The Yale Artificial Intelligence Project: A Brief History

Stephen Slade

This overview of the Yale Artificial Intelligence Project serves as an introduction to Scientific Datalink's microfiche publication of Yale AI Technical Reports

An AI lab is like a greenhouse. Researchers develop new ideas and plant them in programs. The programs are cultivated, hybridized, nurtured. The weaker ideas die out. The stronger ideas are grafted onto new stock and serve as the basis of hearty new strains.

At Yale, there has been a traditional summer seminar series at which graduate students present their unprepossessing theories to the vocal and critical review of their colleagues. The tenor of these discussions is conveyed by their title: "The Friday Fights." Such occasions provide the Yale researcher opportunities for both pruning and growth. Cultivation by candor is the standard. This level of peer review has also been experienced by colloquium speakers. Many visitors to the lab were unprepared for the onslaught. By now though, Yale's reputation for open debate has led many speakers to agree to disagree.

Ideas and theories grow through this process of natural selection. The best are represented here, in this collection of technical reports from the past dozen years. These reports are our harvest—the results of twelve intense years of work by a large crop of researchers.

In the present article, we survey this collection, which is now made available on microfiche through Scientific Datalink. The work falls into several areas.

• *Cognitive Modelling* This category is quite broad, and is used here to describe the work of Roger Schank and his students. It includes natural language processing, models of human memory organization, learning, and explanation.

• *Spatial and Temporal Reasoning.* Drew McDermott and his students have considered a range of topics related to reasoning including nonmonotonic logic, planning with incomplete knowledge, and reasoning about space and time.

• *Cognition and Programming.* Elliot Soloway, and David Barstow before him, have undertaken a scientific inquiry focused on the task of programming itself. How could a computer automatically write programs? How do people learn to program? How can a computer program help people learn to write better programs?

• *Cognitive Science* All the work at the Yale AI Project could be termed part of cognitive science. We have had a long association with members of the Yale Psychology Department, especially Robert Abelson and John Black. Psychologists—faculty, visitors, graduate students—have actively contributed to the AI research efforts, testing and refining theories of human cognitive processing.

Cognitive Modelling

The Yale AI Project began in 1974 when Roger Schank and Chris Riesbeck came from the Stanford AI Lab, via the Istituto per gli Studi Semantici e Cognitivi in Castagnola, Switzerland, to join the Yale Computer Science Department.

The faculty at Yale, especially Alan Perlis and Martin Schultz, were very supportive of the establishment of an AI lab. Robert Abelson, in the Psychology Department, had a longstanding interest in AI and had already begun collaboration with Schank Graduate students were quickly drawn into the work. Jim Meehan, Wendy Lehnert, Rich Cullingford, and Gerry DeJong were among the first students involved.

The main research tool was a DEC PDP-10, equipped with custom-built Sugarman CRT's and Yale's own full-screen "E" editor, courtesy of Ned Irons.

The focus of the initial research was natural language processing. At Stanford, Schank had developed the MARGIE system (Schank 1975) with his students Goldman, Rieger, and Riesbeck. MARGIE was used to demonstrate the effectiveness of *conceptual dependency* (CD) as a language-free, canonical meaning representation MARGIE would read an English sentence, using Riesbeck's expectation-based parser to build a CD form which represented the meaning of the sentence MARGIE would then make inferences based on the meaning of the input sentence using Rieger's inferencing program. Finally, the results of the initial parse, as well as the inferences, could be converted back into natural language with Goldman's generator program, producing paraphrases in either English or German.

The MARGIE program, though considered a toy system, was an effective and productive model for the ensuing research projects. There were several salient characteristics of MARGIE that have developed as themes in Yale cognitive modelling research through to the present.

• **Task Orientation.** An AI program should address a specific, real-world task. The program should model something that a person actually does, rather than an artificial abstraction of intelligent behavior. MARGIE's tasks included reading, paraphrase, and translation. Subsequent programs have created stories, answered questions, summarized stories, skimmed stories, professed opinions, related old stories to new ones, and engaged in conversations. There are several reasons for choosing real tasks. First, these tasks are in the realm of the possible—people provide an existence proof. Second, the researcher has tangible ways of assessing the results of the program through comparisons with human performance Third, the researcher has a ready supply of data. It is preferable for a program to use real data instead of canned examples. In the latter case, even the most objective researcher may find himself tailoring the examples to just those cases which he knows the program can handle. Finally, the experimental paradigm implicit in this approach requires the researcher to build an actual computer program. The program is the crucible in which theories are tested and molded. Without a program, many of the unstated suppositions in a theory are never revealed or examined. In writing a program, the researcher must confront these assumptions.

• **Psychological Process Model.** The MARGIE program was a cognitive simulation. Not only did it try to perform tasks that people perform, but it tried to simulate the manner in which the human mind works. By comparison, a computer chess program which exhaustively searches ahead several moves may be able to play a fine game of chess, but it is unrealistic to consider such a program a model of the way in which a person plays chess. The underlying process model in MARGIE comprised three stages: parsing, inferencing, and generation. This basic triad has been the foundation for the subsequent generations of programs. The primary focus has been on integrating the three processes to allow more interaction with memory. In recent years, the role of memory has transcended the specific natural language concerns and has come to encompass learning and explanation processes. This development can be viewed as a natural evolution of the original central inference process.

