To appear in *Computer Science Education*
Vol. 00, No. 00, Month 20XX, 1–28

Metonymy and Reference-point Errors in Novice Programming

Craig S. Miller*
School of Computing, DePaul University, 243 S. Wabash Ave., Chicago, IL 60604, USA

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

When learning to program, students often mistakenly refer to an element that is structurally related to the element that they intend to reference. For example, they may indicate the attribute of an object when their intention is to reference the whole object. This paper examines these reference-point errors through the context of metonymy. Metonymy is a rhetorical device where the speaker states a referent that is structurally related to the intended referent. For example, the following sentence states an office bureau but actually refers to a person working at the bureau: *The tourist asked the travel bureau for directions to the museum.* Drawing upon previous studies, I discuss how student reference errors may be consistent with the use of metonymy. In particular, I hypothesize that students are more likely to reference an identifying element even when a structurally related element is intended. I then present two experiments, which produce results consistent with this analysis. In both experiments, students are more likely to produce reference-point errors that involve identifying attributes than descriptive attributes. Given these results, I explore the possibility that students are relying on habits of communication rather than the mechanistic principles needed for successful programming. Finally I discuss teaching interventions using live examples and how metonymy may be presented to non-computing students as pedagogy for computational thinking.

Keywords: reference errors, 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.

*Author email: cmiller@cdm.depaul.edu.

- *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 (1980) make the case that metonymy is not just a rhetorical device but language usage that reflects everyday activity and thinking. Its ubiquity is well documented across many (human) languages (Panther & Radden, 1999). Moreover, Gibbs (1999) argues that speaking and thinking with metonymy "is a significant part of our everyday experience."

With its pervasive use in mind, metonymy may offer insight into student reference errors as they learn to program. 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 (1998), 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 (2007) 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, its mechanisms ostensibly underlie some of the difficulty students face when forming a precise, literal command.

While metonymy has been used for developing design patterns (Noble, Biddle, & Tempero, 2002), the goal of this paper is consider its insights into student mistakes when learning to program. Prior study has identified human language as a potential source of some student errors and misconceptions (e.g. Bonar & Soloway, 1985; Clancy, 2004; L. A. Miller, 1981). Many of these errors involve discrepancies between word meanings in human language and their use in programming. In this paper, the focus is on the *mechanisms* that underlie reference-point constructions in human language and their potential misapplication in programming. The working thesis is that the workings of metonymy provide insight into student difficulty when expressing the exact referent to a command or operation.

Before we further consider the role of metonymy, this paper reviews cases where novice programmers write code that refer to the wrong element. Surveying prior work, I introduce terminology that will be useful for analyzing these errors from a metonymic perspective. Focusing on object-attribute reference errors, I apply insight from metonymy to identify where reference-point errors are more likely to occur. Two experiments are then presented that produce results consistent with these predictions. Implications and recommendations are discussed. They include some direction for developing effective instructional strategies and using metonymy as a vehicle for introducing computational thinking to non-computing students.

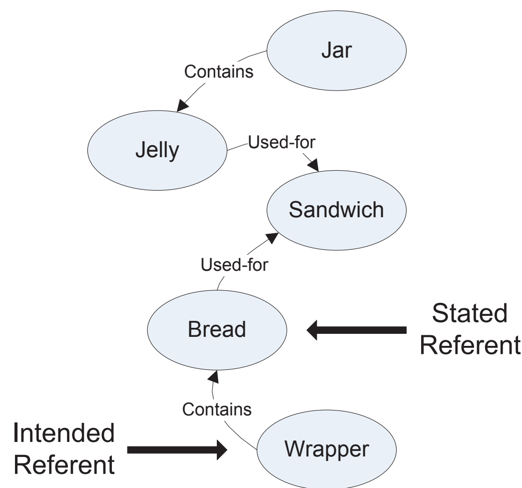

Figure 1. Semantic network demonstrating metonymy

2. Review of Reference-point errors

Reference-point errors involve incorrectly referring to an element that has a structural relationship with the intended element. 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.

Reference-point errors may occur in a variety of contexts. Previous reports have noted the commonality of array errors (Daly, 1999; Garner, Haden, & Robins, 2005). One potential difficulty involves the distinction between the index and the actual value located at the position denoted by the index. For example, a novice programmer may inadvertently add one to the value of an array element when the intention was to add one to the index in order to access the next value in the array.

Reference-point errors may also occur in the context of conflating memory references and the actual content at the reference. Hristova, Misra, Rutter, and Mercuri (2003) 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.

Reference-point errors may also involve entities outside of computer memory. In the context of file references, students frequently specify a file's location at the wrong level in the directory structure (C. S. Miller, Perković, & Settle, 2010). At the conceptual level, an earlier account using Pascal (Anjaneyulu, 1994) 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.

In the next section, I turn to reference-point errors involving objects and attributes, which will be the focus of the paper. Here I apply insights from the use of metonymy to understand the errors and identify where they are likely to occur.

