RESOLUTION OF COLLECTIVE-DISTRIBUTIVE AMBIGUITY USING MODEL-BASED REASONING

Chinatsu Aone*
MCC
3500 West Balcones Center Dr.
Austin, TX 78759
aone@mcc.com

Abstract

I present a semantic analysis of collective-distributive ambiguity, and resolution of such ambiguity by model-based reasoning. This approach goes beyond Scha and Stallard [17], whose reasoning capability was limited to checking semantic types. My semantic analysis is based on Link [14, 13] and Roberts [15], where distributivity comes uniformly from a quantificational operator, either explicit (e.g. *each*) or implicit (e.g. the D operator). I view the semantics module of the natural language system as a hypothesis generator and the reasoner in the pragmatics module as a hypothesis filter (cf. Simmons and Davis [18]). The reasoner utilizes a model consisting of domain-dependent constraints and domain-independent axioms for disambiguation. There are two kinds of constraints, type constraints and numerical constraints, and they are associated with predicates in the knowledge base. Whenever additional information is derived from the model, the Contradiction Checker is invoked to detect any contradiction in a hypothesis using simple mathematical knowledge. CDCL (Collective-Distributive Constraint Language) is used to represent hypotheses, constraints, and axioms in a way isomorphic to diagram representations of collective-distributive ambiguity.

1 Semantics of Collective-Distributive Ambiguity

Collective-distributive ambiguity can be illustrated by the following sentence.

(1) Two students moved a desk upstairs.

(1) means either that two students TOGETHER moved one desk (a collective reading) or that each of them moved a desk SEPARATELY (a distributive reading). Following Link [14, 13] and Roberts [15], distributivity comes from either an explicit quantificational operator like *each* or an implicit distributive operator called the **D operator**. The D operator was motivated by the equivalence in the semantics of the following sentences.

(2) a. *Every* student in this class lifted the piano.
b. Students in this class *each* lifted the piano.
c. Students in this class lifted the piano.
(the distributive reading)

Thus, the distributive readings of (1) and (2c) result from applying the D operator to the subjects.

Now, look at another sentence "Five students ate four slices of pizza." It has 8 POSSIBLE readings because the D operator may apply to each of the two arguments of *eat*, and the two NPs can take scope over each other. Thus, $2 \times 2 \times 2 = 8$.[1] I have extended Link's and Roberts's theories to quantify over events in Discourse Representation Theory (cf. Kamp [10], Heim [9], Aone [2]) so that these readings can be systematically generated and represented in the semantics module. However, the most PLAUSIBLE reading is the "distributive-distributive reading", where each of the five students ate four slices one at a time, as represented in a discourse representation structure (DRS) in Figure 1[2]. Such plausibility comes partly from the lexical semantics of *eat*. From our "common sense", we know that "eating" is an individual activity unlike "moving a desk", which can be done either individually or in a group. However, such plausibility should not be a part of the semantic theory, but should be dealt with in pragmatics where world knowledge is available. In section 2, I'll identify the

*The work described in this paper was done as a part of the author's doctoral dissertation at The University of Texas at Austin.

[1] Actually the two collective-collective readings are equivalent, so there are 7 distinct readings.

[2] (i-part x′ x) says "x′ is an atomic individual-part of x" (cf. Link [12]), and CU, i.e. "Count-Unit", stands for a natural measure unit for students (cf. Krifka [11]).

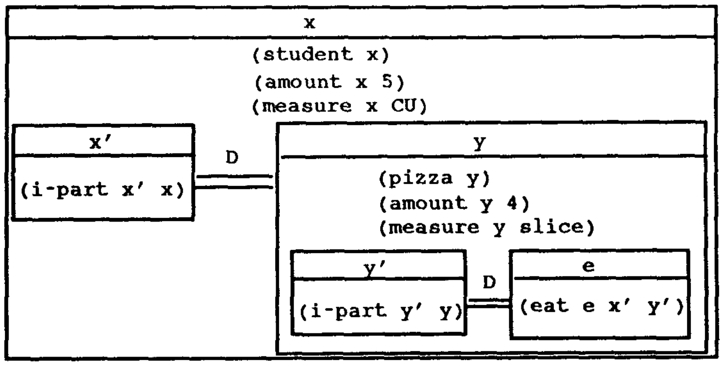

Figure 1: DRS for "Five students ate four slices of pizza"

necessary knowledge and develop a reasoner, which goes beyond Scha and Stallard [17].

