Commonsense Reasoning about Task Instructions

Rakesh Gupta
Honda Research Institute USA, Inc.
800 California Street, Suite 300
Mountain View, CA 94041
rgupta@hra.com

Ken Hennacy
University of Maryland
Institute for Advanced Computer Studies
College Park, MD 20742
khennacy@cs.umd.edu

Abstract

We have developed an approach for commonsense reasoning using knowledge collected from volunteers over the web. This knowledge is collected in natural language, and includes information such as task instructions, locations of objects in homes, causes and effects, and uses of objects in the home. This knowledge stored in tables in a relational database is filtered using statistical methods and rule-based inference. Missing details within natural language task instructions are reasoned to determine the steps to be executed in the task. These missing details are handled by meta-rules which work across the knowledge categories and interact with the appropriate tables to extract the right information. Our reasoning approach is illustrated for common household tasks.

Introduction

Since the work of McCarthy (McCarthy 1959), commonsense reasoning has been widely investigated with formal treatment of logic. Formal representations involve a variety of operator definitions, depending upon the type of logic (e.g. epistemic, modal). Operator evaluations can be embedded within inference rules to generate facts that drive further evaluations. However most real-world problems, require defeasible reasoning, e.g. non-monotonicity (Zernik 1988).

Formal representations for commonsense require many axioms to represent even the most basic household tasks. From an implementation standpoint, a knowledgebase comprised of a collection of such axioms will hold little advantage over other forms of programming unless the axioms are broadly applicable among tasks. This is difficult to achieve and contradictions arise as the knowledgebase is scaled to larger sizes. Tracking which axioms to use for a given problem statement then becomes difficult.

On top of this difficulty, a dynamic world represented within knowledgebases introduces conflicting information over time that may need to be retracted. One method for maintaining psuedo-monotonicity in the face of a dynamic world relies upon the tagging of all information with a timestamp (Elgot-Drapkin & Perlis 1990). Such schemes can be used to identify conflicts however they typically do not perform well for real-time applications. Various maintenance rules for nonmonotonicity have been explored over the years to handle the various difficulties associated with change and scalability (Minker 1991).

As an alternative to formal methods that guarantee solutions, implementational approaches place more emphasis on adaptation and knowledge acquisition. Studies involving human interactions and natural language, for example, have shown promise in developing agents that can learn and reason about human-centric information (W. L. Johnson 2004). According to Rasmussen *et al* (1983), human behavior in familiar environments doesn't require extensive decision making. It is controlled by a set of rules or steps which have proven successful previously. The sequence of steps may have been derived empirically in the past or communicated from another person's know-how as instructions.

A variety of knowledge sources and knowledge representations have been proposed for such approaches. Knowledge bases such as CYC (Guha *et al.* 1990) have been manually developed. Their rule base utilizes domain heuristics and is manually designed. OpenMind projects (Stork 1999; 2000) have been developed to capture information provided by the public in natural-language form. Applications such as a hypermedia authoring agent have been developed with OpenMind knowledge (Lieberman & Liu 2002).

Implementation of logic takes many different forms, as reflected in the base languages and systems that have been developed over the years. Traditional logic programming languages such as Prolog (Colmerauer 1990) favor the use of backward chaining for applications that involve, for example, planning or theorem proving. CLIPS (Giarratono & Riley 1998) is optimized for production systems (forward chaining). A situational calculus within GOLOG (Levesque *et al.* 1997) or fluent calculus within FLUX (Thielscher 2002) facilitate planning in dynamic environments. Systems built upon semantic networks such as SNEPS (Shapiro & Rapaport 1987) provide a suite of powerful search techniques on semi-structured knowledge to support belief revision with natural language text. Finally, modules such as memory chunking in SOAR (Rosenbloom & Laird 1990) provide a method to implement reflective introspection.

Considerations for choosing an appropriate reasoning paradigm include the knowledge acquisition method, scalability, knowledge representation, and known uncertainties that the system might encounter. In this paper, it is assumed that a user requests the robot to carry out a household task such as making coffee or washing clothes. For this kind of knowledge we bind task instructions (e.g. change filter) from the user to explicit action-object-state assignments via class rules and meta-rules. Such assignments can then be executed by the robot hardware to accomplish the task. OpenMind Indoor Common Sense (OMICS) data (Gupta & Kochenderfer 2004) is used as a source for the task instructions. The various categories of collected data are discussed in the section on Knowledge Capture.

