Concept Formation Rules: An Executable Cognitive Model of Knowledge Construction

Veronica Dahl[1] and Kimberly Voll[1]

[1] Simon Fraser University
8888 University Dr. Burnaby, BC,
Canada, V5A 1S6

Abstract. We develop a cognitive model of knowledge construction inspired by constructivist theory as well as by recent natural language processing methodologies. Our generalization of these methodologies into a new and directly executable cognitive paradigm, concept formation, has been tested in non-trivial applications with very promising results. Our system accommodates user definition of properties between concepts as well as user commands to relax their enforcement. It also accepts concept formation from concepts that violate principles that have been declared as relaxable, and produces a list of satisfied properties and a list of violated properties as a side effect of the normal application of the rule.

1 Introduction

Constructivism is the idea that we construct our own world rather than it being determined by an outside reality (see [1] for an overview). Since the idea of constructing solutions or proofs is at the heart of both traditional computing science and logic programming, constructivism can constitute a natural bridge between the cognitive and computing sciences.

In the field of learning, constructivist theories such as those introduced by [2] view learning as the construction of new concepts from previously known concepts. During this process, "the learner selects and transforms information, constructs hypotheses, and makes decisions, relying on a cognitive structure to do so". The formation of new concepts from known ones is also central to cognitive logic, and also to logic programming and its recent sub-paradigm, Constraint Handling Rules (CHR) [3].

We exploit these natural connections to develop a cognitive model of knowledge construction that can be directly executed through a specialized system implemented using CHRs. In our model, information is selected automatically by the system as a side effect of searching through applicable CHR rules. It is automatically transformed, or simply augmented, when a rule triggers. Hypotheses are then made in the form of assumptions and decisions follow from the normal operation of the rules. The necessary cognitive structure is provided by those properties that must be satisfied based on the concepts a given rule is trying to relate. Moreover, some latitude is provided so that the user can declare under what circumstances a given property or

Dahl V. and Voll K. (2004).
Concept Formation Rules: An Executable Cognitive Model of Knowledge Construction.
In *Proceedings of the 1st International Workshop on Natural Language Understanding and Cognitive Science*, pages 29-36
DOI: 10.5220/0002685700290036

properties can be relaxed rather than having to satisfy all properties that have been defined as necessary. Concepts formed under relaxed properties result in output which signals not only what concepts were formed but also which of the properties associated with that concept's construction were satisfied and which were not. This allows us human-like flexibility while still maintaining direct executability.

In this paper, Section 2 presents our cognitive model while Section 3 provides the preliminary background. In Section 4 we discuss our methodological approach to implementing our model, and present two applications in Section 5. Finally, Section 6 presents our concluding remarks.

2 A Computational Metaphor for the Mind

We can imagine the human mind as an evolving store of knowledge, which constantly updates itself from new information built from previous information through some kind of reasoning.

In this model, the problems, events, or feelings, et cetera, of which we are consciously aware at any given time trigger a partly, or wholly, unconscious search through the knowledge store for those pieces of information that relate to the problem. Once found, these pieces of information permit the inference of new knowledge. For instance, a rule that is appropriate for modeling the formation of the new knowledge might look roughly as follows:

$$c1, c2, \ldots cn \rightarrow newc\ . \qquad (1)$$

Here *ci* and *newc* are concepts expressed as logic atoms. We call these rules *concept formation rules*.

Occasionally, we may need to test some condition before being able to form a new concept. For example, consider the knowledge that if the speed limit is *X* and our car's velocity is *Y*, then we will likely receive a speeding ticket if *Y* is greater than *X*. If we also know that we are traveling at 100km/hour and the speed limit is 50km/hour, then we might form the new concept that it is likely we will receive a speeding ticket. In other words, using a primitive (i.e., system-defined) unary atom *test*, whose argument contains the test to be performed, we can model the above concept-formation rule as the following:

$$speed\_limit(X), current\_speed(Y) \Rightarrow test(Y>X),\ speeding\_ticket(likely)\ . \qquad (2)$$

We next examine each feature that our concept-formation paradigm must have in order to a) constitute an adequate, constructivist metaphor for the workings of the human mind, b) provide some of the flexibility exhibited in human reasoning, and c) be directly executable.

2.1 Information Selection

Most of the rules in our knowledge store will be irrelevant to the situation at hand, so the first problem in our model, and in constructivism in general, is the problem of

selecting the appropriate information in order to form new concept(s) (which are actually relevant to the problem). We solve this by using a system that constantly searches for any information that may match the new information (i.e., the initial problem's concepts, or the concepts formed while attempting a solution) and consequently triggers those rules whose left-hand side at least partially matches some of the new information.