There is a special reading called a **cumulative reading** (cf. Scha [16]).

(3) 500 students ate 1200 slices of pizza.

The cumulative reading of (3) says "there were 500 students and each student ate some slices of pizza, totaling 1200 slices." The semantics of a cumulative reading is UNDERSPECIFIED and is represented as a collective-collective reading at the semantic level (cf. Link [13], Roberts [15], Aone [2]). This means that a cumulative reading should have a more specific representation at the pragmatics level for inferencing. Reasoning about cumulative readings is particularly interesting, and I will discuss it in detail.

2 Model-Based Reasoning for Disambiguation

Although scope ambiguity has been worked on by many researchers (e.g. Grosz *et al.* [8]), the main problem addressed has been how to generate all the scope choices and order them according to some heuristics. This approach might be sufficient as far as scope ambiguity goes. However, collective-distributive ambiguity subsumes scope ambiguity and a heuristics strategy would not be a strong method. I argue that the reason why some of the readings are implausible (and even do not occur to some people) is because we have access to **domain-dependent** knowledge (e.g. constraints on predicates) along with **domain-independent** knowledge (e.g. mathematical knowledge). I have developed a reasoner based on the theory of model-based reasoning (cf. Simmons and Davis [18], Fink and Lusth [6], Davis and Hamscher [5]) for collective-distributive ambiguity resolution. The model that the reasoner uses consists of four kinds of knowledge, namely predicate constraints, two types of axioms, and simple mathematical knowledge. First, I will discuss the representation language CDCL[3]. Then, I will discuss how these four kinds of knowledge are utilized during reasoning.

2.1 CDCL

CDCL is used to represent collective-distributive readings, constraints and axioms for reasoning. There are three types of CDCL clauses as in (4), and I will explain them as I proceed[4].

(4) Core clause: (1 ((5) a0 → 4 a1))
Number-of clause: (number-of a1 ?q:num)
Number comparison clause: (<= ?q:num 1)

2.1.1 Expressing Collective and Distributive Readings in CDCL

CDCL is used to express collective and distributive readings. Below, a's are example sentences, b's are the most plausible readings of the sentences, and c's are representations of b's in CDCL.

(5) a. "5 students ate 4 slices of pizza."
b. Each of the 5 students ate 4 slices of pizza one at a time.
c. (eat a0 a1): (5 (1 a0 → 4 a1))

[3]CDCL stands for "Collective-Distributive Constraint Language".

[4]Though not described in this paper, CDCL has been extended to deal with sentences with explicit quantifiers as in "Every student ate 4 slices of pizza" and sentences with *n*-ary predicates as in "2 companies donated 3 PC's to 5 schools". For example:
(i) (eat a0 a1): (every (1 a0 → 4 a1))
(ii) (donate a0 a1 a2): (2 (1 a0 → (5 (1 a2 → (3) a1))))
See Aone [2] for details of CDCL expressed in a context-free grammar.

(6) a. "5 dogs had (a litter of) 4 puppies."
b. Each of the 5 mother dogs delivered a litter of 4 puppies.
c. (deliver-offspring a0 a1): (5 (1 a0 → (4) a1))

(7) a. "5 alarms were installed in 6 buildings."
b. Each of the 6 buildings was installed with 5 alarms one at a time.
c. (installed-in a0 a1): (6 (1 a1 → 5 a0))

First, consider (5c). The representation should capture three pieces of information: scope relations, distributive-collective distinctions, and numerical relations between objects denoted by NP arguments. In CDCL, a0 and a1 signify the arguments of a predicate, e.g. (eat a0 a1). The scope relation is represented by the relative position of those arguments. That is, the argument on the left hand side of an arrow takes wide scope over the one on the right hand side (cf. (5) vs. (7)). The numerical relation such as "there is an eating relation from EACH student to 4 slices of pizza" is represented by the numbers before each argument. The number outside the parentheses indicates how many instances of such a numerical relation there are. Thus, (5c) says there are five instances of one-to-four relation from students to slices of pizza. CDCL is designed to be isomorphic to a diagram representation as in Figure 2.

```
s  — p   s  — p   s  — p   s  — p   s  — p
   \- p     \- p     \- p     \- p     \- p
   \- p     \- p     \- p     \- p     \- p
   \- p     \- p     \- p     \- p     \- p
   s = a student
   p = a slice of pizza
```

Figure 2: "5 students ate 4 slices of pizza."