We then discuss how *class rules* process distributed knowledge using symbolic techniques. Statistical methods are used to filter out variations in word choice and sentence structure. However in this paper, it will be assumed that either a single user's instructions are being used, or a composite set of available choices are available. We use the processed OpenMind knowledge with manually designed *meta-rules* to fill in missing information for household tasks. These class and meta-rules are illustrated with the task of making coffee. The system implementation is described in the following section on the integration of knowledge, memory and rules. This is followed by Conclusions and Future Work.

Our long-term goal is to implement this approach in robots that interact with humans to help them perform common household tasks. Task steps may be communicated explicitly to a robot. However typical instructions will rely upon implied understanding and provide sketchy details. The robot will need to use commonsense principles to seek the details in real time to perform these tasks.

Knowledge Capture

Research on distributed knowledge capture methods is growing in popularity due to concepts such as the Semantic Web. There is vast amount of internet information available, however contextual analysis of natural language statements still represents a holy grail for AI research. To address this challenge, OpenMind projects worldwide collect knowledge in a structured format with contextual information. The goal of the OpenMind Indoor Common Sense (OMICS) [1] project is to develop a relational database of indoor home and office knowledge from volunteers (netizens) over the web. The database table entries are created from responses to fill-in-the-blank prompts generated by seed tables that contain information related to common household items and tasks. These prompts help to preserve context understanding by enforcing structural relationships within the OpenMind schema. Such relations are interpreted for a variety of reasoning scenarios including causes-and-effects, task execution, and the determination of relationships among objects.

As an example of the information captured in the SQL knowledgebase, netizens are asked to enter the steps to accomplish a particular task. These statements contain references to specific actions associated with the task. For example, steps in the task of making coffee entered by a typical user are:

```
<change the filter in the coffeemaker>
<add ground coffee onto the filter>
<pour water into coffeemaker top>
<turn on the coffeemaker>.
```

Other relational tables we have used from OMICS include *Locations*, *Uses*, and *Causes*. The *Locations* activity associates objects with the rooms where they are typically found. For example, the user might be prompted with, 'A room where you generally find a dinner table is the ______.' The *Uses* activity associates objects with their uses. For example, the user might be prompted with the form, 'A hanger is used to ______.'

The Causes activity captures causality. An example of the Causes data is 'When the tap is on, the sink becomes full.' These are stored as pairs of object properties, the first being the *cause* and the second being the *effect*.

In general, natural language phrases are parsed and stored in appropriately tagged fields in a SQL database with different tables mapping to different relations. The OMICS knowledge base is also the source for the grammars, object states and action descriptions. Such knowledge representation is relatively self-contained, and can be augmented with natural language statements by interactions with humans.

Processing Distributed Knowledge with Class Rules

Our approach towards reasoning is implementation-based with constraints expressed in SQL queries and pattern recognition implemented with regular expressions. Raw text task instructions captured in OMICS must be processed using a Penn Treebank parser (Klein & Manning 2001). Interpretation of parses is performed via class-rules, i.e. rules specific to grammatical constructs and knowledge content (such as action references). The class rules work effectively as filters— requiring specific, unique relationships to be stated in order for logic to construct facts for processing.

Our discussion of class rules is illustrated with commonsense reasoning as it applies to the coffee-making task. To begin with, the database is queried whenever a request for a task is identified. Other causes, such as observances of change-of-state variables tied to certain behaviors of the robot could also have initiated the query. Given a set of natural-language instructions, the first class rule triggered involves the extraction of action-object-state information from each given step using regular expressions. Such information serves as parameters for closed-loop feedback and control mechanisms (CLFC) in the robot. The extracted action-object-state information for the coffee-making task discussed in the section on knowledge capture is:

```
<change, filter, in coffeemaker>
<add, ground coffee, onto filter>
<pour, water, into coffeemaker top>
<turn on, switch>.
```

Class rules are also utilized for finding the likely location of objects. There are tables in OMICS that represent typical object locations in a home. Such information gathered

[1] http://openmind.hri-us.com

from netizens cannot be specific to an actual home. The netizen responses, however, may suggest that a couch is located in multiple places such as the living room, family room, or den. Thus, upon arriving at a new home, the robot will have a common sense notion as to where household objects are likely to be located. Over time, such knowledge is replaced by actual observations.

