# Modelling Space and Time in Narratives about Restaurants

**Erik T. Mueller**
IBM Thomas J. Watson Research Center, PO Box 704,
Yorktown Heights, NY 10598, USA

## Abstract

This study investigated the automatic modelling of space and time in narratives involving dining in a restaurant. We built a program that (1) uses information extraction techniques to convert narrative texts into templates containing key information about the dining episodes discussed in the narratives, (2) constructs commonsense reasoning problems from the templates, (3) uses commonsense reasoning and a commonsense knowledge base to build models of the dining episodes, and (4) generates and answers questions by consulting the models. We describe the program and present the results of running it on a corpus of web texts and American literature.

**Correspondence:**
Erik T. Mueller
IBM Thomas J. Watson
Research Center,
PO Box 704,
Yorktown Heights,
NY 10598, USA.
**E-mail:** etm@us.ibm.com



## 1 Introduction

Starting in the 1970s, a number of computer programs have been written that attempt to perform automatic narrative comprehension, which involves reading a narrative text, understanding it, and then answering questions of the sort a person would be able to answer after reading the text (Charniak, 1972; Schank and Riesbeck, 1981; Dyer, 1983; Norvig, 1989; Miikkulainen, 1993; Hobbs *et al.*, 1993; Schank, Kass, and Riesbeck, 1994; Shapiro and Rapaport, 1995; Narayanan, 1997; Frank *et al.*, 2003).[1] Each such program can handle only a small number of short texts (McCarthy *et al.*, 2002), which is not surprising given the complexity of the understanding task. As van Dijk and Kintsch (1983) put it, 'It borders on the miraculous how many subtasks people perform, how many points they keep track of, and how many constraints they respect when comprehending discourse—all normally without apparent effort, in a routine manner' (p. 333). What is more, understanding narratives requires an enormous amount of commonsense knowledge (Davis, 1990; Lenat and Guha, 1990).

In this study, we chose to narrow down the problem of building a narrative comprehension program in two ways. First, we limited the scope of understanding to those portions of a narrative that involve one or more characters dining in a restaurant. Dining is a common activity discussed in many narratives. Second, we focused on modelling the spatial and temporal aspects of restaurant dining at the expense of other aspects.[2] We chose to focus on these aspects because they are important to narrative comprehension. Narratologists define the content of a narrative ('narrated' or 'story') as a succession of events occurring to characters in time and space (Prince, 1982; Rimmon-Kenan, 2002). Readers view a narrative through a shifting window that represents the who, what, where, and when of the narrative as it unfolds (Duchan *et al.*, 1995). Psychological experiments confirm that readers can construct and update detailed spatial models of characters and objects (Morrow *et al.*, 1989; Wilson *et al.*, 1993; Rinck and Bower, 1995).

We have constructed a 13,000-line program that builds models of space and time in narrative texts involving dining in a restaurant. The program builds models by combining two

Literary and Linguistic Computing Vol. 22, No. 1, 2007. 
doi:10.1093/llc/fql014 Advance Access Published on 4 April 2006

techniques: information extraction (Cowie and Lehnert, 1996; Cardie, 1997) and commonsense reasoning (Lifschitz, 1990; Reiter, 2001; Mueller, 2006). It operates as follows:

(1) It uses information extraction to build a template—a frame with slots and slot fillers—containing key information about the dining episode discussed in the narrative.
(2) It constructs a commonsense reasoning problem from the template.
(3) It uses commonsense reasoning and a commonsense knowledge base to solve the reasoning problem and build a model of space and time in the dining episode.
(4) It generates and answers questions by consulting the model.

Consider this sample condensed narrative:

Nicole went to a vegetarian restaurant. She ordered lentil soup. The waitress set the soup in the middle of the table. Nicole enjoyed the soup. She left the restaurant.

Given this narrative, the program is able to generate and answer questions such as the following:

Q: What did Nicole walk through?
A: The main entrance to the vegetarian restaurant.
Q: Was Nicole standing before Nicole sat on a chair?
A: Yes.
Q: Was the cook present when Nicole ordered the lentil soup from the waitress?
A: No.
Q: Who walked through the kitchen door?
A: The waitress.
Q: Was the lentil soup in the kitchen before the waitress walked out the kitchen door?
A: Yes.
Q: Was Nicole hungry after Nicole ate the lentil soup?
A: No.
Q: What did the waitress set down?
A: The lentil soup and the bill.
Q: Was Nicole present when the waitress set down the bill?
A: Yes.

The model produced by the program represents time and space as follows: time is represented as a sequence of timepoints. Space is represented at two scales: room-scale and object-scale. In room-scale space, at each timepoint each character and object of the narrative is located in a particular location—room or outside area. From one timepoint to the next, a character or object may move from one location to another through a door. In object-scale space, at each timepoint characters and objects stand in certain relations to each other—a character may hold an object, or an object may be on top of another object.

The remainder of this article is organized as follows: in Section 2, we review previous work. In Section 3, we describe the development and test corpora used in this study. In Sections 4 through 6, we describe how our program builds models of space and time in narratives involving dining. In Section 7, we describe how our program generates and answers questions by consulting the models. In Section 8, we present an evaluation of the models produced by the program. In Section 9, we discuss the limitations of the program. Finally, we present our conclusions and discuss future work.

## 2 Previous Work

Our model-based approach derives from past work on mental models (Craik, 1943). Johnson-Laird (1983) and van Dijk and Kintsch (1983) have argued that readers understand a narrative by forming a mental model or situation model of the narrative. A number of researchers have investigated, expanded upon, and debated this view (Bower, 1989; Morrow *et al.*, 1989; McKoon and Ratcliff, 1992; Graesser *et al.*, 1994; van Oostendorp and Zwaan, 1994; Goldman *et al.*, 1999).

A previous program that processed narratives about dining in a restaurant was the SAM program (Cullingford, 1978; Schank and Riesbeck, 1981). This program used representations of stereotypical activities or scripts (Schank and Abelson, 1977; Abelson, 1981) to make inferences and fill in missing events. SAM answered questions about the narratives through the use of the QUALM program and model of question answering (Lehnert, 1978).

Some differences between SAM and our program are as follows: our program is robust and able to process an unlimited number of authentic texts touching on restaurant dining, while SAM was able to handle only several invented restaurant stories and several edited newspaper articles. Our program creates a more detailed spatial and temporal model of the script events. For example, our program represents that the waiter walks into the kitchen and picks up some food, and then walks back into the dining room and places the food on the table. SAM represented that the waiter performs an abstract transfer of possession (ATRANS) of the meal to the customer. Our program deals only with one script, whereas SAM could cope with several scripts active at a time.

Several other programs modelled space and time in a small number of narratives. The BORIS program (Dyer, 1983) used a knowledge structure called a scenario participant map to represent settings and the movement of characters from one setting to another. ThoughtTreasure (Mueller, 1998) used spatial occupancy arrays and time intervals to represent the location and movement of narrative characters and objects.

Numerous natural language processing programs are able to perform shallow parsing of spatial and temporal information: information extraction programs are able to extract the location (city or neighbourhood) and date of important events in news stories (MUC, 1991). Question-answering programs are able to extract information about the location and time from text in order to answer fact-based where and why questions (TREC, 2002).

## 3 Corpora

We developed our program using a development corpus and evaluated it on a web test corpus and a Gutenberg test corpus.

We created the development and web test corpora as follows: we downloaded from the Internet 800 texts likely to involve dining in a restaurant through the use of the Boolean query *'the menu' AND ('the waiter brought' OR 'the waiter placed' OR 'the waiter set' OR 'the waiter put' OR 'the waiter poured')*. We then randomly drew 400 texts and assigned them to the development corpus. We assigned the remaining texts to the web test corpus.

