The Medication Advisor Project: Preliminary Report

George Ferguson, James Allen, Nate Blaylock, Donna Byron, Nate Chambers,
Myroslava Dzikovska, Lucian Galescu, Xipeng Shen, Robert Swier, and Mary Swift

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 776

May 2002

Abstract

The Medication Advisor is the latest project of the Conversational Interaction and Spoken Dialogue research group at the University of Rochester. The goal of the project is an intelligent assistant that interacts with its users via conversational natural language, and provides them with information and advice regarding their prescription medications. Managing prescription drug regimens is a major problem, particularly for older people living at home who tend to have both complex medication schedules and, often, somewhat reduced faculties for keeping track of them. Patient compliance with prescribed regimens is notoriously low, leading to incorrect and sometimes harmful usage of both prescribed and over-the-counter medications. The Medication Advisor builds on our prior experience constructing conversational assistants in other domains. In addition to providing new challenges, the project allows us to validate previous efforts in areas such as portability. This brief report details our initial efforts and outlines our future direction.

This material is based upon work supported by Dept. of Education (GAANN) grant no. P200A000306; ONR research grant no. N00014-01-1-1015; DARPA research grant no. F30602-98-2-0133; NSF grant no. EIA-0080124; and a grant from the W. M. Keck Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

1 Background and Motivation

1.1 Conversational Systems

We have for some years been interested in developing *conversational assistants*: computer systems that interact with their users using spoken natural language in order to help them solve problems. Starting from corpus studies gathered using a “Wizard of Oz” setup [Gross *et al.*, 1993; Heeman and Allen, 1995], we have built a sequence of prototype collaborative planning systems [Allen *et al.*, 1995; Ferguson *et al.*, 1996; Ferguson and Allen, 1998]. More recently, we have been extending this work to several new domains including a “911 dispatcher” in a crisis management center, a kitchen design assistant, and the Medication Advisor described in this paper.

Conversational interaction enables effective human-computer collaboration for a variety of reasons. First, it is natural for people, requiring no training in its use. It is efficient, since it exploits humans’ highly developed, built-in mechanisms for spoken interaction. It is also expressive, in that one can express complex, novel ideas using the standard mechanisms of language (and understand same). Spoken conversation is also hands-free, which is very desirable for systems designed to be inobtrusive assistants, for example in a home environment. And finally, it is naturally mixed-initiative, meaning that control of the conversation can switch easily between the participants. This is crucial for collaborative systems that must take maximum advantage of the different skills of both human and computer participants. For these and other reasons, we have been pursing the development of models of conversational collaboration and implementing these in end-to-end, intelligent, dialogue-based systems that help people solve (simple) problems.

1.2 The Prescription Compliance Problem

Our latest project, the Medication Advisor, is being carried out in conjunction with the Center for Future Health at the University of Rochester.[1] Located in the University’s Medical Center, and affiliated with Strong Memorial Hospital, the Center is a multidisciplinary research facility dedicated to the creation of an integrated system of affordable, easy to use, intelligent health care tools for use by consumers in their homes. Why focus on the home? First, people are increasingly taking control of their health care, through mechanisms ranging from finding health information on the Internet to spending billions of dollars annually on alternative and non-traditional health products. Second, the health care system is already over-constrained and overburdened, to the point that if people want something more they have to do it themselves. And most significantly, there is the demographic imperative of our aging population—whereas twenty years ago the ratio of caregivers to patients was 20:1, by 2030 it is predicted to be only 6:1. Put another way, most households will have at least one person needing some degree of medical care living with them. There will simply not be enough hospitals and nursing homes, nor the qualified personnel to run them, to look after everyone. Thus the Center’s mandate is to develop technologies that complement the health care system, while allowing people to live healthier, longer lives in their own homes.

The health care system itself is, in some sense, to blame for one major health-related problem facing people today. With the huge growth in the number of pharmaceutical therapies, patients tend to end up taking a combination of several different drugs, each of which has its own characteristics

[1] `http://www.centerforfuturehealth.org`

and requirements. For example, each drug needs to be taken at a certain rate: once a day, every four hours, as needed, and so on. Some drugs need to be taken on an empty stomach, others with milk, others before or after meals, and so on. Some drugs are prescribed for chronic conditions (patients will take them for a long time), others are prescribed as the need arises. It doesn't take very many of these prescriptions to get to the point where the patient either can't figure out what to do or spends an inordinate amount of time figuring it out. The result is that patients simply do not (or cannot) comply with their prescribed drug regimen.

The statistics back up the intuition that non-compliance is a serious problem. A 1989 study showed that non-compliance causes 125,000 deaths annually in the U.S. [Smith, 1989]. A recent review of the literature based on electronic monitoring of drug consumption concluded that mean dose-taking compliance ranged from an already-poor 71% (+/- 17%) for once-per-day drugs to just 51% (+/- 20%) for drugs meant to be taken four times per day [Claxton *et al.*, 2001]. The New York Times has labelled non-compliance the world's "other drug problem" [Zuger, 1998].

In discussion with geriatricians (as well as through our own personal experience with older relatives), it is clear that this problem is even more acute for older patients. These patients are typically taking a larger number of medications, and these medications are often prescribed by several different doctors. As well, older patients often have perceptual or cognitive deficits (poor eyesight or memory, for example) that make the problem even harder for them.

On the other hand, from a technical perspective, the medication compliance problem seems like a relatively circumscribed domain in which it might be possible to develop an intelligent computer assistant. The major problems involve maintaining the patient's medication schedule (notifying them as appropriate, helping with rescheduling, *etc*.) and answering questions about the schedule and the drugs involved. We were therefore optimistic from the outset that we would be able to adapt our work on planning and scheduling assistants to the task of the Medication Advisor. The remainder of this report discusses our approach and some of the technical issues we encountered. This should be taken as a preliminary report on a work in progress. We will conclude with a discussion of where we are going next on this project.

2 System Architecture and Implementation

Our initial prototype of the Medication Advisor is based on the architecture developed for TRIPS, The Rochester Interactive Planning System [Allen *et al.*, 2001], shown in Figure 1. TRIPS is designed as a loosely-coupled collection of components that exchange information by passing messages using the KQML [Labrou and Finin, 1997] language. A detailed discussion of all the components is not required for this report, but some understanding of how the pieces go together is necessary to appreciate our work on the Medication Advisor. This section provides a brief overview of the Medication Advisor system architecture. For more details, see [Allen *et al.*, 2000].

Components are divided among three main categories: Interpretation, Behavior, and Generation. As shown in the figure, some components straddle categories, meaning they represent state and provide services necessary for both sorts of processing. The Interpretation Manager (IM) interprets user input coming from the various modality processors as it arises. It interacts with Reference to resolve referring expressions and with the Task Manager (TM) to perform plan and intention recognition, as part of the interpretation process. It broadcasts recognized speech acts and their interpretation

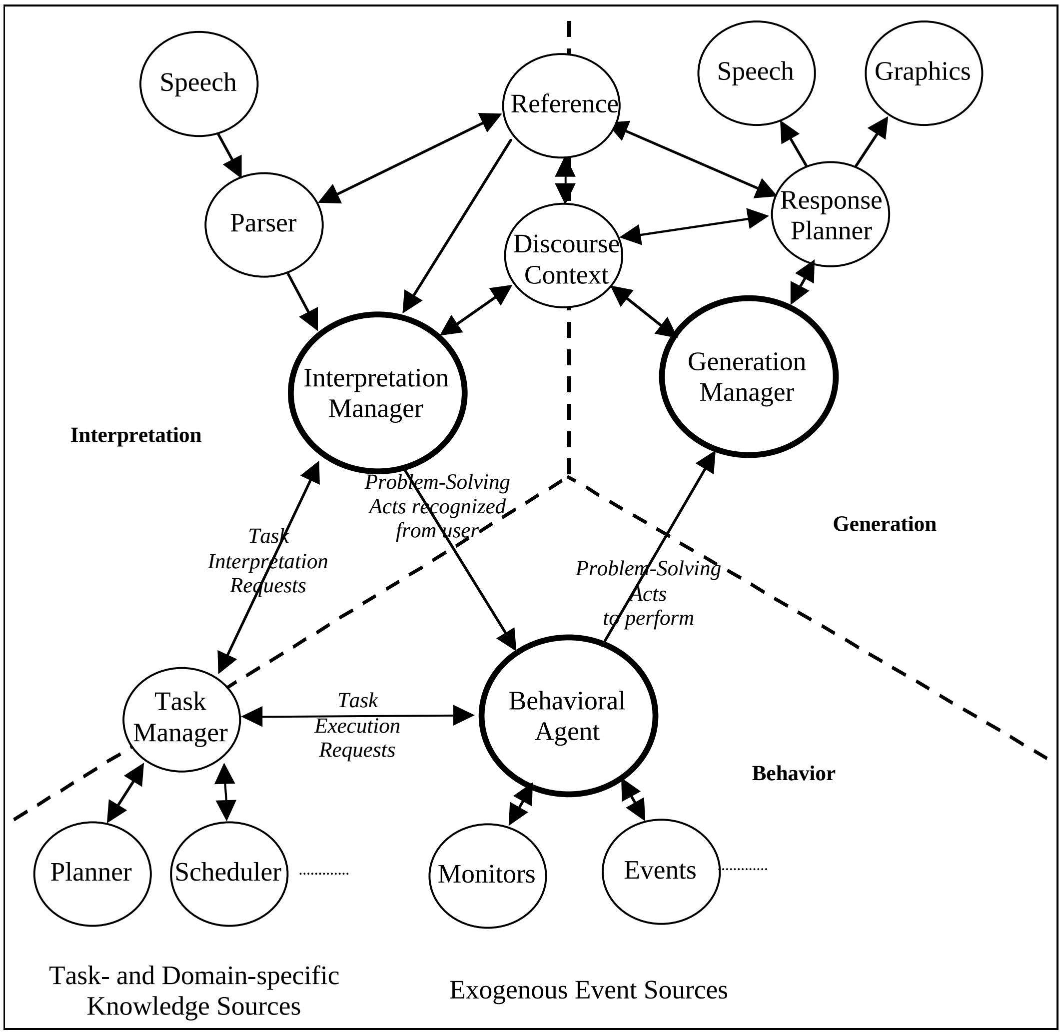

Figure 1: TRIPS collaborative system architecture, from [Allen *et al.*, 2001]

