CHAPTER

Acquisition of Commonsense Knowledge 19

How do we acquire the knowledge necessary for commonsense reasoning? In this chapter, we discuss systems and methods for acquiring this knowledge. First, we discuss the manual acquisition of commonsense knowledge in Cyc, WordNet, and ThoughtTreasure. We then describe systems for crowdsourcing commonsense knowledge, including Open Mind Common Sense (OMCS) and LEARNER. Next, we review games for acquiring commonsense knowledge, including Verbosity and Common Consensus. We then present methods for mining relation instances and implicit knowledge from natural language text. Next, we describe how to perform reasoning in the event calculus using knowledge acquired through crowdsourcing. After that, we compare the systems and methods for acquiring commonsense knowledge. Finally, we review estimates of the size of human commonsense knowledge.

19.1 MANUAL ACQUISITION OF COMMONSENSE KNOWLEDGE

One method for acquiring commonsense knowledge is to enter the knowledge by hand. We discuss several systems for which this is the primary method.

19.1.1 CYC

Douglas B. Lenat started development of the Cyc commonsense knowledge base and reasoning system in 1983. The Cyc representation language, CycL, was originally based on frames (lists of slot name-value pairs), and knowledge acquisition was performed using a frame editor. Starting in 1989, Cyc began to shift away from frames, and CycL evolved into a language based on first-order logic, with extensions for default reasoning and reification. In 1990, *contexts*, also known as *microtheories*, were added to Cyc. Contexts provide a way of keeping alternative and possibly conflicting axiomatizations separate. For example, we can keep our axiomatizations of relational space and metric space in two different contexts. A context consists of a context name and a set of CycL sentences. Contexts are also organized into a hierarchy. As of 2014, Cyc contains over 500,000 terms and about 7,000,000 assertions (Cycorp, 2014e).

Knowledge acquisition

In Cyc, acquisition of commonsense knowledge is performed by *ontological engineers* or knowledge engineers who assert CycL sentences into contexts using a graphical user interface. For example, the engineer enters

```
#$FoodMt
```

into the microtheory dialog box and the sentence

```
(#$implies (#$isa ?x #$RedApple)
           (#$colorOf ?x #$Red))
```

into the sentence dialog box and clicks `Assert Formula`. A variant of CycL called *knowledge entry text* (KE text) can also be used for entry:

```
In Mt: FoodMt
F: (implies (isa ?x RedApple) (colorOf ?x Red))
```

Text files containing CycL sentences can also be loaded into Cyc.

Rapid Knowledge Formation tools

In 1999, the Rapid Knowledge Formation (RKF) program was started by the Defense Advanced Research Projects Agency (DARPA). Its goal was to develop technologies allowing subject matter experts to enter knowledge directly into knowledge bases instead of requiring knowledge engineers to enter knowledge. A set of acquisition tools were developed for Cyc as part of this project. Tools are provided for

- identifying microtheories relevant to a topic
- listing the similarities and differences between two concepts
- deciding where to place a new concept in the hierarchy
- finding concepts related to a concept
- suggesting relations between two concepts
- defining a concept using a natural language phrase
- defining a fact using a natural language sentence
- defining a concept by modifying the definition of a similar concept
- defining IF-THEN rules
- defining a script, its participants, its events, and the ordering of its events

Feedback to the ontological engineer is provided by *knowledge entry facilitation* (KE facilitation) rules. Four types of feedback are provided: requirements, strong suggestions, weak suggestions, and neighbor suggestions. For example:

- *Requirement*: An arity must be defined for a relation.
- *Strong suggestion*: A state change event type should have a type defined for the object of the state change.
- *Weak suggestion*: A duration should be defined for an event type.
- *Neighbor suggestion*: Something that is true of one concept may also be true of a similar concept.

Several other acquisition tools have been developed for Cyc. The Factivore tool allows entry of facts such as properties of objects. The Predicate Populator allows entry of relations between concepts, such as the ingredients of a salad. The Abductive Sentence Suggestor uses machine learning on facts stored in the knowledge base to suggest new rules. Cyc also runs machine learning at night to identify inconsistencies and possible omissions in the knowledge base.

19.1.2 WORDNET

George A. Miller started development of the WordNet lexical database in 1985. WordNet contains words, phrases, and *synsets* (synonym sets), which are sets of words and phrases with the same meaning. It contains two types of relations useful for commonsense reasoning: *lexical relations* between words/phrases and *semantic relations* between synsets. Examples of lexical relations are antonym and participle. Examples of semantic relations are hyponym/hypernym (IS-A), member, part, and cause. WordNet 3.0 contains 117,659 synsets and 220 cause relations between verb synsets such as the following:

```
cause|affect,impress,move,strike|feel,experience
cause|encourage|hope
cause|enrage|rage
cause|feed,give|eat
cause|give|have,have_got,hold
cause|incite,instigate,set_off,stir_up|act,move
cause|keep_up|stay_up,sit_up
cause|lay,put_down,repose|lie
cause|open,open_up|open,open_up
cause|remind|remember,retrieve,recall,call_back,call_up,
recollect,think
cause|resonate,come_across|understand
cause|resuscitate,revive|come_to,revive,resuscitate
cause|strengthen,beef_up,fortify|strengthen
```

In WordNet, acquisition is performed by lexicographers who enter data into plain text files called *lexicographer files*. Each line of a lexicographer file defines a synset. Relations are defined using short character sequences such as `!` (antonym), `<` (participle), `@` (hypernym), `#m` (member), `#p` (part), and `>` (cause).

Here is an example of a synset definition:

```
{ canine, [ dog, cat1,! ] pooch, domestic_dog, canid,@ }
```

This specifies a synset consisting of the words `canine`, `dog`, and `pooch`, and the phrase `domestic dog`. If a word or phrase occurs in more than one synset in a file, it is followed by a number that serves to uniquely identify the occurrence. The notation `[ dog, cat1,! ]` defines an antonym relation between the word `dog` in this synset and the word `cat` in another synset in the same file where it appears as `cat1` in that synset's definition. The notation `canid,@` defines a hypernym relation between this synset and another synset that contains the word `canid`.

Feedback to the lexicographer is provided when WordNet's `grind` program is run to convert the lexicographer files into a WordNet database. Error messages are produced when syntax errors are detected and when a reference to a word or phrase in a relation specification cannot be resolved.