We created the Gutenberg test corpus by downloading thirty American literature texts (Library of Congress class PS) from the Project Gutenberg archive (Hart, 2005), such that the texts satisfied the Boolean query *'the waiter brought' OR 'the waiter placed' OR 'the waiter set' OR 'the waiter put' OR 'the waiter poured'*.

## 4 Information Extraction for the Restaurant Script

In this section, we describe the information extraction module of our program that converts a narrative text into a template for the restaurant script (Schank and Abelson, 1977). For example, given the sample narrative of Section 1, the information extraction module produces the following template:[3]

| | |
|---|---|
| SCRIPT: TYPE | RESTAURANT |
| SCRIPT: LAST EVENT | LEAVE |
| RESTAURANT | 'the vegetarian restaurant' |
| WAITER | 'waitress' |
| CUSTOMER | 'Nicole' |
| FOOD | 'lentil soup': 'Nicole' |

In general, a restaurant script template consists of the following slots and fillers:

**SCRIPT: TYPE** The type of script. The filler of this slot is always RESTAURANT.

**SCRIPT: LAST EVENT** The last event of the script that has been performed. One filler per template. The events of the restaurant script, in order, are: ARRIVE, SIT, READMENU, ORDER, SERVE, EAT, REQUESTBILL, RECEIVEBILL, PAY, and LEAVE.

**RESTAURANT** A name of the restaurant. Zero or more fillers per template.

**WAITER** The name of a waiter. Zero or more fillers per template.

**CUSTOMER** The name of a customer. Zero or more fillers per template.

**FOOD** A food ordered or eaten, cross-referenced to CUSTOMER, the name of the customer who ordered or ate the food. The filler and cross-reference are separated by a colon. Zero or more fillers per template.

**PAYING CUSTOMER** The name of the customer that paid the bill. Zero or more fillers per template.

**TIP** The size of the tip. Empty or one filler per template. Possible fillers are: ZERO, SMALL, NORMAL, and LARGE.

We perform information extraction as follows: we first feed the narrative text through components of the GATE natural language processing architecture (Cunningham *et al.*, 2002); we feed the text through the tokenizer, sentence splitter, part-of-speech tagger (Hepple, 2000), person name recognizer, and coreference resolver (Dimitrov, 2002). We then use pattern matching to recognize mentions of the following entities: bills, foods, menus, restaurants, tables, and waiters. We recognize foods and restaurants using word and phrase lists constructed using WordNet (Fellbaum, 1998). Finally, we use pattern matching to fill slots of the template.

We manually constructed 84 information extraction patterns for recognizing expressions related to the restaurant script, by using concordance tools on the development corpus. Sample patterns are shown in Table 1. When a pattern is matched, fillers are added to the slots of the template as specified by the actions associated with the pattern. The last event of the template is updated to the last event associated with the pattern or the last event already in the template, whichever is the later event.

## 5 Generation of Reasoning Problems from Templates

In this section, we describe the method used by our program to generate a commonsense reasoning problem from a restaurant script template produced by the information extraction module.

We first construct a room-scale space consisting of three locations and two doors. The locations are: (1) a street outside the restaurant, (2) a dining room

**Table 1** Sample information extraction patterns, associated actions, and associated last events for the restaurant script

| Pattern | Associated actions | Associated last event |
|---|---|---|
| Person *walked in/into* restaurant | CUSTOMER ← person<br>RESTAURANT ← restaurant | ARRIVE |
| Person *sat at* table | CUSTOMER ← person | SIT |
| Person *glanced/looked at* menu | CUSTOMER ← person | READMENU |
| Waiter *took* person's order | CUSTOMER ← person<br>WAITER ← waiter | ORDER |
| Person *ordered* food | CUSTOMER ← person<br>FOOD ← food: person | ORDER |
| Waiter *brought* person food | CUSTOMER ← person<br>FOOD ← food: person<br>WAITER ← waiter | SERVE |
| Waiter *returned with* food | FOOD ← food<br>WAITER ← waiter | SERVE |
| Person *ate/chewed/tasted* food | CUSTOMER ← person<br>FOOD ← food: person | EAT |
| Person *asked/called for* bill | CUSTOMER ← person | REQUESTBILL |
| Waiter *brought* bill | WAITER ← waiter | RECEIVEBILL |
| Person *paid/settled* bill | CUSTOMER ← person<br>PAYING CUSTOMER ← person | PAY |
| Person *refused to leave* tip | CUSTOMER ← person<br>TIP ← ZERO | PAY |
| Person *exited/left* restaurant | CUSTOMER ← person<br>RESTAURANT ← restaurant | LEAVE |

of the restaurant, and (3) a kitchen of the restaurant. The doors are: (1) a main entrance that connects the street and the dining room and (2) a kitchen door that connects the dining room and the kitchen.

We next define several characters: customers dining together, a cook, and a waiter. Initially the customers are in the street, the cook is in the kitchen, and the waiter is in the dining room. The customers are hungry, the cook and the waiter are not hungry, and the characters are all standing.

We then define several objects: a table and bill for the customers, a chair and menu for each customer, and food for each customer. Initially the table, bill, chairs, and menus are in the dining room, and the food is in the kitchen. The menus are on the table, and the food is not prepared.

Finally we generate a sequence of events that stops with the last restaurant script event mentioned in the narrative, as specified in the template:

(1) At the first timepoint, the customers walk through the main entrance.
(2) In the next timepoint, the waiter greets the customers. If the last event is ARRIVE, we stop here—this is the last timepoint and we generate no further events.
(3) Next, the customers sit on their respective chairs. If the last event is SIT, we stop here.
(4) The customers pick up their respective menus. If the last event is READMENU, we stop here.
(5) The customers order their respective foods from the waiter and place their menus back on the table. If the last event is ORDER, we stop here.
(6) If the last event is SERVE, the last timepoint of the model is the timepoint in which the waiter serves the customers, and we generate no further events.
(7) After the waiter serves the customers, the customers eat their respective foods. If the last event is EAT, we stop here.
(8) The paying customer requests the bill from the waiter. If the LAST EVENT is REQUESTBILL, we stop here.
(9) If the last event is RECEIVEBILL, the last timepoint of the model is the timepoint after the waiter sets the bill on the table, and we generate no further events.
(10) After the waiter sets the bill on the table, the paying customer pays the bill. If the TIP is not ZERO, the paying customer tips the waiter. If the last event is PAY, we stop here.
(11) The customers rise from their chairs, say goodbye to the waiter, and walk through the main entrance back out onto the street. (The last event is LEAVE.)

Appendix A shows the commonsense reasoning problem built using this method given the template shown in Section 4, which derives from the sample narrative of Section 1.

Note that many details of the model are left unspecified in the aforesaid. The model is filled in using commonsense reasoning. Examples of details filled in are as follows:

- After a customer walks from the street through the main entrance, the customer is in the restaurant.
- After a customer sits in a chair, the customer is no longer standing.
- After a customer picks up a menu, the menu is no longer on the table and the customer is holding the menu.
- After the customer orders food from the waiter, the waiter walks into the kitchen, orders the food from the cook, waits for the food to be prepared, picks up the food, walks into the dining room, and sets the food down on the table.
- After the waiter orders food from the cook, the cook prepares the food.
- After a customer eats, the customer is no longer hungry.

Our program includes a simple natural language generator, which is used to generate questions and answers in English. The generator is similar to previous generators we have constructed (Mueller, 1990; Mueller, 1998). It is a recursive descent, phrasal generator (Jacobs, 1985) that makes use of pattern-concept pairs (Arens, 1986) such as the following:

fluent Holding(character,object)
pattern character <be> "holding" object

The extracted names of the restaurant, customer, food associated with each customer, and waiter provided in the restaurant script template are associated with logical constants, so that they can be generated by the generator. If the information extraction module is unable to extract such names, which often occurs, the generic expressions *the restaurant*, *the customer*, and *the food* are instead used.