as collaborative problem solving actions, and incrementally updates the Discourse Context (DC). The Behavioral Agent (BA) is in some sense the autonomous "heart" of the agent. It plans system behavior based on its own goals and obligations, the user's utterances and actions, and changes in the world state. Actions that require task- and domain-dependent processing are performed by the Task Manager. Actions that involve communication and collaboration with the user are sent to the Generation Manager (GM) in the form of communicative acts. The GM coordinates planning the specific content of utterances and display updates and producing the results. Its behavior is driven by discourse obligations (from the DC), and the directives it receives from the BA. The glue between the layers is an abstract model of problem solving in which both user and system contributions to the collaborative task can be expressed. A more detailed analysis of the problem-solving model, including an example in the Medication Advisor domain, is presented in [Allen *et al.*, 2002].

It is important to note that, in TRIPS, all three major components (IM, BA, and GM) operate asynchronously. For instance, after (or even during) interpretation by the IM, the GM might be generating an acknowledgement while the BA is still deciding what to do. And if the user starts speaking again, the IM will start interpreting these new actions (the GM may or may not stop speaking immediately). The Discourse Context maintains the shared state needed to coordinate interpretation and generation. The Task Manager maintains the task-level information about current problems, solutions, objectives, and so on, as well as coordinating the invocation and integration of the various specialized reasoners that provide the system's capabilities in a given domain.

2.1 Challenge Dialogue

Figure 2 shows our initial "challenge dialogue" for the prototype Medication Advisor system. The user is an elderly woman, who is taking a small number of commonly-prescribed medications. She wakes up with a headache and decides to ask her Medication Advisor system for help. Although the system is fully implemented, for this domain some aspects of the problem-solving are not fully implemented (see subsequent discussion). Utterances are labeled with the speaker and a number indicating the sequence of speaker turns.

The user starts by asking a simple question:

U: Can I take an aspirin? (1)

The system interprets this as either a yes-no question about the ability to take aspirin, or as a suggestion to take aspirin (now, *i.e.*, at the time of the utterance). In either case, the system must first verify whether taking aspirin is possible. It cross-references its knowledge about the user's prescriptions with its knowledge base of drug interactions and side-effects. Since this search turns up an interaction with one of the prescribed medications, the system can immediately answer that taking aspirin is not permissable:

S: No, you are taking Celebrex and Celebrex interacts with aspirin. (2)

In fact, this response would be appropriate for either of the two possible interpretations of the user's utterance.

The user acknowledges the system's utterance:

U:	Can I take an aspirin?	(1)
S:	No, you are taking Celebrex and Celebrex interacts with aspirin.	(2)
U:	Oh.	(3)
U:	Could Celebrex cause headaches?	(4)
S:	No, headaches are not an expected side-effect of Celebrex.	(5)
S:	Do you have a headache?	(6)
U:	Yes.	(7)
S:	How severe is your headache on a scale of one to five.	(8)
U:	One.	(9)
S:	Your doctor says you can take Tylenol for mild headaches.	(10)
U:	Ok, I'll try that.	(11)
	Why am I taking Celebrex?	(12)
S:	You are taking Celebrex to help with your arthritis.	(13)
U:	Should I take one now?	(14)
S:	Yes, you are taking Celebrex every morning.	(15)
U:	Is there anything else I need to take now.	(16)
S:	Yes, you should take your Prinivil.	(17)
U:	Which one is that?	(18)
S:	It's the yellow one. [shows picture]	(19)
U:	Could that be causing my headaches?	(20)
S:	Yes, headaches are an expected side-effect of Prinivil.	(21)
S:	Tylenol should help.	(22)
U:	Ok, thanks.	(23)
S:	You're very welcome.	(24)

Figure 2: Challenge dialogue for initial Medication Advisor prototype

U: Oh. (3)

and then goes on to ask another question:

U: Could Celebrex cause headaches? (4)

At first glance, this is yet another yes-no question. The system could simply look up the answer using its knowledge bases and tell the user the answer.

However, it is here that we begin to see the difference between a simple natural language front-end to a database query system and a true conversational assistant. As part of interpreting the user's utterance, the system attempts to recognize the intention behind the utterance—that is, why she said what she did. This intention recognition process is one of the keys to effective collaboration (see Section 6). In this case, the question about headaches adds evidence to the hypothesis that the user is trying to treat a symptom (headache) with an over-the-counter remedy (aspirin). This hypothesis regarding the user's intention was initially created during the processing of utterance (1), but at that time there was not yet sufficient evidence for the system to take action based on it. With the additional evidence from (4), the system is reasonably confident that the user has a headache that they are trying to treat. As part of its response to this type of user action, it consults its knowledge bases to determine whether such a situation requires action on its part. It turns out that, in this scenario, one of the other medications that the user is prescribed for can result in severe headaches, and that if these occur, it is a serious condition that should be reported to the doctor immediately.

So, to summarize, the literal answer to the question about Celebrex causing headaches is no. However, the system decides to take the initiative in the conversation and resolve the issue about the user's headache, since it is potentially serious. The result is the following utterances:

S: No, headaches are not an expected side-effect of Celebrex. (5)
S: Do you have a headache? (6)

The user answers:

U: Yes. (7)

The following sub-dialogue ensues to determine what action to take as a result of the user's having a headache:

S: How severe is your headache on a scale of one to five? (8)
U: One. (9)
S: Your doctor says you can take Tylenol for mild headaches. (10)
U: Ok, I'll try that. (11)

The form of the question in (8) was suggested to us by our medical colleagues. The information used to produce the system utterance (10) is assumed to come from a combination of explicit instructions associated with a prescription (*i.e.*, prescriptions are more than just a drug and a dosing), together with general "what if" knowledge provided in drug information databases or patient information handouts.

With the headache issue resolved, the system returns the initiative in the conversation to the user, who decides to pursue her original line of questioning regarding Celebrex. The remainder of the dialogue is intended to be illustrative of the sorts of questions that might reasonably arise in a Medication Advisor context (although admittedly it might seem a bit unlikely to have them all come up in this one dialogue).

U: Why am I taking Celebrex? (12)
S: You are taking Celebrex to help with your arthritis. (13)

Again, the reason for taking a medication is an important aspect of the extended notion of "prescription" required to support this kind of dialogue.

Another capability that we wanted to emphasize is the system's ability to reason about when the medications are supposed to be taken (as opposed to the "why" emphasized thus far in the challenge dialogue). We see a terrific opportunity for some fairly simple (in AI terms) scheduling technology to help people by giving reminders and answering questions about when they should take their pills (see Section 7). So, for example, the user continues:

U: Should I take one now? (14)
S: Yes, you are taking Celebrex every morning. (15)
U: Is there anything else I need to take now? (16)
S: Yes, you should take your Prinivil. (17)

Note the fairly complex use of referring expressions and metonymy throughout this exchange (see Section 5.1).

Another possibility we have been pursuing is using the computer's multimodal capabilities to improve the interaction when possible. An obvious example is helping with the very common problem of identifying medications. So in response to being told to take Prinivil, the user might well ask:

U: Which one is that? (18)
S: It's the yellow one. [shows picture] (19)

The system responds by showing a picture from its medication knowledge base. Of course, if the user is interacting with the system over the phone, say, then the visual modality is not available, and the generation components of the system would have to reason about how to achieve the goal of identifying the specific medication using only language (for example, by describing its color, as above, or by its location, its purpose, *etc.*).

In the rest of the dialogue, the user follows up on her earlier headache question:

U: Could that be causing my headaches? (20)
S: Yes, headaches are an expected side-effect of Prinivil. (21)
S: Tylenol should help. (22)
U: Ok, thanks. (23)
S: You're very welcome. (24)

The remainder of this report describes various technical challenges that we encountered while developing the initial prototype of the Medication Advisor system. Although these are presented as independent issues, in reality they interact in various ways, as do the solutions. We conclude the report with some discussion of future directions.

3 Knowledge Acquisition

A crucial component in the Medication Advisor is the system's knowledge base of prescription drugs and over-the-counter medications. The knowledge base was built using a process that combined automatic knowledge extraction and manual correction of the extracted knowledge. This section describes the knowledge acquisition process and also the representation chosen for the system's knowledge about medications and their effects.

3.1 Information Source

The knowledge base for the Medication Advisor project is extracted from the online MEDLINEplus Health Information web site[2], provided by the National Library of Medicine. The site is a page by page listing of legal prescriptions for the U.S. (and Canada) and its purpose, quoted from the main page, is to be "a guide to more than 9,000 prescription and over-the-counter medications provided by the United States Pharmacopoeia." Since the web-site was designed for the average web user, most of the information is described with English sentences and the main task during knowledge acquisition was to extract the relevant information from this English.

