak predecessors of C without P. If there are several parents with equal maximum numbers of weak predecessors, we randomly choose one. Maximality is important, because it allows us to prove a result (Lee and Geller, 1996a) discussed at the end of step (3) below.

2. *Assign tree pairs:* We assign a pair of preorder and maximum numbers to every node of the spanning tree. Preorder numbers and maximum numbers are generated as in Section 3, Step 2(b).

3. *Propagate pairs:* Our transitive closure reasoning mechanism works based on an extension of a number pair propagation algorithm introduced in Lee and Geller (1996a). There we proved that if there is a tree path from σ to τ, we can achieve the effect of having all graph pairs of σ at τ, without actually propagating these pairs to τ, resulting in an additional saving of space. In other words, instead of propagating number pairs along all graph arcs, we propagate them to weak predecessors only. Above 'achieve the effect of having all graph pairs' means

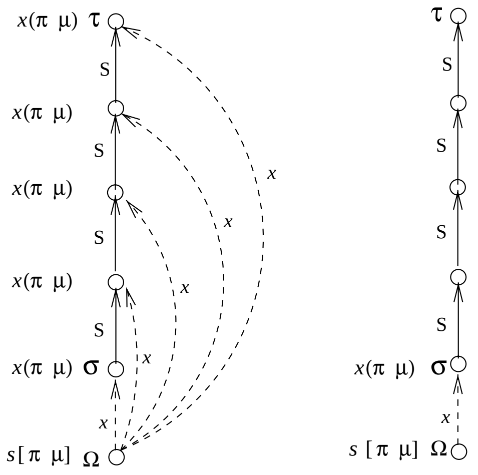

Fig. 4. (a) Agrawal's propagation; (b) maximally reduced propagation.

that we can perform constant time subclass verification and all operations that rely on subclass verification, including propagation itself. We call the resulting structure the *maximally reduced tree cover* (Fig. 4(b)). If a number pair would be propagated to a node where an existing number pair encloses it, this propagation is not performed.

Now we need to integrate relations other than IS-A into our number pair representation. For this purpose, we introduce a data element called *relation type*. Each relation is associated with a relation type and a priority. Figure 4(a) shows propagation based on Agrawal's approach (1989) and Fig. 4(b) shows our approach. When a graph arc is inserted from Ω to σ with the relation type x, we propagate the pair $x(\pi\ \mu)$ only to σ instead of propagating $x(\pi\ \mu)$ up to all tree predecessors of σ (including τ).

In this paper we assume that the possible relation types are s (for IS-A; the letter s is from subclass), p (PART-OF), and c (Contained-in). In Fig. 5, we assign the lowest priority (1) to IS-A, the intermediate priority (2) to PART-OF, and the highest priority (3) to Contained-in (Winston et al., 1987).

Now we have to describe how the relation type of a number pair is transformed during propagation. For this we define two rules. The propagation behavior depends on the link type and the relation type that is propagated.

Rule 1. If a tree pair is propagated along a link, it is transformed in the process into a graph pair with the relation type R of the link.

$$s[x\ \ y] \xrightarrow{R} R(x\ \ y)$$

In the above schema, s stands for the IS-A relation type, and R stands for the relation type of the link. Since the priority of the relation type s (IS-A) is lowest among all the relation types, the relation type of the propagated pair will always be replaced by the relation type R associated with the link.

