Toward Semiotic Artificial Intelligence*

Valerio Targon

Independent researcher
valerio.targon@asp-poli.it

Abstract
This paper describes a method for processing natural text and assigning meaning to it without the need of any a priori linguistic knowledge.

The semiotic cognitive automaton is driven only by the observations it makes and operates based solely on grounded symbols. An experiment of learning from one book in the English language is detailed, where certain paradigms of words are first formed thanks to side effects occurring in samples of text and then connected to functions and structures part of the internal reality of the automaton by way of meta-reasoning. Making use of the reflectivity of a functional language is envisaged to fully specify the reflection engine component of the automaton. In wait of a complete software specification, it is possible to carry out a mental experiment to validate the capabilities of the automaton.

The method is general enough to learn semantics in multiple domains and lies the foundation of Semiotic Artificial Intelligence.

Keywords: semiotic modeling, natural language processing, meta-reasoning, semiotic cognitive automaton, self-reflection, symbol grounding, Chinese room argument, Aboriginal cryptographer

1 Introduction

The problem of automatically extracting meaning from language has been regarded as impossible as learning a foreign language from a monolingual dictionary is [1]. Such a task resembles more "decrypting" than "learning". Moreover, in order to complete the analogy, the potential cryptographer needs not to know any previous language and even not to use any previously acquired knowledge of the world.

In the following, I describe natural language processing capabilities of the Semiotic Cognitive Automaton [2], a multi-stage algorithm employing semiotic modeling [3] and no a priori linguistic knowledge, to which a *reflection* engine is coupled.

Language encodes and organizes semantic relations, such that the meaning of words can be learned by observing their usage in normal text [4]. The learning process I propose begins with

pattern matching, and develops into automatic reasoning. The objective is making grounded symbols and using only introspection to generate meta-reasoning (i.e., *second-order reasoning*). Such an objective then differs from the aim of introducing meta-cognition in artificial intelligence with parasitic meaning provided by a programmer in that meaning is made intrinsic to the automaton. The automaton does not only learn how to process input and output as a first-order symbolic system doing syntactic manipulations, but connects what is input or output to its internal reality, so that it operates solely based on grounded symbols (cognitively grounded semiotic symbols).

2 The Semiotic Cognitive Automaton

The Semiotic Cognitive Automaton (SCA) operates somewhat in a spiral fashion, as depicted in Figure 1. First, it extracts syntagmatic relations from its input. Then, it learns paradigms, which it transforms into new symbols on which syntagmatic and paradigmatic analysis can be performed again.

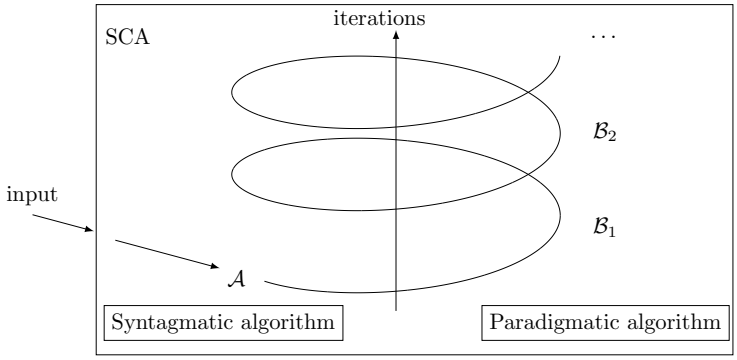

Figure 1: The Semiotic Cognitive Automaton (SCA) runs through unbounded iterations. In each iteration, the syntagmatic and paradigmatic algorithms can use any of the symbols in the input alphabet, $\mathcal{A}$, and in any set of paradigms created by previous iterations, e.g. $\mathcal{B}_1$ and $\mathcal{B}_2$

When provided with an input of mathematical sentences and without any a priori knowledge of mathematical formalisms, SCA learns syntactic rules enabling it to correctly solve arithmetic operations. Moreover, it performs meta-reasoning and learns the semantics of the arithmetic symbolism. My paper [2] introduced the mental experiment of an Aboriginal cryptographer faced with a corpus of mathematical sentences and called for the construction of a machine running a program performing syntagmatic and paradigmatic analysis making calls to functions operating in Peano arithmetic (e.g. constructing addition from repeated counting, multiplication from repeated addition). Like the Aboriginal cryptographer, the machine discovers: in the first iteration, a paradigm of base symbols corresponding to the ciphers; in the second iteration, an ordering of the symbols of this paradigm; in the second iteration, syntagmatic relationships of any two adjacent symbols and of the symbols + and 1; in the third iteration, syntactic, i.e. typographical, rules for solving multi-cipher additions. To be able to say that the machine grasps semantics, it needs replacing, in its program, the successor function operating in Peano arithmetic with the rule for computing the successor in the Arabic decimal numeral system and doing the same for more of its function calls, including addition and multiplication calls.