There were three main subjects of interest that we desired from each page: substance to brand name mappings, dosing schedules, and substance effects. Each of the pages follow the same HTML structure, so we are often able to use the HTML tags as pointers to the required information. For instance, brand names are always listed with numbers that indicate their ingredients. These number/ingredient mappings can be found between `<PRE>` tags later in the page. In addition, the dosing schedules are in a hierarchy between `<UL>` and `<LI>` tags that is also parsable. We can categorize the drugs with their doses and by searching for keywords in the English description that follows, the script is able to extract the relevant information. Only in rare cases does the script fail and human intervention is needed. Unfortunately, our third data type, substance effects, is not as simple. Our automated script for parsing these pages was helpful in determining when a drug interaction was listed, but pulling the semantics out of the natural English was difficult and left to the human researcher.

3.2 Knowledge Representation

The Medication Advisor System must know about the side effects of taking drug substances in order to converse with patients about their symptoms. The goal of our Knowledge Representation was to represent these external effects described in MEDLINE in a descriptively rich, yet simple coding structure. The ambiguity of the explanations that are inherent from such a source as MEDLINE,

[2] http://www.nlm.nih.gov/medlineplus/druginformation.html

whose main purpose is to communicate with humans, required us to make many assumptions during classification. The representation we created and the problems we faced during its inception are described here.

3.3 Development

One of the main tasks of the KR is to represent drug interactions and their effects. However, classifying drug interactions proved to be a difficult problem; the following examples show the ambiguity that arises from MEDLINE's descriptions of drug interactions[3]:

1. Troleandomycin–Use of these medicines with astemizole or terfenadine may cause heart problems
2. If you are now taking, or have taken within the past 2 weeks, any of the MAO inhibitors, the side effects of the antihistamines, such as drowsiness and dryness of mouth, may become more severe
3. Make sure you know how you react to the antihistamine you are taking before you drive, use machines, or do anything else that could be dangerous if you are not alert.
4. Antihistamines are used to relieve or prevent the symptoms of hay fever and other types of allergy

Example (1) is the most prevalent of interactions listed in MEDLINE. It is a drug-to-drug interaction that specifies the substances that interact and what effect results. Obviously, drugs interact with other drugs. However, the temporal constraints on such an interaction are both ambiguous and undefined.

A representation of time is clearly needed in the Medication Advisor KR. We concluded that any situation where time is not explicitly mentioned (example (2) is an example when it is) will assume concurrent consumption, interpreted as a couple of hours by the system. However, future work in this area is needed to safeguard the system against improper assumptions.

Example (3) shows that we need to represent substance interactions with other "events" instead of just other substances. In this case, the act of operating machinery should be avoided when taking antihistamines. Finally, example (4) shows a case that is neither drug-to-drug nor drug-to-event, it describes the effects of a substance on a symptom: drug-to-symptom.

There are many examples of different interactions and the above is only a small sample. Clearly, defining different interaction rules for each combination produces too much overhead in the language and we found it to be unnecessary. Instead, we created three types of *events* to define every possible description of a person's daily schedule:

- `(having` *`condition`*`)`
- `(taking` *`substance`*`)`
- `(doing` *`activity`*`)`

[3]This text is quoted from Medline's description of Antihistamines (Systemic) http://www.nlm.nih.gov/medlineplus/druginfo/antihistaminessystemic202060.html

```
(DRUG-INTERACTION-RULE
   :condition C
   :timing T
   :prescription-constraint P
   :side-effect
     (effect
        :effect E
        :frequency-of-occurrence F
        :severity-of-problem S
        :response R )
   :side-effect ...  )
```

Figure 3: Drug Interaction Rule schema

All events are considered to occur in the present time. In order to represent example (2), the `history-of` operator maps present events to an inclusive past time period. Thus for example (2) we get:

```
(history-of (taking MAO-inhibitor) (weeks 2))
```

The semantics of this statement imply that an MAO-inhibitor was taken any time during the last two weeks. We have no need in the Medication Advisor domain to define exact times when defining drug interactions because they never arise. There is never a case such as, "do not take drug A if you took drug B 36 hours ago." Rather, you would see, "do not take drug A if you took drug B within the last 36 hours."

3.4 Drug Interaction Rule

With all activities represented as events, and time periods adequately defined, we developed the Drug Interaction Rule to define the effects of a substance when taken within different contexts. Figure 3 shows the schematic form of the rule.

The semantics of this rule can be given as: "Under conditions C with timing T, the prescription constraint P applies. For each side-effect, the effect E will occur with frequency F. Further, if it is observed (under conditions C with timing T), then it indicates a problem of severity S and the patient's response should be R." The condition C is the drug interaction, such as `(AND (taking MAO) (doing machinery))`. The prescription constraint P has a variety of options, but the most common is `doctors-permission`.

Finally, we needed to formalize the representation of the *effect* of an interaction. The common example of aspirin would lead us to a conclusion that substances lessen the degree of a side-effect (as aspirin lessens pain), however this is only one in a set of many types of effects. The following are a few examples from the MEDLINE database that we needed to represent:

1. Although not all of these side effects may occur, if they do occur they may need medical attention… cough; diarrhea; difficulty swallowing; dizziness; fast heartbeat; fever; headache…

2. (acetaminophen taken while having) liver disease–The chance of serious side effects may be increased.

3. Acetaminophen is used to relieve pain and reduce fever.

4. Salicylates can make this condition (Gout) worse and can also lessen the effects of some medicines used to treat gout.

It is obvious that effects of substances are not limited to increasing or decreasing the severity of a side-effect. In fact, example (2) raises another issue of classifying types of effects. In the example, it is ambiguous between whether the chance of the serious-side-effects of liver disease is increased, or whether the chance of all-side-effects of liver disease is seriously increased. The ambiguous nature of MEDLINE led us to the conclusion that we could not draw lines between serious effects, mild effects, and inconsequential effects without creating even more ambiguity. For examples like (2), we classified the interpretation as an increased chance that all liver disease side effects will occur. The representation for like examples is as follows:

```
(increase|decrease (chance-of event))
```

However, the `chance-of` operator is still not enough to represent the MEDLINE data. Example (3) describes effects that do not deal with the chance of an event; the effects act on already occurring effects in the patient. We originally created a scheme that indicated a "higher" and "lower" state of a side-effect by using `HI` and `LO` operators on side-effects. This coding is too confined, however, and only describes three states; LO, normal, and HI (where having HI implies having both normal and LO). We decided we needed a more encompassing coding and created the generic `severity-of` operator which acts as the `chance-of` operator and can be both increased and decreased:

```
(increase|decrease
    (severity-of condition))
```

This representation allows us to give the reasoning system more leeway in how to deal with different severity levels. Unfortunately this was still not enough to capture all of the knowledge in MEDLINE. Example (4) describes a situation where Salicylates do not directly alter any side-effects, instead they alter the effectiveness of other medicines that alter side-effects. There is no way of capturing this using `chance-of` and `severity-of`. We need another modifier that maps onto how effective substances are on their listed side-effects. As a result, we use the `effectiveness-of` operator:

```
(increase|decrease
    (effectiveness-of substance))
```

This and the other two clauses above are the main *effect* representations for the MEDLINE database.

3.5 Prescribed Use and Dosing

After formalizing the representation for drug interactions and side-effects, the final task was to represent the section of MEDLINE which describes proper dosing for the drug substances. The

```
(PUR
    :action A
    :form F
    :rate
        (rate
            :quantity Q
            :time-interval T)
    :restrictions R
    :missed-dose-action M
    :effect E)
```

Figure 4: Prescribed Use Rule schema

schematic form of our Prescribed Use Rule is shown in Figure 4. The action (A) and form (F) fields are trivial, and we have already discussed the effect (E) representation already. The missed-dose-action M takes one of several constants describing the action to take if the patient has forgotten to take a dose (`take-now-skip-if-close`, `discuss-with-doctor`, *etc.*). The rate field describes both the amount to take on each dose, and the time range between each dose on any given day.

> "1 to 2 milligrams (mg) every eight to twelve hours as needed"

A straightforward encoding would look like the following:

```
(rate :quantity (MG (range 1 2))
      :time-interval (hours (range 8 12)))
```

However, many substances do not define time intervals, but are centered around other events, such as:

> "1 to 2 milligrams (mg) before bedtime."

There are several other cases involving timing constraints, many of which appear in the Medication Advisor's scheduler implementation described later but, very briefly, we developed a richer `:time-interval` format that allows for time specifications around events:

```
:time-interval (before|after|during event time)
```

4 Pronunciation modeling

Two important components of the speech recognition system are the language model and the lexical model (also called pronunciation dictionary). In our previous work on spoken dialogue applications, we have devised a technique for quickly building language models for new task domains that has delivered very good performance [Galescu *et al.*, 1998]. In the past, however, we didn't have to worry about the lexical model. Large machine readable pronunciation dictionaries have been developed for large vocabulary speech recognition systems (*e.g.*, the CMU dictionary [Weide, 1998]), and they provided the coverage required for our applications; for the few words not covered it was easy to add new entries. In the Medication Advisor domain, however, we found a larger than usual proportion of words that cannot be found in existing dictionaries. Moreover, as new drugs are continuously marketed, no static pronunciation dictionary is ever going to be satisfactory. Nonetheless, recognizing drug names, their ingredients, and the names of various diseases is crucial for this application, the rarity of those words notwithstanding.

The seriousness of the problem is compounded by the fact that, even if a canonical pronunciation can be obtained from a specialist, it is not safe to assume that uninformed users of the system will necessarily know the proper pronunciation. In previous research it has been shown that adding non-canonical pronunciations in the lexical model helps improve the speech recognizer's performance (for a review of the literature on modeling pronunciation variation, see [Strik and Cucchiarini, 1999]). We expect that the same will be true in the Medication Advisor domain, where many specialized words will be unfamiliar to users. Our underlying hypothesis is that, although many medical terms don't follow the typical grapho-phonotactic rules of English, these may be followed by users in their interaction with the system.