Rule 2. If a pair with a relation type K with relational priority X is propa-

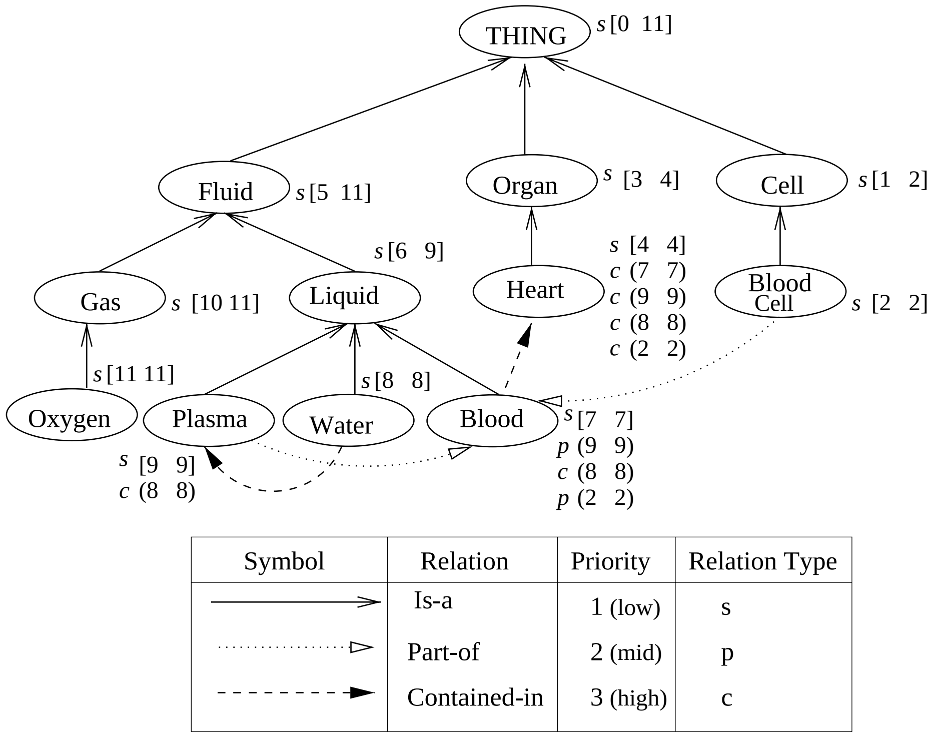

Fig. 5. An example of a combined relational hierarchy.

gated along an edge with a relation type L with relational priority Y then the result of

$$K(x\ \ y) \xrightarrow{L} R(x\ \ y) \quad is \quad R = \begin{cases} K & \text{iff } Y \leqslant X \\ L & \text{iff } Y > X \end{cases}$$

and R is the relation type of the pair at the head of the edge. Informally, the result of the propagation of a pair along a link is the relation type of higher priority of the relation types of the pair and of the link.

In Lee and Geller (1996a) we have shown that the number of number pairs introduced in this way is smaller than in the optimal tree cover of Agrawal (1989). In (Lee, 1997) we have shown that the maximally reduced tree cover for combined relational hierarchies is sufficient for the efficient *parallel* execution of all necessary retrieval and update operations.

4. *Eliminate links:* The set of all nodes together with their tree pairs and graph pairs is generated.

5. *Map nodes onto processors:* All the nodes and their associated number pairs are mapped onto the Double Strand Representation (Fig. 6).

In Fig. 5, ‘Water is a Fluid’ exactly because the number pair of Water s[8 8] is contained in the pair for Fluid s[5 11]. For more details, see (Lee, 1997; Lee and Geller, 1993, 1995, 1996a).

For an example of propagation (Fig. 5) assume that a PART-OF arc from

Plasma to Blood was just inserted. Now the tree pair $s[9\ \ 9]$ and the graph pair $c(8\ \ 8)$ need to be propagated to the nodes (Blood, Heart) (Lee and Geller, 1996a). Therefore, this pair $s[9\ \ 9]$ is propagated through a PART-OF relation from Plasma to Blood, and a Contained-in relation from Blood to Heart. The tree pair $s[9\ \ 9]$ of Plasma is propagated through PART-OF to Blood, resulting, by Rule 1, in the pair $p(9\ \ 9)$. Continuing from Blood to Heart, the pair $p(9\ \ 9)$ needs to be changed to $c(9\ \ 9)$, by Rule 2. This allows us to process queries such as 'Is Plasma a part of Blood'? and 'Is Plasma contained in Heart?'

In contrast, the graph pair $c(8\ \ 8)$ at Plasma has a Contained-in relation type and its priority is higher than that of the PART-OF relation of the arc from Plasma to Blood and is equal to the priority of the Contained-in arc from Blood to Heart. Therefore, the pair $c(8\ \ 8)$ is propagated to Blood and Heart with its own relation type, by Rule 2. Because Heart has a pair $c(8\ \ 8)$ and Water has a pair $s[8\ \ 8]$, we can conclude that Heart contains Water. This uses the number pairs to determine that a relation exists, and the relation type c to determine the kind of the relation.

5. Transitive Reasoning in Combined Relational Hierarchy

In this section we show that we can achieve constant time responses (for a given machine size) for parallel transitive closure queries in a combined hierarchy. Assume that $R_1, R_2, \ldots, R_n$ are hierarchical relations.

Definition 5.1. A *target of transitivity*, τ, is a node at the upper end of a path that is used for transitive closure reasoning.

Definition 5.2. A *source of transitivity*, σ, is a node at the start (lower end) of the path that is used for transitive closure reasoning.

Definition 5.3. A path P from σ to τ is *purely transitive* iff $P = \sigma\ R_1\ \sigma_1\ R_2\ \sigma_2 \ldots R_n\ \tau$, and $R_1 = R_2 = \ldots = R_n$.

Definition 5.4. A path P from σ to τ is *combined transitive* ($\sigma\ R^x\ \tau$) iff $P = \sigma\ R_1\ \sigma_1\ R_2\ \sigma_2 \ldots R_n\ \tau$, and R^x is such that Priority(x) = Maximum(Priority(R_1), Priority(R_2) , ..., Priority(R_n)), and at least one $R_i \neq R_j$.

Both transitivities satisfy the following property: If $\sigma\ R_1\ \sigma_1 \ldots \sigma_n\ R_n\ \tau$ holds, then $(\sigma\ R\ \tau)$ & $(R = R_1$ *or* $\ldots\ R = R_n)$. Importantly, pure transitivity reasoning and combined transitivity reasoning can be done in one step. We will present how these mechanisms can be integrated during reasoning.