2.1. *Object instances and attributes*

Holland, Griffiths, and Woodman (1997) 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:

```
temperatureField = TextField.new
temperatureField.id = "temperature"
temperatureField.size = 5
temperatureField.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 produces errors in programming statements:

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

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

# mistakenly refer to the id when changing
#    the width
temperature.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 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.

3. Implications and Predictions

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 mechanisms underlying metonymy also contribute 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:

(1) In place of an intended reference, humans may state a related referent.
(2) The stated referent has a structural relationship with the intended referent.
(3) The stated referent is useful for uniquely identifying the intended referent.

These insights are consistent with findings from cognitive linguistics, where part-whole relationships are a common source for the use of metonymy (Langacker, 1993). Moreover, they coincide with findings that metonymy is more likely to occur when the speaker wants to emphasize a particular property (Radden & Kövecses, 1999). These findings thus make predictions about what to expect for when students construct references in the context of programming. 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.

4. Overview of Empirical Studies

This section introduces two experiments that explore whether students are more likely to mistakenly refer to an identifying attribute (e.g. name or title) in place of the object itself. Both experiments refer to two types of attributes: identifying attributes (e.g. name, title) and descriptive attributes (e.g. color, size). As already discussed, identifying attributes are characterized by their ability to uniquely name some entity, often within some context. Descriptive attributes may characterize the entity but are generally not useful for uniquely identifying one within a set. Here the attribute **type**, whether identifying or descriptive, is a product of human language and should not be confused with type (e.g. integer, character) used in the context of programming languages.

The context of use for the two experiments is database access using an Object-Relational Mapping (ORM). This context provides a natural use of objects and attributes for server-side web development. A previous study of student errors in this context has already been reported (C. S. Miller, 2012). Results from this study revealed students making errors consistent with metonymic use. Moreover, errors

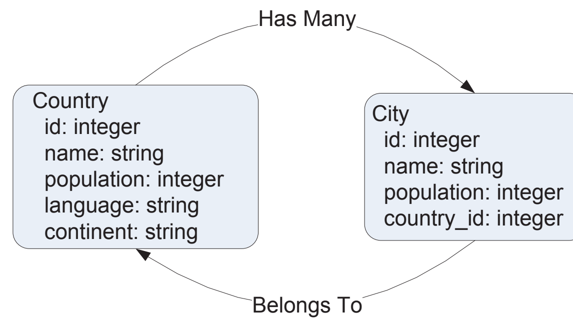

Figure 2. Example models with Rails' relations.

were most prevalent for object-identifying attributes (e.g. name) than descriptive attributes (e.g. size). In this paper, the predicted use is more rigorously tested by controlling for presentation order and structure. Before presenting the studies, ORM access is reviewed in the context of metonymy.

4.1. *Metonymy and Object-relational Mapping*

The experiments for this paper were conducted within several web development courses. In these courses, students access a database through the Active Record ORM, used in the Ruby on Rails web development framework (Bachle & Kirchberg, 2007). A previous article (C. S. Miller, 2012) discusses the choice of this framework and its ORM. Here we review some examples of ORM access in order to see how students might mistakenly refer to objects and their attributes. Examples are based on two schema models presented in Figure 2.

The Country model and the City model suffice to explore student errors and assess the role of metonymy for this study. In addition to a few attributes (e.g. name, population, language), both models have a primary key (id), which Active Record provides by default. Also note that city has a foreign key (country_id). This foreign key indicates a relationship between the two models. Using Active Record terminology, we say that a City *belongs to* a Country and that a Country *has many* Cities.

Once the models are specified and the relationships declared, Active Record provides object-oriented access to the database by automatically defining classes and methods for the models. Class methods selectively query records from the database:

```
france_country = Country.find_by_name('France')
london_city  = City.find_by_name('London')

all_city_objects = City.all
some_objects = City.where('population > 1000')
```

The assigned variable names document what the methods return. In the examples above, the Ruby on Rails framework dynamically defines methods such as `find_by_name`, which are based on the model's attributes.

Given an object (record) from the database, instance methods can be called with them:

```
france_city_objects = france_country.cities
```

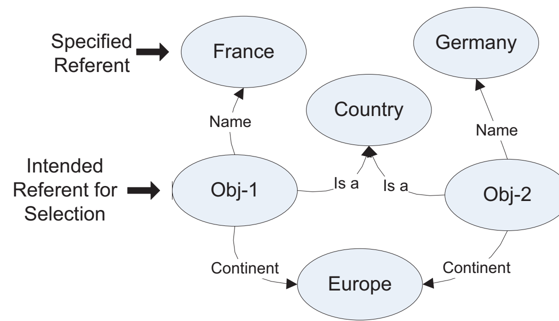

Figure 3. Reference error.

```
uk_country_object = london_city.country
```

Finally, attribute values can be assigned to an object, which can then be saved to the database:

```
new_country = Country.new
new_country.name = 'Outland'
new_country.population = 12000
new_country.save
```

Even though Active Record simplifies database access using object-based notation, student developers are still required to correctly state the intended referent. For example, if the goal is to list a country, more precisely, the name of the country, the reference must explicitly reference the name attribute with the country object:

```
new_country.name
```

However, given the prevalence of metonymy in human languages, students may be inclined to simply indicate the object:

```
new_country
```

Other circumstances may call for simply referencing the object (representing the entire database record), perhaps to gain access to other attributes associated with it. Considering the use of metonymy, students may be inclined to simply indicate the name (a string) since it is the name that clearly distinguishes it from other country objects. Figure 3 provides a working example. As the figure indicates, `Obj-1` and `Obj-2` are usefully identified by their 'name' attribute. However, simply providing the name attribute will not be sufficient if the application requires a reference to the object.

The next section reviews the actual problems used in the first experiment for exploring reference errors and where they are likely to occur.

5. Experiment 1

5.1. *Study Problems and Predictions*

A short exercise with ORM problems was developed to assess student ability to access database content using the Rails ORM, Active Record. The exercise presented the schema shown in Figure 2. The exercise also presented a few working exam-

ples similar to those presented in the previous section. Because students needed to volunteer to participate in the study, the length of the exercise was limited to a minimal number of problems in order to encourage participation and completion of all problems. The first two problems (presented with answers below) ask students to provide expressions that require an explicit reference to the **name** property:

(1) The country name with id 5.
Answer: Country.find(5).name
(2) The population of the city of Toronto.
Answer: City.find_by_name('Toronto').population

While there are many correct answers for each problem, all correct answers require an explicit reference to the **name** property. However, as previously discussed, students may (mistakenly) just reference the object. For example, for the first problem, they may omit the name reference and thus just provide the object: *Country.find(5)*. For the second problem, they may mistakenly omit the name reference when matching the city name: *City.Toronto.population*. Consistent with the use of metonymy, an identifying attribute (e.g. name) can be interchanged with the referent itself.

The last two problems in the exercise were expressly developed to explore how students specify referents. Both problems ask students to delete an object from an array:

(3) Provide ruby code that uses the remove method to delete the country named 'France' from the country_list array.
Answer:
country_obj = Country.find_by_name('France')
country_list.remove(country_obj)
(4) Provide ruby code that uses the remove method to delete the country whose language is "Norwegian" from the country_list array.
Answer:
country_obj = Country.find_by_language('Norwegian')
country_list.remove(country_obj)

These two questions have similar structure in that both contain explicit identifying information in the same order: object, attribute and then value. Moreover, the phrasing is natural for both attributes. Using the exact sentence structure for both questions was considered (e.g. "...to delete the country whose name is 'France' from...") but was rejected out of the concern that the identical wording would prime the answer for the next question. The discussion and the second experiment addresses whether the difference in question structure affects student answers.

The *answers* to both problems are structurally identical. However, the first requires use of the name attribute and the second requires the use of the language attribute. Since the name attribute is commonly used to identify the object, we hypothesize that it is more likely to be used in place of the object itself.

5.2. *Method*

The four ORM problems described in the previous section were assembled as an exercise quiz. In order to mitigate order effects on the last two problems (Problem 3 and Problem 4), two versions were prepared. One version had the name-based

problem appearing before the language-based problem. A second version had the language-based problem appearing before the name-based problem. The appendix provides the actual instructions with examples (Appendix A) and the set of exercise questions (Appendix B).

The ORM problems were offered as an exercise in two classes of a web development course at a large university. The two versions were distributed so that each version was received by an equal number of students. While the exercise was offered to all students, students actively chose to participate in the study by submitting their answers in a sealed envelope, which were then turned over to the study's investigator. Sixteen students (15 male, 1 female) submitted their answers for analysis. The average age was 21 and the average number of prior programming courses was 4. Since the prerequisite to this course included web development courses covering HTML and CSS, it is likely that these courses were included in the count of programming courses.

5.3. *Results*

For the first problem, a correct answer consists of obtaining the object (id = 5) and then referencing its name attribute. Almost all students (15 of 16) produced expressions for accessing the object—of varying degree of correctness—but only two students referenced the name attribute. Of these two answers only one answer was fully correct:

```
temp = Country.find_by_id(5)
temp.name
```

Another answer referenced the name attribute but did not form a correct expression:

```
Country.where = Country.name(5)
```

Some answers (5) successfully accessed the object[1] but did not then obtain the requested attribute. Here is one example:

```
Country.find(5)
```

Other answers (8) used incorrect methods but with the clear intention of accessing the object by its id, again failing to reference the requested attribute. Examples include:

```
Country.country_id(5)

Country.find_by_country_id(5)
```

For the second problem, a correct answer consists of obtaining an object by explicitly specifying the name attribute *and* its value, followed by the reference to the population attribute. Six students produced expressions that correctly referenced the object and the specified population attribute. Here is one example:

```
City.find_by_name("Toronto").population
```

[1]For Active Record, an object may be retrieved by its id using either the `find` method or the dynamically defined `find_by_id` method.

Table 1. Frequency counts for last two problems

		Remove by Language	
		No attribute	Explicit attribute
Remove by Name	No attribute value	2	8
	Explicit attribute	0	4

The most common error (7 cases) involved efforts to obtain the correct object without specifically indicating that it must match the `name` attribute. Incorrect examples belonging to this case include:

```
city_toronto.city.population
```

```
County.population(Toronto)
```

Only one student obtained the correct object without resolving to the required population attribute:

```
City.find_by_name('Toronto')
```

The last two problems ask students to provide an expression that removes an object from an array by specifying its name attribute in one case (Problem RN) and specifying its language attribute in another case (Problem RL). Half the students were presented Problem RN and then RL as the third and fourth problems. The other half were presented Problem RL and then RN.

For both answers, a correct answer requires obtaining the object by explicitly referencing the attribute. Student answers for this case include:

```
country_list.remove(Country.find_by_name('France'))
```

```
c = Country.find_by_language('Norwegian')
country_list.remove(c)
```

Student examples that incorrectly omitted the explicit attribute include:

```
country_list.remove('France')
```

```
country_list.remove('Norwegian')
```

Fourteen of the sixteen students provided answers to both problems. These answers were categorized and counted by whether the attribute was explicitly referenced in order to obtain the whole object. Table 1 presents the frequencies of how each student responded. For example, two students (upper-left cell) specified just the value ("France" for RN and "Norwegian" for RL) while omitting the attribute (`name` for RN and `language` for RL) for both problems. Eight students (upper-right cell) correctly referenced the attribute for problem RL while omitting it for problem RN. In contrast, no student correctly referenced the attribute for problem RN while omitting it for problem RL.

To test for a significant pair-wise difference between the Remove-by-name (RN) problem and the Remove-by-language (RL) problem, an answer was scored as a 1 if it included an explicit reference to the required attribute (`name` for RN and `language` for RL) and 0 if the attribute is omitted. The non-parametric sign test on the pair-wise difference produces a significant two-tailed p-value of 0.0078.

5.4. *Discussion*

Answers to all four problems produced results that are consistent with the metonymy-based analysis. That is, students are more likely to use identifying attributes in place of whole objects, and vice versa, than if the attributes are merely descriptive. In problem 1, almost all students produced the whole object without resolving to the identifying name attribute (Problem 1). In contrast, only one student produced just the whole object without some reference to the population attribute. These findings replicate the qualitative findings from the previous study with similar problems (C. S. Miller, 2012). Moreover this present study rules out effects due to presentation order by reversing the order of the last two questions for half of the participants.

Returning to the insights of the preceding section, the study's results demonstrate each of them:

(1) Results produce reference substitutions: most student responses involved offering a reference that was different than the correct reference.
(2) Observed substitutions commonly follow part-whole structural relationships: most substitutions involve providing a reference to the whole object (e.g. country object) in place of the attribute (e.g. name) and vice versa.
(3) Substitutions are more common with identifying attributes: substitutions were significantly more common using the identifying `name` attribute than the descriptive `language` attribute.

Perhaps most compelling is the finding that none of the participants fully referenced the name attribute while also omitting the language attribute. Consistent with findings from cognitive linguistics, constructions involving salient or identifying attributes are strong candidates for metonymic use.

While this first experiment provides some evidence that the kind of attribute contributes to reference-point errors, it leaves open the possibility that the wording structure in the question is the primary contributor. In particular, it is possible that the phrase "the country named France" invokes the reference error, but the phrase "the country whose name is France" would less likely invoke the error. The next study explores this possibility by controlling question structure and varying attribute type across comparable questions. At the same time, the next study explores a broader range of reference-point errors with a larger sample from multiple web development courses.

6. Experiment 2

The first experiment demonstrated a variety of reference-point errors consistent with the use of metonymy. It revealed students producing an object when the question asked for a naming attribute. It also revealed cases where students are more likely to reference just the attribute when the whole object was required. However, the first experiment was limited in several ways. It did not systematically vary the type of attribute over a range of questions, nor did it fully control for word structure. Finally, the first experiment involved a relatively small sample from one web development course.

This second study addresses the limitations of the first experiment. It makes use of two new data models: Part and Manual. Both models have identifying at-

tributes (name for the Part model and title for the Manual model) and descriptive attributes (e.g. color, weight, level). Based on these models, six new questions were developed. Since none of these new questions involve relations between models, the introductory examples were further simplified. This change permitted the inclusion of students from less advanced courses.

The first two questions were similar to the first two questions in the preliminary study in that the question asked for an attribute, which must be obtained from the object:

(1) The part weight with id 205.
Answer: Part.find(205).weight
(2) The name of the part whose tag is 'm108'.
Answer: Part.find_by_tag('m108').name

Note that the first question resolves to a descriptive attribute and the second question resolves to an identifying attribute. While both questions involve find operations, the focus of analysis is whether students will add the explicit attribute reference (i.e. weight or name) at the end of their expression. In order to control for effects of word structure, a second set of questions was created, which uses identical language except the two attributes of interest were swapped:

(1) The part name with id 205.
Answer: Part.find(205).name
(2) The weight of the part whose tag is 'm108'.
Answer: Part.find_by_tag('m108').weight

Randomly assigning the two versions to students allows us to explore the effect of the attribute while controlling for all other language in the question. A metonymy-based interpretation predicts that students are more likely to omit the identifying attribute (i.e. name) than the descriptive attribute (i.e. weight).

Four additional questions were created that require object references derived from the attribute. Again, both identifying attributes (i.e. name and title) and descriptive attributes (i.e. color and level) were used. Like the first experiment, two of these questions involved deleting an object from an array. This time, however, two different word structures were deliberately selected:

(3) Provide ruby code that uses the remove method to delete the 'wheel' part from the parts_list array.
Answer:
part_obj = Part.find_by_name('wheel')
parts_list.remove(part_obj)