Elaboration of entered data consists of filling in inverse relations. For example, if the synset containing `canid` is a hypernym of the synset containing `dog`, then the synset containing `dog` is a hyponym of the synset containing `canid`.

19.1.3 THOUGHTTREASURE

The ThoughtTreasure commonsense knowledge base and architecture for natural language processing was begun in 1993. It contains both declarative and procedural knowledge. Its declarative knowledge includes English and French words, phrases, and inflections; verb argument structure; names; concepts; relations among concepts including IS-A and part; facts; scripts; and spatial grids. Its procedural knowledge consists of a collection of agents for textual entity recognition, planning, model finding, question answering, and question asking. ThoughtTreasure 0.00022 contains 27,093 concepts and 51,305 assertions (Mueller, 2000). The contents of ThoughtTreasure 0.00022, including linguistic knowledge, were converted to CycL for loading into OpenCyc. This dump consists of 72,554 constants and 547,651 assertions.

Acquisition of declarative knowledge

In ThoughtTreasure, acquisition of declarative knowledge is performed by knowledge engineers who enter knowledge into plain text files using a concise format. The format is designed to speed knowledge entry and enable the presentation of lots of related knowledge on the screen during entry. Here is an example using the format:

```
=concept//
==action//
===action-open//open.Véz/ouvrir.Véy/
|r1=grasper|r2=physical-object|leadto2=state-open|
==state//
===state-open//open.Az/ouvert.Ay/
|r1=physical-object|
```

Indentation is used to express hierarchy. This text defines the concept `concept`; `action`, which is a `concept`; `action-open`, which is an `action`; `state`, which is a `concept`; and `state-open`, which is a `state`. This also specifies words for expressing the concepts `action-open` and `state-open`. The concept `action-open` is expressed in English by *open*. The *feature character* `z` indicates English. This is a verb (`V`), and it takes a direct object (`é`). This concept can also be expressed as `ouvrir` in French (`y`). The feature character `A` indicates an adjective. Words and phrases are specified using their citation forms, such as the infinitive for a verb and the singular form for a noun. An inflection dictionary maps citation forms such as *open* to inflections:

```
open.Az/
open.Vz/
open.f/
open.p1S/opens.p3S/open.pP/
opened.i/opened.d/opening.e/
```

Several types of feedback on declarative knowledge are provided. When the knowledge is loaded into ThoughtTreasure, messages are produced when syntax errors are detected and when a concept is defined multiple times. ThoughtTreasure also provides several tools to mine the knowledge base for patterns and potential problems, such as phrase derivation patterns, suffixes that identify the gender of French nouns, and lexical entries ambiguous as to part of speech.

The example text also specifies that the type of the first argument of `action-open` is `grasper`, the type of the second argument of `action-open` is `physical-object`, and the effect of this action is `state-open`. The notation

```
leadto2=state-open
```

inside the definition for `action-open` is an abbreviation for

```
[leadto2 action-open state-open]
```

which in turn is an abbreviation for

```
[leadto [action-open ?x ?y] [state-open ?y]]
```

Acquisition of procedural knowledge

In ThoughtTreasure, acquisition of procedural knowledge is performed by programmers who write code in the C language. For example, the following code defines the `TVSetOff` planning agent, which turns off a television set:

```
void PA_TVSetOff(Context *cx, Subgoal *sg, Ts *ts, Obj *a, Obj *o)
{
  Obj *p;
  switch (sg->state) {
    case STBEGIN:
      if (!(p = DbRetrievePart(ts, NULL, N("power-switch"),
      I(o, 1)))) {
        goto failure;
      }
      SG(cx, sg, 1, STFAILURE, L(N("switch-off"), p, E));
      return;
    case 1:
      WAIT_PTN(cx, sg, 0, STSUCCESS, L(N("TV-set-off"),
      I(o, 1), E));
      WAIT_TS(cx, sg, ts, (Dur)10, STFAILURE);
      return;
    }
  failure:
    TOSTATE(cx, sg, STFAILURE);
  }
```

A planning agent is a finite state machine. In state `STBEGIN`, the `DbRetrievePart` function is used to retrieve the power switch of the television set. The `SG` function is used to initiate a subgoal for the power switch to be off (which is achieved by another planning agent). If the subgoal succeeds, then the agent transitions to state 1. Otherwise, the agent transitions to state `STFAILURE`, and the agent terminates with failure. In state `1`, the `WAIT_PTN` function is used to wait until the television set is off, upon which the agent transitions to state `STSUCCESS` and the agent terminates with success. The `WAIT_TS` function is used to wait for 10 s, after which the agent transitions to state `STFAILURE` and the agent terminates with failure. The programmer tests procedural knowledge by integrating it into the system and testing the system on existing or new test cases.

19.2 CROWDSOURCING COMMONSENSE KNOWLEDGE

Manual entry of commonsense knowledge tends to be a slow process, and it tends to require specialists. An alternative is to design crowdsourcing systems that allow many nonspecialists to enter commonsense knowledge quickly.

19.2.1 OPEN MIND COMMON SENSE

The OMCS project to acquire commonsense knowledge from volunteer contributors over the Internet was started in 1999 by Push Singh. Users sign up for an account and then log in to enter knowledge into the OMCS knowledge base. In 2002, OMCS contained 456,195 facts (Singh et al., 2002). In 2014, OMCS contains about 1.1 million facts (Catherine Havasi, personal communication, June 11, 2014).

Acquisition of commonsense knowledge is performed using a collection of *elicitation activities*, each with its own entry screen. Some of the activities require knowledge to be entered into templates, and other activities allow entry of free-form text.

