CONSTRAINT BASED COMPOSITIONAL SEMANTICS

VAN DEN BROECK, WOUTER

Sony Computer Science Laboratory, 6, rue Amyot,
Paris, 75005, France
wouter@csl.sony.fr

Abstract

This paper presents a computational system that handles the grounding, the formation, the interpretation and the conceptualisation of rich, compositional meaning for use in grounded, multi-agent simulations of the emergence and evolution of artificial languages. Compositional meaning is deconstructed in terms of semantic building blocks which bundle a semantic function together with the relevant grounding and learning methods. These blocks are computationally modelled as procedural constraints, while the compositional meaning is declaratively represented as constraint programs. The flexibility of the data flow in such programs is utilized to adaptively deal with interpretation and learning. The conceptualisation is performed by a sub-system that composes suitable constraint programs. The various methods used for managing the combinatorial explosion are discussed.

1. Introduction

One way to study the evolution of language is to simulate the emergence and evolution of artificial languages in multi-agent experiments. An important issue in such experiments concerns the involved meaning. This meaning has to be "rich" for experiments that focus on the emergence of grammar. This paper presents an integrated system that handles the grounding, the formation, the interpretation and the conceptualisation of such meaning.

The kind of meaning covered by the system consists of *concepts* and *semantic functions*. Concepts are here considered to be category-like entities such as colors, shapes, events, relations, roles, etc. The grounding of these concepts requires some *grounding method*. Examples of such methods are neural networks as used in (Plunkett, Sinha, Moller, & Strandsby, 1992), probability density estimation as used in (Roy & Pentland, 2002), or discrimination trees as used in (Steels, 1996). Each concept that is grounded with a particular grounding method, corresponds to a particular set of *concept parameters*, which are used by that method.

Each agent constructs and maintains its own repertoire of concepts. The acquisition of a grounded concept requires a *learning method* for use with

the concerned grounding method. Back-propagation can for instance be used as the learning method for concepts grounded in terms of multi-layered perceptrons.

The role of a concept in the meaning of some utterance depends on the *semantic function* that uses this concept. Concepts are for instance used to categorise perceived entities in order to filter out those that do not fit the category. Interpreting "the red ball", for example, involves a filtering of the context such that what is retained is the object that best corresponds with both the color category RED and the shape prototype BALL. Other semantic functions are quantification, predication, negation, deictic reference, etc. Semantic functions are considered to be recruited from the general cognitive capabilities. Their evolutionary origin is thus not considered here.

2. Semantic building blocks

Rich, compositional meaning often involves different types of concepts. There is, however, no grounding method that is equally well suited for all types of concepts. The proposed system therefore accommodates different grounding methods. The structurally coupled evolution of language and concept repertoires furthermore requires a close interaction between the grounding and learning methods on one hand and the semantic functions on the other. Each semantic function is therefore bundled together with the relevant grounding and learning method, and encapsulated in a *semantic building block*. Each such block is equipped with a number of *slots*. These slots are used to get or set the arguments, such as the concepts and contexts, over which the semantic function operates.

An example of a semantic block is called *filter-set-prototype*. This block has three slots for the arguments it takes, i.e. a *source-set*, a *target-set* and a *prototype* concept. The behaviour of this semantic block depends on the availability of the arguments. If the source-set and the prototype are given, which is the case in a regular interpretation process, then the block can derive the target-set. This set contains all entities in the source-set that match the given prototype. If the source-set for instance contains all the objects in the observed scene shown in figure 1, and the prototype concept is for example the shape BALL, then the target-set will contain all ball-like objects in the source-set, i.e. o3, o4 and o5. The meaning of the utterance "the balls" could thus be represented by a structure that includes this *filter-set-prototype* block.

Different arguments are available in a learning situation. Consider for instance a situation in which the speaker used the utterance "the frouple" to discriminate object $o1$ in figure 1. The hearer indicated that he/she could not understand this utterance. The speaker then drew the attention to the topic by pointing to it. This presents a learning opportunity for the hearer. The filter-set-prototype block now has the source-set, which includes all objects in the scene, and the target-set, which contains the topic. It can try to infer the concept that could account for the filtering from the source-set to the target-set. The hearer could assume that this concept is the one meant by the word "frouple" and add this mapping in his/her lexicon.