(4) Provide ruby code that uses the remove method to delete the manual whose title is "Model Painting" from the manuals_list array.
Answer:
manual_obj = Manual.find_by_title('Model Painting')
manuals_list.remove(manual_obj)

To successfully answer the question 3, the student must infer the attribute name and include it in the answer. Question 4 explicitly includes all required elements. In this presented set, the stated elements are identifying attributes (i.e. name and title). In the alternate version, questions 3 and 4 use descriptive attributes (i.e. color and level) as will be presented for questions 5 and 6.

Finally, questions 5 and 6 introduce a display method that also requires an object reference derived from an attribute:

(5) Provide ruby code that uses the display method to display the 'green' part.
Answer:
part_obj = Part.find_by_color('green')
part_obj.display

(6) Provide ruby code that uses display method to display the manual whose level is "advanced".
Answer:
manual_obj = Manual.find_by_level('advanced')
manual_obj.display

In this presented set, the stated elements are descriptive attributes (i.e. color and level). In the alternate version, questions 5 and 6 use identifying attributes (i.e. name and title).

Note that the six questions from each set uses different wording so that a student answer is less likely to be influenced by previous questions. At the same time, identical language, except for the swap of identifying and descriptive attributes, is used *across* the two sets of questions.

The interpretation based on metonymy predicts that references involving descriptive attributes are more likely to be correctly stated than those based on identifying attributes. For questions 1 and 2, the prediction is that students are more likely to add the descriptive attribute to the object than the identifying attribute. For questions 3, 4, 5 and 6, the prediction is that students are more likely to retrieve an object based on the explicit mention of the attribute if a descriptive attribute is involved than an identifying object. Note that for these last four questions, the difference between a correct response and the predicted incorrect response includes both an object reference (with a find method) and the attribute. With that difference in mind, analysis of student answers will include noting both object reference and attribute reference. For comparison, a third coding category will note additional errors in the student response.

6.1. *Method*

The two sets of the six ORM questions described in the previous section were assembled as an exercise quiz. The appendix provides the actual instructions with examples (Appendix C) and both sets of the exercise questions (Appendix D).

The ORM problems were offered as an exercise in three web development courses at a large university. One course (IT 231) had three participating sections. IT 231 is an introductory server-side course that has one client-side course as a prerequisite. The second course (IT 232) had one participating section. IT 232 has a general programming course and IT 231 as prerequisites. The third course (IT 432) is a server-side web development course targeted for graduate students who are not programmers. IT 432 has one client-side programming course as a prerequisite. All courses used Ruby on Rails and the Active Record ORM for the coursework.

The two versions of the exercise were randomly assigned by alternately distributing them to individual students. While the exercise was offered to all students, students actively chose to participate in the study by submitting their answers in a sealed envelope, which were then turned over to the study's investigator. Forty

Table 2. Proportions for each Coding Category

Question Number	Attribute Type	N	Attribute Present	Object Present	Otherwise Correct
1	Descriptive	19	0.63	0.95	0.63
	Identifying	21	0.33	0.95	0.81
2	Descriptive	20	0.90	0.95	0.40
	Identifying	17	0.41	0.88	0.47
3	Descriptive	18	0.17	0.22	0.89
	Identifying	18	0.06	0.06	0.72
4	Descriptive	18	0.33	0.39	0.78
	Identifying	18	0.11	0.06	0.72
5	Descriptive	16	0.38	0.88	0.69
	Identifying	18	0.17	0.83	0.89
6	Descriptive	15	0.47	0.60	0.60
	Identifying	18	0.33	0.72	0.72

students (34 male, 5 female, 1 unspecified) submitted answers for analysis. The average age was 24 and the average number of prior programming courses was 3. Since the prerequisites to the courses included web development courses covering HTML and CSS, it is likely that these courses were included in the count of programming courses.

6.2. *Results*

The 40 response sheets produced 216 responses and included sheets with only some of the questions answered. All 216 responses were evaluated and coded using the following definitions:

Attribute present Coded as one if the experimental attribute is present anywhere in the student answer. Otherwise, coded as zero.

Object present Coded as one if the student answer includes evidence of object use. The object must ostensibly represent the whole database record and not just an attribute (which technically are objects too). Evidence includes use of a **find** method or an object-like reference (e.g. using the class name) followed by dot (e.g. **green_obj.display** or **green_part.display**). Otherwise (e.g. **green.display**), coded as zero.

Otherwise correct Coded as one if the answer was otherwise correct, not including the above errors. Also, errors involving quotations and capitalization were not considered. Presence of any other errors warranted a coding of zero.

The 216 responses were independently coded by the author and one other evaluator. The resulting codes were then compared revealing an agreement on 202 (94%) of the attribute-present code, 198 (92%) object-present code and 179 (83%) otherwise correct code. The student responses for differing codes were reexamined and the differences resolved by correcting for obvious evaluator errors or by further clarifying the coding rules. Table 2 shows the resulting proportions for the three coding categories, broken down by question number and attribute type.

The next subsections review the results in detail by coding category. For each coding category, I first examine the within-subject effect of attribute (descriptive vs. identifying) for all complete sets of answers (N=29). The net effect for each

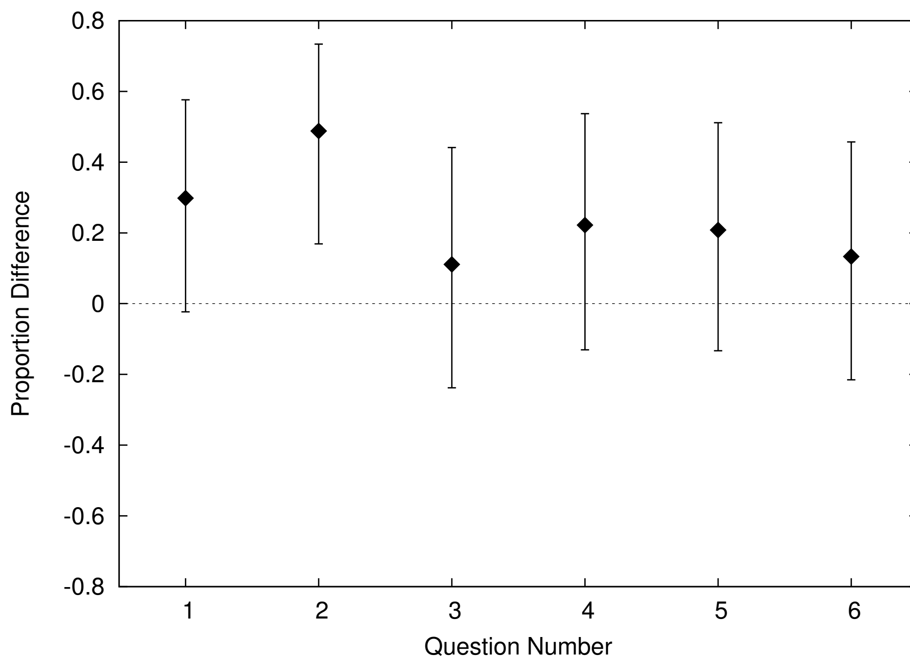

Figure 4. Proportion differences for attribute-present coding.