Template-based activities

Template-based activities display a template with zero or more blanks filled in and ask the user to fill in the remaining blanks. Examples of templates are the following:

```
People want __________
__________ is for __________
The effect of __________ is __________
```

Free-form activities

Free-form activities allow the user to enter arbitrary sentences. One free-form activity asks the user to supply a story given a title. For example, in response to the title `Playing Hockey`, a user entered the following story:

```
I grabbed my ice skates and hockey stick.
I walked over to the ice rink.
I saw that the hockey game had already started.
```

Another free-form activity presents a story and asks the user to enter knowledge useful for understanding the story. For example, a user was presented with the following:

```
Frank and Will are friends. Frank punched Will.
```

In response, the user entered the following:

```
Frank got upset.
Frank may not like Will anymore.
Will is probably hurt.
Frank could get in trouble for hurting Will.
```

Generation of plausible facts

A mechanism was designed and implemented by Thomas Lin to allow OMCS to suggest new plausible facts when a user enters a fact (but an interface to it was not deployed on the OMCS web site). Plausible facts are generated using a collection of *analogical inference rules* such as the following:

```
?NP1 ?V1 ?NP2                    (19.1)
?NP3 ?V1 ?NP4
?NP3 ?V2 ?NP4
→
?NP1 ?V2 ?NP2
```

NP is a noun phrase and V is a verb (possibly including auxiliary verbs). These can be recognized using part-of-speech taggers and syntactic parsers. For example, suppose the user enters

```
A mother can have a baby.
```

and the following facts were previously entered:

```
A child can have a goldfish.
A child can feed a goldfish.
```

Rule 19.1 then generates the following plausible fact:

```
A mother can feed a baby.
```

A more complex analogical inference rule is the following:

```
?NP1 ?V1 ?NP2                    (19.2)
?NP2 ?V2 ?NP3
?NP1 ?V3 ?NP3
```

```
?NP4 ?V1 ?NP5
?NP5 ?V2 ?NP6
→
?NP4 ?V3 ?NP6
```

The score for a plausible fact generated from the consequent of a rule is computed based on the number of distinct matches of the antecedent of the rule against facts in the knowledge base. A plausible fact that is generated more ways gets a higher score.

Extracting structured knowledge

Patterns were used to convert commonsense knowledge expressed as free-form text in OMCS's knowledge base into a more structured representation. For example, using the pattern

```
The effect of ?NP1 is ?NP2
```

facts such as the following were extracted:

```
(The effect of (buying a hamburger) is (having less money))
(The effect of (celebrating) is (being happy))
(The effect of (chatting with friends) is (a cozy feeling))
(The effect of (creating art) is (pleasure))
(The effect of (eating) is (feeling full))
(The effect of (giving a gift) is (to make someone happy))
(The effect of (going on strike) is (to harm management))
(The effect of (hurting someone else) is (pain))
(The effect of (mailing a letter) is (communicating with someone))
```

19.2.2 LEARNER

The LEARNER system for acquiring commonsense knowledge from volunteer contributors was begun in 2001 by Timothy Chklovski. It uses analogy to drive the acquisition of single-sentence facts such as `cats eat mice`. The user selects a concept such as `cat` or `newspaper`, the user enters several facts about the concept, and then the system asks follow-up questions about the concept. For example, for `newspaper`, LEARNER may ask:

```
newspapers contain information?
all newspapers have pages?
newspapers are for reading?
newspapers can contain recipes?
```

The user then answers each question with either `Yes`, `No`, `Some/Sometimes`, `Matter of opinion`, or `Nonsensical question`.

For a natural language sentence to be accepted by LEARNER, (1) the parser must be able to parse it, (2) it must not be an imperative sentence, (3) it must not use the past tense or a past perfect or present perfect construction, (4) it must not start with a pronoun, demonstrative determiner, or possessive determiner, and (5) it must not contain conjunctions.

Facts acquired by Learner are automatically classified into the following categories: ISA, QUALIFIED-ISA, DEFINITION, ACTION, QUALIFIED-ACTION, ACTION-ON, PROPERTY, COMPARATIVE, FUNCTION, MADE-OF, PART-OF, REQUIRES, and POSSIBLE STATE. Learner accepts simple facts about actions such as qualified actions (`kangaroos jump very high`) and requirements (`writing requires literacy`). But Learner does not address knowledge about scripts, goals, and the effects of events (Chklovski, 2003b, p. 37).

Learner was used to elaborate a subset of the OMCS knowledge base. The OMCS facts accepted by Learner were used as the initial knowledge base, which was then grown by contributors who entered new facts and responded to follow-up questions posed by Learner. A total of 31,620 new facts were acquired, starting from an initial set of 53,447 accepted OMCS facts (Chklovski, 2003b, p. 129).

Generation of follow-up questions

In Learner, follow-up questions are generated based on previously entered facts as follows:

1. Other concepts in the knowledge base similar to the selected concept are identified. The similarity of two concepts is computed based on how many facts they share, giving greater weight to concepts sharing properties that are more rare. For example, for the selected concept `newspaper`, the similar concept `book` is identified.
2. Follow-up questions are generated from facts about a similar concept that involve properties not already asserted about the selected concept. The similar concept in these facts is replaced by the selected concept. For example, if the property of containing information is not already asserted about `newspaper`, then `book` in

   ```
   books contain information
   ```

   is replaced by `newspaper`, which yields the follow-up question

   ```
   newspapers contain information?
   ```

Natural language resources are used to inflect words properly.

Representation of sentences

Learner represents sentences as atoms of the form

$$property(concept) = truth\text{-}value$$

The sentence `cats eat mice` is converted into both

```
cat-eat(mouse)=1
```

and

```
eat-mouse(cat)=1
```

Natural language resources are used to convert an inflected word into its base form. LEARNER also deals with negation. For example, `cats do not fly` is converted into

```
fly(cat)=0
```

Computing similarity

The similarity of two concepts is computed in LEARNER as follows. We first define *FreqWt*, the weight of a property, as follows:

$$FreqWt(p) = \begin{cases} 2 & \text{if } \mathcal{C}(p) = \varnothing \\ 1 + 1/\log_2(|\mathcal{C}(p)| + 1) & \text{otherwise} \end{cases}$$

where $\mathcal{C}(p)$ is the set of all concepts for which p is true in the knowledge base ($\{c \mid p(c) = 1\}$). *FreqWt* gives a higher weight to properties that are more rarely true in the knowledge base. We then define *Wt*, the weight assigned to two truth values of a predicate, as follows:

$$Wt(tv_1, tv_2, p) = \begin{cases} FreqWt(p) & \text{if } tv_1 = tv_2 = 1 \\ -1.5 & \text{if } tv_1 \neq tv_2 \\ 0 & \text{if } tv_1 = tv_2 = 0 \end{cases}$$

We then compute the similarity of concepts c and c' as follows:

$$Sim(c, c') = \sum_{p \in P} Wt(p(c), p(c'), p)$$

where P is the set of all properties in the knowledge base and $p(c)$ is the truth value of c in the knowledge base.