As for the collective-distributive information in CDCL, it was implicitly assumed in (5c) that both arguments were read DISTRIBUTIVELY. To mark that an argument is read COLLECTIVELY, a number before an argument is written in parentheses where the number indicates cardinality, as in (6c).

There are two additional symbols, *anynum* and *anyset* for representing cumulative readings. The cumulative reading of (3) is represented in CDCL as follows.

(8) (500 (1 a0 → anynum0 a1)) &
(1200 (1 a1 → anynum1 a0))

In (8), the situation is one in which each student (a0) ate a certain number of pizza slices, and the number may differ from student to student. Thus, ***anynum0*** represents any positive integer which can vary with the value of a0.

2.1.2 Constraints in CDCL

CDCL is also used to express constraints. Each predicate, defined in the knowledge base, has its associated constraints that reflect our "common sense". Thus, constraints are **domain-dependent**. There are two kinds of constraints: **type constraints** (i.e. constraints on whether the arguments should be read collectively or distributively) and **numerical constraints** (i.e. constraints on numerical relations between arguments of predicates.) There are 6 type constraints (C1 - C6) and 6 numerical constraints (C7 - C12) as in Figure 3.

C1. (?p:num (1 ?a:arg → ?q:num ?b:arg))
:⇒ inconsistent
"Both arguments are distributive."

C2. (1 (?p:set ?a:arg → ?q:set ?b:arg))
:⇒ inconsistent
"Both arguments are collective."

C3. (?p:num (1 a0 → ?r:set a1)) :⇒ inconsistent
C4. (1 (?q:set a1 → ?r:num a0)) :⇒ inconsistent
"1st argument distributive and 2nd collective."

C5. (1 (?p:set a0 → ?q:num a1)) :⇒ inconsistent
C6. (?p:num (1 a1 → ?q:set a0)) :⇒ inconsistent
"1st argument collective and 2nd distributive."

C7. (?p:num (1 ?a:arg → ?q:num ?b:arg))
⇒ (<= ?q:num ?r:num)

C8. (?p:num (1 ?a:arg → ?q:num ?b:arg))
⇒ (<= ?r:num ?q:num)

C9. (?p:num (1 a0 → 1 a1)) :⇒ inconsistent
"A relation from a0 to a1 is a function."

C10. (?p:num (1 a1 → 1 a0)) :⇒ inconsistent
"A relation from a1 to a0 is a function."

C11. (1 (?p:set a0 → 1 a1)) :⇒ inconsistent
"Like C9, the domain is a set of sets."

C12. (1 (?p:set a1 → 1 a0)) :⇒ inconsistent
"Like C10, the domain is a set of sets."

Figure 3: Constraints

Predicate constraints are represented as rules. Those except C7 and C8 are represented as "anti-rules". That is, if a reading does not meet a constraint in the antecedent, the reading is considered inconsistent. C7 and C8 are ordinary rules in that if they succeed, the consequents are asserted and if they fail, nothing happens.

The notation needs some explanation. Any symbol with a ?-prefix is a variable. There are 4 variable types, which can be specified after the colon of each variable:

(9)

?a:arg	argument type (e.g. a0, a1, etc.)
?b:num	positive integer type
?c:set	non-empty set type
?d:n-s	either num type or set type

If an argument type variable is preceded by a set type variable, the argument should be read collectively while if an argument type variable is preceded by a number type variable, it should be read distributively.

To explain type constraints, look at sentence (6). The predicate (deliver-offspring a0 a1) requires its first argument to be distributive and its second to be collective, since delivering offspring is an individual activity but offspring come in a group. So, the predicate is associated with constraints C3 and C4.