Lemma 5.1. Pure transitivity in IS-A path. Let R^x be an IS-A relation ($R^x = R^s$). If $\sigma\ R^x\ \tau$ holds through a pure inference path, then, after applying our propagation algorithm from Step 3 of Section 4, the target τ or one of its tree successors will have a number pair with the relation type $x = s$ propagated from σ or from one of its tree predecessors.

Proof. By contradiction, assume that the target τ or a tree successor of τ has a graph pair $s(\pi\ \ \mu)$ from the source σ or one of its tree successors, although an arc A in the inference path from σ to τ is not an IS-A relation. Since the IS-A relation has the lowest relational priority among all relations, the pair $s(\pi\ \ \mu)$ cannot be propagated through an arc A unless the relation type of $s(\pi\ \ \mu)$ changes to the relation type of the arc A (Rule 2 of Section 4). Therefore, the pair $(\pi\ \ \mu)$ cannot be associated with the relation type $x = s$. This results in a contradiction. □

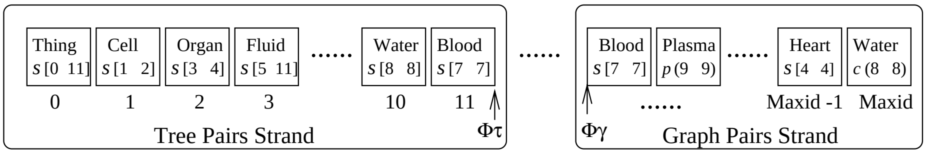

Fig. 6. Double Strand Representation.

Lemma 5.2. Combined transitivity. Let R^x be a relation. If a source σ relates by R^x to a target τ because of a combined inference path from σ to τ, the target τ or one of its tree successors must have a number pair with relation type x propagated from the source σ or one of its tree predecessors.

Proof. By using Rule 2 of Section 4 and maximally reduced propagation, this is trivial. □

Theorem 5.1. If τ (or a tree successor of τ) has a pair that contains (or is equal to) a pair from σ (or a tree predecessor of σ) then the relation type of the pair of τ (or a tree successor of τ) tells the relation which actually holds between σ and τ.

Proof. We prove the theorem by using the above two lemmas. If the relation is an IS-A relation, the query is an instance of IS-A pure transitivity reasoning (Lemma 5.1). Otherwise, the query is an instance of combined transitivity reasoning or pure transitivity reasoning with other relations (Lemma 5.2). □

Now we will show how to answer queries within our paradigm in parallel. We divide pure transitivity into two subcases: one for an IS-A relation and another for other hierarchical relations. A description of pure transitivity with other relations will be omitted for space reasons (Lee, 1997).

First we introduce some specialized parallel processing terminology. A processor may be active or inactive. A parallel variable (*pvar*) is an array (possibly multi-dimensional) where every component is maintained by its own processor and all values are usually changed in the same way and in parallel on all active processors. In the algorithm, variables marked with !! are parallel variables, and operations marked with !! or involving parallel variables are parallel operations (TMC, 1988).

The parallel variable pre!! contains for every node its preorder number, max!! stands for a parallel variable that contains for every node its maximum number, and reltype!! stands for a parallel variable that contains for every number pair its relation type. The functions prenum(), maxnum(), and tree-pair() retrieve the preorder number, maximum number, and the tree pair, respectively, for the given argument.

Additionally, the variable Φ_γ represents the lower bound of the graph pairs strand and Φ_τ represents the upper bound of the tree pairs strand. The parallel function *self-address!!* returns IDs of all active processors and *oddp!!* contains TRUE on a processor if the processor's ID is an odd number. The parallel control structure 'For all active processors in Parallel' describes what operations should be performed on all active processors.

Informal Algorithm: Pure Transitivity with IS-A (σ, τ)

1. Activate every processor that contains a pair with the IS-A relation type (s).

2. (Case 1: a tree path from σ to τ)
   Among active processors, check whether the tree pair of σ is contained in or equal to the tree pair of τ.
3. (Case 2: a graph path from σ to τ)
   Among active processors, check whether any processor has a tree pair of τ or a pair of a tree successor of τ at an odd processor ID = x, and the processor with ID = $x + 1$ contains a pair [propagated] from the tree predecessor of σ or from σ itself.
4. Iff Case 1 or Case 2 is the case, return 'yes'.