student participant is obtained by averaging their respective codes, where descriptive attribute questions are scored positively and identifying questions are scored negatively. Consequently, the net effect per participant ranges from -1 to 1 with 7 possible values. Assuming approximate normality in the distribution, a t-test can then be applied to determine if there is a significant net effect across all students with complete answer sets. Finally between-subject analyses will show the effects of attribute type for each individual question. These between-subject analyses draw upon all 216 responses.

6.2.1. Attribute-present results

The mean net difference between attribute types on the presence of the attribute in the answer was 0.12, meaning that questions with the descriptive attribute more frequently produced answers with the varying attribute present. A two-tailed t-test reveals a significant net effect, $t(28) = 3.55$, $p = 0.0014$. Between-subjects effects for each question are graphically depicted in Figure 4. The plotted points indicate the proportion differences between the questions with descriptive attributes and those with identifying attributes. Points above the zero line indicate that the attribute was more frequently present in the answer for questions with descriptive attributes than for those with identifying answers. The limiting bars depict exact 95% confidence intervals for the differences.

6.2.2. Object-present results

The mean net difference between attribute types on the presence of the object in the answer was 0.03, meaning that questions with the descriptive attribute (slightly) more frequently produced answers showing evidence of the object. A two-tailed t-test does not reveal a significant net effect, $t(28) = 0.50$, $p = 0.62$.

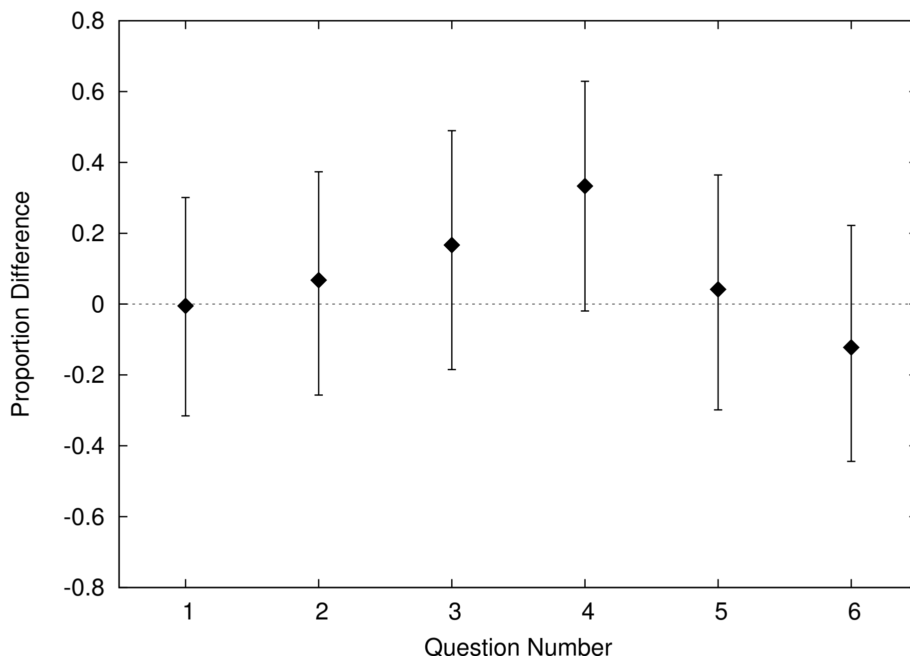

Figure 5. Proportion differences for object-present coding.

Between-subjects effects for each question are graphically depicted in Figure 5. The plotted points indicate the proportion differences between the questions with descriptive attributes and those with identifying attributes. Points above the zero line indicate that the object was more frequently present in the answer for questions with descriptive attributes than for those with identifying answers. The limiting bars depict exact 95% confidence intervals for the differences.

6.2.3. *Otherwise correct results*

The mean net difference between attribute types on the otherwise correctness of the answer was -0.03, meaning that questions with the identifying attribute (slightly) more frequently produced answers that were otherwise correct (ignoring errors missing attribute and object). A two-tailed t-test does not reveal a significant net effect, $t(28) = -0.96$, $p = 0.34$. Between-subjects effects for each question are graphically depicted in Figure 6. The plotted points indicate the proportion differences between the questions with descriptive attributes and those with identifying attributes. Points above the zero line indicate that students more frequently produced answers that were otherwise correct for questions with descriptive attributes than for those with identifying answers. The limiting bars depict exact 95% confidence intervals for the differences.

6.3. *Discussion*

The overall net effect of attribute type on the presence of the varying attribute was reliably consistent with the metonymy-based prediction. In particular, when working with identifying attributes, students were more likely to reference just the object when asked for the attribute and more likely to omit the attribute name when the problem called for the object. Consistent with the use of metonymy,

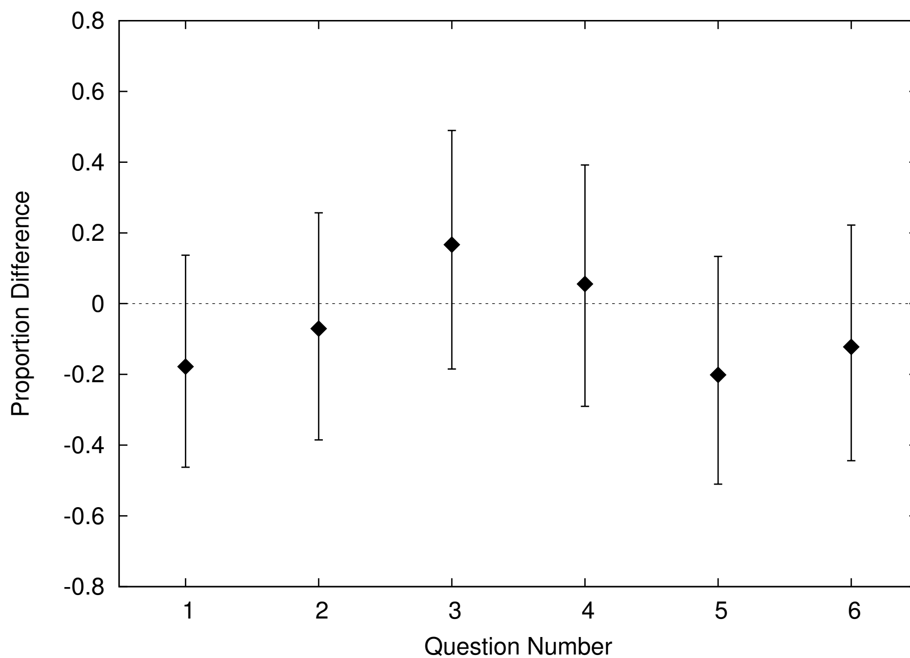

Figure 6. Proportion differences for otherwise-correct coding.

students substituted object for identifying attribute, and vice versa, in favor of the simpler expression. Because other possibly contributing factors were effectively counter-balanced across two sets of questions, the statistically significant effect provides strong evidence that just varying the attribute affects student answers.

A potential concern is that the effect from varying the attribute is not caused by its type (i.e. identifying vs. descriptive) but by some other characterizing difference among the attributes. Perhaps most notable is the presence of the weight attribute as one of the examples in the instructional materials (presented in Appendix C). Seeing its use there might prompt more students to attach it to the object than for the comparable question with the name attribute. To some extent, the potential effect from the example is mitigated by several differences in how it appears in the example and how it appears in questions 1 and 2. First, the English description in the example differs from the wording in the questions. Second, the example expression just uses an object, but the questions require an object retrieval and then a reference to the attribute. Nevertheless, the attribute's presence in the example may account for an increased presence in questions 1 and 2. To isolate possible interference from the example, an additional analysis was conducted with just the last four questions, none of which used attributes in the examples on the instructional page. Subtracting average codes for identifying-attribute questions from those of descriptive-attribute questions produces a range of 5 possible values for each of the 32 students who completed all four questions. Given the reduced range of possible values, the non-parametric Wilcoxon ranked sign test was applied since it makes no assumptions on distribution. This test produces a two-tailed p-value of 0.048. This significant difference provides more confidence that the attribute type is influencing student responses. In any case, more conclusive evidence would come with additional studies involving a range of identifying and descriptive attributes in otherwise controlled questions.

More nuanced is the effect on presence of object. As already noted, questions

1 and 2 do not pertain to the metonymy-based interpretation since the object retrieval does not involve the manipulated attribute. Consequently no effect is predicted. Questions 3 and 4 are pertinent and indeed they produced results consistent with the interpretation's prediction. Most notable is the difference for question 4, whose confidence interval nearly clears the zero mark. Moreover, averaging the proportions for questions 3 and 4 reveals a significant two-tailed difference using a Wilcoxon two-sample test ($p = 0.030$).

In contrast to questions 3 and 4, questions 5 and 6 do not produce the predicted effect on presence of object. A closer examination of these questions offers an explanation. Both correct answers require placement of an object before the dot and method name:

```
example_object.display
```

An answer without an object is conspicuously missing:

```
.display
```

The prominence of the object may override the effect of attribute type and thus explain why no reliable object-present differences were produced for questions 5 and 6.

The results revealed no net effect of attribute type on other kinds of errors. This finding is useful in that it shows how the metonymy-based interpretation effectively isolates the kinds of errors that result by varying the attribute in terms of its identifying qualities. Most useful is the observation that the presence of the weight attribute in the examples did not produce more success in this category for questions 1 and 2. In fact, students did worse (albeit not statistically reliably so) for the questions involving the weight attribute than those involving the identifying name attribute.

Given the goal of isolating the effect of attribute type, the experiment was not designed to explore other factors such as sentence structure and question order. Nevertheless some results indicate a likely effect based on the wording structure in the question. Table 2 shows that, for descriptive attributes, more answers (correctly) included the object for question 4 (39%) than for question 3 (22%). One set of student responses illustrates this difference. For question 3, where the attribute name (i.e. color) must be inferred, the student provided the following answer:

```
part_list.remove('green')
```

Question 4 explicitly names the attribute (i.e. level) in the question. The student provided the following answer for this question:

```
part_list.remove(manual_list.find_by_level('advanced'))
```

In the margin, the student had underlined 'level' in the question and wrote, "This word is throwing me off." A plausible interpretation is that the appearance of the attribute name in question 4 prompted the student to include a **find** method and the attribute name in the answer (however note that the answer has other mistakes). While the attribute's effect in the question is compelling, the experiment did not control for question order, nor for the attributes used in the questions. It is possible that a second question prompted additional thinking or that choice of descriptive attribute (e.g. color or level) have their own effects. Additional study would be required in order to conclusively identify wording structure as a con-

tributing factor to reference-point errors.

7. General Discussion

The experiments provide evidence that an attribute's identifying trait, as used with human language and communication, contributes to reference-point errors. Previous work has also identified student programming errors that carry over from human language and has characterized them as misconceptions (Clancy, 2004). While insight from metonymy usefully predicts the errors in this paper, its status as a misconception is open to debate. This section explores several possibilities.

Students arguably do not have a coherent theory of metonymy and explicitly apply it when writing a programming statement. More plausibly, students may 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. For this interpretation, 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 may be based on their previous experience with language.

A useful comparison for misapplied experience includes accounts of how students acquire physics knowledge. diSessa (1993) 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). With this interpretation, 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 different 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 identifying attributes may have nevertheless induced a habit of communication, which is effective for inter-personal conversation but incorrect for human-to-computer commands. Even in these cases, domain knowledge may play a role in how students construct a referent in their code. A student's mental model of domain entities contain the connecting relationships for indicating an intended referent. These mental models correspond to what Lakoff (1987) calls idealized cognitive models (ICMs). As we saw, hypothesizing a student's ICM of a country's attributes and relationships, allowed us to predict likely reference errors when students construct an expression. An interesting question for future study is whether students are less likely to make reference mistakes on abstract structures that do not correspond to real-world entities. On one hand,

an abstract structure with no corresponding real-world ICM may prevent referent substitutions. On the other hand, lack of supporting real-world knowledge might incur additional errors.

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 using identifying attributes in place of the whole object, the metonymy-based analysis demonstrates the utility of examining how prior experiences may negatively influence how students learn computational concepts.

8. Future Directions

The role of metonymy in student errors suggests several directions for education research and instructional strategies. For assessing student achievement, a metonymy-based analysis may be useful for constructing diagnostic problems 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. This analysis may then aid further development of assessment instruments. Moreover, given that reference-point construction errors are not unique to any particular language, further study may be particularly useful for constructing language-independent assessments such as the assessment developed by Tew and Guzdial (2011).

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.

While the experiments presented students with some examples, the examples included minimal explanation. Nor is it clear that students took time to study the examples before completing the exercise since they were already familiar with the content. Consequently students may benefit by seeing worked examples after introducing the concept of metonymy to students. While problem solving is an important end-goal for computing disciplines (see Robins, Rountree, & Rountree, 2003 for a discussion of its role in programming), previous research indicates that novices greatly benefit from seeing worked examples in the early stages of learning. Extensive studies across a broad range of technical disciplines demonstrate advantages to presenting worked examples (see Kirschner, Sweller, & Clark, 2006 for a review). More specific to computing and reference construction, a study on references shows the relative advantage of studying examples in place of working practice problems (C. S. Miller & Settle, 2011).

Showing worked examples includes the practice of live-coding demonstrations

endorsed by Robins et al. (2003). Teaching metonymy and the pitfalls of reference errors are particularly amenable to instruction based on these live demonstrations. Working a live example gives the instructor the opportunity to show the correct reference-point construction and contrast it with the seemingly natural—yet incorrect—metonymic construction. In this sense, careful instruction of reference-point constructions, here supported with everyday examples, addresses a call to teach students general purpose strategies that promote their development as what Robins et al. (2003) characterize as "effective novices." In this way, these live examples demonstrate habits that serve students in a broad range of contexts.