# 6 Commonsense Reasoning about Dining in a Restaurant

In this section, we describe the method used by our program to perform commonsense reasoning given a reasoning problem in order to fill in details and construct a model of the dining episode. We perform commonsense reasoning using the *event calculus* (Shanahan, 1997; Shanahan, 1999; Miller and Shanahan, 2002), which is based on many-sorted first-order logic (Walther, 1987).

## 6.1 Event calculus

The event calculus includes sorts (or types) for *fluents*, *events*, *timepoints*, and domain objects. A fluent (McCarthy and Hayes, 1969) represents a time-varying property such as the fact that a particular object is in a particular room or that a particular character is hungry. The notation HoldsAt(Hungry(Jim1),3) means that the fluent Hungry(Jim1) is true at timepoint 3. That is, the character Jim1 is hungry at timepoint 3. The notation ¬HoldsAt(Hungry(Jim1),3) means that the fluent Hungry(Jim1) is false at timepoint 3, or that Jim1 is not hungry at timepoint 3.

For events we use the notation Happens (Eat(Jim1,Food1),3) to mean that the event Eat(Jim1,Food1) occurs at timepoint 3. That is, Jim1 eats Food1 at timepoint 3. We use a variant of the event calculus called the *discrete event calculus* (Mueller, 2004a, 2006) in which events that occur at timepoint $t$ result in changes to fluents at timepoint $t+1$.

We use *effect axioms* to specify how events influence the truth values of fluents. For example, the following axioms specify that when a character eats, the character will be satiated and no longer hungry:

∀character,food,time Initiates(Eat(character, food),Satiated(character),time).
∀character,food,time Terminates(Eat (character,food),Hungry(character),time).

The meaning of Initiates and Terminates is given by the following axioms of the discrete event calculus:

∀event,fluent,time Happens(event,time) ∧ Initiates(event,fluent,time) ⇒ HoldsAt(fluent,time+1).

∀event,fluent,time Happens(event,time) ∧ Terminates(event,fluent,time) ⇒ ¬HoldsAt(fluent,time+1).

That is, if an event occurs at a timepoint and the event initiates a fluent, then the fluent will be true at the next timepoint; if an event occurs at a timepoint and the event terminates a fluent, then the fluent will be false at the next timepoint.

Now suppose we are told that Jim1 is hungry and not satiated at timepoint 3, and that Jim1 eats Food1 at timepoint 3:

HoldsAt(Hungry(Jim1),3).
¬HoldsAt(Satiated(Jim1),3).
Happens(Eat(Jim1,Food1),3).

From the above axioms, we can deduce that at timepoint 4, Jim1 is not hungry and is satiated:

¬HoldsAt(Hungry(Jim1),4).
HoldsAt(Satiated(Jim1),4).

## 6.2 Commonsense knowledge base

We have assembled eighty-seven axioms for commonsense reasoning about dining in a restaurant. These axioms are part of a larger commonsense knowledge base we are developing for narrative comprehension. We have so far applied the knowledge base to the understanding of a children's story (Mueller, 2003), terrorism news stories and American literature texts (Mueller, 2004c).

Table 2 summarizes the effect axioms relating to food and hunger. The value of a fluent before an

**Table 2** Effects of events for food and hunger, where c is a character and f is a food

| Fluent before | Event | Fluent after |
|---|---|---|
| ¬FoodPrepared(f) | FoodPrepare(c,f) | FoodPrepared(f) |
| ¬Satiated(c) | Eat(c,f) | Satiated(c) |
| Hungry(c) | Eat(c,f) | ¬Hungry(c) |

**Table 3** Effects of events for object-scale space, where c is a character and o, o1, and o2 are objects

| Fluent before | Event | Fluent after |
|---|---|---|
| ¬Holding(c,o) | PickUp(c,o) | Holding(c,o) |
| ¬Holding(c,o1) | TakeOffOf(c,o1,o2) | Holding(c,o1) |
| Holding(c,o) | LetGoOf(c,o) | ¬Holding(c,o) |
| Holding(c,o1) | PlaceOn(c,o1,o2) | ¬Holding(c,o1) |
| ¬On(o1,o2) | PlaceOn(c,o1,o2) | On(o1,o2) |
| On(o1,o2) | TakeOffOf(c,o1,o2) | ¬On(o1,o2) |

**Table 4** Effects of events for room-scale space, where c is a character, d is a door, l1 and l2 are locations, Side1(d) = l1, and Side2(d) = l2

| Fluent before | Event | Fluent after |
|---|---|---|
| ¬At(c,l2) | WalkThroughDoor12(c,d) | At(c,l2) |
| ¬At(c,l1) | WalkThroughDoor21(c,d) | At(c,l1) |
| At(c,l1) | WalkThroughDoor12(c,d) | ¬At(c,l1) |
| At(c,l2) | WalkThroughDoor21(c,d) | ¬At(c,l2) |

event occurs is shown in the first column, the event is shown in the second column, and the value of the fluent after the event occurs is shown in the third column. In addition to effect axioms, *precondition axioms* are also provided. In this case, precondition axioms state that for a character to eat or prepare food, the character must be at the same location as the food.

Table 3 summarizes the effect axioms relating to object-level space. We also have a number of precondition axioms, one of which states that for a character to let go of an object, the character must be holding the object. We also have several *state constraints*, one of which says that an object cannot be on top of itself.

The *commonsense law of inertia* (Shanahan, 1997) states that the value of a fluent persists unless it is directly affected by an event. A fluent can be *released* from this law so that the value of the fluent may fluctuate. When a fluent is released it may be indirectly affected by events. For example, the location of an object normally persists. But when a character is holding an object, the location of the object is released and varies with the location of the character. We have a number of *release axioms*. One such axiom states that if a character picks up an object, the location of the object will be released from the commonsense law of inertia:

∀character,object,time Releases(Hold (character,object),At(object,location),time).

We also have appropriate state constraints, one of which says that if a character is holding an object and the character is at a location, then the object is also at that location. Finally, we use effect axioms to make fluents again subject to the commonsense law of inertia. We use an effect axiom to represent that if a character is at a given location and lets go of an object, the object will be at that location, and the object's location will be subject to the commonsense law of inertia.

Table 4 summarizes the effect axioms relating to room-scale space. Precondition axioms also state that for a character to walk through a given side of a door, the character must be at that side of the door and the character must be standing. A state constraint says that an object is at one location at a time.

Table 5 summarizes the effect axioms relating to body posture. A precondition axiom also states that for a character to sit or lie on an object, the character must be at the same location as the object. A state constraint says that a character can sit or lie on one object at a time.

Table 6 summarizes the effect axioms relating to some simple speech acts. A precondition axiom states that for a character to order or request something from another character, the first character must be at the same location as the second character.

The behaviour of a waiter in the restaurant script is simulated using the collection of effect axioms shown in Table 7, which resembles a finite

**Table 5** Effects of events for body posture, where c is a character and o is an object

| Fluent before | Event | Fluent after |
|---|---|---|
| ¬LyingOn(c,o) | LieOn(c,o) | LyingOn(c,o) |
| LyingOn(c,o) | RiseFrom(c,o) | ¬LyingOn(c,o) |
| ¬SittingOn(c,o) | SitOn(c,o) | SittingOn(c,o) |
| SittingOn(c,o) | RiseFrom(c,o) | ¬SittingOn(c,o) |
| ¬Standing(c) | RiseFrom(c,o) | Standing(c) |
| Standing(c) | LieOn(c,o) | ¬Standing(c) |
| Standing(c) | SitOn(c,o) | ¬Standing(c) |

**Table 6** Effects of events for speech acts, where c1 and c2 are characters and o is an object

| Fluent before | Event | Fluent after |
|---|---|---|
| ¬KnowOrder(c2,c1,o) | Order(c1,c2,o) | KnowOrder(c2,c1,o) |
| ¬KnowRequest (c2,c1,o) | Request(c1,c2,o) | KnowRequest (c2,c1,o) |