Therefore, the solution we propose is to use a grapheme-to-phoneme (G2P) conversion technique to obtain pronunciations for words outside our base lexical model. To account for multiple pronunciations that might be encountered by the system and for possible errors in the output of the G2P converter, we add several pronunciations for each lexeme. This procedure is applied both to out-of-dictionary words and to uncommon medical terms that are included in the dictionary, in order to obtain alternative pronunciations.

We have developed a statistical model for G2P conversion based on non-uniform units, which showed excellent performance in our tests [Galescu and Allen, 2001]. Trained on more than 100,000 words from the CMU dictionary, it achieved about 72% word accuracy and close to 94% phoneme accuracy when tested on unseen data. In Table 1 we give examples of pronunciations obtained automatically with this method for a few drug names in our lexicon that are not included in the CMU dictionary. Table 2 shows examples of alternative pronunciations obtained automatically; the top pronunciation is the one included in the CMU dictionary.

Although we use a phone set (based on ARPAbet) containing only non-accented phonemes, it can be observed that much of the variation in pronunciation can be traced to differences in stress assignment. Another factor is vowel quality, a known source of errors in grapheme-to-phoneme conversion.

We expect that the grapho-phonotactic knowledge gleaned out of a general purpose dictionary will be suboptimal for the medical domain. We plan to improve the quality of our models by learning automatically domain-specific constraints from medical dictionaries that give guides to pronunciation (*e.g.*, `http://www.cancer.gov/dictionary`). For example, in Table 2, pronunciations under the heading "Alternative (1)" are obtained from the word OSTEOPOROSIS. Additional alternatives (listed under the heading "Alternative (2)" in Table 2) can be obtained by running the G2P converter on the string "OSS-tee-oh-pa-ROW-sis", which is the pronunciation indicated by the National Cancer Institute on their website.

Additional improvements may be obtained by associating probabilities to alternative pronunciations and adapting them dynamically; this way we can personalize the pronunciation dictionary to

LASIX	L AA S IH K S
	L EY S IH K S
NIZORAL	N IH Z AO R AH L
	N IH Z ER AH L
	N AY Z AO R AH L
	N AY Z ER AH L
SALICYLATE	S AE L AH S IY L EY T
	S AA L IY S AH L EY T
	S AH L IY S AH L EY T

Table 1: Examples of pronunciations obtained automatically with the G2P converter.

In dictionary	AA S T IY AA P ER OW S IH S
Alternative (1)	AA S T IY AA P ER OW S AH S
	AO S T IY AA P ER OW S IH S
	AO S T IY AA P ER OW S AH S
Alternative (2)	AO S T IY OW P AH R OW S IH S
	AA S T IY OW P AH R OW S AH S

Table 2: Examples of alternative pronunciations obtained automatically with the G2P converter for the word `OSTEOPOROSIS`.

specific users and correct the possible errors introduced by our G2P converter.

5 Interpretation

5.1 The Lexicon and Ontology

Our development strategy is to keep the language interpretation components in TRIPS as domain-independent as possible so that it is (relatively) straightforward to develop systems in different task domains. In particular, we maintain a generic parser and lexicon and have developed a customization method to rapidly specialize these domain-independent components to new domains [Dzikovska *et al.*, 2002].

Prior to the development of the Medication Advisor, the generic parser in TRIPS was tested mainly on transportation-related domains. While the existing parser provided sufficient natural language coverage for planning and conversational interaction, some additions were required to handle utterances specific to the Medication Advisor domain. For example, we augmented our existing lexicon as needed with relevant lexical entries, including new words such as names for medications, symptoms, and medical conditions, as well as new senses for existing words, such as a `LF_TAKE-ACQUIRE` sense for *take* to handle utterances such as (1) in the challenge dialogue in Figure 2, and a similar sense for *have*, as in "I like to have an antacid before bedtime."

To specialize the lexicon for the Medication Advisor, we used the customization method we have developed for adapting our wide-coverage, domain-independent parser to specific domains. The generic parser uses a domain-independent ontology (the LF ontology) to process the natural language input, and the reasoning components use a domain-specific ontology (the KR ontology) that is optimized for reasoning in the medication domain, using specialized concepts as described in Section 3.2. We define mappings between the semantic representations in the LF ontology and the domain-specific representations in the KR ontology that allow us to specialize the lexicon by identifying domain-specific word senses (such as the `LF_TAKE-ACQUIRE` sense for *take*) which we use to tighten selectional restrictions on arguments and increase preferences for domain-specific senses. This method of specializing the lexicon to the domain substantially improves parsing speed and accuracy compared to the generic lexicon. The mappings also refine how the language interpretation components communicate with the back-end reasoners by specifically tailoring the parser output to the needs of the reasoning components built for the Medication Advisor domain.

This division between generic and domain-specific information allows the parser and grammar to be used across domains with no changes other than additions for new words and word senses as needed, and the domain-specific information is obtained from the KR ontology. The advantage of separating domain-independent and domain-specific information is illustrated with the following example. Our transportation and medication scheduling domains both deal with people. In the transportation domain, the most important information regarding physical objects is whether they can be moved, so people are a subclass of cargo. In the medication domain, however, people are living beings who receive treatment for medical conditions, and there is no notion of cargo. To keep the parser stable across domains, the LF ontology only makes domain-independent distinctions (*e.g.*, people are living beings), and the parser obtains domain-specific information for the lexicon by mapping it to the domain representation. In both domains, `LF_PERSON` will be mapped to the person class, but in the transportation domain PERSON will be defined as a subclass of CARGO, while in the medical domain PERSON is a subclass of AGENT.

This division in information representation allows us to organize concepts differently for parsing as opposed to domain-specific reasoning. For example, from the point of view of the language ontology, medication names have similar distributions across syntactic contexts, so they are represented as leaves under the LF_DRUG type, *e.g.*, LF_DRUG*celebrex, LF_DRUG*aspirin. In contrast, the KR ontology distinguishes between medications in a way appropriate to the medication domain, such as prescription vs. over-the-counter medications, creating a set of divisions that is not necessary to replicate on the language side. The transform below maps any word identified as LF_DRUG into a class with the same name as its canonical lexical form (:lf-form, *e.g.*, aspirin), provided that the resulting class is a descendant of the MEDICINAL-SUBSTANCE class:

```
(LF-to-frame-transform drugs
                        :preds (LF_drug :lf-form)
                        :lf-form-default MEDICINAL-SUBSTANCE)
```

Thus, a single template mapping for all LF_DRUG children converts the generic representation produced by the parser into a more specific representation suited to the medication domain.

We specialize the generic lexicon to the domain by pre-processing every lexical entry to integrate domain-specific information from the KR ontology by determining all possible mappings that apply to its LF. For each domain-specific mapping that is found, we create a new sense definition identical to the generic definition with the addition of a new feature, KR-TYPE which is the KR ontology class that results from applying the mapping to the lexical entry. We then propagate type information into the syntactic arguments, making tighter selectional restrictions in the lexicon.

We also improve parsing speed and accuracy by increasing the preference values of the word senses for which mapping rules were applied. All entries in the lexicon are assigned a preference value which determines the order in which they are attempted by the parser's beam search algorithm. Entries with LFs that have domain-specific mappings have higher preference values so they are tried first, allowing the parser to find a correct domain-specific interpretation more quickly. This also serves to suppress word senses that do not apply in the current domain. This is especially helpful for words such as *take*, which have a large number of senses, resulting in a high degree of ambiguity that impedes parsing. For example, in the Medication Advisor domain, the LF_TAKE-ACQUIRE sense of *take* as in (1), is strongly preferred and will be considered first, as opposed to, say, the transport sense of *take* as in "Take the oranges to Avon," which is the preferred sense in the transport domain.

5.2 Adjectives and Adjective Modification

The medication domain also required an improved representation of adjectives. In the transportation domain, we needed only a basic handling of adjectives. If adjectives referring to the same property are represented as leaves of the same LF (e.g., LF_INTENSITY*mild, LF_INTENSITY*severe) it is not possible, for instance, to indicate explicitly whether the adjective is gradable, or to associate pairs of antonymic adjectives together into an adjective scale. While this schema was adequate for distinguishing between possible referents in a transportation domain (e.g., "use the red truck"), it fails on some language the system must capture for the medical domain. Extending the system to a medical domain has been the impetus for bringing the adjectives in the generic TRIPS parser closer in line with linguistic theories about their meaning and behavior.

Specifically, we believe we must encode adjectives as scales of antonymic pairs to provide the lexical semantic information necessary to reason about language where adjectives play a key role. The examples in section 3 show that the knowledge representation has to rate the side effects in terms of severity. Correspondingly, the Medication Advisor needs to capture the linguistic knowledge that, in this case, the antonymic pair *mild* and *severe* defines an abstract scale of values. Moreover, since the scale itself is valid in different domains, this is information that needs to be encoded in the generic lexicon. To address this issue, we have recently developed an interpretation of adjective scales similar to the one provided in [Rusiecki, 1985], and we are in the process of implementing this system within the TRIPS parser.

5.3 Interpretation and Reference Resolution

Besides these issues, we found two new challenges for interpretation in the Medication Advisor domain. One is an increased occurrence of reference to kinds rather than individuals. Medicinal substances are of course frequently mentioned in this domain, and often they must be interpreted as kinds, as in e.g. dialogue sentence (2). Our previous logical representation could specify individual instances as variable bindings for referring expressions, but not kinds, so we added a new logical form to the Interpretation Manager output to handle kind reference. For example, an individual instance of Celebrex is represented as `(A x (TYPE x CELEBREX))`, while the kind Celebrex is represented as `(KIND x (TYPE x CELEBREX))`.