If we consider our previous example, then as soon as the new information that the speed limit is 50km/hr and the current speed is 100km/hr arrives in the knowledge store, a search through that store is initiated which ultimately yields rule (2), whose left hand side will be matched with unifier {X=50, Y=100}. The unifier will apply to the rest of the rule as well, and a test to see whether 100 > 50 will be performed. Since the test will succeed, the new information *speeding_ticket(likely)* will be added to the knowledge store.

2.2 Flexible Concept Formation

Binary logic is notoriously insufficient for modeling the human mind. The main alternatives are based either on probabilistic or fuzzy reasoning. As pointed out in [4], although probability theory is appropriate for measuring randomness of information, it is inappropriate for measuring the *meaning* of information. For many applications, measuring vagueness (e.g. through a membership function that measures the "degree" to which an element belongs to a set) is more important than measuring randomness.

While the fuzzy set approach has successfully overcome some of the problems with probabilistic reasoning, it relies on fairly detailed comparative knowledge of the domain being described. Not only must one say that Tom is tall, for instance, but one must also state specifically to what degree.

Here we propose a simpler model based on properties between concepts. Vagueness can be expressed by relaxing the properties between concepts in accordance with a user's criteria. The criteria can be flexibly and modularly adjusted for experimental purposes while still maintaining direct executability.

For our model described so far, we need only introduce the following enhancement. That is, rather than inflexibly allowing a concept to be formed if a test succeeds while disallowing its formation if that test fails, we instead denote those tests which we wish to make flexible as *properties*. Properties are like any other test, except that their failure does not necessarily result in the failure of the rule itself. Rather, the concept will still be formed and two lists will be associated with it: a list of the properties that the concept satisfies, *S*, and a list of those that it violates, *V*. This allows us to construct possibly incorrect or incomplete concepts while collecting the information as to what way they are not totally justified. The user then has all the information pertaining to the construction of a particular concept and can interpret these results in a much more informed, holistic way. This is in contrast to the blind computation of the degree of randomness or vagueness from those assigned a priori to each individual piece of the reasoning puzzle.

For instance, if we want to accept incorrect sentences in a parsing system we might designate as properties all tests on rule applicability that would correspond to correct parses, and then choose some to relax (for example, the number agreement property).

This might prove useful in a second-language tutoring system which allows the user to make certain types of mistakes, while pointing out the reason why those are mistakes (i.e. which properties are not being satisfied).

2.3 Transforming the Knowledge Store

Just as rule selection follows automatically from the system's normal search for applicable rules, the transformation of the knowledge store is achieved as a side effect of the system's normal procedure of applying a concept formation rule.

In some cases the newly formed concepts will need to coexist with the concepts that participated in their formation, such as when forming a concept based on some transitive property. In other cases, however, it will be more appropriate for the new concepts to replace the concepts that led to it. For instance, imagine a rule which uses the concept that a teacher is free at time *T* and that a parent is free at the same time *T* in order to add the concept that a meeting between these two individuals should be scheduled for time *T*. Obviously, once a meeting at time *T* has been arranged between them, they can no longer be considered as free at that time.

Yet another type of rule is needed to remove redundant concepts such as "John must meet Alice at six" and "Alice must meet John at six".

In this paper we primarily use the first type of rule, but all three types maintain computational counterparts within our model.

2.4 Making Decisions

It is also interesting to note that decisions also follow from the normal operation of the rules. That is, depending on which of the currently considered concepts matches which rule, the decision on the appropriate rule to trigger is made, determining which new concepts will be formed. Newly added concepts then help further trigger rules, and so on until a solution is found or no new, useful, concepts can be constructed.

2.6 Cognitive Structure

Cognitive structure (such as schemas or mental models) provides meaning and organization to experiences and allows individuals to draw conclusions beyond the information actually given.

In our model, cognitive structure is provided by the rules themselves. The properties that are required between the concepts a given rule is trying to relate are particularly important, both for the formation of new concepts and achieving human-like flexibility

3 Methodological Background: CHR

Imagine an evolving store of knowledge, which represents concepts as logic atoms (called "constraints" in the CHR literature), and uses rules of the following form to construct new pieces of knowledge from previously known ones:

$$\text{Head} \Rightarrow \text{Guard} \mid \text{Body} . \quad (3)$$

Here head and Body are conjunctions of atoms and Guard is a test constructed from built-in or system-defined predicates; the variables in Guard and Body occur also in Head. If the Guard is the constant "true" then it is omitted together with the vertical bar. Its logical meaning is the formula ∀ (Guard → (Head → Body)) and the meaning of a program is given by conjunction.