A class rule that performs verb tense search and replacement is typically triggered as well. This might be used to provide feedback to the user about the robot's current state and actions as well as target specific robot functionality. Task steps in OMICS typically contains the present tense of a verb. Most verbs follow the same rule for converting to past or present participle tenses (adding "ed", "ing"). However there are exceptions such as the word make: (make, made, making). These exceptions are stored in a look-up table (provided by WordNet) associated with the rules. For converting verb references to present tense form, we first check for exceptions and then apply the rule to extract the proper tense of the verb.

Other grammar rules involve synonym search and replacement to match verbs and objects to those verbs and objects explicitly referenced in logic. An example verb is the word *change*. Other synonyms for change include *replace* and *alter*. These synonyms are extracted from the lexical utility Sentry Thesaurus from WinterTree Software. Once replacement candidates are identified, a user's utterance can be translated into a form that is more likely to trigger further processing within the robot's fact/rule knowledgebase.

Finally, class rules are used to find CLFC processes for natural language statements that ultimately do not bind to a known CLFC process. For example, if the robot does not know how to fill a glass with water, the action-object-state (fill-glass-with water) can be processed further using facts associated with the individual object references, actions, and states to find appropriate CLFC processes. A description of this process is provided in the next section.

Using Commonsense Rules to Seek Missing Details

In addition to class rules that are specific to knowledge structures, rules are needed that are independent of specific classes, to seek missing details. In general, commonsense meta-rules are triggered to ensure that a robot (and environment) are properly configured to carry out a task by calling multiple class rules to extract missing pieces of information. Figure 1 illustrates the interaction between meta-rules, class rules and the knowledge base.

The class rules are designed to format information for commonsense processing. Such processing is triggered in response to speech events or new observations. For example, a user speech request event is expressed as a multi-field collection of data consisting of the context (e.g. request), and specific task reference (e.g. make coffee). In response, the meta-reasoner looks up the list of instructions for the task and sends them to a task sequencer rule in a reasoning thread for further processing. The task sequencer processes each instruction one at a time, waiting for a fact to be triggered to release the execution of the next task instruction.

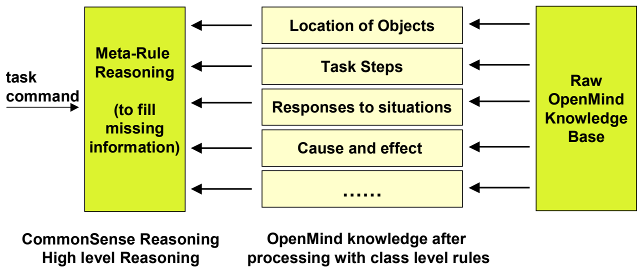
task command
Meta-Rule Reasoning (to fill missing information)
Location of Objects
Task Steps
Responses to situations
Cause and effect
......
Raw OpenMind Knowledge Base
CommonSense Reasoning High level Reasoning
OpenMind knowledge after processing with class level rules

Figure 1: Schema showing interaction between meta-rules, class rules and the knowledge base.

Figure 2 illustrates the knowledge used in the interactions between meta-rules and class rules for responding to a request to make coffee. As each task instruction is extracted, it is processed according to the methods described in the previous section. We now illustrate the processing that must take place to extract additional details for the example task of making coffee. With the first instruction,

```
<change the filter in the coffeemaker>
```

a complete action-object-state description is recognized. At this point the robot is prepared to perform an action on objects it recognizes, however it must use commonsense rules to interact with these objects. Whenever a *direct* object is referenced, a query is triggered to locate the object in the Locations table. Whenever an action request is made, an implicit self-reference is assumed, hence another query is triggered to locate the robot.

By invoking the class rule for locations, the filter is found to be in the kitchen. Hence, the robot will first choose to act upon the filter in the kitchen. It would be better if the robot had an understanding that change is an action that is only performed on used filters, and that a filter box contains only new filters, however such information will not always be present. As part of commonsense processing, in principle, a consistency check can be performed to verify the causes and effects associated with a given action to notice such rules.