• **Canonical Representation of Knowledge.** The heart of the MARGIE system was the conceptual dependency knowledge representation system. CD provided a means of representing actions and states in a canonical, language independent fashion. A concept represented using the dozen CD primitives might be expressed in any number of ways in any number of languages. More specifically, CD addressed a broad range of problems associated with meaning in language.

Translation, Synonymy and Paraphrase CD insured an identical representation for two different sentences having the same meaning.

Inference. CD included a process model for determining the implicit meaning of a sentence.

Ambiguity. The same word can have a variety of meanings. ("John gave Mary a kiss." versus "John gave Mary a book.") The CD paradigm provided a means of distinguishing among multiple word senses.

As the domain of concepts expanded in the subsequent years, new types of knowledge representations were developed These included primitives for social acts, attitudes, and objects, as well as larger knowledge structures built from these primitives, such as scripts, plans, goals, memory organization packets (MOPs), thematic organization packets (TOPs), and explanation patterns (XPs).

One additional feature of the MARGIE system that carried over to Yale researchers was the habit of giving programs names of people. From SAM and ELI through BORIS and CYRUS, this convention has been frequently adopted.

Scripts, Plans, Goals, and Understanding

The Yale AI research reports begin with SAM, the Script Applier Mechanism, the first computer program to understand stories in context. MARGIE had been able to understand simple sentences in terms of the actions or states which they represented. However, connected text—a story—could cause a problem if the series of actions and states could not be connected through simple inferences. How can a program infer context? Two brief stories illustrate the problem.

> John picked up a rock. John threw the rock at Mary. The rock hit Mary.

In this first case, the actions are causally linked. A reader can infer the connections from one action to another and build a *causal chain* based on the inferences associated with each action. For example, picking up an object can enable a person to propel that object.

> John went to a restaurant. He ordered a lobster. He paid the check and left.

This second story demonstrates that some causal chains are derived from context. In this case, the reader can infer a lot about what John did at the restaurant simply because the reader knows a lot about restaurants. For example, we presume that John ate something—probably a lobster. The story, though, does not mention eating at all. However, the reader knows that people usually go to restaurants to eat. This type of knowledge about stereotypical events has been termed a *script* Scripts were originally proposed by Schank and Abelson (1975, 1977) for story understanding and are similar to the frame knowledge structure proposed by Minsky (1975).

Scripts comprise a set of roles, props, goals, locations, and events. In the restaurant script, notated as $RESTAURANT, the roles might include customer, waitress, and cook; the props could be a menu, table, and silverware; the locations could be the bar, dining area, and kitchen; and the events would include arriving, seating, ordering, and so forth. Other script-based situations would be taking a plane, attending a concert, buying a car, or going to the dentist. These are situations in which people have sets of expectations based on numerous previous episodes. The constituent events of these episodes are sufficiently similar to allow a person (or a program) to make inferences when details are not explicitly stated.

SAM was a test of the script knowledge structure. After development using simple stories about restaurants, SAM was applied to real newspaper stories using scripts describing events such as automobile accidents. Like MARGIE, the SAM project was a group effort. SAM's parser was ELI (English Language Interpreter), which was a revised version of Riesbeck's original parser. Anatole Gershman developed a method for parsing complex noun-groups (for example, "Frank Miller, 32, of 593 Foxon Rd., the driver,"). Rich Cullingford built the actual script applier which was the core of the system. Wendy Lehnert developed a conceptually-based question-answering system that allowed the program to respond to queries about the content of the stories. Rick Granger built a parsing module that could infer the meaning of unknown words from context.

SAM had a small repertoire of scripts, and would read news stories in great detail, looking for every bit of meaning it could find. Another approach to reading news stories was explored by Gerald DeJong in FRUMP (Fast Reading Understanding Memory Program). FRUMP was connected directly to the United Press International news wire and could skim news stories in dozens of different domains, and produce summaries in several languages. On the DEC-20 (which by 1978 had replaced the PDP-10), FRUMP could process an average news story in under ten seconds. FRUMP's domain knowledge was represented in sketchy scripts that lacked the detail of SAM's scripts, but provided a feasible method for capturing the salient details of news stories. FRUMP's world knowledge comprised a range of news events including natural disasters, such as floods and earthquakes, international incidents, such as breaking diplomatic relations or armed conflict, and deaths of famous people.

It is important to note that the scripts in FRUMP and SAM were not triggered by keywords like "earthquake" or "death," but by the concepts that embody the underlying meaning. This point can be illustrated by an error FRUMP made in processing a certain story. From a UPI wire dispatch, FRUMP produced the summary "There was an earthquake in San Francisco. Two people were killed." This summary initially appeared plausible, but FRUMP usually provided more details for an earthquake story, such as the severity of the earthquake and the amount of damage reported. As it happened, the original news report was not about earthquakes at all, but concerned the tragic shooting death of the Mayor of San Francisco. FRUMP had taken the lead sentence literally: "San Francisco was shaken by the death of Mayor Moscone and City Councilman Harvey Milk." The figurative use of language remains an open question for researchers in natural language processing.

In addition to looking at scripts, Yale researchers explored intentionality One of the earliest programs to embody goals and plans within the CD paradigm was Jim Meehan's TALESPIN, which made up stories similar to the fables of Aesop. The program would start with a set of characters who wanted to achieve certain goals. The story would be a narration of the characters' attempts at executing plans to satisfy their goals. Many of the unsuccessful stories are striking in exposing inferences that the program had been unable to make.

