Towards Defining a Cognitive Linguistics of Programming and Using Eye Tracking to Verify Its Claims

Sebastian Lohmeier

Technische Universität Berlin
sl@monochromata.de

Keywords: POP-II.B program comprehension, POP-V.A cognitive linguistics, POP-V.B eye tracking

1 A Cognitive Linguistics of Programming

There are cognitive models that explain programming (Parnin, 2010) and psychological experiments based on models from cognitive psycholinguistics (Burkhardt, Détienne, & Wiedenbeck, 1997). In computer science, attempts have been made to enhance programming languages based on knowledge of natural languages (Knöll, Gasiunas, & Mezini, 2011; Knöll & Mezini, 2006; Lopes, Dourish, Lorenz, & Lieberherr, 2003). Combining these three relations, I propose a cognitive linguistics of programming that is based on three-level semantics, a theory of text comprehension from cognitive linguistics (Schwarz, 1992; Schwarz-Friesel, 2007). The cognitive linguistics of programming enables psychological experiments and is used to enhance programming languages.

Three-level semantics was chosen as the basis of the cognitive linguistics of programming, because it specifies structures and processes that can be implemented computationally and are comparable to those found in object-oriented programming languages. Furthermore, three-level semantics is suitable for generating testable psychological hypotheses and incorporates the reader. That is why I describe a cognitive linguistics of *programming*, not of *programming languages*: three-level semantics assumes an active role of the reader – she constructs meaning while reading. Using such a reader-centric model of language, it becomes possible to explain problems related to programming language understanding during the act of programming.

To make a start, the cognitive linguistics of programming is realized by adding indirect anaphors known from linguistics to the Java programming language (Lohmeier, 2011).

2 Indirect Anaphors in Linguistics

How are indirect anaphors described in linguistics? Example (1) contains the indirect anaphor *the Expression* whose definite article *the* signals that *the Expression* is known, either due to world knowledge or prior mention.

(1) An if-then statement is executed by first evaluating the Expression. If the result is of type Boolean, it is subject to unboxing conversion (Gosling et al., 2005, 372)

According to Schwarz-Friesel (2007), the indirect anaphor *the Expression* is understood in relation to its antecedent *An if-then-expression* and based on the knowledge that if-then statements contain an expression. Using conceptual schemata, this fact is expressed as a part-whole relation between *EXPRESSION* and *IF-THEN-STATEMENT* (upper case denotes concepts). Reading the sentence can then be described as follows: While reading the antecedent *An if-then-statement*, the default *EXPRESSION* part of the *IF-THEN-STATEMENT* concept is activated in the mind and is still active when *the Expression* is read. The definite article *the* signals that *the expression* refers to a known *EXPRESSION* and since the default *EXPRESSION* part of the *IF-THEN-STATEMENT* is the only active referent of a matching concept, *the expression* refers to it. During reading, the reader retrieves from memory the part-whole-relation that is underspecified in the text, but available from long-term memory. Likewise, she reconstructs that *the result* is *the result of the Expression*. By resolving the underspecification of the indirect anaphors *the Expression* and *the result*, the reader understands a text that would be longer otherwise:

(2) An if-then statement is executed by first evaluating *its* Expression. If the result *of the Expression* is of type Boolean, it is subject to unboxing conversion

If the underspecified relations are well known to the reader, underspecification shortens texts and improves learning of new relations explicated in the text (McNamara, Kintsch, Butler-Songer, & Kintsch, 1996).

3 Indirect Anaphors in Programming

Knowledge of part-whole and other semantic relations is encoded in object-oriented programs. It is thus desireable to shorten source code by introducing indirect anaphors to programming languages in order to improve programmers' learning from source code. This requires compilers that are able to process indirect anaphors similar to programmers. Listing 1 shows an example of an indirect anaphor in programming.

Listing 1. Indirect anaphor `.Result` in a modified org.junit.runner.JUnitCore (JUnit 4.8.2)

```
52 public static void runMainAndExit(JUnitSystem system, String...
       args) {
53     new JUnitCore().runMain(system, args);
54     system.exit(.Result.wasSuccessful() ? 0 : 1);
55 }
```

The indirect anaphor `.Result` in line 54 refers to the `Result` returned by the invocation of `runMain`, eliminating a local variable from the original source code shown in Listing 2.