19.2.3 OPEN MIND COMMONS AND CONCEPTNET

The Open Mind Commons interface for knowledge acquisition was begun in 2006 by Robert Speer. The user navigates to a concept, and then the user is asked to evaluate existing facts involving the concept, evaluate plausible facts about the concept generated by analogy, and enter new facts about the concept by filling in templates.

For example, when the user navigates to `orchestra`, the user is asked several things. First, the user is asked to evaluate the following existing facts:

```
You are likely to find a flute in an orchestra.
You are likely to find a snare drum in an orchestra.
An orchestra plays music.
Orchestra is related to music group.
```

Second, the user is asked to evaluate the following new, plausible facts:

```
You are likely to find a trumpet in an orchestra.
You are likely to find a harmonica in an orchestra.
```

Third, the user is asked to fill in the following templates:

```
You are likely to find __________ in an orchestra.
An orchestra is __________.
```

Open Mind Commons replaces the previous interface to OMCS as well as its back-end database. Open Mind Commons adds knowledge directly to the ConceptNet commonsense knowledge base developed by Hugo Liu, Catherine Havasi, Robert Speer, Jason B. Alonso, and colleagues. ConceptNet was originally populated by extracting ⟨*relation*, *concept*, *concept*⟩ triples from the OMCS knowledge base. Each ConceptNet record consists of the following fields:

- Relation
- Concept1
- Concept2
- Value: positive for true and negative for false, with a higher magnitude corresponding to a higher degree of validation by sources
- Sources
- Surface text: original natural language sentence

The relations in ConceptNet include IsA, UsedFor, HasA, CapableOf, Desires, CreatedBy, PartOf, Causes, HasFirstSubevent, AtLocation, HasProperty, LocatedNear, DefinedAs, SymbolOf, ReceivesAction, HasPrerequisite, MotivatedByGoal, CausesDesire, MadeOf, HasSubevent, and HasLastSubevent.

ConceptNet 5 contains 36,963 records with a relation of Causes, which represents the effects of events. The following patterns are used to extract Causes records from OMCS:

```
The effect of ?XP1 is that ?S2
The effect of ?XP1 is ?NP2
The consequence of ?XP1 is that ?XP2
The consequence of ?XP1 is ?XP2
Something that might happen as a consequence of ?XP1 is that ?XP2
Something that might happen as a consequence of ?XP1 is ?XP2
?AdvP ?NP1 causes you to ?VP2
?AdvP ?NP1 causes ?NP2
```

NP is a noun phrase, VP is a verb phrase, AdvP is an adverb phrase, S is a sentence, and XP is an NP, VP, or S.

ConceptNet 5 includes knowledge from other projects including DBpedia, Wiktionary (Wikimedia Foundation, 2014b), and WordNet.

19.2.4 ANALOGYSPACE

Open Mind Commons uses a technique called AnalogySpace, developed by Robert Speer, Catherine Havasi, and Henry Lieberman, to generate plausible facts to be evaluated by contributors. AnalogySpace is inspired by the method of latent semantic analysis (LSA) used in information retrieval. Like LSA, Analogy space uses the technique of singular value decomposition. AnalogySpace uses the *property*(*concept*) representation of LEARNER.

Knowledge base

We define an AnalogySpace *knowledge base* as a set KB of atoms of the form

$$\mathit{property}(\mathit{concept}) = \mathit{value}$$

such that, if $p(c) = v_1 \in KB$ and $p(c) = v_2 \in KB$, then $v_1 = v_2$. That is, $p(c)$ can only have one value in KB. As in ConceptNet, positive values indicate true and negative values indicate false.

We convert ConceptNet into an initial knowledge base KB_0 by forming the set of atoms

$$\mathit{concept1\text{-}relation}(\mathit{concept2}) = \mathit{value}$$
$$\mathit{relation\text{-}concept2}(\mathit{concept1}) = \mathit{value}$$

for every $\langle \mathit{relation}, \mathit{concept1}, \mathit{concept2}, \mathit{value} \rangle$ in ConceptNet. We then form a set $\{c_1, \ldots, c_m\}$ of all concepts in KB_0 and a set $\{p_1, \ldots, p_n\}$ of all properties in KB_0. We then convert KB_0 into an m-by-n real matrix A whose (i, j)th element is

$$\begin{cases} v & \text{if } (\mathrm{p_j(c_i) = v}) \in \mathrm{KB}_0 \\ 0 & \text{otherwise} \end{cases}$$

Singular value decomposition

We define singular value decomposition as follows.

Definition 19.1. If A is an m-by-n real matrix, then a *singular value decomposition* of A is a factorization $A = U\Sigma V^T$, where

- $U = [u_1, \ldots, u_m]$ is an orthogonal m-by-m real matrix
- $V = [v_1, \ldots, v_n]$ is an orthogonal n-by-n real matrix
- $\Sigma = \mathit{diag}(\sigma_1, \ldots, \sigma_p)$ is an m-by-n real matrix, where $\sigma_1 \geq \ldots \geq \sigma_p \geq 0$ and $p = \min(m, n)$

The following theorem guarantees that such a decomposition exists.

Theorem 19.1. *If A is a real m-by-n matrix, then a singular value decomposition of A exists.*

Proof. See Theorem 2.5.2 of Golub and Van Loan (1996) or Theorem 4.1.1 of Watkins (2010). ■

The *rank* of A is the number of nonzero values in $\{\sigma_1, \ldots, \sigma_p\}$.

Truncated singular value decomposition

We define truncated singular value decomposition as follows.

Definition 19.2. If $U\Sigma V^T$ is a singular value decomposition of a real matrix A and $k < \operatorname{rank}(A)$, then

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

is a *k-truncated singular value decomposition* of A.

Truncated singular value decomposition gives the closest approximation of a given rank to a given matrix.

Theorem 19.2. *If A is a real matrix, $k <$ rank(A), and A_k is a k-truncated singular value decomposition of A, then*

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2$$

Proof. See Theorem 2.5.3 of Golub and Van Loan (1996) or Theorem 4.2.15 of Watkins (2010). ■

Generation of plausible facts

We generate plausible facts as follows. Given A, we compute its k-truncated singular value decomposition A_k, with $k < \text{rank}(A)$. For a large knowledge base, AnalogyS-pace typically uses $k = 150$. We convert A_k into a plausible knowledge base KB_1 by forming the set of atoms

$$\{p_j(c_i) = (A_k)_{ij}\}$$

The plausible facts are then defined as the first N atoms $p(c) = v$ in KB_1, sorted in descending order by v, such that $p(c)$ is not in the original knowledge base KB_0.