The second interpretation challenge in this domain is that the language used to speak about medical conditions and treatments abounds with metonymy (mentioning one item in a conventional way to refer to a related item), e.g., “I need to know when to take my prescription,” where *prescription* is a metonymic reference to the speaker’s prescribed medication. Metonymic reference must be corrected by the system interpretation components or down-stream reasoners will fail. The back-end processors cannot schedule a `TAKE-MEDICINE` event using a `PRESCRIPTION` object since they expect a `MEDICATION` object, so planning will fail. Parsing these sentences is problematic because the arguments violate semantic restrictions for their argument positions. The parser must be augmented to accept certain semantic restriction violations while still disallowing others that are truly unacceptable, such as “*I need to know when to take my shoe.”

Our solution is to create a set of inference rules that specify the relationships between allowable semantic types of arguments in specific domain actions. Using the inference rules, the interpretation of “my prescription” in “I need to know when to take my prescription,” can be transformed into “my medication.” Below is an example of an inference rule specifying that the parser can accept a prescription in an argument position where a medication is expected. The rule also specifies which relation name (`prescribed-on`) will be used to find the correct object during reference resolution:

```
(define-operator prescribed-on
                 :return medication
                 :arguments prescription)
```

Rather than creating a literal interpretation of the referring expression as a prescription, which would be represented in the logical form as:

```
(the x (and (owner x USER) (type x prescription))
```

The new logical form includes a position for the hidden (relation) argument, and specifies that the parser wants a medication to fill the argument position. The *coerce* operator is used to ask reference resolution if it has a way to turn the user's prescription into an appropriate variable binding that is a medication:

```
(the x (and (owner x USER) (type x prescription)))
(the y (and (type y medication) (coerce prescribed-on x y)))
```

Reference resolution uses this form to find the salient prescription object, and uses that object in the query to find the prescribed medication:

```
(and (type y medication) (prescribed-on y PRESCRIPTION1))
```

It uses the result to create the final logical form for the referring expression that contains the expression's binding:

```
(the x (and (type x medication) (refers-to x CELEBREX)))
```

6 Plan and Intention Recognition

A vital step of dialogue understanding in the Medication Advisor domain is intention recognition. Intention and plan recognition go beyond *what* was said and look at *why* it was said. What did the user hope to accomplish by uttering this? What prompted him to utter it?

Intention recognition also goes beyond simple pattern matching on sentences, since similar language patterns can be based on different underlying user goals and need to be handled in very different ways. Take, for example, the following two utterances, the second of which is from our challenge dialogue in Figure 2:

1. "Can I take a Celebrex?"

2. "Can I take an aspirin?"

At the surface level, these utterances are very similar, differing only in the name of the medication. Depending on the context, however, the user's *underlying goals* behind these utterances, can be very different.

Let us assume, for this example, that we know that the user has a prescription for Celebrex, but does not have a prescription for aspirin. Because we know that the user has a prescription for Celebrex, we can infer that the intention behind the first utterance is that he is querying about a previously-defined medication schedule. Knowledge of the underlying goals gives back-end reasoners the context they need to answer the user's query. Such reasoning may include the following:

According to the user's medication schedule, is it time to take the Celebrex? Have the prerequisites for taking the medication safely (*e.g.* no alcohol, full/empty stomach, *etc.*) been satisfied?

The second utterance, on the other hand, though structurally similar, has a very different underlying user intention. Here, the user does not have a prescription to take the substance, so presumably, there is some sort of other motivation for asking about taking the medication. It may turn out (as it does in our sample dialogue, see Section 6.3 below) that the user has a headache, and desires to take aspirin to get rid of the symptoms. This intention requires a very different response from the system than that of the first utterance.

In the rest of this section we discuss how intentions are represented in the system and discuss the acquisition of a domain plan library for the Medication Advisor domain. Finally we give a further example of how plan and intention recognition are necessary within the system.

6.1 Communicative Intentions

Intention recognition is the process of inferring what the user intended us to understand by his utterance. The more general process of *plan recognition* is the inferring of a user's plans based on what we have observed (regardless of whether we were intended to recognize the plans or not.)

We model communication as *collaborative problem solving* between the user and the system. Together, both agents collaboratively plan and act in order to achieve their objectives. Communicative intentions represent how each agent tries to individually affect the joint problem-solving state. Together, the system and user do things like adopting objectives, formulating and choosing recipes, and creating and executing solutions. The details of our collaborative problem solving model can be found in [Allen *et al.*, 2002; Blaylock, 2002].

Implementation of the plan and intention recognizer is still a work in progress. Our preliminary plan-recognition system is described in [Blaylock, 2001]. We are currently expanding this recognizer to perform intention recognition using our collaborative problem-solving model.

6.2 Domain Plans

Our collaborative problem-solving model is domain independent. It utilizes a domain-dependent plan library, which provides the system with knowledge of domain-level objectives, recipes, and resources. We represent plans similar to [Pollack, 1986], who separates objectives from the recipes intended to achieve them. Recipes may be composed of atomic actions (which are directly executable by the system), or subobjectives. This means that plans can be hierarchical.

In order to get an idea of what kinds of objectives users might have in the Medication Advisor domain, we analyzed a set a questions from an online medication FAQ. We expected that the domain would be sufficiently circumscribed and that there would be a relatively small number of domain plans involved. It turned out, however, that the domain held a large number of plans. For example, some representative objectives found in the FAQ were:

- Evaluate possible drug interactions
- Take over-the-counter drug

- Build medication schedule
- Add new drug to the schedule
- Modify schedule
- Query schedule
- Stop undesirable symptom
- Get recommendation for over-the-counter drug
- Assess symptom as possible side-effect
- Get recommended action for symptom
- Query general drug information
- Define medical terminology
- Suggest prescription medication

We are currently coding a subset of these objectives (those which our back-end components can handle), together with corresponding recipes, to be used with the plan/intention recognition system.

6.3 Plan/Intention Recognition Example

We analyze here, the first few exchanges of the sample dialogue in Figure 2 in order to show how plan and intention recognition proceed in the system. We reprint part of the dialogue here for the reader's convenience.

U:	Can I take an aspirin?	(1)
S:	No, you are taking Celebrex and Celebrex interacts with aspirin.	(2)
U:	Oh.	(3)
	Could Celebrex cause headaches?	(4)
S:	No, headaches are not an expected side-effect of Celebrex.	(5)
	Do you have a headache?	(6)

The user utters utterance 1, which the system must interpret. Intention recognition sees that the user does not have a prescription for aspirin, but that aspirin is an over-the-counter drug. The communicative intention inferred is that the user and system jointly evaluate the domain plan of the user taking an over-the-counter drug, namely aspirin. With this intention now known, the rest of the system (back-end reasoners) determines that it is not a good idea for the user to take aspirin, since he is taking Celebrex and aspirin and Celebrex interact. This is generated and becomes utterance 2 by the system.

The user does not seem to have intended that the system understand *why* he wants to take aspirin. However, through plan recognition, the system realizes that taking an over-the-counter drug is not an end goal, or an objective unto itself (*cf.* [Kautz, 1990].) Rather, it is a means towards a higher-level objective. However, there is not yet enough information for the system to make an inference.

The user then says utterance 3, which grounds utterance 2, followed by utterance 4. Intention recognition recognizes this as an intention to identify some fact in the situation, specifically, if headaches are a side-effect of Celebrex. Back-end reasoners find that this is not the case, and the system says utterance 5.

At the same time, utterance 4 gives plan recognition enough information to make a guess about the user's higher-level goals. In the system's plan library, the plan on taking an over-the-counter medication is part of a larger objective to relieve a symptom. The system knows that aspirin can relieve headaches and infers that the user's higher-level objective is possibly relieving a headache. However, the system is not sure about this inference, so it utters utterance 6 to verify it. Such plan recognition enables the system to be helpful (*cf.* [Allen, 1983a].) Later in the exchange the system, based on this inference, is able to take initiative and suggest that the user take Tylenol for his headache.

Intention recognition is vital to dialogue, since it recognizes what the user intended for the system to understand. Even in a question-answer system such as the Medication Advisor, pattern matching and form filling on utterances is not sufficient to provide understanding, since structurally similar utterances can have very different underlying intentions. Plan recognition, although not vital to systems, allows them to be much more helpful and proactive.

7 Scheduling

The goal of the scheduling component of the Medication Advisor is to generate an optimal, safe, and dynamically adaptive schedule given necessary information about the patient's prescriptions. Input to the scheduler consists of two types of information:

1. Information regarding prescribed medications (name, dosage and rate, *etc.*);

2. Information describing constraints on taking the medication, including timing constraints (*e.g.*, "take before bed"), activity constraints (*e.g.*, "do not take with food"), and drug interactions, including with over-the-counter drugs.

Additional scheduler input comes in the form of events such as the patient missing a dose.

From this information, the Medication Advisor scheduler tries to generate the optimal schedule for patients to take their medicine, taking into account all known constraints. If a new prescription is added, the system should adjust its schedule to satisfy the new constraints if needed. In addition, some of the constraints are "soft", such as "It is preferable to take Medicine A at least 3 hours before a meal". The system should be able to find the best schedule which satisfies as many soft constraints as possible. Finally, when some dynamic changes happen, the system should be able to adjust the schedule to adapt to the new conditions.