As for constraints on numerical relations between arguments of a predicate, there are four useful constraints (C9 - C12), i.e. constraints that a given relation must be a FUNCTION. For example, the predicate ***deliver-offspring*** in (6) has a constraint of a biological nature: offspring have one and only one mother. Therefore, the relation from a1 (i.e. offspring) to a0 (i.e. mothers) is a function whose domain is a set of sets. Thus, the predicate is associated with C12. Another example is (7). This time, the predicate (installed-in a0 a1) has a constraint of a physical nature: one and the same object cannot be installed in greater than one place at the same time. Thus, the relation from a0 (i.e. alarms) to a1 (i.e. buildings) is a many-to-one function. The predicate is therefore associated with C9.

In addition, more specific numerical constraints are defined for specific domains. For example, the constraint "each client machine (a1) has at most one diskserver (a0)" is expressed as in (10), given (disk-used-by a0 a1). It is an instance of a general constraint C7.

```
(10) (?p:num (1 a1 → ?q:num a0))
     ⇒
     (<= ?q:num 1)
```

2.1.3 Axioms in CDCL

While constraints are associated only with particular predicates, axioms hold regardless of predicates (i.e. are **domain-independent**). There are two kinds of axioms as in Figure 4. The first two are **constraint axioms**, i.e. axioms about predicate constraints. Constraint axioms derive more constraints if a predicate is associated with certain constraints.

```
CA1.  (?m:num (1 ?a:arg → 1 ?b:arg))
      ⇒
      (number-of ?a:arg ?m:num) &
      (number-of ?b:arg ?n:num) &
      (<= ?n:num ?m:num)
CA2.  (?l:num (?s:set ?a:arg → 1 ?b:arg))
      ⇒
      (number-of ?a:arg ?m:num) &
      (number-of ?b:arg ?n:num) &
      (<= ?n:num ?m:num)
RA1.  (?m:num (1 ?a:arg → ?y:n-s ?b:arg))
      ⇒
      (number-of ?a:arg ?m:num)
RA2.  (?m:num (1 ?a:arg → ?y:num ?b:arg)) &
      (<= ?y:num ?z:num)
      ⇒
      (number-of ?b:arg ?n:num) &
      (<= ?n:num (* ?m:num ?z:num))
RA3.  (?m:num (1 ?a:arg → ?y:num ?b:arg)) &
      (<= ?z:num ?y:num)
      ⇒
      (number-of ?b:arg ?n:num) &
      (<= ?z:num ?n:num)
```

Figure 4: Axioms

The others are **reading axioms**. They are axioms about certain assertions representing particular readings. Reading axioms derive more assertions from existing assertions.

The constraint axiom CA1 derives an additional numerical constraint. It says that if a relation is a function, the number of the objects in the range is less than or equal to the number of the objects in the domain. This axiom applies when constraints C9 or C10 is present. For example:

```
(11)  C9.   (?p:num (1 a0 → 1 a1))
      CA1.  (?m:num (1 ?a:arg → 1 ?b:arg))
            ⇒
            (number-of ?a:arg ?m:num) &
            (number-of ?b:arg ?n:num) &
            (<= ?n:num ?m:num)
      ------------------------------------
            (number-of a0 ?m:num) &
            (number-of a1 ?n:num) &
            (<= ?n:num ?m:num)
```

The constraint axiom CA2 is similar to CA1 except that the domain is a set of sets.

The reading axiom RA1 asserts the number of all objects in the domain of a relation. For example:

```
(12)  A1.   (5 (1 a0 → 6 a1))
      RA1.  (?m:num (1 ?a:arg → ?y:n-s ?b:arg))
            ⇒
            (number-of ?a:arg ?m:num)
      ------------------------------------
            (number-of a0 5)
```

Given an assertion A1, RA1 asserts that the number of objects in the domain is 5.

The reading axiom RA2 is for a relation where each object in the domain is related to less than or equal to *n* objects in the range. In such a case, the number of the objects in the range is less than or equal to the number of objects in the domain multiplied by *n*. For example:

```
(13)  A2.   (5 (1 a0 → ?x:num a1))
            & (<= ?x:num 2)
      RA2.  (?m:num (1 ?a:arg → ?y:num ?b:arg))
            & (<= ?y:num ?z:num)
            ⇒
            (number-of ?b:arg ?n:num) &
            (<= ?n:num (* ?m:num ?z:num))
      ------------------------------------------
            (number-of a1 ?n:num) &
            (<= ?n:num (* 5 2))
```

The last axiom RA3 is similar to RA2.