We now show a function IS-A-VERIFY that performs pure subclass verification. If τ is a tree predecessor of σ (by IS-A-VERIFY-1) or τ is a graph predecessor of σ (by IS-A-VERIFY-2), then σ IS-A τ. As every tree pair has a relation type *s*, it is not necessary to check the relation type in IS-A-VERIFY-1.

```
IS-A-VERIFY ($\sigma$, $\tau$: Node): BOOLEAN
// B is-a A iff IS-A-VERIFY returns TRUE.
  return(IS-A-VERIFY-1($\sigma$, $\tau$) OR
         IS-A-VERIFY-2($\sigma$, $\tau$))
```

```
IS-A-VERIFY-1 ($\sigma$, $\tau$: Node): BOOLEAN
// If $\tau$ is a tree predecessor of $\sigma$, then
// the tree pair of $\tau$ subsumes the tree
// pair of $\sigma$. Note that all tree pairs
// in the tree pairs strand are associated with
// the relation type s (IS-A), thus a relation type
// check is not required.
  For all active processors in Parallel
      If (prenum(tree-pair($\sigma$)) $\geqslant$!!
              prenum(tree-pair($\tau$)) &&
          maxnum(tree-pair($\sigma$))$\leqslant$!!
              maxnum(tree-pair($\tau$)) &&
          self-address!!() $\leqslant$!! $\Phi_\tau$) Then
         return TRUE
```

Now we show how to verify that σ IS-A τ when τ is a graph predecessor of σ. Remember that a pair (U, V) in the graph pairs strand is used to represent a propagation. The tree pair in the odd processor (U) is used to represent a node τ and the graph pair in the even processor (V) is used to represent a node which propagates its tree pair to τ. Therefore, we are looking for a pair of processors (U, V) such that the tree pair of τ or of one of its tree successors is contained in the processor U and the graph pair of σ or one of its tree predecessors is contained in the processor V. In the following *mark*!![x] := y means that the pvar *mark*!! on the processor with the ID x is assigned the value y.

The first part of IS-A-VERIFY-2 locates a U processor that represents the node τ or a tree successor of τ (by maximally reduced propagation). If any such processor is found, then the mark is set on the V processor with the address of $U + 1$. In the second part, all marked processors are activated. If any pairs that were propagated from σ or any of its tree predecessors (by maximally reduced propagation) are found on V, then TRUE is returned. We omit the initialization of *mark!!*.

```
IS-A-VERIFY-2 ($\sigma$, $\tau$: Node): BOOLEAN
// Activate every occurrence of the tree pair of
// $\tau$ or its tree successors associated with the
// s relation type in the graph pairs strand. Set
// the parallel flag mark!! on the right neighbor
// processors of the active processors.
  For all active processors in Parallel
        If (reltype!! =!! 's' &&
            pre!! $\geqslant$!! prenum(tree-pair($\tau$)) &&
            max!! $\leqslant$!! maxnum(tree-pair($\tau$)) &&
            self-address!!() $\geqslant$!! $\Phi_\gamma$ &&
            oddp!! (self-address!!())) Then
            mark!![self-address!!() +!! 1]:= 1

// Test whether any marked processor has the tree
// pair with the relation type s from $\sigma$ or from
// a tree predecessor of $\sigma$, as a graph pair. If this
// is the case, return TRUE.
  For all active processors in Parallel
        If (reltype!! =!! 's' &&
            pre!! $\leqslant$!! prenum(tree-pair($\sigma$)) &&
            max!! $\geqslant$!! maxnum(tree-pair($\sigma$)) &&
            mark!![self-address!!()] =!! 1) Then
          return TRUE
```

Consider again our example of pure transitivity in Fig. 5: Is Water a Fluid? The tree pair of Fluid s[5 11] contains the tree pair of Water s[8 11]. The answer 'yes' can be given by comparing these two tree pairs (by IS-A-VERIFY).

```
COMBINED-RELATION-VERIFY
  ($\xi$: Relation Type; $\sigma$, $\tau$: Node): BOOLEAN
// Activate every occurrence of the tree pair of $\tau$ and
// of its tree successors in the graph pairs strand
// associated with the relation type s. Set the parallel
// flag mark!! on the right neighbor processors of
// the active processors. Here s stands for IS-A.
  For all active processors in Parallel
       If (reltype!! =!! 's' &&
            pre!! $\geqslant$!! prenum(tree-pair($\tau$)) &&
            max!! $\leqslant$!! maxnum(tree-pair($\tau$)) &&
            self-address!!() $\geqslant$!! $\Phi_\gamma$ &&
            oddp!! (self-address!!())) Then
            mark!![self-address!!() +!! 1]:= 1

// Test whether any marked processor has the tree
// pair from $\sigma$, or from a tree predecessor of $\sigma$,
// as a graph pair with the relation type $\xi$. If this
// is the case, return TRUE.
  For all active processors in Parallel
```

Table 1. Experimental results for three approaches

Hierarchy type	Reasoning type		Run-time (s)
SCH	Single transitivity	IS-A-Verify-1	0.00042
		IS-A-Verify-2	0.0082
CRH	Pure transitivity	IS-A-Verify-1	0.00042
		IS-A-Verify-2	0.0084
	Combined transitivity		0.0086

```
If (reltype!! =!! $\xi$ &&
    pre!! $\leqslant$!! prenum(tree-pair($\sigma$)) &&
    max!! $\geqslant$!! maxnum(tree-pair($\sigma$)) &&
    mark!![self-address!!()] =!! 1) Then
  return TRUE.
```

In the combined transitivity case the following algorithm COMBINED-RELATION-VERIFY is used. The query 'Is Water contained in Organ?' would be processed as follows. The procedure COMBINED-RELATION-VERIFY will be invoked with a list of arguments (*c*, Water, Organ). First, we are looking for nodes representing Organ and its tree successors. These nodes are associated with tree pairs contained in *s*[3 4]. Whenever any of these nodes has received a propagated pair from another node, its tree pair will appear on a *U* processor of a pair (*U*, *V*) in the graph pairs strand.