> One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe threatened to hit Irving if he didn't tell him where some honey was.

Here, the program did not know that it could infer the location of an object from the location of the container of that object. Once this piece of knowledge was added, the program tried again.

> One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe walked to the oak tree. He ate the beehive.

Examples such as these serve to increase an AI researcher's respect for the complexity and diversity of com-

An AI program should address a specific, real-world task. The program should model something that a person actually does, rather than an artificial abstraction of intelligent behavior.

monsense knowledge.

Goals and plans received a different treatment in Jaime Carbonell's POLITICS program, which modelled subjective beliefs using a hierarchy of goals. Thc program would read a news headline and interpret it from one of two opposing perspectives: conservative and liberal. The program would reason about its opponent's behavior (in this case, the Soviet Union) based on a model of the goals of the opponent. In the event of competing goals, the program would use *counterplanning* to block an opponent's goal, or to make sure that its own goals were not blocked by the opponent.

The domain of political judgement in POLITICS led to the development of a new set of semantic primitives to represent institutional actions. These social acts could capture the difference between "John gave Mary a book" and "The policeman gave Mary a ticket." The first case is a simple transfer of possession, while the second embodies an institutional act based on the implicit authority of the polity for which the policeman is an agent. The social acts were designed to represent institutional actions and mediated disputes.

Two other domains were the subject of new semantic primitives: objects and attitudes Wendy Lehnert with Mark Burstein developed a set of primitives to capture inferences about physical objects, such as bottles, pens, sponges, umbrellas, and shopping carts. Object representations allow the inference of default information such as location, associated scripts, and relations. Object primitives embody a psychological approach to the problems of naive physics.

The attitudes primitives attempted to represent a number of dimensions of attitudes including fondness-antipathy, fascination-disinterest, fear-security, attraction-repulsion, jealousy-concern, irritation-comfort, respect-disdain, and trust-distrust. These semantic primitives provided a basis for making inferences about interpersonal behavior.

At this point, it should be apparent that a wide range of problems were under investigation. The next stage of development involved both exposition and synthesis. Yale researchers had produced a number of theories and techniques that could be adopted by others outside Yale. Within the Yale AI lab, the new challenge was to combine the various knowledge structures and theories in an integrated system.

In the summer of 1978, Yale was the site for a workshop in cognitive science. Psychologists, linguists, philosophers, and anthropologists convened in New Haven for four weeks of AI, programming, CD, scripts, and theoretical cross-pollination. Most of these researchers had little programming background Chris Riesbeck and Gene Charniak (a frequent visitor at Yale) developed "micro" versions of SAM and ELI that made those programs more accessible to neophyte program

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe walked to the oak tree. He ate the beehive.

This pedagogical approach was later expanded into a book which included five Yale AI programs (Schank and Riesbeck 1981).

Meanwhile, Rick Granger, and later Mike Dyer, addressed the synthesis problem. BORIS (Better Organized Reading and Inference System) was an effort to combine disparate knowledge types including CD actions, scripts, plans, goals, interpersonal relations, role themes, and affect. Dyer's program could read stories in great depth—making numerous inferences and tying together various pieces of information The domain of Dyer's BORIS was melodramatic divorce stories, such as one might encounter on a soap opera.

These various programs reflected the natural language processing paradigm that began with MARGIE. The role of syntax was secondary to the problems of meaning representation. Larry Birnbaum provided arguments for the roles that meaning and world knowledge must play in language tasks, pointing out the problems associated with the "syntax module" approach to language processing. Carbonell, Cullingford, and Gershman discussed the crucial role of meaning in machine translation. Later, Steve Lytinen's MOPTRANS program embodied an approach to machine translation which integrates semantics and syntax in a psychologically motivated fashion.

Learning, Memory, and Explanation

The work on conceptual dependency, scripts, plans, and goals demonstrated the importance of knowledge representation for cognitive modelling tasks. It was clear that people relied on a considerable amount of knowledge about the world. It was also apparent that much of this knowledge was not innate. People acquire knowledge. To get computers to simulate human cognitive behavior, the programs would have to learn as well.

One of the first Yale learning programs was developed by Mallory Selfridge. His program modelled the learning of language during a child's second 12 months, that is, between the ages of one and two. At this stage, a child presumably understands certain basic concepts such as specific physical objects, movement, eating, and so forth. In learning language, the child develops a mapping between sounds—the words—and these concepts. Selfridge's program was based on extensive protocols, and concentrated on learning the meaning of words and word sequences, rather than syntax rules. Selfridge demonstrated the generality of his system by having it learn not only English, but Japanese as well.

Beyond the specific task of language acquisition, there lies the general problem of learning. The issue of learning can be seen from many perspectives. The computational metaphor leads one to view learning as a matter of updating memory. Several questions arise.

- How is human memory organized?
- How are new events assimilated into memory?
- How are memories indexed for retrieval?

Given the psychological claims made for the AI knowledge structures, it was important that the predictions made by the theories reflect empirical findings In an experiment to explore scripts, the psychologists Bower, Black, and Turner (1979) tested subjects who had read script-based stories. The results demonstrated that subjects would confuse elements of different stories if the underlying events were similar For example, a visit to a doctor's office is similar to going to the dentist This confusion suggested that there were not in fact distinct, mutually exclusive scripts $DOCTOR-VISIT and $DENTIST-VISIT.