CHR rules are then interpreted as rewrite rules over such stores of knowledge. We use the CHR version that is an extension of SICStus Prolog and standard Prolog notation (i.e. capital letters for variables, et cetera).

CHR have rules of three types: *propagation* rules, which *add* new information to the store, *simplification* rules, which *remove* old information in favour of new, and *simpagation* rules, which remove redundant information.

4 Concept Formation Programs/Grammars

It is clear that CHRs can provide most of the features needed for a computational incarnation of our concept formation rules model:

- Information selection is a side effect of the CHR engine's search over applicable rules.
- The transformation of information takes place automatically when a rule triggers, in the way sanctioned by the rule (namely, *propagation* rules update the knowledge store with the concept newly formed).
- Hypotheses can be made through assumptions.

In addition, we need to provide flexible cognitive structure through relaxable, directly executable properties between concepts. This is achieved through concept formation rules, which have the same general form as CHR rules except that the guard may include any number of *property calls* for properties that have been defined by the user. These calls are automatically handled by the system, provided that the user defines the properties as follows:

a) A property must be named and defined through the binary predicate *prop*/2, whose first argument is the property's name and whose second argument is the list of arguments that are involved in checking the property and signaling the results. For instance, in a grammar that needs to check for number agreement between the determiner, noun and verb, and produce either the number of all three, or an indication of mismatch, we can choose the name "agreement" for the property, and define it as follows:

```
prop(agreement,[Ndet,Nn,Nv,N]):-  Ndet=Nn,
                                  Nn=Nv, !,
```

```
                                        N=Nv.
prop(agreement,[Ndet,Nn,Nv],mismatch).
```

b) The acceptability of a thus-defined property must be checked within the rule concerned through the binary, system-predicate *acceptable*/2. The first argument of this predicate takes the *prop* atom just described, and the second argument evaluates to true, false, or a continuous value indicating the degree of acceptability. For our example, we write:

```
determiner(Ndet,P1,P2),noun(Nn,P2,P3),v(Nv,P3,P)=>
acceptable(prop(agreement,Ndet,Nn,Nv,N),B)
                                        | sentence(N,P1,P).
```

N.B. the last two arguments of each atom, *Pi*, are explicit word boundaries. If left implicit, we instead have *Concept Formation Grammars*, which must run under CHRG rather than CHR.

c) In order to relax a property named *N* (i.e. to allow the derivation of concepts that require it but for which it is not satisfied), we simply write the following:

```
relax(N).
```

Degrees of acceptability can be defined through a binary version of the relaxing primitive, where *L* is the *prop* atom with all its arguments and *D* is a measure of acceptability:

```
relax(L,D).
```

A list of satisfied and violated properties, together with the degree of violation when appropriate, will be output for each property defined in a given CF program.

5 Two Case Studies

5.1 Scheduled Meetings

Consider the problem of scheduling meeting events between two groups of people given the availability of each person and the specific individuals that must meet.

Regular scheduling software systems typically work in stages. The schedule is partially assigned until it reaches a meeting that it cannot schedule. By re-shuffling the other meetings that have already been scheduled, it is then often possible to open up matching timeslots for the individuals in question.

Our concept formation framework allows us to produce a higher-level solution in a concise and effective manner. We first initialize the knowledge store with the preliminary concepts, namely, the available intervals of each of the participants and the requirement of who must meet with whom. For example:

```
start:-available(snape,1,3), available(dumbledore,2,4),
       available(potter,1,3), available(grange,1,5),
       req(potter,dumbledore), req(potter,snape),
       req(snape,grange), req(grange,dumbledore).
```

We can form the concept of free timeslots from the interval definitions of availability:

```
available(Person,T1,T) <=> T1 = T | true.
available(Person,T1,T) <=> T2 is  T1+1, |
                           free_slot(Person,T1,T2),
                           available(Person,T2,T).
```

To schedule all required meetings we need just one rule:

```
free_slot(Prof,Start,End), free_slot(Par,Start,End),
req(Par,Prof) <=> meet(Prof,Par,Start,End).
```

To schedule all required meetings we then need only one rule:

```
free_slot(Prof,Start,End), free_slot(Par,Start,End),
req(Par,Prof) <=>  meet(Prof,Par,Start,End).
```

This simplification rule says that provided that *Prof* and *Par* have the same free timeslot, and that *Par* is required to meet *Prof*, then this information (i.e. the free timeslots of each and the requirement to meet) will be deleted from the knowledge store and replaced by the concept that *Prof* and *Par* must meet in that timeslot. (Notice that no parent will be assigned two intervals with the same professor or vice versa since the parent's requirement to meet that professor is removed by our simplification rule.)