There we find Heart *s*[4 4] with a right neighbor Water *c*(8 8) (Fig. 6). In the second stage we are looking for a tree predecessor of Water *s*[8 8] (or Water itself), but with *s* replaced by the value *c*. This matches the pair *c*(8 8) identified in the first step, and we can conclude that the answer is 'Yes, Water is contained in Organ.' Due to parallel processing, the combined transitive closure query can be answered in constant time.

An analysis of the parallel operations involved shows that both kinds of queries can be answered with our parallel representation in constant time, independent of the size of the knowledge base (assuming constant machine size) (Wang et al., 1989).

6. Experiments with the InterMED

We have performed a set of experiments that analyze the run-times for purely transitive IS-A queries and combined relational queries using data from the InterMED. For this experiment, we have used 2495 nodes, 3372 IS-A links, and 682 Pharmaceutic-component-of links from the InterMED. Note that the fan-in and fan-out of the hierarchy are not of immediate experimental importance because the node set representation eliminates the explicit IS-A links. The only relevant factor is the number of graph pairs generated. The experiments were performed on a Connection Machine CM-5 (TMC, 1988) programmed in *LISP. For this purpose: (1) we created an IS-A hierarchy (SubClass Hierarchy, *SCH*); (2) We created a combined relational hierarchy (Combined Relational Hierarchy, *CRH*).

The results in Table 1 show that the run-times for transitive closure reasoning in both hierarchies are the same within unit processor space (8K virtual processors). We show run-times for 'normal' transitive closure reasoning in the *SCH* hierarchy, for pure transitivity reasoning and combined transitivity reasoning in the *CRH* hierarchy.

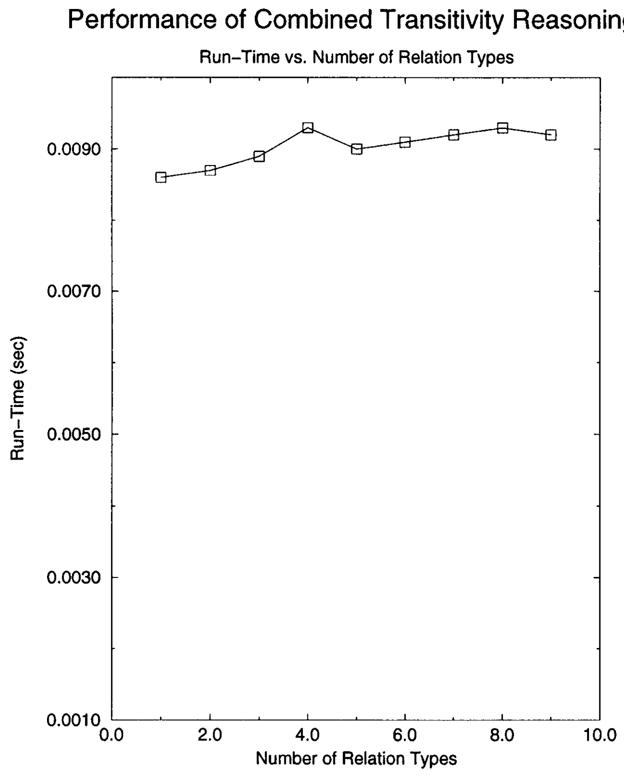

Fig. 7. An experiment of combined transitivity reasoning.

As another experiment, we measured how the number of relations is related to the run-time of combined relational queries. Additional relation types were created by a random generator. Fig. 7 shows that the run-time for combined relational transitivity reasoning is independent of the number of relations in a hierarchy. In summary, our combined transitive queries were executed in constant time, as in a pure IS-A hierarchy.

7. Conclusions

Human reasoning relies heavily on hierarchical knowledge, including IS-A hierarchies and PART-OF hierarchies. One kind of reasoning which makes use of such hierarchies is transitive closure reasoning. In this paper, we have discussed the need for fast transitive closure reasoning in combined relational hierarchies. We have shown a representation and algorithms that allow correct, fast reasoning in combined relational hierarchies. The representation relies on a pointer-free materialized transitive closure which is adapted to a parallel high-performance computer. We have shown how to combine IS-A, PART-OF, and Contained-in relations, using a relation type, with the Double Strand Representation, and we have introduced correct propagation rules for relation types. Our maximally reduced tree cover requires fewer propagated pairs than other previously published materialized transitive closures. Our reasoning algorithms have been tested with a large medical vocabulary that contains IS-A, PART-OF, and Contained-in links. Our analysis and experiments have shown that combined transitivity reasoning can be performed in constant time, assuming constant processor space.