In implementing the initial version of the Medication Advisor scheduler, we need to solve the following problems:

1. How to represent time?

2. How to represent various constraints?

3. How to (efficiently) compute the schedule given the constraints?

4. How to deal with the soft constraints?

5. How to adjust the schedule to adapt to dynamic changes?

The remainder of this section describes these problems and our initial solutions to them in more detail.

7.1 Representing Time

The literature on the nature and representation of time is full of different theories—continuous point-based models based on the real number line, discrete time, interval logics, branching time, and the like [Shoham, 1987]. In the area of temporal reasoning, the two chief kinds of representation of time is period-based representation, and point-based representation. The period-based representation, which is also called interval-based representation, is proposed in [Allen, 1983b]. Allen proposed 13 relations to denote all the possible relationships between two intervals. That theory is used widely in temporal reasoning and other artificial intelligence areas. It is very helpful to denote the qualitative temporal relations in the constraints, such as “medicine A must be taken before bedtime”. And the reasoning algorithm proposed in [Allen, 1983b] can maintain the qualitative temporal relations when new constrains are added.

However, since our goal is to generate a schedule, which is quantitative, we need some other representation. The representation should have the following features. First, it should be good at representing quantitative temporal information. Second, the representation of time should be suitable for the later temporal reasoning. Finally, it should also be good at representing the temporal relationships.

Our new representation is to divide one day into 48 units evenly, and use the 48 integers between 0 and 47 to denote each unit, which is a half of an hour. The most important feature of this representation is that it simplifies the following medication scheduling work greatly. Using this form, what we face in making the schedule is just the 48 integers. It make it much easier for us to deal with time. The other advantage of this form is that the final schedule is natural to people. Clearly, it is unreasonable to ask a patient to take a medicine at some exact time (*e.g.,* 2:20). We should give a slot of time for each time to take a medicine. Using this form, since each integer denotes a slot of time (a half of an hour), it satisfies the requirement very well. The only problem of this representation is that it can’t represent accurate time, such as a minute, 12:22 and so on. However, since generally the prescriptions of medication do not use accurate time at all, this is not a problem in this application.

7.2 Representing Events

An important aspect of the temporal content of prescriptions is that information is often expressed relative to prototypical events, such as breakfast, lunch, dinner, mealtime, bedtime, and so on. For example, “medicine A must be taken within 3 hours of lunch time,” “medicine B can be taken

6 hours before bedtime," and so on. Our initial approach was to use the following qualifiers to describe the time of an event:

```
:mealtime (within N hours)
:bedtime (before M hours)
```

which means the medicine should be taken within N hours of mealtime and also must be M hours before bedtime.

There are two main problems with this representation. The first one is ambiguity. Using that form, it is difficult to distinguish the following three meanings:

1. The time the medicine should be taken must be before bedtime and the space between that time and bedtime should be no less than k hours.

2. The time the medicine should be taken must be exactly at the time that is exactly k hours before bedtime.

3. The time the medicine should be taken must be before bedtime and the space between that time and bedtime should be no more than k hours.

It seems that they all can be denoted by "`:bedtime (before k hours)`". One solution to eliminate the ambiguity could be to add some more keywords in the denotation. For example, we might add `BeforeMore`, `BeforeExact`, and `BeforeLess` to distinguish those three conditions. But that would add more keywords to the representation, and it is not flexible either.

The other problem for this representation is that it is hard to denote complicated constraints. For example,

> "Medicine A should be taken 2 hours before lunch or 3 hours after lunch, and must be 4 hours before bedtime."

One solution to this problem is to add more specific keywords and some logical connectives such as `and` and `or` to expand the representation. Clearly, this would make the representation very complicated.

Instead, we chose to represent constraints on events in conjunctive normal form (*i.e.*, as a conjunction of disjuncts). Each primitive constraint is of the form

```
(event S E)
```

where *event* is one of a fixed set including `breakfast`, `lunch`, *etc.* The parameters *S* and *E* are the offset in time units from the start and end of the event, respectively. Supposing that $event$ occurs at some time t, the described interval spans from $(t + s)$ to $(t + e)$. Both s and e may be positive or negative integers. Thus

```
(lunch -2 2)
```

```
(prescription
    :med-name Name
    :rate N
    :separation Sep
    :quantity (Amt Unit)
    :sch-con S
    :med-con M
    :prefer P)
```

Figure 5: Prescription information schema

describes the interval from one hour (two half-hours) before lunch until one hour afterwards. Two special parameters are `L` (for "left"), which denotes the start of the day, and `R` (for "right"), which denotes the end of the day. So

```
(lunch L -4)
```

describes the interval from the beginning of the day until two hours before lunch. The actual times of the events for a particular run are specified as part of the input to the scheduler.

7.3 Representing Prescriptions

After deciding the representation of time, the next problem is to represent the content of prescriptions, including the constraints between various events. Figure 5 shows the schematic form used to express this knowledge.

The basic information associated with a prescription is the name of the drug (*Name*), how often it is taken per day (*Rate*), the minimum time between doses (*Sep*), and how much is taken each time (*Amt* units *Unit*, for example, "6 milligrams"). We should note this is not sufficient to capture the complete range of dosing information seen in MedLINE. For example, some medications are prescribed to be taken "as needed for pain." Others involve taking varying amounts depending on other circumstances. In any case, our representation can handle a fairly wide range of dosings, in particular those that are possible to schedule definitively.

If a prescription specifies that a medication should be taken at a specific time, we specify these "schedule constraints" using the `sch-con` slot. For example:

```
:sch-con (((lunch L -2) (lunch 2 R)) ((bedtime L -4)))
```

says that the medication must be taken at least one hour before lunch or at least one hour after (*i.e.*, not within an hour of lunch), and that it must be taken at least two hours before bedtime (recall that each constraint is a CNF expression).

If one medication should not be taken with another medication, we specify this using the `med-con` slot. For example, in the prescription for drug `A`, we might have:

```
:med-con (((B L -4) (B 4 R)))
```

This means that, subject to other constraints, A can be taken from the start of the day until two hours (four half-hours) before taking B, or from two hours after taking B until the end of the day. That is, A cannot be taken within two hours of taking B.

Finally, soft constraints (preferences) are expressed using the `prefer` slot. For example:

```
:prefer (((bedtime L -6)))
```

means that the best time for it to be taken is at least three hours (six half-hours) before bedtime.

It can be seen that this representation can denote complicated constraints very succinctly. Using our representation, the ambiguity mentioned above can be solved easily. The three conditions are represented as follows. They can be distinguished clearly:

1. The time the medicine should be taken must be before bedtime and the space between that time and bedtime should be no less than 3 hours.

   ```
   :sch-con (((bedtime L -6)))
   ```

2. The time the medicine should be taken must be exactly at the time that is exactly 3 hours before bedtime.

   ```
   :sch-con (((mealtime -3 -3)))
   ```

3. The time the medicine should be taken must be before bedtime and the space between that time and bedtime should be no more than 3 hours.

   ```
   :sch-con (((bedtime -3 0)))
   ```

We feel that this representation of time and constraints allows clear and concise translation of the information contained in prescriptions. It also supports the scheduling algorithms, which are described next.

7.4 Scheduling Algorithm

In this section, we describe the algorithms used in the system to make the best schedule given prescription constraints. Since this is clearly a constraint-satisfaction problem (CSP), the main algorithm used is constraint propagation with backtracking. After describing the basic approach, we will describe the extensions to soft constraints, as well as the approach adopted to handling dynamic information that affects the schedule.

7.5 Constraint Satisfaction Problems

Constraint satisfaction problems are a large class of problems that arise often in AI and other areas of computer science. The definition of a CSP can be stated as follows (following [Kumar, 1992]):

> We are given a set of variables, a finite and discrete domain for each variable, and a set of constraints. Each constraint is defined over some subset of the original set of variables and limits the combinations of values that the variables in this subset can take.The goal is to find one assignment satisfies all the constraints.

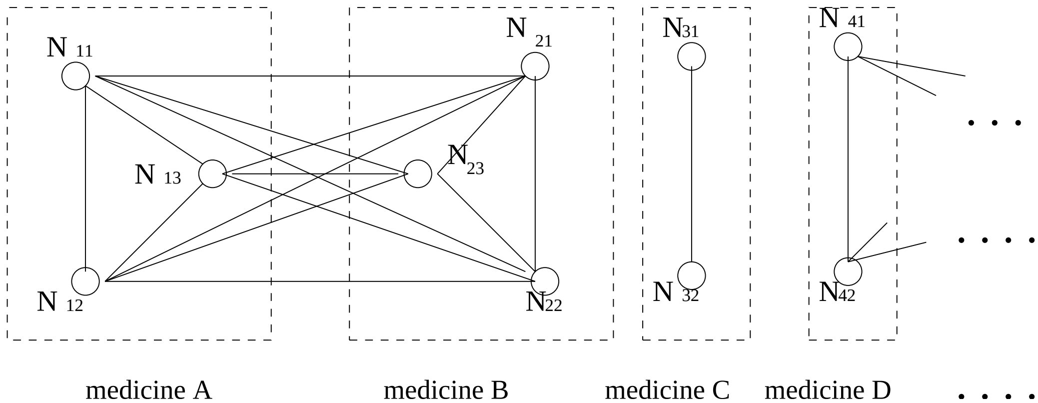

Figure 6: Abstract description of the medication scheduling problem

The problem of making a reasonable schedule is shown abstractly in Figure 6. Each box outlined by broken lines denotes one medicine. Each node denotes one time of taking a medicine. Each edge denotes the constraints between two nodes. The edges connecting two nodes in one box denote the constraints between two times of taking one medicine. In reality, the constraint of this type is like "the medicine can't be taken twice within 10 hours". The edges connecting two nodes in different boxes denote the constraints between two medicines. In the conditions shown in Figure 6, medicine A and medicine B are exclusive medicines, while medicine C has nothing to do with other medicines.