Example: flat similarity

Like Learner, AnalogySpace can generate plausible facts based on a notion of *flat similarity*, namely, the degree to which two concepts share properties. Suppose we have atoms that represent that flutes and oboes are woodwinds, bass guitars and electric pianos are electric instruments, flutes are in orchestras, and bass guitars are in jazz bands.

```
IsA-woodwind(flute)=1
IsA-woodwind(oboe)=1
IsA-electric_instrument(bass_guitar)=1
IsA-electric_instrument(electric_piano)=1
AtLocation-orchestra(flute)=1
AtLocation-jazz_band(bass_guitar)=1
```

Flutes and oboes are similar because they share the property of being a woodwind. Bass guitars and electric pianos are similar because they share the property of being an electric instrument. Given these atoms, AnalogySpace with $k = 2$ produces the following plausible facts:

```
AtLocation-orchestra(oboe)=0.4472
AtLocation-jazz_band(electric_piano)=0.4472
```

Because flutes and oboes are similar, and flutes are in orchestras, it is plausible that oboes are also in orchestras. Similarly, because bass guitars and electric pianos are similar, and bass guitars are in jazz bands, it is plausible that electric pianos are also in jazz bands.

Example: deep similarity

Unlike Learner, however, AnalogySpace can generate plausible facts based on a notion of *deep similarity*, namely, the degree to which two concepts share properties that themselves are similar. Suppose we have atoms that represent that violins are classical instruments and in orchestras, and that electric guitars are twentieth century instruments and in jazz bands:

```
IsA-classical_instrument(violin)=1
AtLocation-orchestra(violin)=1
IsA-20th_century_instrument(electric_guitar)=1
AtLocation-jazz_band(electric_guitar)=1
```

Being a classical instrument and being in an orchestra are similar properties, because these are both properties of violins. Similarly, being a twentieth century instrument and being in a jazz band are similar properties, because these are both properties of electric guitars. Now suppose we also have atoms that represent that flutes are in orchestras, bass guitars are in jazz bands, oboes are classical instruments, and electric pianos are twentieth century instruments:

```
AtLocation-orchestra(flute)=1
AtLocation-jazz_band(bass_guitar)=1
IsA-classical_instrument(oboe)=1
IsA-20th_century_instrument(electric_piano)=1
```

Given these atoms, AnalogySpace with $k = 2$ produces the following plausible facts:

```
AtLocation-orchestra(oboe)=0.5
AtLocation-jazz_band(electric_piano)=0.5
```

Flutes and oboes do not share any properties. But, because flutes and oboes share the properties of being in an orchestra and being a classical instrument, which are similar properties, and flutes are in orchestras, it is plausible that oboes are in orchestras. Similarly, bass guitars and electric pianos do not share any properties. But, because bass guitars and electric pianos share the properties of being in a jazz band and being a twentieth century instrument, which are similar properties, and bass guitars are in jazz bands, it is plausible that electric pianos are in jazz bands.

19.3 GAMES FOR ACQUISITION OF COMMONSENSE KNOWLEDGE

Related to crowdsourcing efforts like OMCS is the idea of using games to motivate people to contribute commonsense knowledge. We review several representative systems.

19.3.1 ANIMAL

An early example of a multiuser game for acquiring knowledge was the ANIMAL program written by Nathan Teichholtz in 1973. The program stores knowledge acquired from multiple users in a common data file `ANIMAL.GME`. The user thinks of an animal and the computer tries to guess the name of the animal by asking yes-no questions. When the computer guesses incorrectly, the computer asks the user to provide a yes-no question that distinguishes the animal guessed by the computer from the animal the user was thinking of. As games are played, the program grows a knowledge base about properties of animals (or any other category, after changing `ANIMAL` to that category in the program).

19.3.2 GAME FOR INTERACTIVE OPEN MIND IMPROVEMENT

In 2002, David Garcia, Catherine Havasi, and Katherine Todd implemented the Game for Interactive Open Mind Improvement (GIOMI), a game for rating facts in the OMCS knowledge base. The user selects a topic, the user is presented with facts about that topic from OMCS, and then the user is asked to rate them as good or bad. Users receive points for each rating, and the number of points per rating increases during a session.

19.3.3 VERBOSITY

In 2006, Luis von Ahn, Mihir Kedia, and Manuel Blum introduced Verbosity, a game for acquiring commonsense knowledge. A user logs in and is paired with another randomly selected logged-in user. One user is assigned the role of *narrator*, whereas the other user becomes the *guesser*. A concept such as `book` is presented to the narrator but not the guesser. The narrator's goal is to get the guesser to type the name of the concept. The narrator can provide hints to the guesser by filling in templates such as the following:

```
__________ is a kind of __________
__________ is used for __________
__________ is typically near/in/on __________
__________ is the opposite of __________
__________ is related to __________
```

The guesses are provided to the narrator, and the narrator can tell the guesser whether a guess is or is not getting closer to the right answer. 212,090 facts from Verbosity were added to ConceptNet (Speer, Havasi, & Surana, 2010, p. 107).

19.3.4 COMMON CONSENSUS

In 2006, Henry Lieberman, Dustin A. Smith, and Alea Teeters introduced Common Consensus, a game for acquiring commonsense knowledge about goals. The system selects a goal such as `watch television` from the knowledge base. It then asks the user a question using one of the following templates designed to elicit particular types of goal-related knowledge:

- *Supergoal*: `Why would you want to` ***goal*** `?`
- *Subgoal*: `What is something you can do if you wanted to` ***goal*** `?`
- *Similar goal*: `What is another goal similar to` ***goal*** `?`
- *Duration*: `About how long would it take to` ***goal*** `?`

- *Instrument*: `What are some things you would use to` ***goal*** `?`
- *Location*: `Where are some places you would` ***goal*** `?`

The user types in multiple answers (except for Duration questions) and receives points for each answer equal to the number of other users who gave that answer or a similar answer.

19.4 MINING COMMONSENSE KNOWLEDGE

Instead of having specialists or nonspecialists contribute commonsense knowledge, another approach is to extract, learn, or mine commonsense knowledge from large natural language text corpora.