Suppose, for example, the school director wishes to see parents at random, whether they need to meet him or not. We simply define *req*/2 as a property rather than as an intervening concept, and relax it for the individual in question:

```
req(potter,dumbledore).     req(potter,snape).
req(snape,grange).          req(grange,dumbledore).

free_slot(Prof,Start,End),free_slot(Par,Start,End) <=>
acceptable(req(Par,Prof),B) | meet(Prof,Par,Start,End).

relax(req(Par,dumbledore)).
```

The last clause relaxes the requirement property only for Professor Dumbledore.

Duplicate meetings can be avoided using the *simpagation rule*:

$$\text{req(X,Y)} \setminus \text{req(Y,X)} \Leftrightarrow \text{true} . \qquad (4)$$

This rule removes the second requirement, which is symmetric to the first if both appear in the knowledge store.

Notice that any unsatisfied requirements will print automatically, since their *req*/2 constraint remains in the knowledge store at the end of the computation.

Similarly, free timeslots that have not been used in the solution of the problem are reflected in leftover constraint list *free*/3, which also prints out automatically. Further utilities to organize all this information in various useful ways can be provided according to the specific application.

5.2 Error Detection/Correction

Notice that by simply treating as properties those conditions whose failure we wish identify but tolerate, detected errors will now appear in the list of violated properties that is automatically produced. Consider the parsing example again, if number agreement has been relaxed, then in a language tutoring system a sentence disagreeing in number will be accepted but the error will be indicated as a side effect of parsing.

Error correction is now not far away. All that remains is to decide on a heuristic. For example, in parsing number agreement consult the user as to which of the two mismatched elements should change in number, or alternatively, take the most common number if one exists (e.g. changing "boys" into "boy" if everything else is singular). Once the heuristic is chosen, it simply remains to effect the change, since our concept formation rules have already done the diagnosis part of the work.

In joint work with Kimberly Voll, we are presently investigating the use of CF rules to aid in the interpretation of free-text reports, particularly within the medical domain. We treat semantic correctness as a property, so that nonsensical but grammatical sentences resulting from mistakes in input, such as those occurring through the use of speech recognition systems, can be detected and corrected. This is an important aid because while such automatic transcription systems have accuracy rates as high as 98% (which is a quite encouraging number in most settings), such rates can have dangerous, even fatal consequences in the medical setting. Semantic compatibility between parts of a sentence is checked with the aid of semantic type hierarchies in the case of this particular application.

6 Conclusions

We have shown a powerful yet relatively simple cognitive model for concept formation inspired by constructivism. The methodology we present for implementing directly executable concept formation paradigms generalizes parsing methods we have developed specifically for natural language processing [6]. We have discussed our methodology for a directly executable, CHR-based rendition of this model, and exemplified its promise through two examples: (time) resource allocation, and error detection and correction. Other approaches need considerable more apparatus to solve the same problem (for example, [7] uses a fairly involved abductive model based on CHRGs for error detection and correction in string processing problem domains). We are also currently working on applying concept formation to the problem of identifying biomarkers in cancer diagnosis systems [8].

Concept formation rules, as we have seen, have a grammatical counterpart (CFG), and are applicable to many other AI and cognitive problems as well, most notably, those involving the need to reason with incomplete or incorrect concepts.

7 Acknowledgements

Thanks are due to Alma Barranco-Mendoza for her help with typesetting, and to her and the anonymous referees for their useful comments on this article's first draft. This work was supported in part by NSERC grant 31-611024 and the NSERC PGS program. The linguistic work that inspired our cognitive model was developed with Phillipe Blache during Veronica Dahl's visit to Universite de Provence, invited by CNRS as Chercheur Etranger.

References

1. Kearsley, G.: Explorations in Learning and Instruction: The Theory into Practice Database, http://tip.psychology.org (2003)
2. Bruner, J.: Going Beyond the Information Given. New York: Norton. (1973)
3. Fruhwirth, T.: Theory and Practice of Constraint Handling Rules, J. Logic Programming, Vol. 37 (1-3), 95—138 (1998)
4. Zadeh, L.,: Commonsense knowledge representation based on fuzzy logic, Computer, 16 (1983)
5. Christiansen, H.: CHR grammars. In Theory and Practice of Logic Programming special issue on Constraint Handling Rules (2004)
6. Dahl, V. and Blache, P.: Directly Executable Constraint Based Grammars. TR, Univ. de Provence (2004)
7. Christiansen, H. and Dahl, V.: Logic Grammar for Diagnosis and Repair. In'tl J. of AI Tools, Vol. 12, No. 3, 227—248 (2003)
8. Barranco-Mendoza, A., Persaud, D. R., and Dahl, V.: A property-based model for lung cancer diagnosis, Poster Proceedings of RECOMB 2004 (2004)