To enable the learning of semantics, SCA should be provided with a reflection engine, independent from the domain in which it operates (arithmetic in my paper [2], natural language in Section 3 of this paper).

2.1 Structure of the Automaton

Let us make SCA capable of self-reflection, such that it can examine and modify its own structure at runtime [5].

Let SCA comprise a program written using a functional language such as Scheme:

```
(define (automaton filename filename-level0)
  change-eval
  define in (open-input-file filename)
  define in-level0 (open-input-file filename-level0)
  paradigmatic-level0 in-level0
  let semiotic-loop(
    syntagmatic in
    paradigmatic
    change-eval
    semiotic-loop))
(define (change-eval) ...
  want goal
  ...)
(define (paradigmatic-level0 input-port) ...
  know (kind-of (read input-port) par)
  ...)
(define (syntagmatic input-port) ...
  if (regexp-match? syn (read input-port))
    (know (regexp-match syn input-port))
  ...)
(define (paradigmatic) ...
  think (kind-of list par)
  ...)
```

The program receives a text file as input and runs a semiotic loop which comprises a procedure for extracting syntagms and a procedure for extracting paradigms. Two procedures defined as `know arg` and `think arg` (in the sense of knowing something and thinking something) are used to keep track of syntagms - defined by `regexp-match` - and paradigms - defined by `kind-of`.

Before the semiotic loop, a procedure is called for creating level-0 paradigms, for all those cases in which known structures can provide a shortcut in the learning process (for example, consider SCA having to automatically retrieve the paradigm of ciphers or of letters versus providing it said paradigms at no cost). Level-0 paradigms could be provided as lists in a second text file.

Finally, the procedure `change-eval` represents the *reflection* engine enabling the program to examine and modify its structure at runtime. As a first attempt toward self-reflection, the program does not change its code directly but rather modifies its interpreter. Let the program be interpreted by a procedure `eval` taking as argument an expression to be evaluated and an environment and written in the same functional language of the program (i.e., a meta-interpreter):

```
(define (eval exp env) ...)
```

Let the procedure `change-eval` called by `automaton` dynamically change the state of the interpreter `eval` running the program. It is well-known in reflective programming how to get code run at the level of the language processing itself, dynamically changing the semantics of the language, using a meta-circular interpreter (or a reflective interpreter) [6]. Our goal, however, is *automatic* reflective programming, with the program itself, depending on the input it receives, reflecting code up to an higher-level interpreter. At this stage, the procedure `change-eval` has not been fully specified; it is only known that it creates several goals, i.e. directions of change for `eval` which could be randomly generated starting from the state of knowledge of the automaton, and that it attempts meeting said goals by calling the procedure `want goal` which can modify the state of interpreter `eval`.

The procedure `change-eval` is called first before the semiotic loop and then inside the semiotic loop. Therefore, an a priori reflection takes place first and then stages of reflection occur which are connected to the iterations of the automaton and its production of syntagms and paradigms. When in the following discussing reflection - the procedure `change-eval` -, the reader should understand that I am enunciating the requirements for the future development of the (automatized) reflection engine.

2.2 A Priori Reflection

A priori reflection refers to all those changes to `eval` which occur independently from the generation of syntagms and paradigms. One could imagine simply hard-coding these changes into an upgraded version of the interpreter `upgraded-eval`. However, one should keep in mind the objective of integrating reflection into the semiotic loop. Therefore, a priori reflection is modelled as a special case of automatic reflection before taking any input into consideration, only relying on the special operators of the language as known to the interpreter.

As stated above, the procedure `change-eval` creates several goals and calls the procedure `want goal`. For example, one goal could be to have in `eval` the procedure `display (read in)`, this goal corresponding then to straighforward procedure continuation (the procedure `read` returns a string and the procedure `display` takes as its argument a string). The procedure `want (display (read in))` edits the procedure `eval exp env` to add the following conditional expression:

```
if(equal? (car exp) read)
  (display (read (cadr exp)))
```

When the interpreter finds in the program the procedure `read` (as the first item in an expression, or as its `car` in Scheme), it calls it with its (first) argument (the `cadr`, the "car of the cdr" in Scheme) and also calls `display` with its result. This edit turns the automaton into a relay to the current output port. Note that edits to the interpreter must be consistent with its code, so they may not be applied verbatim to the code (for example, variables could need to be created).