**Table 7** Finite automaton for the role of waiter, where b is a bill, c is a customer, f is food, k is a cook, r is a restaurant, t is a table, and w is a waiter

| Fluent before | Event | Fluent after |
|---|---|---|
| BeWaiter0(w) | Greet(w,c) | BeWaiter1(w) |
| BeWaiter1(w) | Order(c,w,f) | BeWaiter2(w) |
| BeWaiter2(w) | WalkThroughDoor12 (w,KitchenDoorOf(r)) | BeWaiter3(w) |
| BeWaiter3(w) | Order(w,k,f) | BeWaiter4(w) |
| BeWaiter4(w) | PickUp(w,f) | BeWaiter5(w) |
| BeWaiter5(w) | WalkThroughDoor21 (w,KitchenDoorOf(r)) | BeWaiter6(w) |
| BeWaiter6(w) | PlaceOn(w,f,t) | BeWaiter7(w) |
| BeWaiter7(w) | Request(c,w,b) | BeWaiter8(w) |
| BeWaiter8(w) | PickUp(w,b) | BeWaiter9(w) |
| BeWaiter9(w) | PlaceOn(w,b,t) | BeWaiter0(w) |

automaton (Lewis and Papadimitriou, 1981). The initial state of the automaton is BeWaiter0. When the waiter greets the customer, the automaton enters state BeWaiter1. When the customer orders some food, the automaton enters state BeWaiter2. When the automaton is in state BeWaiter2, the waiter goes to the kitchen and the automaton enters state BeWaiter3. When the automaton is in state BeWaiter3, the waiter orders the food from the cook and the automaton enters state BeWaiter4, and so on.

**Table 8** Finite automaton for the role of cook, where f is food, k is a cook, and w is a waiter

| Fluent before | Event | Fluent after |
|---|---|---|
| BeCook0(k) | Order(w,k,f) | BeCook1(k) |
| BeCook1(k) | FoodPrepare(k,f) | BeCook0(k) |

The behavior of the cook, coordinated with the behaviour of the waiter, is simulated using the collection of effect axioms shown in Table 8.

## 6.3 Commonsense reasoning

Our program uses a method for reasoning in the event calculus based on satisfiability (SAT) (Garey and Johnson, 1979; Du *et al.*, 1997). The method builds on the methods of Shanahan and Witkowski (2004) and Kautz and Selman (1992). Here we provide a short description of our method, which is described in detail elsewhere (Mueller, 2004a; Mueller, 2004b).

A SAT problem consists of a set of Boolean variables and a propositional formula over those variables. The formula is in conjunctive normal form—a conjunction of clauses, where each clause is a disjunction of literals, where each literal is a variable or a negated variable. A solution to a SAT problem consists of truth assignments for the variables such that the formula is true.

Our method consists of (1) encoding a commonsense reasoning problem and the commonsense knowledge base as a SAT problem, (2) running a SAT solver on the problem, and (3) decoding the solutions produced by the SAT solver into models. We used the Relsat complete SAT solver (Bayardo Jr. and Schrag, 1997). The steps of our method are as follows:

(1) All Initiates formulas in the commonsense knowledge base are combined into a single Initiates formula. For example, suppose the knowledge base contains the following formulas:

∀character,food,time Initiates(Eat(character, food),Satiated(character),time).

$$\forall character,time\ Initiates(WakeUp(character), Awake(character),time).$$

These are combined into the following formula:

$$\forall event,fluent,time\ Initiates(event,fluent,time) \iff (\exists character,food\ event = Eat(character,food) \wedge fluent = Satiated(character)) \vee (\exists character\ event = WakeUp(character) \wedge fluent = Awake(character)).$$

This formula states that eating is the only thing that initiates being satiated, and waking up is the only thing that initiates being awake. Furthermore, no other fluents are initiated.

(2) All Terminates formulas are combined into a single Terminates formula.

(3) All Releases formulas are combined into a single Releases formula.

(4) The discrete event calculus axioms are transformed by replacing Initiates, Terminates, and Releases with their definitions as created earlier. For example, the axiom

$$\forall event,fluent,time\ Happens(event,time) \wedge Initiates(event,fluent,time) \Rightarrow HoldsAt(fluent,time+1)$$

is transformed into

$$\forall event,fluent,time\ Happens(event,time) \wedge ((\exists character,food\ event = Eat(character,food) \wedge fluent = Satiated(character)) \vee (\exists character\ event = WakeUp(character) \wedge fluent = Awake(character))) \Rightarrow HoldsAt(fluent,time+1).$$

(5) All Happens formulas in the commonsense reasoning problem are combined into a single Happens formula. For example, suppose we have the following Happens formulas:

$$Happens(Eat(Jim1,Food1),3).$$
$$Happens(Eat(Jim1,Food2),4).$$

These are combined into the following formula:

$$\forall event,time\ Happens(event,time) \iff (event = Eat(Jim1,Food1) \wedge time = 3) \vee (event = Eat(Jim1,Food2) \wedge time = 4).$$

This states that the only events that occur are that Jim1 eats Food1 at timepoint 3, and Jim1 eats Food2 at timepoint 4.

(6) The conjunction of the following is formed: (a) the formulas of the commonsense reasoning problem minus the Happens formulas, (b) the formulas of the commonsense knowledge base minus the Initiates, Terminates, and Releases formulas, (c) the discrete event calculus formulas as transformed earlier, and (d) the single Happens formula as formed earlier.

(7) The domain of each sort is restricted to a finite set, and quantifiers are instantiated. For example, suppose that the object sort is limited to the set {Food1, Menu1} and the timepoint sort is limited to the set {0, 1}. Then the quantifier in the state constraint

$$\forall object,time\ \neg HoldsAt(On(object,object),time)$$

is expanded, which gives:

$$\neg HoldsAt(On(Food1,Food1),0) \wedge \neg HoldsAt(On(Menu1,Menu1),0) \wedge \neg HoldsAt(On(Food1,Food1),1) \wedge \neg HoldsAt(On(Menu1,Menu1),1).$$

(8) The formula is simplified using standard techniques (Nerode and Shore, 1997).

(9) The formula is converted into a compact conjunctive normal form using the technique of renaming subformulas (Plaisted and Greenbaum, 1986; Giunchiglia and Sebastiani, 1999).

(10) A map such as the following from ground atoms to Boolean variables is built:

$$HoldsAt(On(Food1,Food1),0) \rightarrow v1$$
$$HoldsAt(On(Menu1,Menu1),0) \rightarrow v2$$
$$HoldsAt(On(Food1,Food1),1) \rightarrow v3$$
$$HoldsAt(On(Menu1,Menu1),1) \rightarrow v4$$

(11) A propositional formula is constructed by replacing ground atoms with the corresponding Boolean variables. For example, the expanded state constraint given earlier becomes:

$$\neg v1 \wedge \neg v2 \wedge \neg v3 \wedge \neg v4.$$

(12) The propositional formula is sent to the SAT solver.
(13) The results from the SAT solver are decoded. For each truth assignment, a model is constructed by mapping Boolean variables back to ground atoms.

Appendix B shows the model that results from solving the commonsense reasoning problem of Appendix A, which was built from the template of Section 4 that results from the sample narrative of Section 1. Table 9 provides runtime statistics on solving this commonsense reasoning problem.

## 7 Generation of Questions and Answers

In this section, we describe the method used by our program to generate and answer questions about the dining episode by consulting the model produced by commonsense reasoning.

We generate and answer yes–no questions about space as follows: for each character *c* in the model, for each unique event *e* that occurs in the model at one or more timepoints such that *c* is not the actor of *e*: (1) if for every occurrence of *e* in the model, the location of *c* is equal to the location of the character of *e*, we generate 'Q: Was *c* present when *e*? A: Yes.' and (2) otherwise, if for every occurrence of *e* in the model, the location of *c* is not equal to the location of the character of *e*, we generate 'Q: Was *c* present when *e*? A: No.'