Listing 2. Original snippet from org.junit.runner.JUnitCore (JUnit 4.8.2)

```
52 public static void runMainAndExit(JUnitSystem system, String...
       args) {
53     Result result= new JUnitCore().runMain(system, args);
54     system.exit(result.wasSuccessful() ? 0 : 1);
55 }
```

4 Claims

Experienced programmers should not have difficulty understanding both listings. While it is not clear whether they would take less time to read the code, if the findings for underspecification in natural language hold for indirect anaphors in programming languages, programmers can be expected to better recall new information from source code underspecified with indirect anaphors. Like in the case of underspecification in natural language, indirect anaphors will of course impede understanding for those programmers who do not possess the underspecified knowledge. Identifying what knowledge is available to programmers so as to be able to present code with or without indirect anaphors would require use of a cognitive architecture (Hansen, Lumsdaine, & Goldstone, 2012; Lohmeier & Russwinkel, 2013) and is beyond the scope of this work.

5 Eye Tracking

To test whether source code with indirect anaphors is slower or faster to comprehend than traditional source code, eye tracking could be applied relatively naturally by integrating eye tracking into an existing IDE (see `http://monochromata.de/eyeTracking`). This would allow a comparison of eye movements and reading times for indirect anaphors in natural language texts (Garrod & Terras, 2000) and source code. Presenting programmers source code with varying numbers of indirect anaphors would permit comprehension studies in the fashion performed for natural language texts (McNamara et al., 1996). For both reading times and comprehension of indirect anaphors in source code, the effect of a reader's possession or lack of knowledge required to overcome underspecification will be of interest.

References

Burkhardt, J.-M., Détienne, F., & Wiedenbeck, S. (1997). Mental representations constructed by experts and novices in object-oriented program comprehension. In S. Howard, J. Hammond, & G. Lingaard (Eds.), *Human-computer interaction: INTERACT '97.* Chapman & Hall.

Garrod, S., & Terras, M. (2000). The contribution of lexical and situational knowledge to resolving discourse roles: Bonding and resolution. *Journal of Memory and Language*, *42*, 526–544.

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *The Java language specification* (3rd ed.). Boston: Addison Wesley.

Hansen, M. E., Lumsdaine, A., & Goldstone, R. L. (2012). Cognitive architectures: A way forward for the psychology of programming. In *Onward! 2012: Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software* (pp. 27–37).

Knöll, R., Gasiunas, V., & Mezini, M. (2011). Naturalistic types. In *Onward '11: Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software* (pp. 33–47).

Knöll, R., & Mezini, M. (2006). Pegasus: first steps toward a naturalistic programming language. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (pp. 542–559). New York: ACM.

Lohmeier, S. (2011). *Continuing to shape statically-resolved indirect anaphora for naturalistic programming: A transfer from cognitive linguistics to the Java programming language.*

Lohmeier, S., & Russwinkel, N. (2013). Issues in implementing three-level semantics with ACT-R. In *Proceedings of the 12th international conference on cognitive modeling (ICCM).*

Lopes, C. V., Dourish, P., Lorenz, D. H., & Lieberherr, K. (2003). Beyond AOP: toward naturalistic programming. *SIGPLAN Notices*, *38*(12), 34–43.

McNamara, D. S., Kintsch, E., Butler-Songer, N., & Kintsch, W. (1996). Are good texts always better? Interactions of text coherence, background knowledge and levels of understanding in learning from text. *Cognition and Instruction*, *14*(1), 1–43.

Parnin, C. (2010). A cognitive neuroscience perspective on memory for programming tasks. In *PPIG 2010, 22nd annual workshop.*

Schwarz, M. (1992). *Kognitive Semantiktheorie und neuropsychologische Realität: repräsentationale und prozedurale Aspekte der semantischen Kompetenz.* Tübingen: Niemeyer.

Schwarz-Friesel, M. (2007). Indirect anaphora in text: A cognitive account. In M. Schwarz-Friesel, M. Consten, & M. Knees (Eds.), *Anaphors in text : cognitive, formal and applied approaches to anaphoric reference* (pp. 3–20). Amsterdam: Benjamins.