We say that the Medication Scheduling Problem is a typical CSP. Each node is a variable in the CSP. The possible value set for each variable is those times that this medicine can be taken according to those constraints about mealtime and bedtime. The constraints are those denoted by those edges mentioned above. Our goal is to find one assignment for all of the variables that satisfies all the constraints.

7.6 Constraint Propagation and Backtracking

Our method for solving the medication scheduling problem can be divided into four steps, shown in Figure 7. The second and third steps are the process of propagating constraints. Here, our constraint propagation is simple. We only check one edge once (two directions). Thus, we cannot guarantee that at last, each edge of the constraint graph is consistent. If we want to guarantee consistency, when we eliminate some values for some variable according to one edge constraint, we need to check those edges that we have already checked to make sure that they are still satisfied. However, through our once-check constraint propagation, we eliminate much of the value space and greatly improve the efficiency of the backtracking search. The reason for us to do only once-check constraint propagation is that the number of exclusive medicines in medication prescriptions is small and the total number of variables is also not large. If we do the complete constraint propagation, the efficiency won't improve much. Thus, in this application, we think once-check constraint propagation is enough. And this method is much simpler.

The sequence of assignment of variables in our backtracking search is to first assign all variables

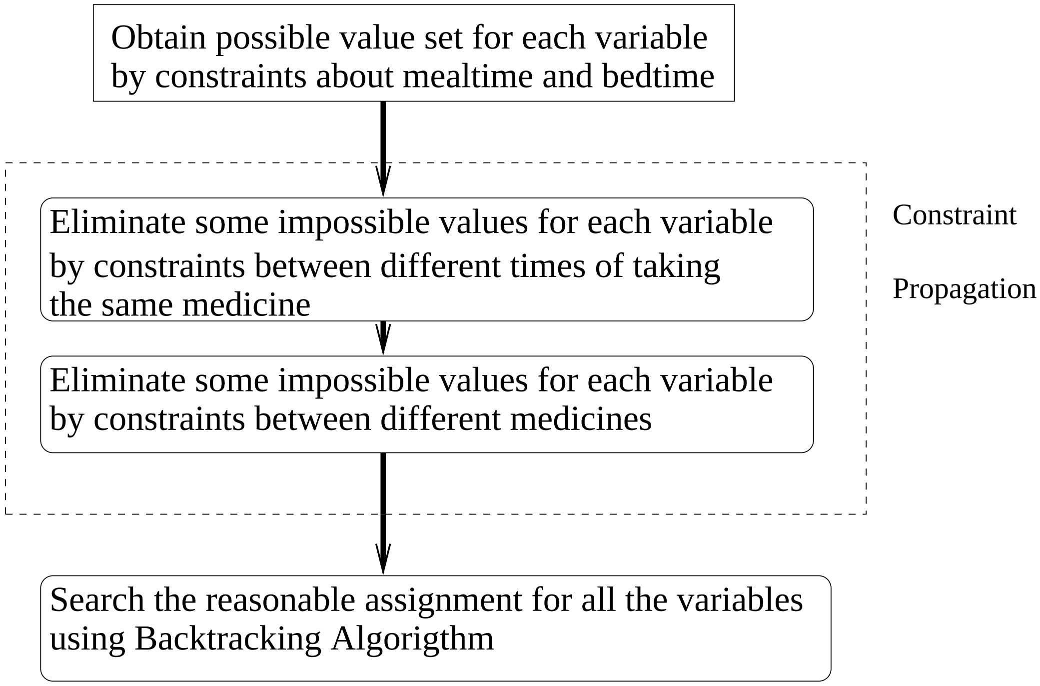

Figure 7: Steps in medication scheduling

in one box (one medicine) and then assign those variables in the other boxes. This sequence is easier to implement and not inefficient. There are some better sequences that can be chosen, such as first assigning those variables that have the fewest remaining alternatives. But since in this application there are not large number of variables, we choose the simpler strategy.

7.7 Handling Soft Constraints

As mentioned above, some constraints are "soft". It is better to satisfy these constraints if possible, but if they can't be satisfied they may be ignored. Soft constraints can arise from the properties of particular medicines, or from patients' habits or preferences. For example, "It is better if medicine A is taken 2 hours before bedtime" is one example of a soft constraint. In our system, we try to satisfy as many soft constraints as possible.

To do this, we modified our algorithm to construct two value sets for each variable in the first step (see Figure 7): *OptSet* and *CandSet*. The *OptSet* is composed of all possible values for the variables when soft constraints are considered, while *CandSet* is composed of all possible values for the variable without considering soft constraints. When making the schedule, we first try to make a schedule only using values in the *OptSet*s. If this proves impossible, we gradually replace more and more subsets of the *OptSet*s using some subsets in *CandSet*s until a reasonable schedule can be made. Since we try to use as many subsets of the *OptSet*s as possible, the final schedule will be the one that satisfies all the hard constraints and as many soft constraints as possible.

It should be noted that after finishing making one schedule, we need to recover those subsets

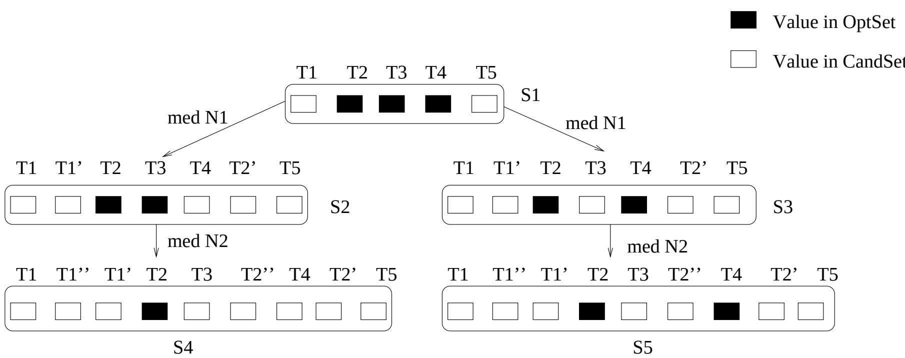

Figure 8: Problem in handling soft constraints

replaced in the *OptSet*s. Otherwise, when a new medicine needs to be added into the schedule, we may miss the optimal schedule. Figure 8 shows such an example. In the figure, a black-filled box denotes the time value in *OptSet*s, and the white box denotes the time value in *CandSet*s.

`S1` is an existing schedule. When a new medicine `N1` needs to be added into the schedule, the algorithm found schedule `S2` to be the optimal one. (It can be seen, in `S2`, `T4` has been replaced by a slot of time in *CandSet*). While, in fact, schedule `S3` is also an optimal schedule here. It also has two optimal slots of time as schedule `S2`. Our system first obtains schedule `S2`, so uses it as the optimal schedule. That is all right since no other one (including `S3`) is better than `S2`. However, if we do not recover the *OptSet*s, there will be a problem when another new medicine `N2` needs to be added into the schedule. If we try to make the new schedule based on the *OptSets* after making schedule `S3`, we can get schedule `S5`, which has two optimal slots of time. But since we choose `S2` in last step, we make our schedule based on the *OptSets* obtained after making schedule `S2` and so get schedule `S4` instead, which has only one optimal slot of time. The real optimal schedule is missed. Since it is impossible to predict which of schedule `S2` or schedule `S3` is better before `N2` is input, to solve that problem, we have two options. One is to store all *OptSets* for later use after making all possible optimal schedules. The other one is to recover the *OptSets* to the original content after making one schedule. It seems that the latter one is less efficient than the former one, since we need to make the schedule from scratch each time when some medicine needs to be added. However, there are two problems with the former method. One is that many sets need to be restored if there are many medicines and few constraints. When some new medicine needs to be added, we need to try to make out the new schedule based on all the restored sets. Thus, using the former method, we cannot get much time or space gains. So, in our system, we use the latter method, which is simpler.

7.8 Dynamic Schedule Adaptation

The scheduling system can adjust its schedule in light of certain changes, such as the patient forgetting to take a medicine at the right time. The system will try to find a proper time for him to take the missed medicine according to the current time and the medicine's requirement. This requirement is specified in MedLINE as something like:

> "Medicine A should be taken at once if missed. But if the time is less than 2 hours to take the next dose, you should skip it."

We input this to the system using an addition slot in the prescription information for a medication. This information should be input to the system. The input form is:

:missing N

This means that the medication should be skipped if it is within N half-hours of the time of the next medication. This does not cover all types of instructions seen in MedLINE, but it handles the most common cases.

Given this information, the scheduler will check if there is any time that satisfies all the constraints on the medicine and the missing requirement. If it finds one, it will tell the patient to take the medicine at that time. Otherwise, it will suggest the patient to skip the dose. The current system only checks if there is any suitable time for the patient to take the specific medicine while keeping the rest of the schedule (for other medications) unchanged. In the future, we will try to adjust the other schedules to look for a better solution.

7.9 Sample output of the medication scheduling system

The following traces show the results of the prototype medication scheduling system on a variety of inputs. In these examples, the times of events are fixed as follows:

breakfast	8:00-8:30
lunch	12:00-12:30
supper	18:00-18:30
bedtime	22:00

First we have a medication named A, for which six milligrams should be taken twice a day, with lunch and supper.

```
(makesch :med-name 'A :rate 2 :quantity '(6 mg)
         :sch-con '(((lunch 0 0) (supper 0 0))))
Medication Schedule:
A> 6MG (12:00 12:30) (18:00 18:30)
```

The schedule says that A should be taken in an mount of six milligrams between 12:00 and 12:30 and between 18:00 to 18:30 each day.