These axioms are necessary to reason about consistency of cumulative readings when numerical constraints are associated with the predicates. For example, given "5 alarms were installed in 6 buildings", intuitively we eliminate its cumulative reading because the number of buildings is more than the number of alarms. I claim that behind this intuition is a calculation and comparison of the number of buildings and the number of alarms given what we know about "being installed in". The constraint axioms above are intended to simulate how humans make such comparisons between two groups of objects related by a predicate that has a numerical constraint. The reading axioms, on the other hand, are intended to simulate how we do such calculations of the number of objects from what we know about the reading (cf. 2.2.2).

2.2 Model-Based Reasoner

In this section, I describe how the reasoner performs disambiguation. But first I will describe special "unification" which is the basic operation of the reasoner[5].

2.2.1 Unification

"Unification" is used to unify CDCL clauses during the reasoning process. However, it is not standard unification. It consists of three sequential matching operations: Syntax Match, ARG Match, and Value Match. First, Syntax Match tests if the syntax of two expressions matches. The syntax of two expressions matches when they belong to the same type of CDCL clauses (cf. (4)). If Syntax Match succeeds, ARG Match tests if the argument constants (i.e. a0, a1) in the two expressions match. If this operation is successful, Value Match is performed. There are two ways Value Match fails. First, it fails when types do not match. For example, (14a) fails to unify with (14b) because ?r:set does not match the integer 4.

```
(14) a. (?p:num (?q:num a0 → ?r:set a1))
     b. (5 (1 a0 → 4 a1))
```

The second way Value Match fails is two values of the same type are simply not the same.

```
(15) a. (1 (?p:set a1 → 1 a0))
     b. (1 ((4) a1 → 5 a0))
```

Unification fails only when the first and second operations succeed and the third one fails, and unification succeeds only when all the three operations succeed. Otherwise, unification neither succeeds nor fails.

2.2.2 Inferences Using A Model

Each reading (i.e. a hypothesis) generated by the semantics module is stored in what I call a **reading record** (RR). Initially, it just stores assertions that represent the reading. As reasoning proceeds, more information is added to it. When the RR is updated and inconsistency arises, the RR is marked as inconsistent and the hypothesis is filtered out.

The reasoner uses a model consisting of four kinds of knowledge. Inferences that use these four (namely Predicate-Constraint inference, Constraint-Axiom inference, Reading-Axiom inference, and the Contradiction Checker) are controlled as in Figure 5.

First, Predicate-Constraint inference tests if each hypothesis satisfies predicate constraints. This is done by unifying each CDCL clause in the hypothesis with predicate constraints. For example, take a type constraint C1 and a hypothesis H1.

```
(16)  H1.  (eat a0 a1): (5 (1 a0 → (4) a1))
      C1.  (?p:num (1 ?a:arg → ?q:num ?b:arg))
           :⇒ inconsistent
      ---------------------------------------
           inconsistent
```

When a predicate constraint is an anti-rule like C1, a hypothesis is filtered out if it fails to unify with the constraint. When a predicate constraint is a rule like C7, the consequent is asserted into the RR if the hypothesis successfully unifies with the antecedent.

[5]The reasoner has been implemented in Common Lisp. Unification and forward chaining rule codes are based on Ableson and Sussman [1] and Winston and Horn [19].

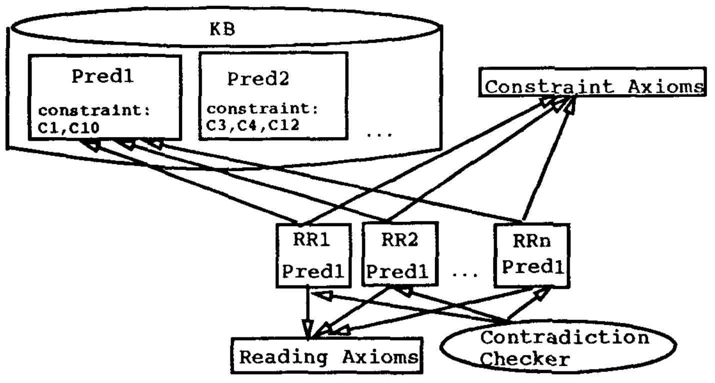

Figure 5: Control Structure

Second, Constraint-Axiom inference derives additional CONSTRAINTS by unifying antecedents of constraint axioms with predicate constraints. If the unification is successful, the consequent is stored in each RR (cf. (11)).