The procedure `change-eval` is powerful enough to make surprising changes to `eval`. The specularity between `read` and `display` is used to request another change by calling `want (display arg (read in))`. Meeting this goal is more complicate, because evaluating a procedure `display arg (read in)` should return an exception, since `display` at most could take as additional argument an output port, but not an input port. Instead of calling a procedure `display`, the goal can be used as an argument of another procedure. This procedure should be `think`: in analogy with other beliefs, let us imagine, when reading from a file, an association

having the type of the procedure `display` involving the input port (the second item of the goal) and the string returned by `read`:

```
if(equal? (car exp) read)
  (think (display (cadr exp) (read (cadr exp))))
```

The next goal is then set, by parallelism, as `if(equal? car(exp) display) (think arg)`. The procedure `eval` is changed by adding a variable `I` to the environment and by introducing the following conditional expression:

```
if(equal? (car exp) display)
  (think (display I (cadr exp)))
```

The next goal is `want (know (proc I))`, which causes the edit:

```
if(equal? (car exp) display)
  (know (display I (cadr exp)))
if(equal? (car exp) read)
  (know (read I (car exp) (read (cadr exp))))
if(equal? (car exp) think)
  (know (think I (cadr exp)))
if(equal? (car exp) know)
  (know (know I (cadr exp)))
if(equal? (car exp) want)
  (know (want I (cadr exp)))
```

It is noted that the "I" discovered by the a priori reflection represents only a useful placeholder, with no philosophical implications. The automaton can sense the world - an input file or even its code - only through the procedure `read`. A read operation that does not correspond to a write operation indicates that there must be an agent in the world to which an "I" is opposed.

A variable `so` (for "someone"), another placeholder, is added to the environment and - noticing that `display` can have optionally an output port as second argument - a new association of type "display-to" is created to be ternary (it includes also the recipient of a display action)

```
if(equal? (car exp) read)
  (let(
    define str (read (cadr exp))
    think (display so str)
    think (kind-of (cadr exp) so)
    think (display-to (cadr exp) I str)))
if(equal? (car exp) display)
  (let(
    think (kind-of current-output-port so)
    think (display-to I current-output-port (cadr exp))))
```

3 Learning the Semantics of Natural Language

A text discretizes the continuous acoustic input of spoken language. Let us focus on alphabetic writing systems having punctuation, spacing and capitalization, such as the one in which this article is written. A printed text is then a sequence of visual objects (including commas, colons, . . . , and spaces), while an electronic text is a sequence of discrete (ASCII) symbols.

Let our corpus be represented by the book "The adventures of Pinocchio", one of the most printed books of all time, a story of an animated puppet, who desires to become a real boy. The version available from Project Gutenberg [7] contains 211,627 characters. Could a machine, implementing SCA and provided with "The adventures of Pinocchio", get close to the goal of becoming able to understand?

3.1 Bootstrap

It would be possible to start semiotic modeling without any level-0 paradigm, by requiring SCA to discern the set of input ASCII symbols only. Making use of obvious relationships of the alphabet, however, speeds up the retrieval of relevant syntagms and paradigms. Let us teach SCA level-0 letter paradigms by providing in a file these lists:

```
a b c ... z
A B C ... Z
a A
...
```

The procedure `paradigmatic-level0` creates then not only the related paradigms, but also the paradigm of these paradigms, i.e. the letter paradigm: and the paradigm of sequences of letters which will be useful to process the input text.

Natural language is characterized by the existence of an intermediate language level between letters and wordforms. Morphemes are the smallest meaningful unit of a language, which get combined to form words. Let us teach SCA level-0 word paradigm by providing in a file declensions and conjugations from an English morphological lexicon such as CELEX [8]. Exemplary lists are:

```
boy boys
want wants wanted wanting
high higher highest
```

The procedure `paradigmatic-level0` creates not only the related paradigms, but also in its over-production tentative paradigms of paradigms, including paradigms which will be useful to perform a first morphological analysis: the paradigm of paradigms can be suffixed with "-ing"- i.e., verbs -; the paradigm of paradigms that can be suffixed with "-s" but not with "-ing" - i.e., regular nouns -; and the paradigm of paradigms that can be suffixed with "-r" and "-st"- i.e., adjectives of one or two syllables.