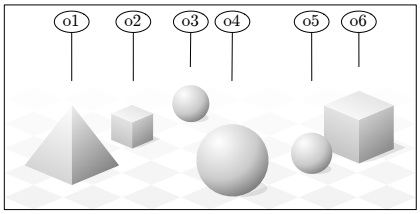

Figure 1. A scene with a number of labelled objects of varying size and shape.

3. Constraint programs

A semantic building blocks can have multiple operational modes depending on the availability of the arguments. Put differently, each block represent an omnidirectional relationship among a number of variables. Such relationships can be computationally modelled as *constraints*. The encapsulated functionality that implements the grounding and learning method and the semantic function *enforce the relationship*. The resulting procedural constraints can however be *declaratively* combined by linking[a] relevant slots. The result is a *constraint program* that represents compositional meaning.

> The constraint paradigm is a model of computation in which values are deduced whenever possible [...]. One may visualize a constraint 'program' as a network of devices connected by wires. Data values may flow along the wires, and computation is performed by the devices. A device computes using only locally available information (with a few exceptions), and places newly derived values on other, locally attached wires. (Steele, 1980)

The interpretation of a constraint program can be seen as a *constraint satisfaction problem*, for which efficient algorithms exist. Our implementation uses a extension of the AC-4 algorithm (Mohr & Henderson, 1986) which implements a strong form of generalized relational arc-consistency. It involves *constraint-ordering* heuristics, and uses a *look-ahead* search to find the actual solutions.

3.1. *Examples*

Figure 2 depicts the constraint program that represents the meaning for the utterance "the bigger ball". The particular values and data flow correspond with the interpretation of this program in the context of the scene shown in figure 1. The *filter-set-protype* constraint takes the context and the BALL prototype, and yields the set that contains all balls. The *filter-set-comparison* constraint takes this set and the comparator BIG and selects the bigger one, i.e. the topic $o4$.

[a] such links represent equality relationships

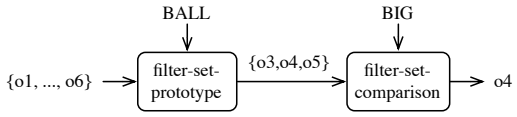

Figure 2. The constraint program and interpretation data flow for "the bigger ball".

Figure 3 shows the data flow involved in a learning situation. The hearer did not understand the modifier but was shown the topic $o4$. The hearer did properly understand "ball" and could thus produce the source-set taken by the filter-set-comparison constraint. This constraint can then, given the topic, infer the modifier BIG, and a new entry can be added to the lexicon.

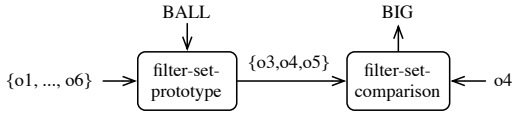

Figure 3. The data flow involved in the inference of the modifier concept.

Figure 4 depicts the program and interpretation data flow for "the box close to the pyramid". The *filter-set-relation* constraint takes the set of boxes as source-set, the pyramid as landmark, and CLOSE-TO as relation concept. Given these parameters, it can properly discriminate the topic $o2$.

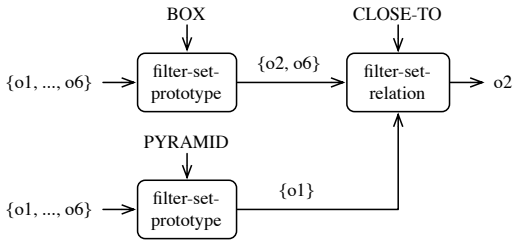

Figure 4. The program and interpretation data flow for "the box close to the pyramid".

4. Conceptualisation

We can now turn our attention to the conceptualisation of the compositional meaning. Since this meaning is represented as constraint programs, its conceptualisation must involve a process that constructs such programs. The input for this process is a communicative goal, such as "discriminate topic

x in the sensory context". It must construct a constraint program that, when interpreted by the hearer, is expected to satisfy that goal. There are typically many potential programs that could fulfil a given goal. Various criteria are defined for measuring their relative strengths, such as the level of ambiguity involved, the expressibility in an utterance, the complexity, etc.

Finding a suitable constraint program is a combinatorial problem. The constraint program composer algorithm used in our system involves a number of techniques and strategies for keeping the combinatorial explosion in check.