19.4.1 MINING RELATION INSTANCES FROM TEXT

In 1992, Marti A. Hearst proposed to mine relation instances from large text corpora and described several pattern-based rules for extracting likely instances of the hyponym or IS-A relation from text. They include the following:

```
?NP0 (such as|including|especially) ?NP1 ?NP2 ...
→
ISA(?NP1, ?NP0)
ISA(?NP2, ?NP0)
⋮

such ?NP0 as ?NP1 ?NP2 ...
→
ISA(?NP1, ?NP0)
ISA(?NP2, ?NP0)
⋮

?NP1 ?NP2 ... (or|and) other ?NP0                              (19.3)
→
ISA(?NP1, ?NP0)
ISA(?NP2, ?NP0)
⋮
```

For example, given

```
... sudden dyspnea, syncope, and other findings including orthopnea
and hypoxemia ...
```

these patterns allow us to extract

```
IS-A(sudden dyspnea, finding)
IS-A(syncope, finding)
IS-A(orthopnea, finding)
IS-A(hypoxemia, finding)
```

Hearst also presented the following method for learning new patterns for extracting instances of a given relation:

1. Start with a list of known instances of the relation.
2. Search a corpus for sentences containing both arguments of any of the known instances of the relation.
3. Generalize the retrieved sentences to form pattern-based rules.

For example, starting with the instances

```
IS-A(syncope, finding)
IS-A(hypoxemia, finding)
```

we retrieve the sentences

```
Syncope is a relatively common finding.
Hypoxemia was the only significant finding.
```

We generalize these sentences into the following rule:

```
?NP1 (was|is) (a|the) ADV ADJ ?NP0
→
ISA(?NP1, ?NP0)
```

19.4.2 MINING IMPLICIT KNOWLEDGE FROM TEXT

In 2002, Lenhart K. Schubert started developing the KNEXT system for extracting commonsense knowledge implicit in natural language sentences. The system takes parsed English sentences in Penn Treebank format as input and produces English sentences, called *factoids*, and logical forms as output.

For example, given the parse

```
((S (NP (NP (NNP Matt))
        (POS \'s) (NN grandfather))
    (VP (VBD created)
        (NP (DT a) (JJ beautiful) (NN painting)))))
```

KNEXT produces the factoids

```
A GRANDFATHER MAY CREATE A PAINTING.
A MALE MAY HAVE A GRANDFATHER.
A PAINTING CAN BE BEAUTIFUL.
```

and the logical forms

```
(:I (:Q THE GRANDFATHER.N) CREATE.V (:Q A{N} PAINTING.N))
(:I (:Q DET MALE*.N) HAVE.V (:Q DET GRANDFATHER.N))
(:I (:Q DET PAINTING.N) BEAUTIFUL.A)
```

Using Charniak's (2000) parser, KNEXT extracted 67,632,550 unique factoids from weblogs and 53,945,110 unique factoids from Wikipedia (J. M. Gordon, Van Durme, & Schubert, 2010, p. 10). The number of unique factoids continued to grow as more text was processed.

Logical forms

The language for logical forms is a simplified version of *episodic logic*, a logic that resembles natural language. An expression of the form

(:I a_1 p a_2 ...)

represents the application of predicate p to arguments $a_1, a_2, \ldots$. The expression

(:Q q p)

represents quantification, where q is a quantifier and p is a predicate.

Extracting logical forms

KNEXT extracts logical forms from a parse tree by recursively applying an ordered list of 80 pattern-based rules to the tree. A pattern-based rule is of the form

tree-pattern
$\rightarrow$
logical-form-template

An example is the following:

```
((* ~ NP VP) (NP) (* ~ VP) (VP) (*))
→
(:I 2 4)
```

The pattern `(* ~ NP VP)` matches zero or more subtrees not of type NP or VP, and the pattern `(NP)` matches a subtree of type NP. Whenever a rule's tree pattern matches a node of the tree, the rule's logical form template is instantiated and returned. Instantiating the logical form template involves replacing each (1-origin) index *i* by an abstracted version of the extracted logical form corresponding to the *i*th element of the tree pattern. Abstracting a logical form involves removing modifiers and generalizing entities to types.

Sharpening quantifiers

Jonathan Gordon and Lenhart Schubert developed a rule-based method for sharpening the quantifiers in logical forms produced by KNEXT such as `DET` into stronger quantifiers such as `some`, `many`, and `all-or-most`. The method takes logical forms as input and produces episodic logic sentences as output. For example, given

```
(:I (:Q DET MALE*.N) HAVE.V (:Q DET GRANDFATHER.N))
```

the method produces

```
(many-or-some x: [x male.n]
    (some e:
        (some y: [y person.n]
            [[x (have-as grandfather.n) y] ** e])))
```

Given an input logical form `(:I` a_1 p a_2 `...)`, a stronger quantifier is selected based on the strength of the association between a_1 and p a_2 ... as measured by pointwise mutual information—see Section 18.4.5.

1.5 million sharpened sentences have been produced (Schubert, Gordon, Stratos, & Rubinoff, 2011).

Extracting the effects of events

Gordon and Schubert developed a method for extracting entailment rules from parsed English sentences, including rules for the effects of events. The method is based on identifying expectation violations. For example, the sentence

```
Matt turned on the light, but it didn't light up.
```

can be abstracted into the inference rule

```
If a person turns on a light, then it may light up.
```

The first implementation of this method used TGrep2 patterns to identify candidate sentences for abstracting into inference rules. The TGrep2 patterns were later converted into TTT transductions that identify candidate sentences and transform them into initial logical forms.

19.5 EVENT CALCULUS REASONING OVER ACQUIRED COMMONSENSE KNOWLEDGE

Here is an example of how we can reason using the event calculus over commonsense knowledge acquired through crowdsourcing. We start with 396,018 facts from the OMCS knowledge base. We look through the knowledge base and notice that the effects of events are represented various ways. Some examples are the following:

```
If before you go to a restaurant the situation is you are hungry,
then afterwards the situation is likely to be you are full

The effect of attending a rock concert is ringing ears

Sometimes acting in a play causes the audience to laugh.

In the event "Billy is a soccer player.  Billy scored a goal.",
something that changed was Billy feels happy
```

Unlike the other examples, the first example includes both a fluent that becomes false after the event (`hungry`) and a fluent that becomes true after the event (`full`). Let us extract all facts of this type using the following regular expression:

```
If before you (.*) the situation is (.*) then afterwards
the situation is likely to be (.*)
```

This yields 215 facts. We assume that the first group matched with `(.*)` contains an event and that the second and third groups contain fluents.