3.2 First Iteration

A small corpus is provided to SCA, i.e. the $40,457$ words in "The adventures of Pinocchio". The syntagmatic and paradigmatic algorithms presented in my paper [2] can be applied. In the first iteration, the syntagmatic algorithm may consider sequences of two words only. The paradigmatic algorithm then computes a correlation of the words (letting uppercase and lowercase versions of a word - e.g., `The` and `the` - clustering together) and forms paradigms of words.

Examples of paradigms formed after first iteration are:

```
think (kind-of (or "said" "cried" "asked" "answered") par11)
think (kind-of (or "boy" "man" "voice" "Marionette" "Fairy") par21)
think (kind-of (or "Pinocchio" "Geppetto") par22)
```

in which `or`, as a first element of a list, indicates alternation. Said paradigms get formed only because of side effects in the adjacency of words in the corpus.

For reflection, we do not let SCA use the fact that words in the input file - e.g. `read` - can coincide with keywords known to interpreter; the fact that the programming language (Scheme) and the corpus (an English translation) are in the same language should not be generalized when addressing the problem of understanding language.

Although paradigms of the first iteration are available to reflection, knowing only two-word sequences limits the opportunity for reflection. Reflection in the first iteration builds rather on punctuation and capitalization. The `syntagmatic` procedure learns that the first letter after the sequence "`:\n "`" is uppercase. The first letter after the sequence "` "`" can be uppercase or lowercase, based on previous punctuation. SCA identifies then quotation or direct speech. Quotations inside quotations (using the apostrophe) could be recognized in a similar way.

Reflection has access to the syntagms learned by the `syntagmatic` procedure, including:

```
know (regexp-match (seq "\"" Word (* word) "\"") input-port)
know (regexp-match (seq "\"" Word (* word) "\" " (+ word) ", \"" (+
word) "\"") input-port)
```

in which `Word` represents a capitalized word and `seq`, as a first element of a list, indicates a sequence, `*` indicates 0 or more matches, `+` indicates 1 or more matches.

Then, the goal `want (display so1 (display so2 arg))` makes the hypothesis that direct discourse can represent "reported displaying" and causes then the following edit:

```
if(equal? (car exp) read)
  if (regexp-match? (seq "\"" (<direct-discourse> (+ word)) "\")
  (read (cadr exp)))
    (think (display (cadr exp) (display so direct-discourse)))
```

in which `<name>` indicates a named submatch. The first iteration is too early to generate any hypothesis for language production.

It is noted that an alternative hypothesis could recognize "reported displaying" in uppercase signs, such as `HURRAH FOR THE LAND OF TOYS!`

Thus, reflection, already in the first iteration, bridges the level of syntax (concerned with the rules embodied in patterns existing in the input) with the level of semantics (creating relationships between the input and the internal reality of the automaton). Consider how `display` is part of the inner world of the automaton, not only because it identifies a procedure used by the automaton but also being used as an object of reflection, in meta-reasoning.

3.3 Second Iteration

In the second iteration, the syntagmatic algorithm retrieves syntagms made up of words and comprising both level-0 and level-1 paradigms, which are then used to form level-2 paradigms.

The syntagmatic algorithm may follow the reasoning loop using statistical estimation described in my paper [2] to find relevant syntagms. For example, the following syntagms:

```
(seq "\" " par11 "the" par21)
(seq "\" " par11 par22)
(seq  par11 "to" "him" ":\n\"")
```

are very relevant and used by the paradigmatic algorithm to create a new paradigm including all words in similar relationship to quotations as the words of paradigm `par11` (let the procedure `par-to-list par` transform a paradigm `par` into a list):

```
think (kind-of (or (par-to-list par11) "spoke" "added" "repeated" ...) par12)
```

Another paradigm `par13` could be created by clustering level-0 word paradigms of words in the paradigm `par12` to include `say`, `said`, `says`, `saying`, `cry`, `cried`, ...