We generate and answer yes–no questions about time as follows: for each unique event *e* that occurs in the model at exactly one timepoint *t*, for each unique fluent *f* that is true in the model at one or more timepoints: (1) if *f* is false at *t*, and *f* is true at $t+1$, we generate 'Q: *f* before *e*? A: No' and 'Q: *f* after *e*? A: Yes' and (2) if *f* is true at *t*, and *f* is false at $t+1$, we generate 'Q: *f* before *e*? A: Yes' and 'Q: *f* after *e*? A: No.'

We generate and answer question-word questions as follows: for each unique event *e* that occurs in the model at one or more timepoints, for *i* in 1 through the number of arguments of *e*: (1) we form a conjunction of answers consisting of the *i*th arguments of all events occurring in the model that match *e* without regard to the contents of its *i*th argument and (2) we generate a question consisting of *e* with the *i*th argument replaced with an appropriate question word and an answer consisting of the conjunction of answers.

**Table 9** Statistics on solving sample commonsense reasoning problem (wall times in seconds on machine with 1.8 GHz Intel Pentium 4 processor and 512 megabytes RAM)

| Statistic | Value |
|---|---|
| Number of variables | 5,160 |
| Number of clauses | 32,945 |
| Encoding time | 26.9 |
| Solution time | 1.0 |

## 8 Evaluation

After developing our program, we evaluated it in order to determine how successful it was at modelling space and time in narratives involving dining in a restaurant, and ways in which it could be improved. We conducted evaluations on the previously unseen web test corpus and the previously unseen Gutenberg test corpus.

### 8.1 Evaluation on web test corpus

We extracted 123 excerpts from the first forty-eight texts of the web test corpus as follows: we located the keywords *menu*, *waiter*, *waitress*, and *restaurant* in each text. For each occurrence of a keyword, we formed a twenty-one sentence excerpt centred around the sentence in which the keyword occurred. We then merged overlapping or adjacent excerpts into single excerpts. On average, the resulting excerpts contained 34.6 sentences.

We ran our program on the 123 excerpts. The program produced restaurant script templates for sixty-three of the excerpts. The program then built reasoning problems, produced models, and generated questions and answers for each of these excerpts. A total of 4155 questions and answers were generated.[4]

For each excerpt, we then randomly drew two questions and answers of each type (yes–no space, yes–no time, and question word), resulting in 378 questions and answers. At this point we noticed that

two of the sixty-three excerpts were contained in the Gutenberg test corpus. We removed these excerpts, leaving 366 questions and answers associated with sixty-one excerpts of thirty-six texts. The composition of the excerpts is shown in Table 10.

We then performed a detailed analysis of the 366 questions and answers. Table 11 provides the overall results of our analysis, as well as the results of the analysis broken down by last event and question type. A total of 277 answers (seventy-six percent) were correct, forty-nine answers (thirteen percent) were incorrect, and forty questions (eleven percent) had incorrect presuppositions.

Eighteen questions were considered to have incorrect presuppositions because they resulted from excerpts that consisted of brief reviews of a bunch of restaurants. Ten questions with incorrect presuppositions were due to incorrect extraction of a food or restaurant name—for example, a question asked about entering the dining room of the Main Pier, when Main Pier was not the name of the restaurant. Four questions with incorrect presuppositions assumed the wrong layout of the restaurant—for example, a question referred to the street outside the restaurant when the dining had taken place on a cruise ship. One excerpt turned out not to involve the restaurant script at all; the resulting six questions all contained the incorrect presupposition that dining had occurred.

Twenty-nine of the incorrect answers were considered incorrect because they did not clearly follow from the narrative. The answers typically specified the location of the waiter in cases where the waiter's location could not clearly be determined.

**Table 10** Composition of web test excerpts

| Category | Excerpts (%) |
|---|---|
| Travelogue | 23 (38) |
| Fiction | 22 (36) |
| Restaurant review | 15 (25) |
| Videogame discussion | 1 (2) |
| Total | 61 (100) |

## 8.2 Evaluation on Gutenberg test corpus

Using the same method used to extract excerpts from the web test corpus, we extracted 530 excerpts from the thirty texts in the Gutenberg test corpus. On average there were 25.1 sentences per excerpt. We ran our program on the 530 excerpts. The program produced restaurant script templates for sixty-three of the excerpts. The program then produced models for each of these excerpts and generated a total of 3040 questions and answers.[5]

For each excerpt we randomly drew two questions and answers of each type, giving 378 questions and answers. We noticed that eleven of the sixty-three excerpts were duplicates.

**Table 11** Evaluation on web test corpus

| | | Correct answer (%) | Incorrect answer (%) | Incorrect presupposition (%) |
|---|---|---|---|---|
| Overall | | 277/366 (76%) | 49/366 (13%) | 40/366 (11%) |
| ARRIVE | 0/366 (0%) | 0/0 (0) | 0/0 (0) | 0/0 (0) |
| SIT | 36/366 (10%) | 34/36 (94) | 2/36 (6) | 0/36 (0) |
| READMENU | 24/366 (7%) | 24/24 (100) | 0/24 (0) | 0/24 (0) |
| ORDER | 114/366 (31%) | 90/114 (79) | 19/114 (17) | 5/114 (4) |
| SERVE | 54/366 (15%) | 43/54 (80) | 10/54 (19) | 1/54 (2) |
| EAT | 120/366 (33%) | 70/120 (58) | 17/120 (14) | 33/120 (28) |
| REQUESTBILL | 0/366 (0%) | 0/0 (0) | 0/0 (0) | 0/0 (0) |
| RECEIVEBILL | 6/366 (2%) | 6/6 (100) | 0/6 (0) | 0/6 (0) |
| PAY | 0/366 (0%) | 0/0 (0) | 0/0 (0) | 0/0 (0) |
| LEAVE | 12/366 (3%) | 10/12 (83) | 1/12 (8) | 1/12 (8) |
| Yes-no space | 122/366 (33%) | 94/122 (77) | 18/122 (15) | 10/122 (8) |
| Yes-no time | 122/366 (33%) | 81/122 (66) | 24/122 (20) | 17/122 (14) |
| Question word | 122/366 (33%) | 102/122 (84) | 7/122 (6) | 13/122 (11) |

**Table 12** Evaluation on Gutenberg test corpus

| | | Correct answer (%) | Incorrect answer (%) | Incorrect presupposition (%) |
|---|---|---|---|---|
| Overall | | 217/312 (70%) | 30/312 (10%) | 65/312 (21%) |
| ARRIVE | 30/312 (10%) | 13/30 (43) | 1/30 (3) | 16/30 (53) |
| SIT | 66/312 (21%) | 56/66 (85) | 4/66 (6) | 6/66 (9) |
| READMENU | 6/312 (2%) | 6/6 (100) | 0/6 (0) | 0/6 (0) |
| ORDER | 114/312 (37%) | 81/114 (71) | 11/114 (10) | 22/114 (19) |
| SERVE | 30/312 (10%) | 21/30 (70) | 3/30 (10) | 6/30 (20) |
| EAT | 30/312 (10%) | 15/30 (50) | 5/30 (17) | 10/30 (33) |
| REQUESTBILL | 6/312 (2%) | 4/6 (67) | 2/6 (33) | 0/6 (0) |
| RECEIVEBILL | 12/312 (4%) | 9/12 (75) | 3/12 (25) | 0/12 (0) |
| PAY | 6/312 (2%) | 2/6 (33) | 0/6 (0) | 4/6 (67) |
| LEAVE | 12/312 (4%) | 10/12 (83) | 1/12 (8) | 1/12 (8) |
| Yes-no space | 104/312 (33%) | 64/104 (62) | 18/104 (17) | 22/104 (21) |
| Yes-no time | 104/312 (33%) | 75/104 (72) | 6/104 (6) | 23/104 (22) |
| Question word | 104/312 (33%) | 78/104 (75) | 6/104 (6) | 20/104 (19) |

We removed these duplicates, leaving 312 questions and answers associated with fifty-two excerpts of fifteen texts. We analysed these questions and answers in detail.