Acknowledgements. This work was conducted using the computational resources of the Northeast Parallel Architectures Center (NPAC) at Syracuse University, which is funded by and operates under contract to DARPA and the Air Force Systems Command, Rome Air Development Center (RADC), Griffiss Air Force Base, NY, under contract# F306002-88-C-0031. This research was (partially) done under a cooperative agreement between the National Institute of Standards and Technology Advanced Technology Program (under the HIIT contract, number 70NANB5H1011) and the Healthcare Open Systems and Trials, Inc. consortium. Jim Geller worked on this paper while he was on Sabbatical at the NEC Research Institute in Princeton, NJ. Generous financial and intellectual support by Dave Waltz is gratefully acknowledged.

References

Agrawal R, Borgida A, Jagadish HV (1989) Efficient management of transitive relationships in large data and knowledge bases. In Proceedings of the ACM SIGMOD international conference on the management of data, May–June 1989, Portland, Oregon, Association for Computing Machinery, pp 253–262

Agrawal R, Dar S, Jagadish HV (1990) Direct transitive closure algorithms: design and performance evaluation. ACM Transactions on Database Systems 15(3): 428–458

Ait-Kaci H, Boyer R, Lincoln P, Nasr R (1989) Efficient implementation of lattice operations. In Proceedings of the ACM transactions on programming languages and systems 11(1): 115–146

Barnden JA, Srinivas K (1992) Overcoming rule-based rigidity and connectionist limitations through massively-parallel case-based reasoning. International Journal of Man–Machine Studies 36(2): 221–246

Carino F Jr, Jahnke M (1998) Bank of America case study: the information currency advantage. In Proceedings of 24th annual international conference on very large data bases, San Francisco, CA, pp 641–644

Chu W, Hua Y, Kuorong C, Minock M, Chow G, Larson C (1996) CoBase: a scalable and extensible cooperative information system. Journal of Intelligent Information Systems 6(2–3): 223–259

Cimino JJ, Clayton PD, Hripcsak G, Johnson SB (1994) Knowledge-based approaches to the maintenance of a large controlled medical terminology. Journal of the American Medical Informatics Association 1(1): 35–50

Cohen PR, Loiselle CL (1988) Beyond ISA: structures for plausible inference in semantic networks. In Proceedings of AAAI-88, pp 415–420

Ebert J (1981) A sensitive transitive closure algorithm. Information Processing Letters 12: 255–258

Evett MP, Andersen WA, Hendler JA (1993) Massively parallel support for efficient knowledge representation. In Proceedings of the 13th international joint conference on artificial intelligence. Morgan Kaufmann, San Mateo, CA, pp 1325–1330

Farroha B, Deshmukh R (1995) A novel approach to design a massively parallel application specific architecture for image recognition systems. In Proceedings of IEEE southeastcon '95, New York, pp 292–299

Feldman J (1982) Dynamic connections in neural networks. Biological Cybernetics 46: 27–39

Fischer MJ, Meyer AR (1971) Boolean matrix multiplication and transitive closure. Conference record 12th annual symposium on switching and automata theory, October 1971, East Lansing, Michigan, IEEE, pp 129–131

Ganguly D, Mohan C, Ranka S (1994) A space-and-time efficient coding algorithm for lattice computations. IEEE Transactions on Knowledge and Data Engineering 6(5): 819–829

Geffner S, Agrawal D, Abbadi AE, Smith T (1999) Browsing large digital library collections using classification Hierarchies. In Proceedings of the eighth international conference on information knowledge management (CIKM99), Kansas City, MO, pp 195–201

Geller J (1993) Massively parallel knowledge representation. AAAI spring symposium series working notes: innovative applications of massive parallelism, Stanford University, pp 90–97

Greiner R, Grove A, Roth D (1996) Learning active classifiers. In Proceedings of the thirteenth international conference (ICML '96), San Francisco, CA, pp 207–215

Halper M, Geller J, Perl Y (1992) An OODB 'Part' relationship model. In Proceedings of the first international conference on information and knowledge management, Baltimore, MD, pp 602–611

Halper M, Geller J, Perl Y (1993) Value propagation in object-oriented database part hierarchies. In Proceedings of the 2nd international conference on information and knowledge management, Washington, DC, pp 606–614

Halper M, Geller J, Klas W (1994) Integrating a part relationship into an open OODB system using Metaclasses. In Proceedings of the 3rd international conference on information and knowledge management, Gaithersburg, pp 10–17

Halper M, Geller J, Perl Y (1998) An OODB part-whole model: semantics, notation, and implementation. Data & Knowledge Engineering 27(1): 59–95

Han J, Fu Y, Wang W, Chiang J, Gong W, Koperski K, Li D, Lu Y, Rajan A, Stefanovic N, Xia B, Zaiane O (1996) DBMiner: a system for mining knowledge in large relational databases. In Proceedings of second international conference on knowledge discovery and data mining, Menlo Park, CA, pp 250–255