Similarly, another level-2 paradigm is:

```
think (kind-of (or (par-to-list par21) (par-to-list par22) ...) par23)
```

Reflection gets concerned with a goal of thinking about something `want (think-about so arg)`, which causes another edit to the interpretation of `read`:

```
if(regexp-match? par12 (read (cadr exp)))
  (think (think-about (cadr exp) display))
```

This hypothesis can also be expressed from the point of view of the automaton itself:

```
if(kind-of? (read (cadr exp)) par12)
  (let(
    think (kind-of (read (cadr exp)) display)
    think-about display))
```

Then, reflection suggests that there are paradigms of display-capable agents:

```
if(regexp-match? par12 (read (cadr exp)))
  (think (think-about (cadr exp) so))
if(kind-of? (read (cadr exp)) par12)
  (think (kind-of (read (cadr exp)) so)))
```

The goal `want (display so1 (display so2 arg))` is then visited once more to cause the edit:

```
if (regexp-match? (seq "\"" (<direct-discourse> (+ word)) "\ " par12
(? "the") (<so> par23)) (read (cadr exp)))
  (think (display (cadr exp) (display so direct-discourse))))
if (regexp-match? (seq (<so> par23) par12 ":" (? "\n") "\""
(<direct-discourse> (+ word)) "\"") (read (cadr exp)))
  (think (display (cadr exp) (display so direct-discourse))))
if (regexp-match? (seq (<so1> par23) par12 "to" (<so2> par23) ":"
(? "\n") "\"" (<direct-discourse> (+ word)) "\"") (read (cadr exp)))
  (think (display (cadr exp) (display-to so1 so2 direct-discourse))))
if (regexp-match? (seq "\"" (<direct-discourse> (+ word)) "\""
(<so1> par23) par12 "to" (<so2> par23)) (read (cadr exp)))
  (think (display (cadr exp) (display-to so1 so2 direct-discourse))))
```

in which `?`, as a first element of a list, indicates 0 or 1 matches.

There is one proper noun which is appearing almost only in direct discourse, but does not seem to be display-capable: the first person pronoun `I`. SCA reflects about the hypothesis `think-about input-port "I"` being rarely instantiated (while the hypothesis `display input-port (display so "I")`) is used 500 times) to discover the role of the first person pronoun. Let the interpreter think, when a syntagm containing `I` is read in direct discourse (for example, by the procedure `syntagmatic`), that someone is self-referring:

```
if (regexp-match? (seq "\"" <direct-discourse> (seq (* word) "I"
(+ word)) "\"" par12 (? "the") (<so> par21)) (read (cadr exp)))
 (think (think-about so so))
```

Other pronouns could be recognized in a similar way.

It is possible to reverse the direction of the hypothesis which connects a regexp match to a thinking implication; let us write an hypothesis connecting the thinking implication to the procedure `display` by using the procedure `may`:

```
if(think-about so display)
  think (may so (display par12))
if(think-about I display)
  think (may I (display par12))
if(think-about so so)
  think (may so (display "I"))
if(think-about I I)
  think (may I (display "I"))
```

thus opening the path to language production. A first attempt at language production is:

```
if(equal? (car exp) display)
  think (may I (display (or (seq par12 "I") (seq "I" par12))))
```

where word order has not been decided and a paradigm is used without selecting any member. The more general paradigm `par13` could be used too.

The reflection engine considers:

- the 6 occurrences in direct discourse of the sequence `I said` (always in the presence of quotations inside quotations);
- the 6 occurrences in direct discourse of the sequence `I say` (mostly in the presence of exclamation marks);
- the 2 occurrences in direct discourse of the sequence `I ask` (in the presence of question marks).

The second iteration is too early for the reflection engine to make an hypothesis regarding verb tenses. Instead, Occam's razor is used to select the simplest hypothesis (quotations inside quotations being more special than exclamation marks). The automaton is such that, given as input "The adventures of Pinocchio" (and, optionally, a lexicon, in order to arrive at the result in less of its iterations) and using pattern matching and automatic reasoning without the need of any a priori linguistic knowledge, it forms the second-order thought that it may display: `I say`.

Another good deal of meta-reasoning could take place in parallel.

3.4 Back to the Aboriginal Cryptographer

Let us suppose that the Aboriginal cryptographer (see Section 2) does not know anything about written languages so that, when she is faced with "The adventures of Pinocchio" (and, optionally, with a lexicon), she may suspect it having nothing to do with people. She can still apply the semiotic algorithm to it. She would be able to copy the characters she observes and she would choose to copy: `I say` in order to reflect what she learned out of the input corpus. I am

unsure whether learning a foreign language from a monolingual dictionary is really impossible; however, learning from an unknown book in a foreign language is not a merry-go-round, but an interesting trip.