We then analyze the syntax of events and fluents extracted using the previous regular expression. The top three syntactic patterns for events are as follows:

verb noun	106/215	49%
verb preposition noun	26/215	12%
verb	15/215	7%

The top three syntactic patterns for fluents are as follows:

adjective	114/430	27%
noun	31/430	7%
you are/were *adjective*	23/430	5%

We then take the 215 facts and convert the ones whose extracted events and fluents match these syntactic patterns into event calculus effect axioms. This yields 41 *Initiates* axioms and 41 *Terminates* axioms. Examples are as follows:

```
% If before you sleep the situation is tired, then afterwards
the situation
% is likely to be rested
initiates(sleep(A),rested(A),T) :- holdsAt(tired(A),T).
terminates(sleep(A),tired(A),T) :- agent(A), time(T).

% If before you eat vegetables the situation is hungry, then
afterwards
% the situation is likely to be full
initiates(eatVegetables(A),full(A),T) :- holdsAt(hungry(A),T).
terminates(eatVegetables(A),hungry(A),T) :- agent(A), time(T).
```

We can then reason using these extracted axioms using answer set programming. For example, we specify that John starts out tired and hungry. He sleeps and then eats vegetables.

```
agent(john).
holdsAt(tired(john),0).
holdsAt(hungry(john),0).
happens(sleep(john),1).
happens(eatVegetables(john),2).
:- releasedAt(F,0), fluent(F).
```

We then run `clingo` on this example along with the extracted axioms and the DEC rules. It is correctly inferred that (1) after sleeping, John is rested and no longer tired and (2) after eating vegetables, John is full and no longer hungry:

```
Answer: 1
0
hungry(john)
tired(john)
1
tired(john)
hungry(john)
happens(sleep(john),1)
```

```
2
-tired(john)
+rested(john)
hungry(john)
happens(eatVegetables(john),2)
3
-hungry(john)
+full(john)
rested(john)
```

19.6 COMPARISON OF ACQUISITION METHODS

Systems and methods for acquisition of commonsense knowledge are compared in Table 19.1. The earlier systems tend to rely on specialists for knowledge entry, whereas the later crowdsourcing systems acquire knowledge from nonspecialists. Cyc and many of the crowdsourcing systems provide feedback during knowledge entry. Many years after Cyc was first introduced, it still stacks up very well against the newer systems.

Although many of the systems acquire the effects of events, of the systems reviewed only Cyc and ThoughtTreasure clearly represent the arguments of an event, the arguments of the effects of the event, and how the arguments in the event map to the arguments of the effects of the event. For example, although OMCS contains facts from which argument information could conceivably be extracted such as

```
If before you serve customers the situation is the customers are
waiting, then afterwards the situation is likely to be the
customers are happy
```

it also contains many facts that do not contain this information such as

```
The effect of hurting someone else is pain
```

Table 19.1 Commonsense Knowledge Acquisition[a]

	Cyc	WN	TT	OMCS	L	OMC	G	M
Begun	1983	1985	1993	1999	2001	2006	1973	1992
Enterer	S	S	S	NS	NS	NS	NS	M
Interface	GUI	T	T	GUI	GUI	GUI	G	
Feedback	✓				✓	✓	✓	
Event effects	✓	✓	✓	✓		✓		✓
Arguments	✓		✓					
Disambiguated	✓	✓	✓					

[a] *WN, WordNet; TT, ThoughtTreasure; OMCS, Open Mind Common Sense; L, LEARNER; OMC, Open Mind Commons; G, games; M, mining S, specialist; NS, non-specialist; GUI, graphical user interface; T, text files.*

We would like to clearly represent, for example, that if a_1 hurts a_2, then a_1 will feel pain and a_2 will feel pain. This is represented in the event calculus as follows:

$$Initiates(Hurt(a_1, a_2), Pain(a_1), t)$$
$$Initiates(Hurt(a_1, a_2), Pain(a_2), t)$$

The knowledge acquired by many of the later systems is not disambiguated, although another crowdsourcing system, Open Mind Word Expert (Chklovski & Mihalcea, 2002), has been used to disambiguate facts in OMCS.

19.7 HOW BIG IS HUMAN COMMON SENSE?

How much commonsense knowledge do we need to acquire? How big is human commonsense knowledge? Landauer (1986) produced estimates of the size of an adult's long-term memory ranging from 0.5×10^9 to 3.4×10^9 bits using different methods:

1. Several estimates were based on the rate information is added to long-term memory—1.2 bits per second for text and 2.3 bits per second for pictures, from psychological experiments—and a linear accumulation over 70 years.
2. Another estimate took into account forgetting.
3. Another estimate was based on 346 bits to define each word in a dictionary containing 10^5 words, multiplied by 15 to take account of other domains of knowledge such as history, music, nature, personal life history, and motor skills.

Estimates ventured over the years are summarized in Table 19.2. For comparison, the KNEXT system has extracted 10^8 factoids (J. M. Gordon, Van Durme, & Schubert, 2010, p. 10).

Table 19.2 Estimates of the Size of Human Common Sense

Turing (1950, p. 442)	10^9 bits for a 70% chance of identifying a program as not a human after 5 min of text-based questioning
von Neumann (1958, pp. 63 and 64)	2.8×10^{20} bits of "memory capacity" based on 10^{10} neurons $\times$ 14 bits per second per neuron $\times$ 2×10^9 s, assuming "there is no true forgetting in the nervous system"
Minsky (1968, p. 26)	10^5-10^7 "elements of knowledge in order to behave with reasonable sensibility in ordinary situations"
Landauer (1986)	10^9 bits of "information from experience"
Schwartz (1988, p. 126)	0.8×10^{17} bits of "long-term memory"
Marvin Minsky (Lenat & Guha, 1990, p. 21)	10^7 entries, assuming a child acquires an entry every 10 seconds from ages 0 to 8
Kurzweil (1999, p. 119)	10^8 chunks or "bits of understanding, concepts, patterns, specific skills"
Moravec (1999, p. 56)	0.8×10^{15} bits of "nervous system memory" based on 10^{14} synapses $\times$ 1 byte per synapse
Mahoney (1999)	10^9 bits for predicting "how people respond to arbitrary input during communication"