Third, Reading-Axiom inference derives more ASSERTIONS by unifying reading axioms with assertions in each RR (cf. (12) and (13)).

While these three inferences are performed, the fourth kind, the Contradiction Checker, constantly monitors consistency of each RR. Each RR contains a **consistency database**. Every time new information is derived through any other inference, the Contradiction Checker updates this database. If, at any point, the Contradiction Checker finds the new information inconsistent by itself or with other information in the database, the RR that contains this database is filtered out.

For example, take the cumulative reading of (7a), which is implausible because there should be at least 6 alarms even when each building has only one alarm. The reading is represented in CDCL as follows.

(17) (5 (1 a0 → anynum0 a1)) &
(6 (1 a1 → anynum1 a0))

The Contradiction Checker has simple mathematical knowledge and works as follows. Initially, the consistency database records that the upper and lower bounds on the number of objects denoted by each argument are plus infinity and zero respectively.

(18)

(18)	Number-of-a0	[0 +inf]
	Number-of-a1	[0 +inf]
	Constraint	NIL
	Consistent?	T

Then, when the constraint axiom CA1 applies to the predicate constraint C9 associated with *installed-in* (cf. (11)), a new numerical constraint "the number of buildings (a1) should be less than or equal to the number of alarms (a0)" is added to the database.

(19)	Number-of-a0	[0 +inf]
	Number-of-a1	[0 +inf]
	Constraint	(<= a1 a0)
	Consistent?	T

Now, the reading axiom RA1 applies to the first clause of (17) and adds an assertion (number-of a0 5) to the database (cf. (12)). The database is updated so that both upper and lower bounds on a0 are 5. Also, because of the constraint (<= a1 a0), the upper bound on a1 is updated to 5.

(20)	Number-of-a0	[5 5]
	Number-of-a1	[0 5]
	Constraint	(<= a1 a0)
	Consistent?	T

Finally, RA1 applies to the second clause of (17) and derives (number-of a1 6). However, the Contradiction Checker detects that this assertion is inconsistent with the information in the database, i.e. the number of a1 must be at most 5. Thus, the cumulative reading is filtered out.

(21)	Number-of-a0	[5 5]	
	Number-of-a1	[0 5]	⇐ [6 6]
	Constraint	(<= a1 a0)	
	Consistent?	NIL	

2.2.3 Example

I illustrate how the reasoner disambiguates among possible collective and distributive readings of a sentence. The sentence (7a) "5 alarms were installed in 6 buildings" generates 7 hypotheses as in (22).

(22)	R1	(5 (1 a0 → 6 a1))
	R2	(1 ((5) a0 → 6 a1))
	R3	(5 (1 a0 → (6) a1))
	R4	(6 (1 a1 → 5 a0))
	R5	(1 ((6) a1 → 5 a0))
	R6	(6 (1 a1 → (5) a0))
	R7	(5 (1 a0 → anynum0 a1)) & (6 (1 a1 → anynum1 a0))

The predicate (be-installed a0 a1) is associated with two constraints C1 and C9. Predicate-Constraint inference, using the type constraint C1 (i.e. both arguments should be read distributively), filters out R2, R3, R5, and R6. The numerical constraint, C9, requires that the relation from alarms to buildings be a function. This eliminates R1, which says that each alarm was installed in 6 buildings. The cumulative reading R7 is filtered out by the other three inferences, as described in section 2.2.2. Thus, only R4 is consistent, which is what we want.

3 Conclusion

The work described in this paper improves upon previous works on collective-distributive ambiguity (cf. Scha and Stallard [17], Gardiner *et al.* [7]), since they do not fully explore the necessary reasoning. I believe that the reasoning method described in this paper is general enough to solve collective-distributive problems because 1) any special constraints can be added as new predicates are added to the KB, and 2) intuitively simple reasoning to solve numerical problems is done by using domain-independent axioms.