Eager, incremental search. The algorithm searches for suitable constraint programs by incrementally expanding incomplete programs, one constraint at a time. There can be many candidate constraints at each step. These candidates are handled in separate branches. The expanded programs are evaluated according to some heuristics to decide which branch to expand next. Solutions are found more efficiently with this strategy.

Goal-directed search. If the goal is to discriminate a topic in a context, then the target program must be such that the topic can be inferred from the given concepts and context. In other words, one of the potential data flows in that program must be a coherent, non-cyclic one from the context and concepts to the topic. The algorithm tries to satisfy this requirement by only adding constraints that incrementally extend the data flow backwards. Each constraint is added to support a goal. The initial goal is the topic. Each constraint supports a goal by adding a piece of data flow. The added data flow connects the goal with the new sub-goals introduced by the constraint. When a filter-set-prototype constraint is for example added and its target-set slots is linked with the goal, then the new sub-goals are the source-set, unless it is linked with the context, and the prototype, unless it is expressed in the utterance. A more detailed description of this search process can be found in (Van den Broeck, 2007).

All potential expansions that do not properly contribute to the data flow, are ignored. This significantly reduces the size of the search space. The number of potential combinations of r constraints from an inventory of n constraints is $\left\langle {n \atop k} \right\rangle$ (the multi-set coefficient). The average number of potential links between the slots of r constraints with an average arity of a is $s(k,a) = (k-1)\,a\,((k-1)\,a+1)/2$. The total number of potential constraint programs of size k is thus approximately $\left\langle {n \atop k} \right\rangle 2^{s(k,a)}$, while the size of the incrementally explored search space of constraint programs of maximum size k is approximately $\sum_{i=1}^{k} \left\langle {n \atop i} \right\rangle 2^{s(k,a)}$.

For a small test case with 5 kinds of constraints with an average arity of 2.6 and a maximum program size of 6, the total number of partial constraint programs is approximately 5.199348e29. The goal-directed search does however find a suitable program (if there is one) after on average 262 expansions when conceptualising a program for a randomly chosen topic in our benchmark scene collection.

Interleaved constraint satisfaction. Determining if a constraint program fulfils the goal is done by interpreting it using the aforementioned constraint satisfaction algorithm. This algorithm also identifies branches with inconsistent partial programs, which can be pruned. Interleaving the constraint satisfaction in the incremental search furthermore minimizes the amount of consistency enforcing (when using AC-4), because all enforcing applied on some partial program caries over to the expanded programs.

Chunking An additional technique we are currently exploring is *chunking*. This technique consists of taking a (part of a) successfully used semantic program and wrap it such that it can be re-used as a constraint in future programs. We call these *composite* constraints, since they are composed of a number of component constraints. The initially given constraints are in contrast called *primitive* constraints. Figure 5 depicts a constraint program that involves a composite constraint which wraps two primitive constraints[b]. This composite constraint has four slots, which are internally linked with the appropriate slots of the component constraints.

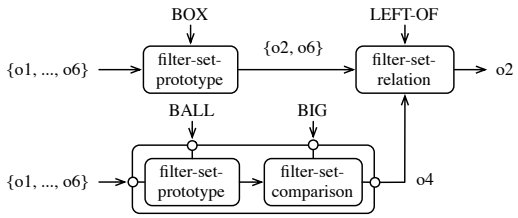

Figure 5. The constraint program and data flow for the interpretation of "the box left of the big ball". This program involves a composite constraint that wraps two primitive constraints.

The composite constraint inventory of an agent is initially empty. New composites are created according to some chunking strategy. We currently use a basic strategy that chunks complete constraint programs. The resulting composite constraints are candidates, just like primitives, with which to expand incomplete programs. Adding a composite corresponds to jumping to a point in the search space that previously proved to be useful. First experiments show that chunking and re-using the resulting composites, significantly improves the performance of the composer algorithm, as shown in figure 6. These telling results were obtained in spite of the basic chunking strategy we currently use.

