3-1-2012

Metonymy and Student Programming Errors

Craig Miller
DePaul University, cmiller@cs.depaul.edu

Metonymy and Student Programming Errors

Craig S. Miller
School of Computing
DePaul University
243 S. Wabash Avenue
Chicago, IL
cmiller@cs.depaul.com

ABSTRACT

The common occurrence of metonymy in everyday language is considered as a negative bias towards successfully learning to state the correct referent when learning to program. Reported errors from previous studies are surveyed and the analysis reveals a pattern consistent with the use of metonymy, a rhetorical device where the speaker states a referent that is structurally related to the intended referent. This analysis suggests an underlying cause for a class of programming errors and provides directions for further research and instructional interventions.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Human Factors, Languages

Keywords

Metonymy, Misconceptions, Novice programming

1. INTRODUCTION

Consider this sentence: *The tourist asked the travel bureau for directions to the museum.* Few people would have difficulty understanding that the tourist did not talk to a building, nor to an abstract government agency, but to an actual person who works at the travel bureau. Moreover, the process of identifying the actual referent (person) through its relationship to the place of employment (travel bureau) comes so effortlessly to most people that they may not even be aware that the sentence uses a rhetorical device for expressing how the tourist acquired directions.

Here the rhetorical device is metonymy. When using metonymy, the speaker does not state the actual referent. In its place, the speaker states something that has a relationship with the referent, often with the goal of emphasizing that aspect of the relationship. Below are additional examples:

- *The White House condemned the latest terrorist action.* The action was not condemned by the White House building but by a person who represents the president who lives at the White House.
- *The pitcher threw the ball to first base.* In the game of baseball, the pitcher throws the ball to the person standing at first base.
- *I thought the first dish we ate was excellent.* The word *dish* does not refer to the physical plate but the food that was on the plate.
- *Open the bread and take out two slices.* The request is to open the wrapper containing the bread, not the bread itself.

In some cases, use of metonymy has become so common with some words, that their dictionary definitions include the extended meaning. For example, the definition of a *dish* includes the food served on it. However, note that in the right context, any food-containing vessel (e.g. bowl, plate, pot) could carry a similar meaning. Offering additional examples, Lakoff and Johnson [10] make the case that metonymy is not just a rhetorical device but language usage that reflects everyday activity and thinking.

Given the pervasive use of metonymy, it would be a surprise if its influence did not extend to how students learn computing concepts and, in particular, programming. In fact, the last listed example using the word *bread* was inspired by an account of student errors from a class exercise. Originally described by Lewandowski and Morehead [11], this exercise asks students to issue precise commands so that the instructor, acting as a robot, assembles a peanut butter and jelly sandwich. A major pedagogical goal of the exercise is to introduce algorithms to students by having them develop a clear set of operational instructions. Among accounts of student errors for this exercise, Davis and Rebelsky [4] present the phrase, "Open the Bread," as an example of student error. While none of the accounts of this exercise have cited the role of metonymy, it ostensibly underlies some of the difficulty students face when forming a precise, literal command.

While metonymy has been used for developing design patterns [12], the goal of this paper is to explore its role as a source of student mistakes when learning to program. The