Then a map algorithm instantiates a chain of responses to the chain of object references generated by the previous queries to navigate the robot to the desired location. This navigation is performed via a series of commands to the meta-reasoner to oversee the operation of the relevant components of the robot (e.g. legs). For example, the following command will be generated in response to the comparison between the robot's location (living room) and the filter's inherited location (in the kitchen):

```
    <go to the kitchen>.
```

Upon binding the action "go" to a legs routine "walk", the robot will initiate a navigation routine to the desired object reference. Upon arriving at the kitchen, a completion fact is initiated which restarts the visual check to determine where the *direct* object is located. The same cycle of rules are executed until the robot is eventually proximate to the *direct*

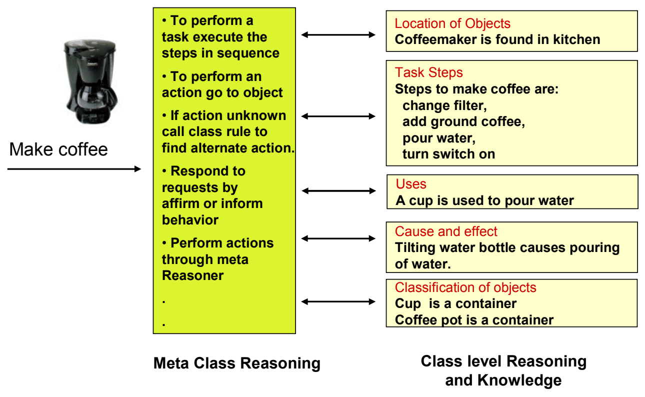
Make coffee
• To perform a task execute the steps in sequence
• To perform an action go to object
• If action unknown call class rule to find alternate action.
• Respond to requests by affirm or inform behavior
• Perform actions through meta Reasoner
Location of Objects
Coffeemaker is found in kitchen
Task Steps
Steps to make coffee are: change filter, add ground coffee, pour water, turn switch on
Uses
A cup is used to pour water
Cause and effect
Tilting water bottle causes pouring of water.
Classification of objects
Cup is a container
Coffee pot is a container
Meta Class Reasoning
Class level Reasoning and Knowledge

Figure 2: Knowledge used and interaction between meta-rules and class rules for a coffee making task.

object location. At this point, a command is sent to the meta-reasoner, requesting that the action take place. In our work, a simulator is used in place of a CLFC process to carry out the actions. When the execution cycle is finished, a completion fact is instantiated which triggers the processing of the next task instruction in the queue.

Another use of commonsense meta-rules involves handling action-object-state facts that the robot does not know how to execute. As discussed in the previous section, a search of its knowledgebase is required in order to decompose the action-object-state reference into known CLFC processes. In traditional logic, many axioms are typically required to express the proper relationships between objects in order to constrain the possible behaviors that the robot can perform. An example of such a pair could be the following step from the make coffee example:

```
<pour, water, into coffeemaker top>.
```

If the robot does not know how to pour water, the action object pair (pour, water) can be processed further using reasoning concerning the interactions between objects and their states (cause-and-effect) to find an alternate way to achieve the same effect. Class rules match the *effected* object and property to all entries in the knowledge base to find a possible *cause*, e.g. (tilt, water bottle). Alternative sources of water, or the implicitly referenced container for pouring can be determined by chaining together multiple cause and effects. This information is extracted from OMICS using meta-rules and parsed into a look-up table for assembling a set of action-object pairs to accomplish the step. This may return an alternate action-object pair to achieve the same result, that the robot may know how to execute:

```
    <tilt, water bottle>
```

System Implementation

The reasoning engine we used for our study was developed in Java and JESS provided by Sandia National Laboratories (Friedman-Hill 2004). The core of the system is implemented within the kernel's SQL schema which represents a self-organizing ontology. By self-organizing, we refer to the ability of the kernel to create and augment tables with new information. This ability accomplishes several key processing capabilities described below.

The operation of the kernel at present is governed by both class-level and meta level logic. It contains the core set of rules and interfaces which gets the system up and running. It is kept separate from meta reasoning, which may be busy handling various action statements and monitoring hardware performance. It is also kept separate from reasoning threads which may involve explicit timing requirements. Figure 3 illustrates the interaction between various types of processes that contribute to common sense reasoning. We utilize Nuance's Speech Verifier and Vocalizer to recognize and generate speech (Nuance 2004). A Penn Treebank parser is used to process text data that is either extracted directly from OMICS, or from a statistical grammar model applied to audible speech. OMICS and Memory data are stored in a SQL schema and is accessed via a MySQL server (Vaswani 2002). The SQL schema facilitates the binding of memory, rules, and knowledge to robot functionality.

We define Memory as consisting of the working knowledge which has been instantiated through observation. It includes a robot's knowledge about the state of the environment, knowledge about its own current state (e.g. whether it is walking or extending an arm), and which room the robot is currently in. As the robot goes from room to room, observations about actual object locations are recorded in the state tables in Memory. These observations are used to update a statistical-based commonsense representation of object locations. More work is needed, however, to properly model this memory for learning in a dynamic environment.

