

Formalizing Psychological Knowledge in Answer Set Programming

Marcello Balduccini

Intelligent Systems, OCTO

Eastman Kodak Company

marcello.balduccini@gmail.com

Sara Girotto

Department of Psychology

Texas Tech University

sara.girotto@ttu.edu

Abstract

In the field of psychology, a considerable amount of knowledge is expressed using only natural language, which complicates accurate studies and comparisons. We believe that Answer Set Programming (ASP) can be used successfully for the formalization of psychological knowledge. To demonstrate the viability of ASP for this task, in this paper we develop an ASP-based formalization of the mechanics of Short-Term Memory, and show how it correctly reproduces the observed behavior of human subjects.

Introduction

In this paper we explore the use of Answer Set Programming (ASP) (Gelfond and Lifschitz 1991) for the formalization of psychological knowledge. The importance of a precise formalization of scientific knowledge has been known for a long time (see e.g. Hilbert's philosophy of physics). Most notably, formalizing a body of knowledge in an area improves one's ability to accurately study the properties and consequences of sets of statements, compare competing set of statements, and design experiments aimed at confirming or refuting sets of statements. In the field of psychology, some of the theories about the mechanisms of the brain have been formalized using artificial neural networks and similar tools. That approach works well for theories that can be expressed in quantitative terms. However, theories of a more qualitative or logical nature are not easy to formalize in this way. We believe that ASP can be used successfully for the formalization of such bodies of knowledge. ASP is a knowledge representation formalism that allows for concise and simple representation of defaults, uncertainty, and dynamic domains. One further advantage of ASP is that it is directly executable, in the sense that the consequences of collections of ASP statements can be directly, and often efficiently, computed using computer programs. In this paper we develop an ASP-based formalization of the mechanics of Short-Term Memory (STM) and chunking. We selected this theory because it is rather mature. Moreover, like the general type of psychological knowledge that we aim to formalize, it is mostly of a qualitative nature, and is expressed,

in the literature, at a rather high level of abstraction. Finally, formalizing the mechanics of STM is challenging, because it involves modeling a sophisticated dynamic domain with fixed-capacity storage and time-based decay. As we hope will become apparent later, it is difficult to find other languages that allow writing a formalization at the level of abstraction of the one shown here, and that are at the same time directly executable. Direct executability of ASP provides a unique opportunity to bridge the gap between formulation of psychological theories and practical applications.

Representation of Dynamic Domains

To save space, we assume the reader's familiarity with the syntax and semantics of ASP. We refer the reader to (Gelfond and Lifschitz 1991) for more information.

Because of the dynamic nature of STM, for its formalization we use techniques for the representation of dynamic domains. The key elements are presented next; details can be found e.g. in (Gelfond and Lifschitz 1998). *Fluents* are first-order ground terms, and intuitively denote the properties of interest of the domain. A fluent literal is either a fluent f or its negation ($\neg f$). Actions are also first-order ground terms. A set of fluent literals is *consistent* if, for every fluent f , f and $\neg f$ do not both belong to the set. The set of all the possible evolutions of a dynamic domain is represented by a *transition diagram*, i.e. a directed graph whose nodes – each labeled by a consistent set of fluent literals – correspond to the states of the domain in which the properties specified are respectively true or false, and whose arcs – each labeled by a set of actions – correspond to the occurrence of state transitions due to the execution of the actions specified. Because of the size of transition diagrams, a direct representation is usually impractical. Hence, here we use an indirect representation based on the research on action languages (Gelfond and Lifschitz 1998), and we adopt the variant of writing such encoding in ASP. The encoding is based on the notion of a path in the transition diagram from a given initial state, corresponding to a particular possible evolution of the domain from that initial state. The steps in a path are identified by integers (with 0 denoting the initial state), and logical statements (*laws*) are used to encode the state transitions. An expression $h(f, i)$ states that fluent f holds at a step i . If $\neg f$ holds, we write $\neg h(f, i)$. An expression $o(a, i)$ states that action a occurs at step i . An *action description* is a collec-

tion of laws describing the evolution of the domain. Given an action description AD , an initial state σ_0 (e.g. $\sigma_0 = \{h(f_1, 0), \neg h(f_2, 0), \dots\}$), and a sequence of occurrences of actions α (e.g. $\alpha = \{o(a_1, 0), o(a_3, 0), o(a_4, 1), \dots\}$), the corresponding path(s) in the transition diagram can be computed by finding the answer set(s) of $AD \cup \sigma_0 \cup \alpha$.