BIBLIOGRAPHIC NOTES

Manual acquisition

Cyc was originally proposed as a way of overcoming knowledge acquisition bottlenecks (Lenat, Prakash, & Shepherd, 1985). An early version of Cyc called Knoesphere is described by Lenat, Borning, McDonald, Taylor, and Weyer (1983). The development of Cyc is documented in a series of papers by Lenat, Prakash, and Shepherd (1985), Lenat, Guha, Pittman, Pratt, and Shepherd (1990), Guha and Lenat (1994), and Lenat (1995). A book-length treatment of Cyc is provided by Lenat and Guha (1990). The Cyc frame editor is mentioned by Lenat, Prakash, and Shepherd (1985, p. 73). Cyc's shift from frames to predicate logic and the evolution of CycL are discussed by Guha and Lenat (1990a, 1994) and Lenat and Guha (1991a). Lenat and Guha (1991a, p. 87) note that CycL began in 1984 as a reimplementation of the frame-based Representation Language Language (RLL), which is described by Greiner and Lenat (1980). RLL was used in the EURISKO learning by discovery program (Lenat, 1984). CycL is described by Cycorp (2014c). The formalization of contexts was proposed by McCarthy (1987) and developed by Guha (1992) and McCarthy (1993). Cyc's contexts are discussed by Guha and Lenat (1994). Entry of knowledge into Cyc is discussed by Cycorp (2014c, chap. 3). Cyc's RKF tools are discussed by Cycorp (2002a, sec. 19.5), Panton et al. (2002), and Witbrock et al. (2003). KE text is described by Cycorp (2014f). KE facilitation rules are described by Cycorp (2014c, sec. 8.5). The Factivore tool is described by Belasco et al. (2004) and Witbrock et al. (2005). The Predicate Populator and the Abductive Sentence Suggestor are described by Witbrock et al. (2005). Cyc's machine learning at night is mentioned by Guha and Lenat (1990a, p. 53).

WordNet is described by Fellbaum (1998). The WordNet lexicographer file format is described in the WordNet reference manual (Trustees of Princeton University, 2006). ThoughtTreasure is described by Mueller (1998). Scripts in ThoughtTreasure are described by Mueller (2000).

Crowdsourcing

Open Mind Common Sense is described by Singh (2002) and Singh et al. (2002). Part of speech tagging and syntactic parsing are discussed by Manning and Schütze (1999) and Jurafsky and Martin (2009). LEARNER is described by Chklovski (2003a, 2003b). Open Mind Commons is described by Speer (2007). An early version of ConceptNet called the Concept Node Graph (30,000 nodes) is described by Liu and Lieberman (2002). Another early version called OMCSNet (140,000 items) is described by Liu (2003). ConceptNet 2 (over 300,000 nodes) is described by Liu and Singh (2004b). ConceptNet 3 (over 150,000 nodes) is described by Speer (2007) and Havasi, Speer, and Alonso (2007, 2009). ConceptNet 5 (3.9 million nodes), which pulls in DBpedia, Wiktionary, WordNet, and other sources, is described by Speer and Havasi (2012).

Analogy

Lenat, Prakash, and Shepherd (1985, pp. 66, 68-70) proposed to use analogy to suggest new concepts and rules to be added to Cyc. T. Lin (2002) and Singh et al. (2002) discuss the generation of plausible facts using analogical inference rules. Chklovski and Fredette (2001) and Chklovski (2003b) discuss the generation of plausible facts using analogy in Learner. Latent semantic analysis is described by Deerwester, Dumais, Furnas, Landauer, and Harshman (1990). AnalogySpace is described by Speer (2007), Speer, Havasi, and Lieberman (2008), and Havasi (2009). Singular value decomposition is described by Golub and Van Loan (1996) and Watkins (2010). Truncated singular value decomposition is described by Hansen (1998). Krishnamurthy and Lieberman (2010) describe an analogy algorithm based on truncated singular value decomposition that finds analogies in ConceptNet.

Games

A description of ANIMAL and the program code are provided by Ahl (1973, p. 17). Stork and Lam (2000) created an Internet version of ANIMAL. GIOMI is described by Garcia, Havasi, and Todd (2002). Verbosity is described by von Ahn, Kedia, and Blum (2006). von Ahn and Dabbish (2008) present templates and strategies for building successful games with a purpose. Common Consensus is described by Lieberman, Smith, and Teeters (2007).

Mining

Extracting and learning information from text corpora is discussed by Zernik (1991), Armstrong (1993), Manning and Schütze (1999), and Jurafsky and Martin (2009). Hearst (1992, 1998) proposes to extract relation instances automatically from large text corpora using patterns. Hobbs (1984, p. 8) mentions a rule similar to (19.3). Richardson, Dolan, and Vanderwende (1998) describe the MindNet lexical resource, which is built by extracting relation instances from dictionaries and encyclopedias. Berland and Charniak (1999) extract instances of the part relation from text. D. Lin and Pantel (2001) learn inference rules such as “X wrote Y $\approx$ X is the author of Y” from text. Chambers and Jurafsky (2008, 2009) learn narrative schemas (scripts) and their participants from text.

Schubert (2002) proposes to extract world knowledge from text. Knext and related research are described by Schubert (2002), Schubert and Tong (2003), Van Durme (2009), Schubert, Gordon, Stratos, and Rubinoff (2011), and J. M. Gordon (2014). Episodic logic is described by Schubert and Hwang (2000). Sharpening of quantifiers is discussed by J. M. Gordon and Schubert (2010). Extracting entailment rules from text is discussed by J. M. Gordon and Schubert (2011) and J. M. Gordon (2014). J. M. Gordon and Schubert (2012) also discuss learning event frequencies from text. The Penn Treebank is described by Marcus, Marcinkiewicz, and Santorini (1993). TGrep2 is described by Rohde (2005), and TTT is described by Purtee and Schubert (2012). Sil and Yates (2011) present a system that learns the preconditions and effects of 40 actions.

EXERCISES

19.1 Give examples in which Open Mind Common Sense analogical inference rule (19.2) succeeds. Give examples in which the rule fails.

19.2 Propose other analogical inference rules along the lines of Open Mind Common Sense rules (19.1) and (19.2).

19.3 Write a program that applies analogical inference rules to a text-based commonsense knowledge base.

19.4 (Research Problem) Design and implement a crowdsourcing system or game to acquire *Initiates* and *Terminates* axioms including event and fluent arguments and their sorts.

19.5 (Research Problem) Mine *Initiates* and *Terminates* axioms including event and fluent arguments and their sorts from text.