The binding operation across rule bases and components using tags forms the basis of a self-organizing ontology. This mechanism is used to identify and assign a preliminary context to incoming information, learn new information, and format OMICS knowledge into logic-based facts for processing. Particular AI algorithms that need to be used can be referred to within this ontology and are instantiated within independent reasoning threads as required. Tags such as *request*, *query*, and *inform* are assigned to task instructions based upon keyword references and sentence structure. Sentence structure is identified by searching through a database of regular expressions and applying them to the text. Once a regular expression is satisfied, the associated tag is identified. Having identified the immediate context, references to behavior choices (or responses) are made via the same tags through a behavior map. When only one choice is present, the response represents a default assignment. Behavior tags and assignments currently originate from the system designer.

After class and meta-rules configure the system for a particular action, the action is scheduled via a meta-reasoner. The meta-reasoner monitors status information generated by an action control loop. The meta-reasoning thread is responsible for scheduling rule executions bound to particular actions and noticing anomalies associated with those executions.

The need to interface reasoning with hardware requires

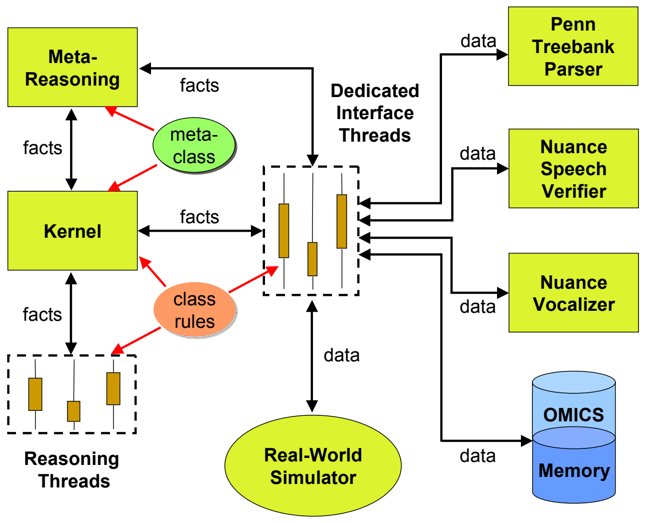

Figure 3: Multi-threaded Reasoning System.

multi-threading. Our architecture utilizes two layers of interfacing to access and control various threads. First, each hardware component (or simulated component) has an exclusive set of JESS rules that are implemented within their own thread. Secondly, within each thread, the JESS rules target specific hardware (or simulator) commands that need to be executed. A reasoning and meta-reasoning thread is used to manage their execution. The reasoning thread is responsible for making a decision to refer to a rule for a particular hardware interface. These references are made through a two step process that first binds natural language references to actions (such as "put") in the appropriate hardware thread (arm motion). Then, other context-specific rules utilize the particular configuration information associated with the action to determine which commands parameters to use.

To illustrate how this architecture is used, we discuss the sequence of events that unfold when a robot is asked to make coffee. First, a user's utterance is translated either into recognized slot parameters by the speech verifier, or extracted text. Slot parameters are used when a precise identification of the utterance is made while text is generated only when a precise match has not been found. Slot parameters are used to create a fact that uniquely identifies the speech event. Text must be processed further by a parser and search algorithm to determine the most likely speech event. After this event is identified, a fact is instantiated in the same manner as before.

When a speech fact is instantiated, a behavior tag associated with the event identifies likely responses. If the system is in an obedient mode, these responses are executed without further reasoning (default response). In our example, the system sends back an *affirm* response: e.g. the system sends text to the speech vocalizer to inform the user that coffee will be made. Another class rule triggers the recall of the task steps associated with making coffee. Then, these steps are sequenced by a reasoning thread that was instantiated by the kernel. As each step is carried out, the state changes to objects are updated in Memory. These updates to Memory are important for future actions. When the steps are completed, the reasoning thread is terminated to free up resources.

As required, the reasoner accesses class rules to send facts to various hardware interfaces for implementation. These hardware-interface threads are dedicated to specific hardware and continue to operate throughout the robot's operation. In our example, facts associated with walking, grabbing, and pouring are sent to threads associated with the legs and hands. These facts are sent via the meta-reasoner which is responsible for last-chance decision-making as well as recognizing problems associated with the hardware interfaces. In our work so far, we have utilized a graphical animation utility to simulate hardware interface commands.