However, the current reasoning capability should be extended further to include different kinds of knowledge. For example, while the cumulative readings of "5 alarms were installed in 6 building" is implausible and is successfully filtered out by the reasoner, that of "5 students ate 4 slices of pizza" is less implausible because a slice of pizza can be shared by 2 students. The difference between the two cases is that an alarm is not divisible but a slice of pizza is. Thus knowledge about divisibility of objects must be exploited. Further, if an object is divisible, knowledge about its "normal size" with respect to the predicate must be available with some probability. For example, the cumulative reading of "5 students ate 4 large pizzas" is very plausible because a large pizza is UNLIKELY to be a normal size for an individual to eat. On the other hand, the cumulative reading of "5 students ate 4 slices of pizza" is less plausible because a slice of pizza is more LIKELY to be a normal size for an individual consumption.

Acknowledgments

I would like to thank Prof. Manfred Krifka and Prof. Benjamin Kuipers for their useful comments. The prototype of the reasoner was originally built using Algernon (cf. Crawford [3], Crawford and Kuipers [4]). Many thanks go to Dr. James Crawford, who gave me much useful help and advice.

References

[1] Harold Abelson and Gerald Sussman. ***Structure and Interpretation of Computer Programs.*** The MIT Press, Cambridge, Massachusetts, 1985.

[2] Chinatsu Aone. ***Treatment of Plurals and Collective-Distributive Ambiguity in Natural Language Understanding.*** PhD thesis, The University of Texas at Austin, 1991.

[3] James Crawford. *Access-Limited Logic – A Language for Knowledge Representation.* PhD thesis, The University of Texas at Austin, 1990.

[4] James Crawford and Benjamin Kuipers. Towards a theory of access-limited logic for knowledge representation. In ***Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,*** Los Altos, California, 1989. Morgan Kaufmann.

[5] Randall Davis and Walter Hamscher. Model-based reasoning: troubleshooting. In H. E. Shrobe, editor, *Exploring Artificial Intelligence.* Morgan Kaufmann, Los Altos, California, 1988.

[6] Pamela Fink and John Lusth. A general expert system design for diagnostic problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3), 1987.

[7] David Gardiner, Bosco Tjan, and James Slagle. Extended conceptual structures notation. Technical Report TR 89-88, Department of Computer Science, University of Minnesota, Minneapolis, Minnesota, 1989.

[8] Barbara Grosz, Douglas Appelt, Paul Martin, and Fernando Pereira. Team: An experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32, 1987.

[9] Irene Heim. *The Semantics of Definite and Indefinite Noun Phrases.* PhD thesis, University of Massachusetts at Amherst, 1982.

[10] Hans Kamp. A theory of truth and semantic representation. In Groenendijk et al., editor, *Truth, Interpretation, and Information.* Foris, 1981.

[11] Manfred Krifka. Nominal reference and temporal constitution: Towards a semantics of quantity. In *Proceedings of the Sixth Amsterdam Colloquium*, pages 153–173, University of Amsterdam, Institute for Language, Logic and Information, 1987.

[12] Godehard Link. The logical analysis of plurals and mass terms: Lattice-theoretical approach. In Rainer Bauerle, Christoph Schwarze, and Arnim von Stechow, editors, *Meaning, Use, and Interpretations of Language.* de Gruyter, 1983.

[13] Godehard Link. Plural. In Dieter Wunderlich and Arnim von Stechow, editors, To appear in: *Handbook of Semantics.* 1984.

[14] Godehard Link. Generalized quantifiers and plurals. In P. Gaerdenfors, editor, *Generalized Quantifiers: Linguistics and Logical Approaches.* Reidel, 1987.

[15] Craige Roberts. *Modal Subordination, Anaphora, and Distributivity.* PhD thesis, University of Massachusetts at Amherst, 1987.

[16] Remko Scha. Distributive, collective, and cumulative quantification. In Janssen and Stokhof, editors, *Truth, Interpretation and Information.* Foris, 1984.

[17] Remko Scha and David Stallard. Multi-level plural and distributivity. In *Proceedings of 26th Annual Meeting of the ACL*, 1988.

[18] Reid Simmons and Randall Davis. Generate, test and debug: Combining associational rules and causal models. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Los Altos, California, 1987.

[19] Patrick Winston and Berthold Horn. *LISP 3rd Edition.* Addison-Wesley, Reading, Massachusetts, 1989.