For medication B, the prescription is three milligrams twice a day. It can't be taken with medicine A within three and a half hours of A (seven half-hours), and it's preferable that it be taken with a meal:

```
(makesch :med-name 'B :rate 2 :quantity '(3 mg)
         :med-con '((A 7))  :prefer '(((mealtime 0 0))))
Medication Schedule:
A> 6MG (12:00 12:30) (18:00 18:30)
B> 3MG (8:00 8:30) (21:30 22:00)
```

For medication C, the prescribed dose is again three milligrams twice a day, and it cannot be taken within three hours (six half-hours) of A. It is also preferable that it be taken at least two hours before bedtime:

```
(makesch :med-name 'C :rate 2 :quantity '(3 mg)
         :med-con '((A 6))  :prefer '(((bedtime L -2))))
Medication Schedule:
A> 6MG (12:00 12:30) (18:0 18:30)
B> 3MG (8:00 8:30) (21:30 22:00)
C> 3MG (8:00 8:30) (15:00 15:30)
```

Note that in this case, the system cannot satisfy the preference that C be taken at least two hours before bedtime.

Next, medication D, three milligrams twice a day, not within one hour of A:

```
(makesch :med-name 'D :rate 2 :quantity '(3 mg)
         :med-con '((A 2)))
Medication Schedule:
A> 6MG (12:00 12:30) (18:00 18:30)
B> 3MG (8:00 8:30) (21:30 22:00)
C> 3MG (8:00 8:30) (15:00 15:30)
D> 3MG (8:00 8:30) (11:00 11:30)
```

Finally, medication E cannot be taken within four and a half hours of C:

```
(makesch :med-name 'e :rate 2 :quantity '(3 mg)
         :med-con '((c 9)) :separate 6
         :prefer '(((mealtime 0 0))))
Medication Schedule:
A> 6MG (12:00 12:30) (18:00 18:30)
B> 3MG (8:00 8:30) (21:30 22:00)
C> 3MG (8:00 8:30) (21:00 21:30)
D> 3MG (8:00 8:30) (11:00 11:30)
E> 3MG (12:30 13:00) (15:30 16:00)
```

We can see that the system adjusted the schedule for C automatically in order to incorporate E into the schedule.

7.10 Future Work in Scheduling

There are still many interesting problems to solve in order to fully address the scheduling requirements of the Medication Advisor system. One of these involves making the system not only be able to make a schedule of one day, but also a long-term schedule. There are some medicines that should be taken differently in different stage of time, such as "after 12 weeks, may be increased to 8 mg once daily." Another issue is when the specification of what to do on a missed dose has something to do with the next day's scheduling, such as "If scheduled for every other day, take when remembered and restart alternate day schedule." Another interesting challenge is to make the scheduler take account of the special conditions of our patients. For some medicines, the schedule could be different for different patients, such as "for patients who have difficulty chewing, take before a meal.". These and other extensions would make the scheduler more intelligent and more suitable for use in a full-blown Medication Advisor system.

8 Conclusions and Future Work

The Medication Advisor project is an application of dialogue systems to the important societal problem of helping people manage their medications. As we have described, it also presents a variety of challenges to our current understanding of how to build such systems. Among our major projects for the future are dealing with the metonymy and type-shifting that is ubiquitous in this domain, as well as extending the conversational assistant to be able to actively monitor patient compliance, thereby enabling a wide range of interesting and useful activities, including automatic reordering of medication and automated analysis and reporting to the physician. Each of these has their own set of challenges to be dealt with.

We believe that the Medication Advisor is just the tip of the iceberg in terms of providing a conversational assistant that can help people take care of their health in their homes. The Center for Future Health is leading the development of the Smart Medical Home, which will integrate a wide variety of sensors and effectors into Intelligent Assistive Technologies that address particular medical needs. As part of this broader vision, we intend to expand the Medication Advisor into a more general "Personal Medical Assistant," which will integrate the information provided by the various technologies and provide a personalized point of contact for the residents of the home. The goal is not to replace doctors, nurses, or pharmacists, since this is both technically difficult (if not impossible) as well as socially undesirable. Rather, we want to provide systems that can help people better manage *their* part of their health care, and connect them to health care providers, family members, and the broader community as appropriate. We are in the early going, of course, but the need is great and growing, and the future looks very interesting.

References

[Allen, 1983a] James Allen, "Recognizing Intentions from Natural Language Utterances," In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*, pages 107–166. MIT Press, 1983.

[Allen *et al.*, 2002] James Allen, Nate Blaylock, and George Ferguson, "A Problem-Solving Model for Collaborative Agents," In *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, Bologna, Italy, July 31–August 2 2002, To appear.

[Allen *et al.*, 2000] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent, "An Architecture for a Generic Dialogue Shell," *Natural Language Engineering*, 6(3 and 4):213–228, September 2000.

[Allen *et al.*, 2001] James Allen, George Ferguson, and Amanda Stent, "An Architecture for More Realistic Conversational Systems," In *Proceedings of the Conference on Intelligent User Interfaces (IUI-2001)*, pages 1–8, Santa Fe, NM, January 14-17 2001.

[Allen, 1983b] James F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, 26(11):832–843, 1983.

[Allen *et al.*, 1995] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum, "The TRAINS Project: A Case Study in Defining a Conversational Planning Agent," *Journal of Experimental and Theoretical AI*, 7:7–48, 1995, Also available as TRAINS Technical Note 93-4, Department of Computer Science, University of Rochester.

[Blaylock, 2001] Nate Blaylock, "Retroactive Recognition of Interleaved Plans for Natural Language Dialogue," Technical Report 761, Department of Computer Science, University of Rochester, December 2001.

[Blaylock, 2002] Nate Blaylock, "Managing Communicative Intentions in Dialogue Using a Collaborative Problem-Solving Model," Technical Report 774, Department of Computer Science, University of Rochester, 2002.

[Claxton *et al.*, 2001] A. J. Claxton, J. Cramer, and C. Pierce, "A Systematic Review of the Associations between Dose Regimens and Medication Compliance," *Clinincal Therapeutics*, 23(8):1296–1310, August 2001.

[Dzikovska *et al.*, 2002] Myroslava Dzikovska, James Allen, and Mary Swift, "Finding the Balance between Generic and Domain-specific Knowledge: A Parser Customization Strategy," In *Proceedings of the Workshop on Customizing Knowledge in NLP Applications (LREC-02)*, Las Palmas, Spain, May 28 2002, To appear.

[Ferguson *et al.*, 1996] George Ferguson, James Allen, and Brad Miller, "TRAINS-95: Towards a Mixed-Initiative Planning Assistant," In Brian Drabble, editor, *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 70–77, Edinburgh, Scotland, 29–31 May 1996.

[Ferguson and Allen, 1998] George Ferguson and James F. Allen, "TRIPS: An Integrated Intelligent Problem-Solving Assistant," In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 567–573, Madison, WI, 28–30 July 1998.

[Galescu and Allen, 2001] Lucian Galescu and James Allen, "Bi-directional Conversion Between Graphemes and Phonemes Using a Joint N-gram Model," In *Proceedings of the 4th ISCA Tutorial and Research Workshop on Speech Synthesis*, Atholl Palace Hotel, Perthshire, Scotland, August 29 - September 1 2001.

[Galescu *et al.*, 1998] Lucian Galescu, Eric Ringger, and James Allen, "Rapid Language Model Development for New Task Domains," In *Proceedings of the First International Conference on Language Resources and Evaluation (LREC)*, Granada, Spain, May 1998.

[Gross *et al.*, 1993] Derek Gross, James F. Allen, and David R. Traum, "The TRAINS-91 Dialogues," TRAINS Technical Note 92-1, Department of Computer Science, University of Rochester, Rochester, NY, 14627, June 1993.

[Heeman and Allen, 1995] Peter A. Heeman and James F. Allen, "The TRAINS-93 Dialogues," TRAINS Technical Note 94-2, Department of Computer Science, University of Rochester, Rochester, NY, March 1995.

[Kautz, 1990] H. Kautz, "A Circumscriptive Theory of Plan Recognition," In P. R. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 105–134. MIT Press, Cambridge, MA, 1990.

[Kumar, 1992] Vipin Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey," *AI Magazine*, 13(1):32–44, 1992.

[Labrou and Finin, 1997] Yannis Labrou and Tim Finin, "A Proposal for a New KQML Specification," Technical Report CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997.

[Pollack, 1986] Martha Pollack, "Inferring Domain Plans in Question-Answering," Technical Report MS-CIS-86-40 LINC LAB 14, University of Pennsylvania, May 1986, PhD thesis.

[Rusiecki, 1985] Jan Rusiecki, *Adjectives and Comparison in English*, Longman, 1985.

[Shoham, 1987] Yoav Shoham, "Temporal Logics in AI: Semantical and Ontological Considerations," *Artificial Intelligence*, 33(1):89–104, 1987.

[Smith, 1989] D. Smith, "Compliance Packaging: A Patient Education Tool," *American Pharmacy*, NS29(2), February 1989.

[Strik and Cucchiarini, 1999] Helmer Strik and Catia Cucchiarini, "Modeling Pronunciation Variation for ASR: A Survey of the Literature," *Speech Communication*, 29:225–246, 1999.

[Weide, 1998] R. Weide, "The CMU Pronunciation Dictionary, release 0.6," 1998.

[Zuger, 1998] Abigail Zuger, "The 'Other' Drug Problem: Forgetting to Take Them," *The New York Times*, June 2:Section F, Page 1, Column 2, 1998.