Conclusions

We have implemented a reasoning architecture that uses knowledge in natural language form, collected from distributed sources. This distributed knowledge does not have to be carefully hand-crafted as in case of expert systems. However, distributed knowledge is inherently noisy, so we use an ontological schema and statistical methods to identify useful sentence structures. In our current system, we have about 15 hand-crafted class rules.

We use meta-rules to implement the commonsense that is required to fill-in missing information handling tasks requests. These rules are common for all household tasks. In out current system we have about 20 hand-crafted meta-rules. Meta-rules are also used to model interactions with humans, manage interactions among multiple actions, and handle anomalous situations that may arise in the execution of the tasks. our architecture is used to control a simulated robot that performs tasks requested by the user.

Although our work is currently limited to performing tasks in home and office environments, our approach is general enough to be used for other domains. Many of the class rules would be the same, but new distributed knowledge and class rules can be added as appropriate. The seamless integration of distributed knowledge, natural language processing, class rules and meta-rules simplifies rule and knowledge updates and makes our architecture and system scalable.

Acknowledgements

This work was performed while Ken Hennacy was supported by Honda Research Institute USA, Inc. The authors would like to thank Mykel Kochenderfer for his helpful comments and suggestions. Thanks are also due to anonymous reviewers and the users of the Open Mind Indoor Common Sense website for their data and feedback.

References

Colmerauer, A. 1990. An introduction to prolog iii. *Communications of the ACM* 33(7):69–90.

Elgot-Drapkin, J., and Perlis, D. 1990. Reasoning situated in time I: basic concepts. *J. of Experimental and Theoretical Artificial Intelligence* 2:75–98.

Friedman-Hill, E. 2004. *Jess in Action: Rule-Based Systems in Java*. Greenwich, CT: Manning Publications Co.

Giarratono, J., and Riley, G. 1998. *Expert Systems Principles and Programming, Third Edition*. Boston, MA: PWS Publishing Company.

Guha, R. V.; Lenat, D. B.; Pittman, K.; Pratt, D.; and Shepherd, M. 1990. Cyc: A midterm report. *Communications of the ACM* 33(8):391–407.

Gupta, R., and Kochenderfer, M. 2004. Common sense data acquisition for indoor mobile robots. In *Nineteenth National Conference on Artificial Intelligence (AAAI-04)*.

Klein, D., and Manning, C. D. 2001. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the penn treebank. *ACL 39/EACL 10* 330–337.

Levesque, H.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

Lieberman, H., and Liu, H. 2002. Adaptive linking between text and photos using common sense reasoning. In *Proceedings of Adaptive Hypermedia and Adaptive Web-Based Systems*.

McCarthy, J. 1959. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*.

Minker, J. 1991. An overview of nonmonotonic reasoning and logic programming. Technical Report UMIACS-TR-91-112, CS-TR-2736, University of Maryland, College Park, Maryland.

Nuance. 2004. *Speech Recognition System 8.5: Introduction to the Nuance System*. USA: Merriam-Webster, Incorporated.

Rasmussen, J. 1983. Skills, rules and knowledge: Signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man and Cybernetics* SMC-13(3):257–266.

Rosenbloom, P. S., and Laird, J. E. 1990. Integrating execution, planning, and learning in soar for external environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1022–1029. Boston, MA: MIT Press.

Shapiro, S. C., and Rapaport, W. J. 1987. *SNePS considered as a fully intensional propositional semantic network*. New York: Springer-Verlag. 263–315.

Stork, D. G. 1999. The Open Mind Initiative. *IEEE Expert Systems and Their Applications* 14(3):19–20.

Stork, D. G. 2000. Open data collection for training intelligent software in the open mind initiative. In *Proceedings of the Engineering Intelligent Systems (EIS2000)*.

Thielscher, M. 2002. Programming of reasoning and planning agents with flux. In *Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2002)*, 263–315. Morgan Kaufmann Publishers.

Vaswani, V. 2002. *MySQL: The Complete Reference*. Emeryville CA: McGraw-Hill/Osborne.

W. L. Johnson, S. Marsella, H. V. 2004. The darwars tactical language training system. In *The Interservice/Industry Training, Simulation and Education Conference, 2004, Orlando, FL.*

Zernik, U. 1988. Default reasoning in natural language processing. In *Proceedings of the 12th Conference on Computational Linquistics*.