Han J, Ng RT, Fu Y, Dao SK (1998) Dealing with semantic heterogeneity by generalization-based data mining techniques. In Papazoglou MP, Schlageter G (eds). Cooperative information systems: current trends and directions. Academic Press, Netherlands, pp 207–231

Han Y, Lee Y (2000) Parallel computation for managing transitive relations knowledge bases. In Proceedings of the twelfth IASTED international conference parallel and distributed computing and systems, November 2000, Las Vegas, NV, ACTA Press, pp 37–43

Harabagiu S, Moldovan D (1998) A parallel system for text inference using marker propagations evaluation. IEEE Transactions on Parallel & Distributed Systems 9(8): 729–747

Harnad S (1989) Minds, machines and Searle. Journal of Theoretical and Experimental Artificial Intelligence 1: 5-25

Hendler J, Stoffel K (1999) Back-end technology for high-performance knowledge-representation systems. IEEE Intelligent Systems 14(3): 53–69

Hernandez M, Stolfo S (1998) Real-world data is dirty: data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery 2(1): 9–37

Hillis WD (1985) The connection machine. MIT Press, Cambridge, MA

Hillis WD (1988) Intelligence as an emergent behavior; or: the songs of Eden. In Proceedings of the American academy of arts and sciences (Daedalus) 117(1): 175–189

Horrocks I, Sattler U (1999) A description logic with transitive and inverse roles and role hierarchies. Journal of Logic & Computation 9(3): 385–410

Ioannidis YE, Ramakrishnan R, Winger L (1993) Transitive closure algorithms based on graph traversal. ACM Transactions on Database Systems 18(3): 512–576

John GH, Langley P (1996) Static versus dynamic sampling for data mining. In Proceedings of second international conference on knowledge discovery and data mining (KDD-96), August 1996, Portland, Oregon, AAAI Press

Joslyn C, Rocha L (1998) Towards a formal taxonomy of hybrid uncertainty representations. Information Sciences 110(3–4): 255–277

Kasparov G (1997) Einsteins muskel. Der Spiegel 18: 222–224

Kitano H, Higuchi T (1991) High performance memory-based translation on IXM2 massively parallel associative memory processor. In Proceedings of IJCAI, Sydney, Australia, pp 149–154

Lee EY, Geller J (1993) Representing transitive relationships with parallel node sets. In Bhargava B (ed). In Proceedings of the IEEE workshop on advances in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos, CA, pp 140–145

Lee EY, Geller J (1995) Parallel operations on class hierarchies with double strand representations. IJCAI-95 workshop program working notes, Montreal, Quebec, pp 132–142

Lee EY, Geller J (1996a) Constant time inheritance with parallel tree covers. In Proceedings of the Florida AI Research Symposium (FLAIRS), Key West, FL, pp 243–250

Lee EY, Geller J (1996b) Parallel transitive reasoning in mixed relational hierarchy. In Proceedings of the conference on knowledge representation and reasoning, Cambridge, MA, pp 576–587

Lee EY (1997) Massively parallel reasoning in transitive relationship hierarchies. CIS Department, New Jersey Institute of Technology

Munro I (1971) Efficient determination of the transitive closure of a directed graph. Information Processing Letters 1(2): 56–58

Nuutila E (1994) Efficient transitive closure computation in large digraphs. PhD thesis, Mathematics and Computing in Engineering, Helsinki University of Technology

Plexousakis D (1993) Integrity constraint and rule maintenance in temporal deductive knowledge bases. In Proceedings of the 19th international conference on very large databases, Palo Alto, CA, pp 146–157

Quillian MR (1968) Semantic memory. In Minsky ML (ed). Semantic information processing. MIT Press, Cambridge, MA, pp 227–270

Rector AL, Bechhofer S, Goble CA, Horrocks I, Nowlan WA, Solomon WD (1997) The GRAIL concept modeling language for medical terminology. Artificial Intelligence Medicine 9(2): 139–171

Rector A, Zanstra P, Solomon W, Rogers J, Baud R, Ceusters W, Claassen W, Kirby J, Rodrigues J, Mori A, Van der Haring E, Wagner J (1998) Reconciling users' needs and formal requirements: issues in developing a reusable ontology for medicine. IEEE Transactions on Information Technology in Biomedicine 2(4): 229–242

Russel S, Norvig P (1995) Artificial intelligence: a modern approach. Prentice-Hall, Englewood Cliffs, NJ

Schmitz L (1983) An improved transitive closure algorithm. Computing 30: 359-371

Schubert LK (1979) Problems with parts. In Proceedings of the 6th international joint conference on artificial intelligence. Morgan Kaufmann, Los Altos, CA, pp 778–784

Schubert LK, Papalaskaris MA, Taugher J (1983) Determining type part, color, and time relationships. Computer 16(10): 53–60