The theory needed revision to account for this data. Schank proposed a generalization of the script notion: a higher-level knowledge structure that organized smaller components. This higher-order script was termed a Memory Organization Packet, or MOP. Thus, there could be a MOP for PROFESSIONAL-OFFICE-VISIT which could have complex and variable sets and orders of scenes This hierarchical view allowed greater flexibility. Each scene component would have less variability, but might be shared among several higher-level knowledge structures In the PROFESSIONAL-OFFICE-VISIT MOP, there would be common components such as making an appointment, going to the office, sitting in the waiting room, and paying the bill. Only the dentist version of this MOP would have a scene for tooth extraction.

MOPs provide an architecture for human memory of episodes, and suggest an explicit mechanism for the process of reminding. However, it was apparent that people often relate events that have little surface similarity, but share an underlying goal structure. Schank proposed Thematic Organizational Packets or TOPs to provide a goal-based indexing mechanism for human memory. These topics are explored in depth in Schank (1982).

An early application of MOPs was Janet Kolodner's program CYRUS (Computerized Yale Retrieval and Updating System). CYRUS represents one of the first attempts to model the organization of episodic human memory CYRUS stored and retrieved episodes in the lives of Secretaries of State Cyrus Vance and Edmund Muskie. When new events were added to its memory, CYRUS integrated them with the events it already knows about. CYRUS answered questions using memory search strategies based on reconstructive memory processes

How could story understanding programs benefit from these theories? The FRUMP program provides an illustration If FRUMP were given the same story to read a dozen times in a row, it would process the story exactly the same way each time, and produce the same set of summaries This repetitive behavior should not be surprising from a computer, but it would be most unusual for a person. We would expect the person's reaction to change over time In particular, the person should remember having already seen the story FRUMP's problem was that it could not remember what it had read.

Michael Lebowitz' IPP program applied MOPs to the problem of reading newspaper stories IPP (Integrated Partial Parsing) had two thrusts. The first was a parsing strategy that allowed the program to focus its attention on the interesting words or phrases, and skip the dull ones. IPP's domain of terrorism had a great deal of intrinsic interest

The second and most significant aspect of IPP was its ability to remember stories that it had read and relate them to new episodes Using this MOP-based mechanism, IPP could detect similarities among stories and arrive at generalizations. For example, IPP would notice that the victims of terrorist acts in Northern Ireland were establishment, authority figures, such as policemen or soldiers, and that the terrorists were members of the IRA. When IPP subsequently read a story about a policeman being shot in Northern Ireland by an unidentified gunman, IPP could infer that the gunman was a member of the IRA.

Another broad area of application for episodic memory is the area of expert systems The central feature of *expertise is experience* An expert is someone who has vast, specialized experience, who has witnessed numerous cases in the domain, and who has generalized this experience to apply it to new situations When confronted with a problem, the expert is reminded of previous, similar problems and their respective resolutions. It may be that the expert has so many exemplars for a given problem that the experiences have been distilled into a general rule to be applied.

In the production system paradigm, the rule is hard-wired into the system. If a rule fails, the system generally requires human intervention to revise the rule. This makes the system more fragile and less robust

Furthermore, most of the knowledge in the system is written strictly in terms of the domain. While it is certainly important for the system to have much domain-specific knowledge, it is also true that people have the valuable ability to generalize their knowledge across domains. That is, they can apply general principles acquired in one setting to a situation involving quite different specific knowledge While the production-system approach offers a general paradigm for expert systems, it does not provide a general mechanism for applying knowledge from one domain to another area of expertise.

A memory-based model of human expertise can be contrasted with the rule-based approach.

- Psychologically valid. People don't become experts simply by being told the rules. They become experts by extracting the rules from experience
- Tolerant of rule-failure. Since the rules of the system are derived from experience, a new experience which violates a rule would be assimilated to modify the rule.
- Generalizable across domains. The general mechanism of developing expertise through experience allows the system to be applied directly to new domains.

An expert system that can extract information from its experience will be able to grow and acquire knowledge on its own This is a crucial step for the long-range success of the expert system concept in AI There are so many tasks to which automated reasoning power might be applied, that it is absolutely necessary to

develop a mechanism that can assimilate new knowledge directly from experience.

There are several complex research issues involved in developing this novel approach to expertise. These include the familiar questions of memory organization and indexing, and the underlying learning mechanism IPP can be viewed as a prototype of this approach.

IPP's learning was based primarily on noticing similarities between events. This similarity-based learning is clearly part of human cognitive processing, but it does not account for the human ability to reject some similarities as mere coincidences while labelling others as significant.

Schank proposed that learning is triggered by *expectation-failure* That is, when we observe a discrepancy between our predictions and some event, we then have something to learn. We need to revise our knowledge structure. The mechanism for updating our knowledge often requires explanation. Schank suggests that explanation plays a central role in learning and intelligence (Schank 1986). He proposes an explicit knowledge structure, *explanation patterns* (XPs), that are used to generate, index, and test explanations.