Short-Term Memory and Its Formalization

Short-Term Memory is “the memory storage system that allows for short-term retention of information before it is either transferred to long-term memory or forgotten” (Nevid 2007). This view is based on the so called *three-stage model of memory*: sensory inputs are first stored in Sensory Memory, which is very volatile and has large capacity; then, a portion of the inputs is processed – and possibly transformed into more rich representations – and moved to Short-Term Memory, which is less volatile than Sensory Memory, but of limited capacity. Short-Term Memory is also often viewed as a working memory, i.e. as a location where information is processed. Finally, selected information is moved to Long-Term Memory, which has larger capacity and longer retention periods. According to recent studies, the capacity limit of STM is about 4 pieces of information (e.g. (Cowan 2000)). Notice that this limit does not affect the *amount* of information (in an information-theoretic sense) that STM can hold. In fact, STM appears to be capable to storing *references* to concepts that are stored in Long-Term Memory. One such reference counts as a single piece of information, but the amount of information it conveys can be large. For example, it has been observed that it is normally hard for people to remember the 12-letter sequence CN NIB MMT VU SA, while most people have no problems remembering the sequence CNN IBM MTV USA, because each triplet refers to a concept stored in Long-Term Memory, and can thus be represented in STM by just 4 symbols. The phenomenon of the detection and use of known patterns in STM is referred to as *chunking*. Another limit of STM is that the information it contains is retained only for a short period of time, often set to about 30 seconds by researchers ((Nevid 2007); notice however that the issue of a time limit on the information stored in STM is somewhat controversial – see e.g. (Cowan 2000)).

Using a common methodology in ASP-based knowledge representation, we begin our discussion on the formalization of STM by condensing the description of STM and chunking in a number of statements still written in natural language, but precisely formulated. The statements are: (1) STM is a collection of symbols; (2) The size of STM is limited to ω elements; (3) Each symbol has an expiration time associated with it (saying when the piece of information will be “forgotten”); (4) Symbols are divided in primitive symbols and chunk symbols; (5) Primitive symbols represent concepts that are innate in the brain (e.g. are direct encoding of sensory input); (6) Chunk symbols represent concepts that are stored in Long Term Memory; (7) New symbols can be added to STM. If a symbol is added to STM when ω elements are already in STM, the symbol that is closest to expiring is removed from STM (“forgotten”). In the case of multiple symbols equally close to expiring, one is selected

arbitrarily; (8) When a symbol is added to STM its expiration time is set to a constant value ε ; (9) *Simplifying assumption*: only a single operation (where by operation we mean either addition or use) can occur on STM at any given time. The next set of statements describes the mechanism of chunking: (10) A chunk is a set of (primitive or chunk) symbols; (11) A chunk symbol is a symbol that uniquely denotes a chunk; (12) A chunk is detected in STM if all the symbols that it consists of are in STM; (13) When a chunk is detected in STM, the symbols it consists of are removed from STM and the corresponding chunk symbol is added to STM; (14) A symbol can be inferred from STM if it belongs to STM, or if it is part of a chunk that can be inferred from STM; (15) *Simplifying assumption*: chunks can only be detected when STM is not in use (i.e. no addition or use operations are being performed); (16) *Simplifying assumption*: the available chunks and contents of STM are such that, at every step, at most one chunk can be detected.

Now we focus on the ASP encoding of the formalization. Fluent $in_stm(s)$ says that symbol s is in STM; $expiration(s, k)$ says that s will expire (i.e. will be “forgotten”) in k units of time, unless the expiration counter is otherwise altered. Action $store(s)$ says that s is stored in STM. Relation $primitive(s)$ says that s is a primitive symbol; chunks are described by $chunk(s)$, saying that s is a chunk symbol, and by $chunk_element(s, s')$, stating that s' is a component of the chunk identified by s ; $symbol(s)$ says that s is a symbol, either primitive or chunk. Relation $stm_max_size(\omega)$ says that the maximum size of STM is ω elements; $stm_expiration(\varepsilon)$ states that the symbols in STM expire after ε units of time. Relation $dur(i, d)$ says that the overall duration of step i , based on the actions that took place, is d units of time.