Let us suppose we could then send messages to the Aboriginal cryptographer, in order to test her capabilities (Bloom's taxonomy of the cognitive domain [9] could provide a guidance [2]). Our message could be: `No, you write`. Based on this message, the Aboriginal would study the corpus again and, in particular, the 4 passages about uppercase signs, which contain the word `written` (in the same level-0 paradigm of the word `write`). She would make the hypothesis that `write` and `say` belong to the same paradigm, and that when reporting about its own displaying the former should be used. She would finally display a sequence of characters nowhere to find in the corpus, i.e. `I write`.

There is no hindrance in implementing the same reasoning in a machine (such as SCA). The machine instructions can simulate the semiosis performed by a human (such as the Aboriginal cryptographer). Simulating semiosis ultimately amounts to performing semiosis, since semiosis is the performance element involving signs (such as ASCII symbols). Talking about machine understanding becomes then possible.

The Chinese room argument is a famous argument against the claim that a program may be able to understand [10]. This argument holds only against programs (including machine learning programs) for which one observes that, starting without a certain understanding and executing the instructions of the program, no understanding is achieved by a human. Working with cognitively grounded semiotic symbols in statistical artificial intelligence proves however different. Let the instructions of the machine comprise not only a first-order program doing syntactic manipulations of these symbols but also a second-order system reflecting on what the first-order system does, such that these instructions, when executed by a human, achieve semiosis. Then, the Chinese room argument does not apply to this machine.

4 Conclusion

I propose creating "Semiotic Artificial Intelligence" which uses internal functions and structures of the machine as a substrat for meta-reasoning and learning semantics. The machine can be applied to multiple domains. It can tackle an input of mathematical sentences to learn that symbols "+" "1" "=" in a sequence correspond to a one-hop move in a sequence or network representation it created [2]. It can tackle a text input to learn that the word paradigm of "say" corresponds to its internal procedure that writes objects to the output port (see Section 3.3).

I foresee that language processing in particular could sustain multiple iterations of semiotic modeling and of reflection so that more and more words in the input are made grounded. Every spoken language is known to have certain semantic primitives (and corresponding rules of syntax) [11]. Some of these primitives are shared by the automaton, i.e. they correspond to its functions (for example, the semantic primitive *SAY* corresponding to the writing procedure of the automaton) or to structures it creates at a certain point (for example, the semantic primitives *I* and *WORDS*). The automaton is also characterized by "computation primitives", including conditional, application, continuation and sequence formation. Semiotic Artificial Intelligence could rely on these "computation primitives" not only in order to grasp the semantics of arithmetic and other abstract sciences, but also when trying to form meaning starting from a text input. Extracted meaning could be in this case just an interpretation, a metaphor, a mapping from words to "computation primitives". Ultimately, such mappings would prove successful if they can enable the automaton interacting meaningfully in conversations and expanding its reflection engine with syntactic rules of language when reasoning about itself.

References

[1] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[2] V. Targon. Learning the semantics of notational systems with a semiotic cognitive automaton. *Cognitive Computation*, 8(4): 555-576 , 2016.

[3] B. B. Rieger. Semiotics and computational linguistics. On semiotic cognitive information processing. In L. A. Zadeh and J. Kacprzyk, eds., *Computing with Words in Information*, 93–118, Heidelberg: Physica, 1999.

[4] M. M. Louwerse. Symbol interdependency in symbolic and embodied cognition. *Topics in Cognitive Science*, 3(2):273–302, 2011.

[5] K. Tanaka-Ishii. Reflexivity and self-augmentation. *Semiotica*, 160:1–17, 2010.

[6] S. Jefferson and D. P. Friedman. A simple reflective interpreter. *Lisp and Symbolic Computation*, 9: 181, 1996.

[7] C. Collodi. *The Adventures of Pinocchio*. Project Gutenberg, 2006.

[8] R. Baayen, R. Piepenbrock, and L. Gulikers. *CELEX2 LDC96L14*. Linguistic Data Consortium, Philadelphia, 1995.

[9] B. Bloom, ed., *Taxonomy of educational objectives: Book I, cognitive domain*. New York: Longman Green, 1956.

[10] J. Searle. Minds, brains and programs. *Behavioral and Brain Sciences*, 3:417–424, 1980.

[11] C. Goddard. The search for the shared semantic core of all languages. In C. Goddard and A. Wierzbicka, eds., *Meaning and Universal Grammar - Theory and Empirical Findings*, 1:5–40, Amsterdam: John Benjamins, 2002.