The theories of failure-driven learning and explanation are being explored in several domains, including economics, law, and cooking. Chris Riesbeck, together with Jim Spohrer and Charles Martin, have developed a program in the domain of political economics. The program begins with a novice view of economic questions, and becomes increasingly knowledgeable through reading the opinions of experts, such as Milton Friedman and Lester Thurow.

William Bain has applied the memory-based approach to legal reasoning. Bain observed several lawyers and judges in the context of sentencing convicted criminals. His program, JUDGE, simulates the process of a judge deciding the appropriate sentence to mete out, based on the judge's experience.

Kristian Hammond's CHEF program has the task of solving problems by reusing plans for similar problems. The plans are recipes, and the problems are the creation of new dishes with specific ingredients. CHEF is able to reason about multiple, interacting goals, and furthermore, can learn more general planning heuristics in the process. When CHEF detects a failure in one of its recipes, it can ask itself questions to reason about the cause of the failure. This process of examining failures can generally lead to an explanation of the failure.

The central feature of expertise is experience.

Computer models of human learning provide insights into how people learn. Schank recognized that these theories can be applied to the teaching of children (Schank 1981). One major lesson learned from building computer programs that read stories is that children, even at the age of 3 or 4, have a tremendous amount of knowledge about the world. Computer programs have to be fed that knowledge in order to read. With children, that knowledge is a rich asset that the education process should exploit. In recent years, Schank and the author have applied AI perspectives to the construction of educational software for microcomputers. Together with Riesbeck and Soloway, we have begun to look at a wide range of issues to create effective educational programs that address the pressing needs of the schools.

In summary, Schank's work in cognitive modelling has three broad agendas. The first is scientific. What is the nature of the human mind? How do people remember, learn, and understand?

The second is technological. How can we build intelligent machines? How can we program computers to communicate in natural language, learn from experience, and explain anomalies?

The final goal is educational. How can the scientific and technological results be applied to primary and secondary education? How can microcomputers best be deployed in our schools?

Spatial and Temporal Reasoning

Drew McDermott came to Yale from MIT in 1976. At MIT, McDermott had worked with Gerald Sussman in developing the CONNIVER AI programming language (McDermott and Sussman 1973), and applied deductive techniques to problem solving and planning. Three themes of McDermott's early work have continued in his past ten years at Yale.

- *Deduction* McDermott has based much of his research on the underlying paradigm of logical deduction. His work with Doyle on non-monotonic logics is one extension to classical logic to address particular AI problems of plausible reasoning.
- *Embedded AI Languages.* The original work on CONNIVER has been extended with the Duck language. Practical issues of portability were the motivation for the NISP language used in implementing Duck.
- *Planning.* The intellectual testbed for McDermott's theories is problem solving in the physical world. McDermott and his students have explored a range of issues involved in reasoning about space and time including knowledge representations, planning strategies, and scheduling heuristics.

McDermott has termed his research domain *theoretical robotics* Just as Schank's work explores the implicit knowledge that people have about intentionality and social domains, McDermott tries to make explicit the knowledge that people have about space and time. A robot must be able to solve problems which require knowledge about spatial and temporal phenomena, and the ability to act in the absence of complete information.

Deduction and Duck

Many early AI programs adopted the robot planning metaphor, but usually in a restricted domain, such as the blocks world. In these cases, it was feasible to represent the complete world. That is, the robot would have

exact knowledge of the location, size, and shape of all relevant objects. It is now clear to most AI researchers that such complete world knowledge is not merely an experimental abstraction—it is delusional. Rarely if ever does a person have exact knowledge The blocks world thus finessed one of the toughest AI problems. reasoning under uncertainty.

The typical reasoning paradigm for the blocks world was deductive retrieval. In a traditional deductive logic system, every item in the database is true—either as an axiom or as a theorem derived through a valid inference process. The addition of facts always increases the number of facts in the database The size of the database is a monotonic increasing function. Deduction provides a tidy way to keep track of the state of a closed world.

When faced with uncertainty, a person or robot must make assumptions. These assumptions become part of the reasoning process. Non-monotonic logic derives its name from the significant property that the addition of a new fact to a database can result in a decrease in the size of the database. This curious state is due to the inclusion of assumptions in the database along with observed and proven facts. The addition of a new fact could result in previous assumptions becoming invalid and then deleted from the database. For example, if we are given the statement "John is married to Mary," we might infer a range of things about John including the following·

1. John is an adult male.
2 John has the usual anatomical structure.
3 John can walk, talk, and chew gum
4. John has a primary interest in the welfare of Mary and himself.

These inferences are based on default reasoning, and they are not facts. In particular, if we later are told "John died," we must revise our beliefs to remove any conclusions that we had inferred based on the assumption, directly or indirectly, that John was alive. Thus, learning of John's demise results in our removing conclusions from the database.

Non-monotonic logics comprise a range of logical assumptions, from the conceivable to the provable, from the arguable to the doubtless. The point is that people must rely on inferred information to understand the world around them Inference is the rule, not the exception. The problem of reasoning from incomplete information pervades disparate AI domains

McDermott implemented default reasoning in the context of a deductive retrieval language: Duck Duck has been the primary vehicle for McDermott's research, and has also been made available to researchers outside Yale, in both universities and industry.