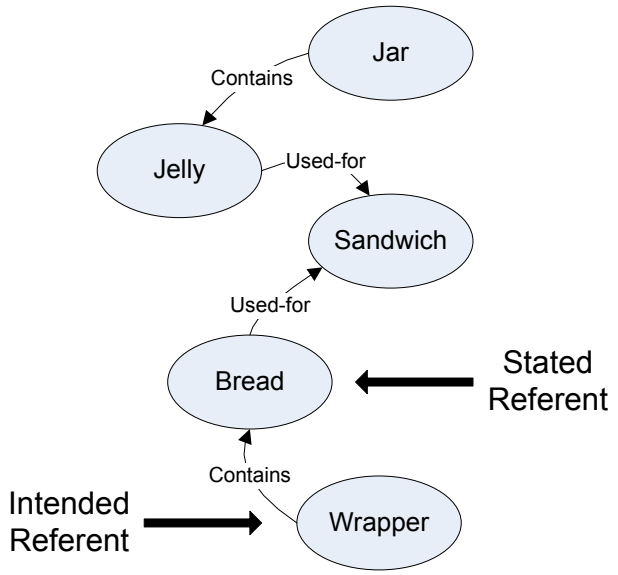

Figure 1: Semantic network demonstrating metonymy

working thesis is that students' prior experience with language leads to their difficulty in expressing the exact referent to a command or operation. To support this thesis, this paper reviews a broad range of student errors and shows how metonymy underlies them. In so doing, this analysis offers a useful means for classifying student errors to track development of student ability. Also this diagnosis of student misunderstanding provides some direction for developing effective instructional strategies. Finally, to the extent that this analysis using metonymy yields helpful diagnoses and strategies, it may offer a useful heuristic for identifying other categories of misconceptions based on prior student experiences, whether they come from language or otherwise.

2. EXAMPLES

Metonymy-based errors involve incorrectly referring to an element that has a structural relationship with the intended element. As we will see, the structural relationship is often, but not always, based on data structures represented in computer memory in the form of arrays, linked lists or objects.

The examples in this paper will make use of the terms *intended referent* and *stated referent*. Figure 1 provides a concrete example using the sandwich exercise. When the speaker says, "Open the bread," the stated referent is the bread, but the intended referent is the wrapper, which is really the item that is to be opened. A listener can infer the intended referent through its structural relationship (contains) to the bread. Of course, students need to learn that most programming environments require that the intended referent is explicitly referenced, otherwise an error will result.

2.1 Object instances and attributes

Holland *et al.* [8] note that students often conflate an object with one of its attributes. For example, they may refer to the whole object when it is the value of a particular attribute that is needed. Consider the following code, which uses statements that are not unlike many programming languages and environments for constructing a graphical user interface:

```
tempField = TextField.new
tempField.id = "temp"
tempField.size = 5
tempField.value = 32
```

This code constructs an object that represents a text field. The code shows three attributes for the text field object. The **id** uniquely identifies the component in the interface. The **size**, specifies the width of the text field. The **value** specifies the contents of the text field. In everyday (non-computer) language, both the id and the value may usefully identify the text field when speaking to a human listener. However, this use of metonymy are considered errors as programming statements:

```
# mistakenly refer to the object to specify
#    the contents
givenTemperature = tempField

# mistakenly refer to the id to specify
#    the contents
givenTemperature = temp

# mistakenly refer to the id when changing
#    the width
temp.size = 10
```

These examples illustrate that metonymic errors are not simply characterized as mistaking one referent for another structurally-related referent. In addition, the mistakenly stated referent tends to be an element that usually distinguishes the intended referent from other items. This analysis suggests that students would not mistakenly refer to the size element in place of the other elements since size generally does not uniquely identify a component in an interface.

While all of these errors involve referents with structural relationships in computer memory, it is also possible that metonymic usage may lead a student to mistakenly refer to other elements when the intended referent is the text field. For example, text fields typically have an adjacent text label. While there is no relationship between the label and the text field with respect to computer memory, there is a meaningful structural relationship between the two components as the application developer and user understand them. In this sense, a novice programmer may mistakenly state the label object (or perhaps an attribute belonging to it) when the intention is to refer to some aspect of the corresponding text field.

2.2 Array elements and indices

While previous reports have noted the commonality of array errors [3, 6], detailed descriptions of student errors with arrays are less common and rarely discuss the underlying cause of the errors. One potential difficulty involves the distinction between the index and the actual value located at the position denoted by the index. For example, let us consider the selection sort, which requires finding the minimal value in order to swap it into the correct position. It is easy

to conflate the minimal value with its position, especially when descriptions of the selection sort refer to the value instead of its location (e.g. Wikipedia presents the first step of the selection sort with "Find the minimum value in the list" [1]). As a consequence students may store the value instead of its position, which is ultimately needed to swap the array element into its correct position. More generally, communication to a human listener may refer to the value even if the intended referent is its location. However, in the context of a computer program, referencing the value (stated referent) when the intended referent is the location results in a programming error.