Table 12 gives the results of our analysis. A total of 217 answers (seventy percent) were correct, thirty answers (ten percent) were incorrect, and sixty-five questions (twenty-one percent) had incorrect presuppositions.

Thirty of the questions with incorrect presuppositions arose from the five of the fifty-two excerpts that turned out not to involve the restaurant script. Other questions with incorrect presuppositions contained the wrong food or restaurant name, or referred to food only when drinks were served.

Twenty-one of the incorrect answers did not clearly follow from the narrative.

## 8.3 Discussion

The models of space and time produced by our program corresponded to a large degree to the space and time described in the excerpts. However, we noted several problems during the analysis.

Space was not modelled correctly for excerpts in which the dining room was not on the ground floor, the dining room was on a ship or train, the dining room was outdoors, or the cook prepared food in the room where customers sat.

Time was not modelled correctly for excerpts in which a drink order was taken separately from the food order, multiple courses were served, or the customer and dining companion arrived separately.

Meals were often described in great detail, but the information extraction module ignored most of this detail. Extraction of foods could be improved by extending the module's patterns and list of foods. Some unrecognized synonyms—*dig into* for *eat*, *flop down* for *set down*, and the old-fashioned *bill of fare* for *menu*—could also be added.

# 9 Limitations

For question answering, the main limitation of our program is that it can only handle a fixed class of questions relating to space and time. In order for the program to answer arbitrary questions posed by users, it would have to be extended to model other aspects of narratives, and to handle other sorts of questions.

For narrative comprehension, the main limitation of our program is that it is based on scripts and, therefore, cannot handle the many novel sequences of situations and events that arise in narratives.

Our program has some limitations with respect to the restaurant script. The program is only able to model several variations of the script such as different numbers of customers, different numbers

of orders, different numbers of foods, different restaurant names, and different stages of completion of the restaurant script (via the last event). The program models only a single spatial layout consisting of a street, dining room, and kitchen. The program does not model different styles of dining such as buffet dining and customers sharing tables with strangers. The program does not model exceptions such as the waiter bringing the wrong food, the waiter bringing the food after a long delay, or the customer getting up from the table to ask for the bill. The program is able to model the roles of only the waiter and the cook, and does not handle other roles such as owner, manager, greeter, assistant waiter, or piano player.

In order to model such variations, the information extraction module could be extended to extract the necessary information (such as DINING STYLE), appropriate reasoning problems could be constructed based on this information, and appropriate commonsense axioms could be added.

## 10 Conclusions

We have demonstrated a program that automatically builds models of space and time in narratives involving dining in a restaurant. The program uses information extraction and commonsense reasoning techniques to build the models. Our evaluation showed that generating such models is feasible; however, much works remains to be done before highly accurate models can be produced. Several improvements to our program are suggested by this study:

- *Recognition and modelling of multiple spatial layouts*: the program could be extended to extract information about spatial layout from text, and use this to construct models of room-scale space.
- *Adding more scripts*: the program could be extended to handle many scripts. Gordon (2001) has identified over 700 scripts. We have begun the process of adding scripts, extending the program to handle four terrorism scripts frequent in news stories and ten scripts frequent in American literature texts (Mueller, 2004c).
- *Adding more variation to scripts*: the script representations used in the program could be enriched. The restaurant script representation could be extended with dining style, exceptional conditions, and additional characters.
- *Handling novel sequences*: the program could be extended to build models of arbitrary sequences of situations and events mentioned in a text. This is a difficult problem, however. It will be essential to develop efficient techniques for searching the large space of possible models.

Powerful narrative comprehension programs could eventually be used to understand texts in greater detail than is currently possible. They could be used to improve the effectiveness of dialogue systems, help systems, news-tracking services, question-answering systems, and search engines. The present work is one step towards this long-term goal.

## References

**Abelson, R. P.** (1981). Psychological status of the script concept. *American Psychologist*, **36**(7): 715–29.

**Arens, Y.** (1986). *CLUSTER: An Approach to Contextual Language Understanding*. PhD thesis, University of California, Berkeley, Berkeley, CA.

**Bayardo Jr., Roberto J. and Schrag, R. C.** (1997). Using CSP look-back techniques to solve real world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, Menlo Park, CA: AAAI Press, pp. 203–8.

**Bower, G. H.** (1989). Mental models in text understanding. In Adrienne F. Bennett and Kevin M. McConkey (eds), *Cognition in Individual and Social Contexts*. Amsterdam: Elsevier, pp. 129–44.

**Cardie, C.** (1997). Empirical methods in information extraction. *AI Magazine*, **18**(4): 65–80.

**Charniak, E.** (1972). Toward a model of children's story comprehension. Technical Report AITR-266, Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

**Cowie, J. and Lehnert, W. G.** (1996). Information extraction. *Communications of the ACM*, **39**(1): 80–91.

**Craik, K. J. W.** (1943). *The Nature of Explanation*. Cambridge: Cambridge University Press.

**Cullingford, R. E.** (1978). Script application: Computer understanding of newspaper stories. Technical Report

YALE/DCS/tr116, New Haven, CT: Computer Science Department, Yale University.

**Cunningham, H., Maynard, D., Bontcheva, K. and Tablan, V.** (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 168–75.

**Davis, E.** (1990). *Representations of Commonsense Knowledge*. San Mateo, CA: Morgan Kaufmann.

**Dimitrov, M.** (2002). A light-weight approach to coreference resolution for named entities in text. Master's Thesis, University of Sofia, Bulgaria.

**Du, D., Gu, J. and Pardalos, P. M. (eds).** (1997). *Satisfiability Problem: Theory and Applications, volume 35 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Boston, MA.

**Duchan, J. F., Bruder, G. A. and Hewitt, L. E. (eds).** (1995). *Deixis in Narrative: A Cognitive Science Perspective*. Hillsdale, NJ: Lawrence Erlbaum.

**Dyer, M. G.** (1983). *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*. Cambridge, MA: MIT Press.

**Fellbaum, C.** (ed.) (1998). *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press.

**Frank, S. L., Koppen, M., Noordman, L. G. M. and Vonk, W.** (2003). Modeling knowledge-based inferences in story comprehension. *Cognitive Science*, **27**: 875–910.

**Garey, M. R. and Johnson, D. S.** (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman.

**Giunchiglia, E. and Sebastiani, R.** (1999). Applying the Davis-Putnam procedure to nonclausal formulas. In *Proceedings of the Sixth Congress of the Italian Association for Artificial Intelligence*, Bologna.

**Goldman, S. R., Graesser, A. C. and van den Broek, P. (eds).** (1999). *Narrative Comprehension, Causality, and Coherence: Essays in Honor of Tom Trabasso*. Mahwah, NJ: Lawrence Erlbaum.

**Gordon, A. S.** (2001). Browsing image collections with representations of commonsense activities. *Journal of the American Society for Information Science and Technology*, **52**(11): 925–29.

**Graesser, A. C., Singer, M. and Trabasso, T.** (1994). Constructing inferences during narrative text comprehension. *Psychological Review*, **101**(3): 371–95.

**Grishman, R. and Sundheim, B.** (1996). Message Understanding Conference 6: A brief history. In *Proceedings of the Sixteenth International Conference on Computational Linguistics*, San Francisco, CA: Morgan Kaufmann.

**Hart, M.** (2005). *Project Gutenberg Archive*. MS: Oxford University Press.

**Hepple, M.** (2000). Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pp. 278–85.

**Hobbs, J. R., Stickel, M. E., Appelt, D. E. and Martin, P.** (1993). Interpretation as abduction. *Artificial Intelligence*, 63: 69–142.