We divide the axioms in our formalization of the STM¹ in a number of categories. We begin with the axioms that describe the effect of storing a symbol s in STM. The first axiom says that an effect of storing s in STM is that s becomes part of STM: $h(in_stm(S), I+1) \leftarrow o(store(S), I)$. The next axiom says that, if adding a symbol to STM would cause the STM size limit to be exceeded, then the symbol that is closest to expiring will be forgotten.

$$\neg h(in_stm(S2), I+1) \leftarrow S1 \neq S2, o(store(S1), I), \\ stm_max_size(MX), curr_stm_size(MX, I), \\ not\ some_expiring(I), oldest_in_stm(S2, I).$$

The axiom relies on a number of auxiliary relations, whose definition is omitted to save space. Relation $curr_stm_size(w, i)$ says that the size of STM at step i is w ; $some_expiring(i)$ states that some symbol in STM will be forgotten when the domain moves to the next step; $oldest_in_stm(s, i)$ says that s is the symbol closest to expiring. The final axioms (omitted) state that an effect of storing s in STM is that the expiration time of s is set to ε , and that it is impossible for two STM-related actions to be executed at the same time. The second group of axioms describes the mechanism of chunking. Three axioms state that, when the components of a chunk are detected in STM,

¹To save space, we omit the most straightforward rules.

the corresponding symbols are replaced by the chunk symbol, whose expiration is set to ε .

$$\begin{aligned} \neg h(in_stm(S), I + 1) \leftarrow \\ \text{detected}(C, I), \text{chunk_element}(C, S). \\ h(in_stm(C), I + 1) \leftarrow \text{detected}(C, I). \\ h(expiration(C, K), I + 1) \leftarrow \\ \text{stm_expiration}(K), \text{detected}(C, I). \end{aligned}$$

Auxiliary relation $\text{detected}(C, I)$, used above, says that chunk C has been detected in STM at step I . The detection occurs, when STM is not in use as per our simplifying assumption above, by checking if there is any chunk whose components are all in STM, as follows:

$$\begin{aligned} \text{detected}(C, I) \leftarrow \text{chunk}(C), \text{not } \neg \text{stm_idle}(I), \\ \text{not } \text{chunk_element_missing}(C, I). \\ \text{chunk_element_missing}(C, I) \leftarrow \\ \text{chunk_element}(C, S), \neg h(in_stm(S), I). \\ \neg \text{stm_idle}(I) \leftarrow o(A, I), \text{memory_related}(A). \end{aligned}$$

It is interesting to notice that the components of a detected chunk are allowed to be located anywhere in STM. However, *now that the model is formalized at this level of detail, one cannot help but wonder whether in reality the focus of the mechanism of chunking is on symbols that have been added more recently*. We were unable to find published studies regarding this issue.

The final group of axioms deals with the evolution of the contents of STM over time when no *store* action occurs. Notice that fluents $in_stm(s)$ and $expiration(s, k)$ are *non-inertial*; in fact, when symbol s is added to STM, $expiration(s, \varepsilon)$ holds at first, but at the next step $expiration(s, \varepsilon)$ becomes false and $expiration(s, \varepsilon - \delta)$ becomes true, where δ is the duration of the latest step, and so on. On the other hand, $in_stm(s)$ is true if-and-only-if $expiration(s, k)$ holds for some $k > 0$.

$$\begin{aligned} \text{noninertial}(in_stm(S)). \text{noninertial}(expiration(S, K)). \\ h(in_stm(S), I + 1) \leftarrow h(in_stm(S), I), \\ \text{not } \text{expiring}(S, I), \text{not } \neg h(in_stm(S), I + 1). \\ \neg h(in_stm(S), I + 1) \leftarrow h(in_stm(S), I), \\ \text{expiring}(S, I), \text{not } h(in_stm(S), I + 1). \\ \neg h(in_stm(S), I + 1) \leftarrow \\ \neg h(in_stm(S), I), \text{not } h(in_stm(S), I + 1). \\ h(expiration(S, K - D), I + 1) \leftarrow \text{expiration_value}(K), \\ h(expiration(S, K), I), \text{dur}(I, D), K > D, \\ \text{not } \text{different_expiration}(S, K - D, I + 1), \\ \text{not } \neg h(in_stm(S), I + 1). \end{aligned}$$