Duck has several facets

- Duck is a logic programming language, similar to Prolog. However, Duck is a descendent of Hewitt's original PLANNER language (Hewitt 1969), which pioneered logic-based programming Duck maintains a relational database that supports predicate calculus deductions, using both forward and backward chaining.
- Duck is built on top of NISP (Nifty LISP)—McDermott's portable dialect of LISP that itself runs on top of other dialects of LISP, including Common LISP, Interlisp, Franz LISP, ZetaLISP, and Yale's own T (Slade 1987). NISP provides a range of features including a structure package, type declarations, stream-oriented I/O, closures, and a workspace manager. Duck has access to all the features of NISP.
- Duck provides a *reason maintenance system* which is a mechanism for plausible reasoning. In particular, Duck maintains justification links, or *data-dependencies*, for assertion-inference chains. If some fact is removed from the Duck database, then Duck can automatically remove all related beliefs that were derived based solely on that fact, and thus are no longer substantiated.
- Duck is a general-purpose AI programming language which is particularly suited to rule-based applications. Duck provides an interactive debugging environment, including a convenient way to integrate English language templates into the program. This means that Duck can explain its actions in a pseudo-English, instead of regurgitated code fragments or stack frames

Duck has been around for nearly a decade. It is a mature programming environment for both teaching and research—especially in domains that are rife with uncertainty. In recent years, Duck has been recognized outside academia as an effective tool for building expert systems.

When faced with uncertainty, a person or robot must make assumptions.

The Realms of the Unknown: Space and Time

The domains on which McDermott and his students have concentrated are space and time—an abundant source of reasoning problems. Spatial reasoning encompasses a range of problems. These include the following:

- *Naive physics.* How do physical forces affect physical objects? For example, suppose a robot accidentally drops a computer out of a third floor office window. Where will the computer end up and in what condition will it be?
- *Navigation.* How could a robot represent knowledge about a familiar terrain to allow it to plan a route? For example, in the previous situation, what is a reasonable path for the robot to take to get to the new location of the computer? Presumably, the robot should try a route other than that taken by the computer itself.
- *Design.* What physical properties of objects can be exploited to solve problems? What available containers could

our forlorn robot use to help carry the pieces of the computer back upstairs? An envelope? A Coke bottle? A desk drawer? A trash can?

Ernie Davis' program MERCATOR was a computational model of memory for spatial relations McDermott and Davis demonstrated that a simple Cartesian representation of space is not appropriate. In general, people do not know the precise location and extent of objects in the world Yet, people can nevertheless perform a range of spatial reasoning tasks based on a *cognitive map,* a conceptual representation of spatial relations

MERCATOR addressed a number of problems, including the following:

- *Inexact knowledge.* Many geographic facts are imprecise or incomplete MERCATOR used a fuzzy representation scheme to capture partial knowledge. Locations could be specified with great imprecision.
- *Retrieval* The program accessed a database of geographic knowledge to accomplish tasks such as finding routes and answering geographic queries. A question like "Is the Chinese restaurant within walking distance?" does not require calculating the exact distance involved.
- *Assimilation.* The program could learn new geographic facts. That is, the program would revise its cognitive map to be congruent with new sensory data This learning component was the primary focus of Davis' thesis

The domain of spatial relations is enormous. Davis argues that almost any physical problem includes spatial reasoning. Furthermore, many abstract problems benefit from spatial analogies. Space invades our thoughts

The companion of space is time Like space, time is pervasive and people reason about time without complete knowledge

Using default logic to reason about time can be revealing and problematic. Data-dependencies and temporal reasoning can have anomalous interactions. For example, in representing a state change that occurs over time, we may invoke a rule such as

Rule: If a person is eating, then his hunger decreases.

We might then read the story.

John was hungry. He started eating a hamburger. He finished lunch.

When we observe John eating, we infer that his hunger subsides. That is, John's diminished hunger is an assumption based on the fact that he is eating. However, once John stops eating, we must remove this assertion from the database. At this point, we can no longer infer a decrease in his hunger, so this fact is erased The system then concludes that John is still hungry after eating lunch.

What is needed to rectify this situation is some notion of effects that persist. Thorny issues of causality often appear in the guise of temporal problems. McDermott and his students have addressed these problems from a variety of perspectives.

McDermott devised a temporal logic for planning and problem solving. A computer program, the Forbin planner, was developed to explore issues of temporal reasoning in planning. Forbin builds plans top-down, expanding and analyzing the subtasks. Forbin may have many alternative plans in its library for accomplishing a given task It must decide which plan best achieves its goal. In this process, Forbin relies on two key components: a temporal database and a task scheduler.

The ordering of events in a plan can be represented using a *time map,* analogous to Davis' cognitive map, but with distinct properties. The cognitive map was a geographical database; the time map is a temporal database Tom Dean implemented a time map maintenance system to allow a planner to reason about persistent and transient states. A planner faces a dynamic world of shifting goals, deadlines, changes in the environment, and revised assumptions. Dean's program provides a methodology for reasoning about these kinds of temporal dimensions to planning.

Dean's temporal database is used by David Miller's task scheduler. A given plan may have a large number of steps which can be arranged in a number of orders. The question for the planner is What is the best order? The answer to that question depends on a variety of factors.

- *Preemption.* Can a task be broken up into parts that may be executed noncontiguously?
- *Resources* Do plan steps have overlapping resource requirements?
- *Precedence* Is a given step a prerequisite to another?
- *Delays* How soon can a given step be initiated?
- *Deadlines* How soon must a given step be completed?
- *Execution time* How long will it take to complete a given step?