**Jacobs, P. S.** (1985). *A Knowledge-based Approach to Language Generation*. PhD thesis, University of California, Berkeley, Berkeley, CA.

**Johnson-Laird, P. N.** (1983). *Mental Models: Toward a Cognitive Science of Language, Inference, and Consciousness*. Cambridge, MA: Harvard University Press.

**Kautz, H. and Selman, B.** (1992). Planning as satisfiability. In Bernd Neumann, (ed.), *Proceedings of the Tenth European Conference on Artificial Intelligence*, Chichester, UK: John Wiley, pp. 359–63.

**Lehnert, W. G.** (1978). *The Process of Question Answering: A Computer Simulation of Cognition*. Hillsdale, NJ: Lawrence Erlbaum.

**Lenat, D. B. and Guha, R. V.** (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Reading, MA: Addison-Wesley.

**Lewis, H. R. and Papadimitriou, C. H.** (1981). *Elements of the Theory of Computation*. Englewood Cliffs, NJ: Prentice-Hall.

**Lifschitz, V. (ed).** (1990). *Formalizing Common Sense: Papers by John McCarthy*. Norwood, NJ: Ablex.

**McCarthy, J. and Hayes, P. J.** (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B. and Michie, D. (eds), *Machine Intelligence 4*. Edinburgh, Scotland: Edinburgh University Press, pp. 463–502.

**McCarthy, J., Minsky, M., Sloman, A., Gong, L., Lau, T., Morgenstern, L., Mueller, E. T., Rieken, D., Singh, M. and Singh, P.** (2002). An architecture of diversity for commonsense reasoning. *IBM Systems Journal*, **41**(3): 530–9.

**McKoon, G. and Ratcliff, R.** (1992). Inference during reading. *Psychological Review*, **99**(3): 440–66.

**Miikkulainen, R.** (1993). *Subsymbolic Natural Language Processing*. Cambridge, MA: MIT Press.

**Miller, R. and Shanahan, M.** (2002). Some alternative formulations of the event calculus. In Kakas, A. C. and Sadri, F. (eds), *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part II, volume 2408 of Lecture Notes in Computer Science*. Berlin: Springer, pp. 452–90.

**Morrow, D. G., Bower, G. H. and Greenspan, S. L.** (1989). Updating situation models during narrative comprehension. *Journal of Memory and Language*, **28**: 292–312.

**MUC.** (1991). *Third Message Understanding Conference (MUC-3)*. San Mateo, CA: Morgan Kaufmann.

**Mueller, E. T.** (1990). *Daydreaming in Humans and Machines: A Computer Model of the Stream of Thought*. Norwood, NJ: Ablex.

**Mueller, E. T.** (1998). *Natural Language Processing with ThoughtTreasure*. New York: Signiform.

**Mueller, E. T.** (2002). Story understanding. In Nadel, L. (ed), *Encyclopedia of Cognitive Science*, vol. 4. London: Nature Publishing Group, pp. 238–46.

**Mueller, E. T.** (2003). Story understanding through multi-representation model construction. In Hirst, G. and Nirenburg, S. (eds), *Text Meaning: Proceedings of the HLT-NAACL 2003 Workshop*, East Stroudsburg, PA: Association for Computational Linguistics, pp. 46–53

**Mueller, E. T.** (2004a). Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, **14**(5): 703–30.

**Mueller, E. T.** (2004b). A tool for satisfiability-based commonsense reasoning in the event calculus. In Barr, V. and Markov, Z. (eds), *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference*, Menlo Park, CA: AAAI Press, pp. 147–52.

**Mueller, E. T.** (2004c). Understanding script-based stories using commonsense reasoning. *Cognitive Systems Research*, **5**(4): 307–340.

**Mueller, E. T.** (2006). *Commonsense Reasoning*. San Francisco: Morgan Kaufmann/Elsevier.

**Narayanan, S. S.** (1997). *Knowledge-based Action Representations for Metaphor and Aspect (KARMA)*. PhD Thesis, University of California, Berkeley, Berkeley, CA.

**Nerode, A. and Shore, R. A.** (1997). *Logic for Applications*. 2nd edn, New York: Springer.

**Norvig, P.** (1989). Marker passing as a weak method for text inferencing. *Cognitive Science*, **13**(4): 569–620.

**Plaisted, D. A. and Greenbaum, S.** (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation*, **2**: 293–304.

**Prince, G.** (1982). *Narratology: The Form and Functioning of Narrative*. Berlin: Mouton.

**Ram, A. and Moorman, K. (eds).** (1999). *Understanding Language Understanding: Computational Models of Reading*. Cambridge, MA: MIT Press.

**Reiter, R.** (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

**Rimmon-Kenan, S.** (2002). *Narrative Fiction: Contemporary Poetics*. 2nd edn, London, Routledge.

**Rinck, M. and Bower, G. H.** (1995). Anaphora resolution and the focus of attention in situation models. *Journal of Memory and Language*, **34**: 110–31.

**Schank, R. C. and Abelson, R. P.** (1977). *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Hillsdale, NJ: Lawrence Erlbaum.

**Schank, R. C., Kass, A. and Riesbeck, C. K.** (eds). (1994). *Inside Case-Based Explanation*. Hillsdale, NJ: Lawrence Erlbaum.

**Schank, R. C. and Riesbeck, C. K. (eds).** (1981). *Inside Computer Understanding: Five Programs plus Miniatures*. Hillsdale, NJ: Lawrence Erlbaum.

**Shanahan, M.** (1997). *Solving the Frame Problem*. Cambridge, MA: MIT Press.

**Shanahan, M.** (1999). The event calculus explained. In Wooldridge, M. J. and Veloso, M. M. (eds), *Artificial Intelligence Today, volume 1600 of Lecture Notes in Computer Science*. Heidelberg: Springer-Verlag, pp. 409–30.

**Shanahan, M. and Witkowski, M.** (2004). Event calculus planning through satisfiability. *Journal of Logic and Computation*, **14**(5): 731–45.

**Shapiro, S. C. and Rapaport, W. J.** (1995). An introduction to a computational reader of narratives. In Duchan, J. F., Bruder, G. A. and Hewitt, L. E. (eds), *Deixis in Narrative: A Cognitive Science Perspective*. Hillsdale, NJ: Lawrence Erlbaum, pp. 79–105.

**TREC.** (2002). The Eleventh Text Retrieval Conference (TREC 2002). Technical Report SP 500-251, United States National Institute of Standards and Technology.

**van Dijk, T. A. and Kintsch, W.** (1983). *Strategies of Discourse Comprehension*. Orlando, FL: Academic Press.

**van Oostendorp, H. and Zwaan, R. A. (eds).** (1994). *Naturalistic Text Comprehension, volume LIII of Advances in Discourse Processes*. Norwood, NJ: Ablex.

**Walther, C.** (1987). *A Many-Sorted Calculus Based on Resolution and Paramodulation*. London: Pitman.

**Wilson, S. G., Rinck, M., McNamara, T. P., Bower, G. H. and Morrow, D. G.** (1993). Mental models and narrative comprehension: some qualifications. *Journal of Memory and Language*, **32**: 141–54.

## Notes

1 By my own count (http://www.signiform.com/erik/pubs/storyres.html), over 50 narrative comprehension programs have been written. Ram and Moorman (1999) and Mueller (2002) review the field of automatic narrative comprehension.

2 Among the many other aspects of a narrative a program could understand are: goals, plans, beliefs, emotions, interpersonal relations, personality traits, plot structures, themes, and morals.

3 We use a format for templates similar to that used in the MUC evaluations of information extraction programs (Grishman and Sundheim, 1996).

4 For the sixty-three excerpts of the web test corpus, 972 yes–no space, 1854 yes–no time, and 1329 question-word questions were generated.

5 For the sixty-three excerpts of the Gutenberg test corpus, 702 yes–no space, 1338 yes–no time, and 1000 question-word questions were generated.