Also related to arrays is the conflation between the array itself and its contents. This error is most likely when the array has only one element. Instead of indexing the first element in the array in order to refer to it, a student may mistakenly refer to the whole array (without using the index).

2.3 References and contents

Hristova et al. [9] note that students often mistake reference comparisons with contents comparisons. For example, in Java, if the intention is to check whether two strings (actually two references to string objects) have the same string contents, students often incorrectly use the == operator, which only checks if the variables refer to the same references, instead of the **equals** method, which actually compares the string contents. The mistake is understandable since reference variables to strings are often just called strings. Following the practice of metonymy, a student may justifiably expect the == operator to do the "reasonable" comparison even if the stated referents are just references. For Java and any language that represents strings as objects, this error is a special case of referencing an object (string object) when the intended referent is an attribute (string content of the object). However, similar mistakes could generalize to any data structure where there is a distinction between a memory reference and the value located at that reference.

2.4 Other structural relationships

As already discussed, the parallel to metonymy may not necessarily involve a structural relationship that is represented in computer memory. An earlier account using Pascal [2] presents a common student mistake when finding the mode (value with the largest number of occurrences) in an array of integers. Instead of returning the value with the highest frequency, the incorrect code returns the frequency itself. Like the use of metonymy, the intended referent (the mode) is related to (defined by) the stated referent, although the two are not directly linked in a computer data structure. Also, like metonymy, the stated referent plays a role in identifying the intended referent.

3. IMPLICATIONS

Computer science instructors have long known that students will confuse one element for another related element. At times, it may be tempting to simply dismiss such mistakes as sloppy thinking, conceptual ignorance or even a general inability to think computationally. Of course, these vague causes offer little direction for understanding and correcting these mistakes. More helpful are efforts that try to identify areas where students are missing important distinctions. Indeed, some of the previous reports of student errors have taken this approach. However, to the extent that experience with metonymy also contributes to these errors, an analysis based on metonymy provides a systematic approach to a range of logical errors spanning multiple programming paradigms. In particular, understanding these errors as metonymic errors adds the following insights:

- Use of metonymy explains the source of these errors or at least suggests a contributing factor for why students are likely to make the errors.
- Use of metonymy explains that the stated referent has a structural relationship with the intended referent.
- Use of metonymy explains that the stated referent is useful for uniquely identifying the intended referent.

As explanations, they make predictions about what to expect for when students mistakenly refer to an element. For example, the last insight predicts that students are not likely to state any referent that has a relationship to the intended referent if the stated referent is not useful for identifying the intended referent. As we saw, the size attribute is not useful for identifying a particular text field and we thus predict that a student would not refer to the size for this purpose in a statement.

3.1 Status as a misconception

While prior use of metonymy is arguably a contributing factor to the student errors reviewed here, its status as a misconception is open to debate. It is doubtful that students have a coherent theory of metonymy and explicitly apply it when writing a programming statement. A more plausible interpretation is that students have a false expectation of a computer's ability to successfully infer the intended referent, without realizing that such an inference would require domain knowledge and a representation of the programmer's goal. In this case, students at least have some awareness of the difference between the intended referent and the stated item, just as they know the difference between a loaf of bread and its wrapper. This interpretation thereby supposes that the students have a concrete expectation that the computer will successfully interpret their statement, an expectation that is based on their previous experience with language.

This interpretation is akin to theory on how students acquire physics knowledge. diSessa [5] asserts that novice students possess "pieces of knowledge" based on their previous experience with the world. Each piece of knowledge, called a phenomenological primitive (p-prim), does not act as a coherent theory of understanding and is not necessarily consistent with other p-prims. For example, a student may predict that a cannonball falling from a mast of a moving ship would land behind the ship, rather than have the correct understanding that the ball would land at the base of the mast. This naive prediction is based on the student's experience that objects falling from a moving vehicle tend to fall behind the vehicle (assuming air resistance is a significant factor). In this sense, the student's incorrect prediction based on previous experience is similar to that of a metonymy-based reference error in programming, although physics p-prims are probably cued by the physical properties of the situation. In contrast, any effort to establish the p-prim equivalent for computational learning would involve identifying cues relating to language or communication.