The chunking strategy is also interesting because it can be relevant at the language level. In particular the potential relationship between composite constraints and grammatical constructions is intriguing, but unfortunately

[b] composite constraints can also be hierarchically composed

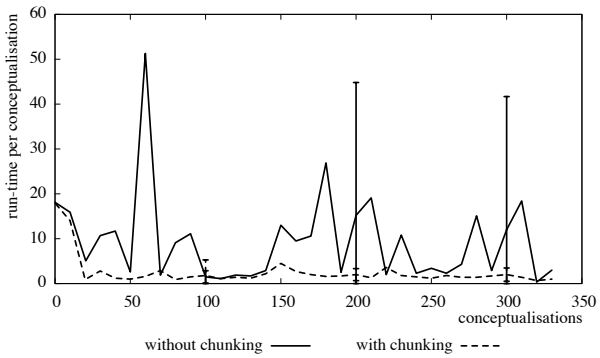

Figure 6. Comparison of run-time needed to conceptualise a series of topics, with and without chunking.

beyond the scope of this paper. Finally we would like to note that the composer is also useful when a hearer could not fully reconstruct the constraint program due to misunderstanding or under-specification. The composer can in these cases propose potential completions of the incomplete program.

5. Conclusions

In this paper we showed how representing rich, compositional meaning in terms of constraints and constraint programs offers a uniform framework for dealing with their interpretation and conceptualisation. We demonstrated how the flexible data flows handles interpretation and appropriately adapts to learning situations.

The bundling the semantic functions together with the grounding and learning methods affords a tight interaction between the interpretation and the concept acquisition. Encapsulating the procedural details of the bundled functionality allows experimenters to combine different techniques transparently.

The constraint based representation of meaning enabled us to draw upon the well-developed body of knowledge on constraint processing in the fields of artificial intelligence and operations research. The interpretation of the constraint based representation constitutes a constraint satisfaction problem, for which optimal algorithms exist. The conceptualisation on the other hand, is implemented as a incremental composer of constraint programs. A number of techniques and strategies were discussed that effectively keep the involved combinatorial explosion in check.

In traditional first-order logic representations of meaning, the concepts are typically represented as predicates. In a constraint based approach, the concepts are rather arguments for the semantic constraints, which can be thought of as relational predicates. A constraint based semantics can thus be regarded as a second-order semantics.

Finally, the proposed system does not favour any particular model or formalism concerning the emergence and evolution of language in general, or grammar in particular. It should thus be adoptable in a wide array of experimental and theoretical settings. One particular setting is presented elsewhere in this collection (Bleys, 2008).

Acknowledgements

This research is supported by Sony Computer Science Laboratory in Paris and the ECAGENTS project funded by the Future and Emerging Technologies programme (IST-FET) of the European Community under EU R&D contract IST-2003-1940. It builds on the work first introduced in Steels (2000) and elaborated on in Steels and Bleys (2005).

References

Blackburn, P., & Bos, J. (2005). *Representation and inference for natural language. a first course in computational semantics.* CSLI Publications.

Bleys, J. (2008). Expressing second order semantics and the emergence of recursion. In A. D. M. Smith, K. Smith, & R. F. i Chancho (Eds.), *The evolution of language: Evolang 7.* World Scientific.

Dechter, R. (2003). *Constraint processing.* Morgan Kaufmann.

Mohr, R., & Henderson, T. C. (1986). Arc and path consistency revisited. *Artificial Intelligence*, *28*(2), 225–233.

Plunkett, K., Sinha, C., Moller, M. F., & Strandsby, O. (1992). Symbol grounding or the emergence of symbols? vocabulary growth in children and a connectionist net. *Connection Science*, *4*, 293—312.

Roy, D. K., & Pentland, A. (2002). Learning words from sights and sounds: a computational model. *Cognitive Science*, *26*, 113—146.

Smith, A. D. M. (2005). The inferential transmission of language. *Adaptive Behavior*, *13*(4), 311–324.

Steele, G. L. (1980). *The definition and implementation of a computer programming language based on constraints.* Unpublished doctoral dissertation, MIT.

Steels, L. (1996). Perceptually grounded meaning creation. In M. Tokoro (Ed.), *Icmas96.* AAAI Press.

Steels, L. (2000). The emergence of grammar in communicating autonomous robotic agents. In W. Horn (Ed.), *Ecai2000* (pp. 764–769). Amsterdam: IOS Press.

Steels, L., & Bleys, J. (2005). Planning what to say: Second order semantics for fluid construction grammars. In A. Bugarin Diz & J. S. Reyes (Eds.), *Proceedings of caepia '05. lecture notes in ai.* Berlin: Springer Verlag.

Van den Broeck, W. (2007). A constraint-based model of grounded compositional semantics. In *Proceedings of langro'2007.*