To demonstrate that our formalization captures the key features of the mechanics of STM and chunking, we subject it to a test of memory span. In a memory-span test, a subject is presented with a sequence of digits, and is asked to reproduce the sequence. From now on, we refer to the above formalization of STM by Π_{STM} . Boundary conditions are encoded by the set Π_P of rules, shown below. We use the expression $seq(p, d)$ to represent the fact that digit d is at position p in the sequence.

$$\begin{aligned} \text{stm_max_size}(4). \text{stm_expiration}(30). \text{dur}(I, 1). \\ \text{position}(1). \dots \text{position}(6). \text{digit}(0). \dots \text{digit}(9). \\ \text{primitive}(seq(P, D)) \leftarrow \text{position}(P), \text{digit}(D). \end{aligned}$$

Initially, no symbols are in STM. The corresponding state is σ_{STM} . In the first instance, STM is presented with the sequence 2, 4, 5, 7. Human subjects are normally able to reproduce this sequence. Let us see if our formalization can do the same. The sequence of digits is encoded by set of rules $SPAN_1 = \{o(\text{store}(seq(1, 2)), 0), o(\text{store}(seq(2, 4)), 1), \dots\}$. To predict the behavior of STM, we look at the path in the transition diagram from σ_{STM} , and under the actions in $SPAN_1$. This is accomplished by finding the answer set of $\Pi_1 = \Pi_{STM} \cup \Pi_P \cup \sigma_{STM} \cup SPAN_1$. It is not difficult to check that, at step 4 (corresponding to the end of the experiment), the state of STM is:

$$\begin{aligned} h(in_stm(seq(1, 2)), 4), h(in_stm(seq(2, 4)), 4), \\ h(in_stm(seq(3, 5)), 4), h(in_stm(seq(4, 7)), 4), \\ h(expiration(seq(1, 2), 27), 4), h(expiration(seq(2, 4), 28), 4), \\ h(expiration(seq(3, 5), 29), 4), h(expiration(seq(4, 7), 30), 4), \end{aligned}$$

which shows that the sequence is remembered correctly. In the next instance, the sequence is 2, 4, 5, 7, 3. The set of actions, $SPAN_2$ is straightforward. This sequence is too long for human subjects to remember. Our formalization of STM exhibits the same behavior: in the answer set of $\Pi_2 = \Pi_{STM} \cup \Pi_P \cup \sigma_{STM} \cup SPAN_2$, the first element of the sequence is forgotten by step 5. In the next instance, we consider the sequence 5, 8, 5 - 8, 0, 6, where “-” represents a 1-second pause, to allow time for chunking. We also assume familiarity with the area codes 585 and 806. Under these conditions, human subjects have demonstrated to be capable of remembering the sequence if they are familiar with those area codes. The knowledge about the area codes is encoded by the set Γ_1 of rules such as:

$$\begin{aligned} \text{chunk}(seq(P, ac(roc))). \\ \text{chunk_element}(seq(P, ac(roc)), seq(P, 5)). \\ \text{chunk_element}(seq(P, ac(roc)), seq(P + 1, 8)). \\ \text{chunk_element}(seq(P, ac(roc)), seq(P + 2, 5)). \end{aligned}$$

As before, $SPAN_3$ encodes the sequence of digits. At step 8, the state of STM predicted by our formalization is:

$$\begin{aligned} h(in_stm(seq(1, ac(roc))), 8), h(in_stm(seq(4, ac(lbb))), 8), \\ h(expiration(seq(1, ac(roc)), 26), 8), \\ h(expiration(seq(4, ac(lbb)), 30), 8) \end{aligned}$$

which shows that the chunking of the two area codes has occurred, allowing STM to store a sequence of digits that is longer than its capacity.

References

- Cowan, N. 2000. The Magical Number 4 in Short-Term Memory: A Reconsideration of Mental Storage Capacity. *Behavioral and Brain Sciences* 24:87–185.
- Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.
- Gelfond, M., and Lifschitz, V. 1998. Action Languages. *Electronic Transactions on AI* 3(16).
- Nevid, J. S. 2007. *Psychology: Concepts and Applications*. Houghton Mifflin Company, second edition.