A final possibility is that students do not even possess a concrete expectation that their intended referent will be successfully resolved by the compiler or interpreter. Perhaps they do not distinguish between the stated referent and the intended referent. In this case, prior use of metonymy may have nevertheless induced a habit of communication, which is effective for inter-personal conversation but incorrect for human-to-computer commands.

Whether prior usage of metonymy has given a student a false expectation for how a computer resolves referents or has merely encouraged a habit of the mind where making distinctions are not important, the metonymy-based analysis demonstrates the utility of examining how prior experiences may negatively influence how students learn computational concepts. This case runs counter to some discussions comparing physics to computing, where the conventional wisdom is that learning physics involves overcoming prior expectations whereas computing students approach programming with a relatively clean slate (see Mark Guzdial's blog [7] for a discussion on the topic).

4. RECOMMENDATIONS

The role of metonymy in student errors suggests several directions for education research and instructional strategies. For assessing student achievement, metonymy-based analysis may be useful for constructing diagnostic questions that generalize across multiple languages and even multiple programming paradigms. As we have seen, student errors that parallel metonymy-based language usage occur in a variety of contexts. To the extent that the metonymy-based analysis unifies student errors across various contexts, we would expect this class of errors to predict similar errors but in other contexts.

For developing effective instructional strategies, the role of metonymy may offer guidance for teaching students how to successfully refer to the correct referent in programming statements. One possible intervention is to explicitly teach students the concept of metonymy in everyday language but then discuss how its practice does not work for computer statements. For topics where it is common to misstate the referent in a statement, the instructor can show examples and refer back to the lesson on metonymy. Explicit awareness of metonymy can support students in two ways. First, it illustrates a programming pitfall that generalizes to many circumstances. Second, its examples can make use of situations that students know well (e.g. making sandwiches). Use of familiar situations effectively reduces the cognitive load of the example and allows students to focus on the underlying relationship between the stated referent and the implied referent.

Finally, the role of metonymy for student errors suggests that there might be other habits and practices in students' everyday lives that affect how they learn programming and computational concepts. In this way, the analysis presented in this paper suggests a methodology for diagnosing other classes of errors and developing strategies for addressing them.

5. REFERENCES

[1] Selection sort. `http://en.wikipedia.org/wiki/Selection_sort` accessed from the web July 29, 2010.

[2] K. S. R. Anjaneyulu. Bug analysis of pascal programs. *SIGPLAN Not.*, 29(4):15–22, 1994.

[3] C. Daly. Roboprof and an introductory computer programming course. *SIGCSE Bull.*, 31(3):155–158, 1999.

[4] J. Davis and S. A. Rebelsky. Food-first computer science: starting the first course right with pb&j. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 372–376, New York, NY, USA, 2007. ACM.

[5] A. A. diSessa. Toward an epistemology of physics. *Cognition and Instruction*, 10:105–225, 1993.

[6] S. Garner, P. Haden, and A. Robins. My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems. In *ACE '05: Proceedings of the 7th Australasian conference on Computing education*, pages 173–180, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[7] M. J. Guzdial. How computing and physics learning differ. `http://computinged.wordpress.com/2010/04/01/how-computing-and-physics-l%earning-differ/`, 2010.

[8] S. Holland, R. Griffiths, and M. Woodman. Avoiding object misconceptions. *SIGCSE Bull.*, 29(1):131–134, 1997.

[9] M. Hristova, A. Misra, M. Rutter, and R. Mercuri. Identifying and correcting java programming errors for introductory computer science students. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 153–156, New York, NY, USA, 2003. ACM.

[10] G. Lakoff and M. Johnson. *Metaphors We Live By*. The University of Chicago Press, Chicago, IL, 1980.

[11] G. Lewandowski and A. Morehead. Computer science through the eyes of dead monkeys: learning styles and interaction in cs i. *SIGCSE Bull.*, 30(1):312–316, 1998.

[12] J. Noble, R. Biddle, and E. Tempero. Metaphor and metonymy in object-oriented design patterns. *Aust. Comput. Sci. Commun.*, 24(1):187–195, 2002.