Schubert LK, Papalaskaris MA, Taugher J (1987) Accelerating deductive inference: special methods for taxonomies colors and times. In Cercone N, McCalla G (eds). The knowledge frontier. Springer, New York, pp 187–220

Schwartz D (1999) Intelligent Indexes with Inheritable Fuzzy Relations. In Proceedings of 18th international conference of the north American fuzzy information processing Society, Piscataway, NJ, pp 371–375

Schwartz JT (1988) The new connectionism: developing relationships between neuroscience and artificial intelligence. In Proceedings of the American Academy of Arts and Sciences (Daedalus) 117(1): 123–141

Shafer JC, Agrawal R (1997) Parallel algorithms for high-dimensional proximity joins. In Proceedings of the 23rd international conference on very large databases, San Francisco, CA, pp 176–185

Shastri L (1988) Semantic networks: an evidential formalization and its connectionist realization. Morgan Kaufmann, San Mateo, CA

Sipper M (1998) Simple + parallel + local = cellular computing. Parallel Problem Solving from Nature – PPSN V. In Proceedings of 5th international conference, Amsterdam, Netherlands

Sipper M (1999) The emergence of cellular computing. Computer 32(7): 18–26

Song Y, Chou C, Min Y (1999) Large-scale economic dispatch by artificial ant colony search algorithms. Electric Machines & Power Systems 27(7): 679–690

Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In Proceedings of the fifth international conference on extending database technology, Berlin, Germany, pp 3–17

Srikant R, Agrawal R. (1997) Mining generalized association rules. Future Generations Computer Systems 13(2-3): 161–180

Stoffel K, Taylor M, Hendler J (1997) Efficient management of very large ontologies. In Proceedings of American Association for Artificial Intelligence conference (AAAI-97), July 1997, Providence, Rhode Island, AAAI/MIT Press

Tarjan R (1972) Depth first search and linear graph algorithms. SIAM Journal on Computing 1(2): 146–160

Taylor MG, Stoffel K, Hendler JA (1997) Ontology-based induction of high level classification rules. In Proceedings of SIGMOD data mining and knowledge discovery workshop, Tuscon, AZ

TMC (Thinking Machines Corporation) (1988) *LISP reference manual version 5.0 edition. Thinking Machines Corporation, Cambridge, MA

Thorpe S, Fize D, Marlot C (1996) Speed of processing in the human visual system. Nature 381(6582): 520–522

Turing AM (1950) Computing machinery and intelligence. Mind 59(236): 433–460

Van Bommel MF, Beck TJ (1999) Incremental encoding of multiple inheritance hierarchies. In Proceedings of the eight international conference on information knowledge management (CIKM99), Kansas City, MO, pp 507–513

Waltz DL (1988) The prospects for building truly intelligent machines. In Proceedings of the American Academy of Arts and Sciences (Daedalus) 117(1): 191–212

Waltz DL, Stanfill C (1988) Artificial intelligence related research on the connection machine. In Proceedings of the international conference on fifth generation computer Systems, Tokyo, Institute for New Generation Computer Technology (ICOT), pp 1010–1024

Waltz DL (1990) Massively parallel AI. In Proceedings of the eighth national conference on artificial intelligence. Morgan Kaufmann, San Mateo, CA, pp 1117–1122

Wang W, Iyengar S, Patnaik LM (1989) Memory-based reasoning approach for pattern recognition of binary images. Pattern Recognition 22(5): 505–518

Warshall A (1962) A theorem of Boolean matrices. Journal of the ACM 9(1): 11–12

Winston ME, Chaffin R, Herrmann D (1987) A taxonomy of part–whole relations. Cognitive Science 11(4): 417–444

Zupan B, Bohanec M, Bratko I, Cestnik B (1997) A dataset decomposition approach to data mining and machine discovery. In Proceedings of the third international conference on knowledge discovery and data mining, Zurich, Switzerland, pp 299–303

Author Biographies

Yugyung Lee is Assistant Professor at the University of Missouri at Kansas City. She received a B.S. degree in Computer Science from the University of Washington in 1990 and a Ph.D. degree in Computer and Information Sciences from the New Jersey Institute of Technology in 1997. Before joining the UMKC, she was working at MCC as a research scientist. Her research interests include high-performance data mining, distributed computing, intelligent agents, agent-based computing and architectures, knowledge representation and reasoning, Web mining and E-Commerce.

James Geller received his M.S. and Ph.D. degrees at the CS Department of the State University of New York at Buffalo. He is currently Professor at the CIS Department of the New Jersey Institute of Technology and Vice Chair of the M.S. and Ph.D. programs in Biomedical Informatics. His research interests include parallel processing for AI, ontologies, semantic networks, knowledge representation, medical vocabularies, object-oriented modeling, Web mining and genomic databases.

Correspondence and offprint requests to: Yugyung Lee, Computer Science Telecommunications, University of Missouri at Kansas City, Kansas City, MO 64110-2499, USA. Email: yugi@cstp.umkc.edu