Miller developed a representation and heuristics for scheduling combinations of plans, such as preparing dinner while washing clothes. The program recognizes which steps could be reordered or interleaved.

The work of McDermott and his students represents a long-term research program. These research issues are broad in scope. They can provide an illuminating perspective to many areas of artificial intelligence. McDermott has applied his vision and methodology in his introductory AI textbook, written with Charniak (Charniak and McDermott 1985).

Reading a computer program is like reading a story–almost.

Cognition and Programming

The one knowledge domain in which AI researchers feel most at home is computer programming. Many fields of computer science develop tools and techniques to make the construction of computer programs easier to learn, more accurate, quicker, and generally more efficient. AI research efforts have attempted to automate and model the process of programming.

David Barstow's work represents the automatic programming paradigm. Barstow's PECOS system is a knowledge-based coding expert. Developed at Stanford, PECOS takes a high-level

representation of an algorithm and converts it into a concrete implementation through a process of gradual refinement. PECOS's programming knowledge comprises a set of transformation rules. Barstow came to Yale after Stanford. He explored both the applicability of algorithm refinement to new problems and the contrasting paradigm of deduction-based theorem proving for program synthesis, before moving on to Schlumberger.

Following Barstow, Elliot Soloway arrived at Yale with a different approach to studying programming. Instead of a rule-based or logical methodology, Soloway's research is goal-based and psychological. Soloway established the Cognition and Programming Project (CAPP) which focuses on two themes:

• *AI and Software Engineering* Rather than prescribe how software should be designed and maintained, CAPP's approach is first to understand how experts (and non-experts) design, develop, debug, and maintain programs. Based on this understanding, one then is in a better position to design tools and make prescriptions.

• *AI and Education*· What should children be taught about programming? How should they be taught? AI offers some answers to these key educational questions. In particular, Soloway has focussed on the development of a curriculum for introductory programming, which should teach more than just the syntax and semantics of Pascal. More generally, Soloway is exploring the development of computer-based instructional systems than can deliver high-quality, individualized instruction.

AI and Software Engineering

The design, development, and maintenance of significant pieces of software are complex tasks. One aim of software engineering is the development of methodologies, languages, and tools that facilitate these tasks. There are several approaches to developing such methodologies or tools. For example, some researchers advocate a more formal approach to software: by providing a mathematical grounding for software, the claim is that the product will be more reliable, easier to maintain, and so forth. Soloway's approach, however, is to look less at software per se, but rather, focus on designers, programmers, and maintainers as they engage actively in building and maintaining software The claim, then, is that by understanding how programmers go about comprehending a program, one can pinpoint where they are having difficulties, and thus be in a position to design methodologies, languages, or tools that address the programmers' problems from a principled, cognitive viewpoint.

Soloway and his group have applied this cognitive approach to a number of issues in software.

• *Language Design.* CAPP has found empirical evidence that in a wide class of situations, Pascal's **while** construct is more difficult to use correctly than is Ada's loop construct.

• *Program Documentation*· Soloway and his students have identified specific types of knowledge (for example, static and causal knowledge) that maintainers need to abstract from documentation in order to carry out effective enhancements. CAPP then has gone on to prescribe a documentation format that enables maintainers easy access to these key types of knowledge.

• *Programming Instruction* Lewis Johnson of CAPP has constructed a system, PROUST, that can identify, for a class of moderately complex introductory programming assignments, the non-syntactic bugs in students' programs PROUST's performance is comparable to a human teaching assistant. Currently, Soloway's group is designing a curriculum for an introductory programming course, as described in the next section.

The particular theoretical perspective Soloway brings to the study of programmers can be summed up by saying. "Reading a computer program is like reading a story—almost." That is, Soloway has explored the notions of schema, goal, and plan that were developed in the story understanding world, and applied them to the programming world. Building on these notions, Soloway has developed fine-grained cognitive models of how programmers design, comprehend, and generate programs. In developing these models, CAPP has carried out traditional controlled-experiments, analyzed talking-aloud protocols, and constructed computer-based simulations. In sum, the theoretical aspects of this research push the frontiers of problem solving and text comprehension research, while practical payoffs for software engineering follow directly from the principled study of the programming process.

AI and Education

A commonly held belief is that teaching kids computer programming really teaches them some important problem solving strategies that transfer to other subject domains. Unfortunately, there is precious little evidence that supports this claim. Soloway's group is carrying out a number of studies in search of this elusive transfer effect. For example, they are looking to see if learning programming helps students to solve certain types of algebra word problems more effectively. Unless transfer can be shown, then programming might best be viewed as simply another job skill, along with drafting. In fact, this view is gaining acceptance in educational circles now. Soloway disputes that vocational view of programming, and is attempting to provide quantitative evidence of the elusive transfer effect.