## A Sample Commonsense Reasoning Problem

BillOf(Restaurant1) = Bill1
CookOf(Restaurant1) = Cook1
KitchenDoorOf(Restaurant1) = KitchenDoor1
Side1(KitchenDoor1) = DiningRoom1
Side1(MainEntrance1) = Street1
Side2(KitchenDoor1) = Kitchen1
Side2(MainEntrance1) = DiningRoom1
TableOf(Restaurant1) = Table1
WaiterOf(Restaurant1) = Waiter1
HoldsAt(Standing(character),0)
¬HoldsAt(FoodPrepared(food),0)
¬HoldsAt(Holding(character,object),0)
¬HoldsAt(KnowOrder(character1,character2, physobj),0)
¬HoldsAt(KnowRequest(character1,character2, physobj),0)
HoldsAt(At(Bill1,DiningRoom1),0)
HoldsAt(At(Chair1,DiningRoom1),0)
HoldsAt(At(Cook1,Kitchen1),0)
HoldsAt(At(Customer1,Street1),0)
HoldsAt(At(Food1,Kitchen1),0)
HoldsAt(At(Menu1,DiningRoom1),0)
HoldsAt(At(Table1,DiningRoom1),0)
HoldsAt(At(Waiter1,DiningRoom1),0)
HoldsAt(BeCook0(Cook1),0)
HoldsAt(BeWaiter0(Waiter1),0)
HoldsAt(Hungry(Customer1),0)
HoldsAt(On(Menu1,Table1),0)
¬HoldsAt(Hungry(Cook1),0)
¬HoldsAt(Hungry(Waiter1),0)
¬HoldsAt(On(Bill1,Table1),0)
Happens(WalkThroughDoor12(Customer1, MainEntrance1),0)
Happens(Greet(Waiter1,Customer1),1)
Happens(SitOn(Customer1,Chair1),2)
Happens(TakeOffOf(Customer1,Menu1, Table1),3)
Happens(Order(Customer1,Waiter1,Food1),4)
Happens(PlaceOn(Customer1,Menu1,Table1),5)
Happens(Eat(Customer1,Food1),11)
Happens(Request(Customer1,Waiter1,Bill1),12)
Happens(Pay(Customer1,Waiter1),15)
Happens(Tip(Customer1,Waiter1),15)
Happens(RiseFrom(Customer1,Chair1),16)
Happens(SayGoodbye(Customer1,Waiter1),17)
Happens(WalkThroughDoor21(Customer1, MainEntrance1),18)

## B Model of Sample Narrative

All fluents that are true at timepoint 0 are shown. Thereafter, fluents that become false are indicated with a minus sign and fluents that become true are indicated with a plus sign. Event occurrences are shown at the end of each timepoint.

0
At(Bill1, DiningRoom1)
At(Chair1, DiningRoom1)

At(Cook1, Kitchen1)
At(Customer1, Street1)
At(Food1, Kitchen1)
At(Menu1, DiningRoom1)
At(Table1, DiningRoom1)
At(Waiter1, DiningRoom1)
BeCook0(Cook1)
BeWaiter0(Waiter1)
Hungry(Customer1)
NearPortal(Bill1, KitchenDoor1)
NearPortal(Bill1, MainEntrance1)
NearPortal(Chair1, KitchenDoor1)
NearPortal(Chair1, MainEntrance1)
NearPortal(Cook1, KitchenDoor1)
NearPortal(Customer1, MainEntrance1)
NearPortal(Food1, KitchenDoor1)
NearPortal(Menu1, KitchenDoor1)
NearPortal(Menu1, MainEntrance1)
NearPortal(Table1, KitchenDoor1)
NearPortal(Table1, MainEntrance1)
NearPortal(Waiter1, KitchenDoor1)
NearPortal(Waiter1, MainEntrance1)
On(Menu1, Table1)
Satiated(Cook1)
Satiated(Waiter1)
Standing(Cook1)
Standing(Customer1)
Standing(Waiter1)
Happens(WalkThroughDoor12(Customer1,
MainEntrance1), 0)
1
−At(Customer1, Street1)
+At(Customer1, DiningRoom1)
+NearPortal(Customer1, KitchenDoor1)
Happens(Greet(Waiter1, Customer1), 1)
2
−BeWaiter0(Waiter1)
+BeWaiter1(Waiter1)
Happens(SitOn(Customer1, Chair1), 2)
3
−Standing(Customer1)
+Sitting(Customer1)
+SittingOn(Customer1, Chair1)
Happens(TakeOffOf(Customer1,
Menu1, Table1), 3)
4
−On(Menu1, Table1)
+Holding(Customer1, Menu1)
Happens(Order(Customer1, Waiter1, Food1), 4)
5
−BeWaiter1(Waiter1)
+BeWaiter2(Waiter1)
+KnowOrder(Waiter1, Customer1, Food1)
Happens(PlaceOn(Customer1,
Menu1, Table1), 5)
Happens(WalkThroughDoor12(Waiter1,
KitchenDoor1), 5)
6
−At(Waiter1, DiningRoom1)
−BeWaiter2(Waiter1)
−Holding(Customer1, Menu1)
−NearPortal(Waiter1, MainEntrance1)
+At(Waiter1, Kitchen1)
+BeWaiter3(Waiter1)
+On(Menu1, Table1)
Happens(Order(Waiter1, Cook1, Food1), 6)
7
−BeCook0(Cook1)
−BeWaiter3(Waiter1)
+BeCook1(Cook1)
+BeWaiter4(Waiter1)
+KnowOrder(Cook1, Waiter1, Food1)
Happens(FoodPrepare(Cook1, Food1), 7)
8
−BeCook1(Cook1)
+BeCook0(Cook1)
+FoodPrepared(Food1)
Happens(PickUp(Waiter1, Food1), 8)
9
−BeWaiter4(Waiter1)
+BeWaiter5(Waiter1)
+Holding(Waiter1, Food1)
Happens(WalkThroughDoor21
(Waiter1, KitchenDoor1), 9)
10
−At(Food1, Kitchen1)
−At(Waiter1, Kitchen1)
−BeWaiter5(Waiter1)
+At(Food1, DiningRoom1)
+At(Waiter1, DiningRoom1)
+BeWaiter6(Waiter1)
+NearPortal(Food1, MainEntrance1)
+NearPortal(Waiter1, MainEntrance1)
Happens(PlaceOn(Waiter1, Food1, Table1), 10)

11
−BeWaiter6(Waiter1)
−Holding(Waiter1, Food1)
+BeWaiter7(Waiter1)
+On(Food1, Table1)
Happens(Eat(Customer1, Food1), 11)
12
−Hungry(Customer1)
+Satiated(Customer1)
Happens(Request(Customer1, Waiter1, Bill1), 12)
13
−BeWaiter7(Waiter1)
+BeWaiter8(Waiter1)
+KnowRequest(Waiter1, Customer1, Bill1)
Happens(PickUp(Waiter1, Bill1), 13)
14
−BeWaiter8(Waiter1)
+BeWaiter9(Waiter1)
+Holding(Waiter1, Bill1)
Happens(PlaceOn(Waiter1, Bill1, Table1), 14)
15
−BeWaiter9(Waiter1)
−Holding(Waiter1, Bill1)
+BeWaiter0(Waiter1)
+On(Bill1, Table1)
Happens(Pay(Customer1, Waiter1), 15)
Happens(Tip(Customer1, Waiter1), 15)
16
Happens(RiseFrom(Customer1, Chair1), 16)
17
−Sitting(Customer1)
−SittingOn(Customer1, Chair1)
+Standing(Customer1)
Happens(SayGoodbye(Customer1, Waiter1), 17)
18
Happens(WalkThroughDoor21(Customer1, MainEntrance1), 18)
19
−At(Customer1, DiningRoom1)
−NearPortal(Customer1, KitchenDoor1)
+At(Customer1, Street1)