One main reason why transfer hasn't been found can be traced to the current content of the vast majority of introductory programming courses. By and large, students are taught the syntax and semantics of a programming language. A quick look at the introductory programming textbooks will convince the reader of this claim. The tables of contents are typically organized in terms of the constructs of the language being taught. Based on studying thousands of programs—and bugs—generated by novice programmers, Soloway feels that language constructs are not the problem, but rather that students are having significant difficulty in "putting the pieces together," that is, composing and coordinating constructs, plans, and goals into a coherent whole. Moreover, based on what CAPP researchers have learned about what expert program-

mers know and about the strategies they employ, Soloway has identified a set of concepts that need to be taught explicitly. In teaching introductory programming, Soloway introduces concepts such as goals, plans, rules of programming discourse, problem simplification, simulation, reflection on past problem solving efforts, and top-down design. Studies are now underway to assess the effectiveness of this new curriculum.

Cognitive Science

Many of the questions posed by artificial intelligence researchers are not new or unique to AI. Other disciplines have examined the problems of language, mind, and cognition. In recent years, researchers from many circles have reached out across traditional academic boundaries to explore contrasting perspectives and paradigms. The intersection of these fields—psychology, philosophy, linguistics, anthropology, neuroscience, and AI—has become known as *cognitive science*

Yale researchers embrace this interdisciplinary approach. Over the years, we have invited visitors from disparate fields to spend time at Yale to share their views. They represent a wide spectrum, including George Lakoff, John Ross, James McCawley, Donald Norman, Joseph Weizenbaum, Hubert Dreyfus, John Searle, and Jerrold Katz.

The primary source of ongoing interdisciplinary research has been Yalc psychologists, in particular, Robert Abelson. In 1979, Schank and Abelson established a formal research and training program at Yale in cognitive science. Under that aegis, they attracted many young psychologists including John Black, James Galambos, Noel Sharkey, Ray Gibbs, Steven Shwartz, and Steven Read Psychology graduate students were likewise attracted to the program They included Brian Reiser, Scott Robertson, Colleen Seifert, and Valerie Abbott.

The experimental paradigm of psychology provided a vehicle for exploring the AI theories of cognitive processing, and the results of the experiments stimulated new AI theories Much of Black's early work focussed on the psychological implications of scripts, and subsequently, MOPs. Story understanding tasks explored issues in causal coherence of texts, hierarchical memory organization, and the thematic organization of the story itself.

This research program provided perspective on the process of psychological experimentation. Researchers noted a marked contrast between discovery and verification research. AI research tended to generate new hypotheses, while cognitive psychology served to test existing ones. While both roles were important and suggested synergism, the Yale psychologists argued that cognitive psychology should develop more hypothesis generation techniques.

This dichotomy between verification and discovery paradigms is reflected in another distinction adopted by Abelson and Schank: *neat versus scruffy* In this view, a theory or discipline or hypothesis (or researcher) can be neat or scruffy. Neat implies formal, quantitative, and observable Scruffy suggests the antithesis: informal, qualitative, and intuitive

Broadly speaking, AI is scruffy and cognitive psychology is neat. There are clear exceptions, such as automatic theorem proving in AI and Freudian mentalism in psychology. More importantly, there is a symbiosis between the neats and scruffies of the world. They each need the other, though they may not be willing to admit it. The scruffies generate bountiful harvests of ideas, while the neats painstakingly sift the grains of truth from the pounds of chaff

The process continues. AI researchers plant the seeds that survived previous harvests. The cultivation proceeds under various guises: scientific, technological, and educational

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency and the Office of Naval Research under contract N00014-85-K-0108, by the Air Force Office of Scientific Research under contract AFOSR-85-0343, and by the National Library of Medicine under contract 1-R01-LM04251

References

Bower, G. H.; Black, J B ; and Turner, T. J. 1979 Scripts in Memory for Text. *Cognitive Psychology* 11·177-220.

Charniak, E , and McDermott, D. 1985. *Introduction to Artificial Intelligence.* Reading, Mass . Addison-Wesley.

Hewitt, C 1969. PLANNER: A Language for Proving Theorems in Robots. In Proceedings of the International Joint Conference on Artificial Intelligence, 295-301. Bedford, Mass : Mitre Corporation.

McDermott, D. V., and Sussman, G. J. 1973. The CONNIVER Reference Manual, Technical Report, 259, AI Laboratory, Massachusetts Institute of Technology.

Minsky, M 1975. A Framework for Representing Knowledge. In *The Psychology of Computer Vision,* ed. P. Winston, 211-277. New York· McGraw-Hill.

Schank, R. C. 1986. *Explanation Patterns. Understanding Mechanically and Creatively* Hillsdale, N J.. Lawrence Erlbaum Associates.

Schank, R. C. 1982. *Dynamic Memory. A Theory of Learning in Computers and People* Cambridge, England: Cambridge University Press

Schank, R. C. 1981. *Reading and Understanding Teaching from the Perspective of Artificial Intelligence.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

Schank, R. C. 1975. *Conceptual Information Processing.* Amsterdam. North-Holland.

Schank, R. C., and Abelson, R. 1977. *Scripts, Plans, Goals and Understanding* Hillsdale, N.J.: Lawrence Erlbaum Associates

Schank, R. C., and Abelson, R. 1975. Scripts, Plans, and Knowledge. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence, 151-157. Los Altos, Calif.· William Kaufmann.

Schank, R. C., and Riesbeck, C. 1981. *Inside Computer Understanding: Five Programs with Miniatures.* Hillsdale, N.J.. Lawrence Erlbaum Associates

Slade, S 1987. *The T Programming Language: A Dialect of LISP* Englewood Cliffs, N.J.: Prentice-Hall.