Teaching metonymy in the context of programming also provides another opportunity to promote computational thinking to general audiences. Computational thinking is a collection of principles and mental habits from the computing disciplines that have general application for non-computing students (Guzdial, 2008; Wing, 2006). Lately its pedagogy has been further developed for other disciplines (Goldberg, Grunwald, Lewis, Feld, & Hug, 2012; Perković, Settle, Hwang, & Jones, 2010). The analysis of metonymy and its contrast to human-computer communication provides yet additional content and examples for cross-disciplinary coursework whose topics involve communication, cognition and language. Not only does it expose students to some of the intricacies of human communication, using computational constructs no less, but it also illustrates limitations of computer languages.

Finally, while the empirical studies of this paper focused on object-attribute reference errors, a metonymy-based analysis may provide useful insight to a broader range of reference errors, including those surveyed at the beginning of this paper. Promising candidates include array errors and hash reference errors. If it is possible to manipulate the identifying qualities of values, indices or keys, an analysis may reveal when reference-point errors for these constructs are likely to occur. If successful, the analysis presented in this paper provides some direction for broadening its coverage.

Acknowledgements

I gratefully acknowledge John Lalor for serving as the second data coder of the experiment 2 results. In addition, both he and Lauren Lucchese pilot-tested questions for the study. A special thanks goes to Ed Allemand, James DeBettencourt, Laura McFall, Kumail Razvi and John Rogers, who helped me conduct the experiments in their classes. Finally I thank the three anonymous reviewers, whose detailed comments greatly contributed to the final version of this article.

References

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

Bachle, M., & Kirchberg, P. (2007). Ruby on Rails. *Software, IEEE*, *24*(6), 105 -108.

Bonar, J., & Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction*, *1*(2), 133–161.

Clancy, M. (2004). Misconceptions and attitudes that interfere with learning to program. In S. Fincher & M. Petre (Eds.), *Computer science education research* (pp. 85–100). Taylor and Francis Group, London.

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

Davis, J., & Rebelsky, S. A. (2007). 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* (pp. 372–376). New York, NY, USA: ACM.

diSessa, A. A. (1993). Toward an epistemology of physics. *Cognition and Instruction*, *10*, 105-225.

Garner, S., Haden, P., & Robins, A. (2005). 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* (pp. 173–180). Darlinghurst, Australia, Australia: Australian Computer Society, Inc.

Gibbs, R. W., Jr. (1999). Speaking and thinking with metonymy. In K.-U. Panther & G. Radden (Eds.), *Metonymy in language and thought* (pp. 61–76). Amsterdam: John Benjamins.

Goldberg, D. S., Grunwald, D., Lewis, C., Feld, J. A., & Hug, S. (2012). Engaging computer science in traditional education: the ECSITE project. In *Proceedings of the 17th acm annual conference on innovation and technology in computer science education* (pp. 351–356). New York, NY, USA: ACM.

Guzdial, M. (2008, August). Education: Paving the way for computational thinking. *Commun. ACM*, *51*(8), 25–27.

Holland, S., Griffiths, R., & Woodman, M. (1997). Avoiding object misconceptions. *SIGCSE Bull.*, *29*(1), 131–134.

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). 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* (pp. 153–156). New York, NY, USA: ACM.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, *41*(2), 75–86.

Lakoff, G. (1987). *Women, fire, and dangerous things: What categories reveal about the mind*. Cambridge Univ Press.

Lakoff, G., & Johnson, M. (1980). *Metaphors we live by.* Chicago, IL: The University of Chicago Press.

Langacker, R. W. (1993). Reference-point constructions. *Cognitive Linguistics*, *4*(1), 1–38.

Lewandowski, G., & Morehead, A. (1998). Computer science through the eyes of dead monkeys: learning styles and interaction in CS I. *SIGCSE Bull.*, *30*(1), 312–316.

Miller, C. S. (2012). Metonymic errors in a web development course. In *Proceedings of the 13th annual conference on information technology education* (pp. 65–70). New York, NY, USA: ACM.

Miller, C. S., Perković, L., & Settle, A. (2010). File references, trees, and computational thinking. In *Proceedings of the fifteenth annual conference on innovation and technology in computer science education* (pp. 132–136).

Miller, C. S., & Settle, A. (2011). When practice doesn't make perfect: Effects of task goals on learning computing concepts. *ACM Transactions on Computing Education (TOCE)*, *11*(4), 22.

Miller, L. A. (1981). Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, *20*(2), 184–215.

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

Panther, K.-U., & Radden, G. (1999). Introduction. In K.-U. Panther & G. Radden (Eds.), *Metonymy in language and thought* (pp. 1–14). Amsterdam: John Benjamins.

Perković, L., Settle, A., Hwang, S., & Jones, J. (2010). A framework for computational

thinking across the curriculum. In *Proceedings of the fifteenth annual conference on innovation and technology in computer science education* (pp. 123–127). New York, NY, USA: ACM.

Radden, G., & Kövecses, Z. (1999). Towards a theory of metonymy. In K.-U. Panther & G. Radden (Eds.), *Metonymy in language and thought* (pp. 17–60). Amsterdam: John Benjamins.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Tew, A. E., & Guzdial, M. (2011). The FCS1: a language independent assessment of CS1 knowledge. In *Proceedings of the 42nd acm technical symposium on computer science education* (pp. 111–116). New York, NY, USA: ACM.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Appendix A: Exercise instructions and examples for Experiment 1

Exercise for Models and Relationships

A Country model and a City model have been created with the following attributes:

Country	City
name: string	name: string
population: integer	population: integer
language: string	
continent: string	country_id: integer

The following relationships have been declared:

A city belongs to a country.

A country has many cities.

The next page asks you to provide ruby expressions and statements. Below are some examples that you might find useful as a reference.

Assume the following assignments:

```
france_country = Country.find_by_name('France')
london_city   = City.find_by_name('London')
```

Note what the following ruby expressions produce:

`City.all` – produces an array of all City objects

`Country.where('population > 1000')` – produces an array of country objects with population greater than 1000

`france_country.cities` – produces an array of City objects that belong to france_country

`london_city.country` – produces the Country object

These statements write out the languages for each country:

```
country_list = Country.all
country_list.each do |c|
   puts c.languages
end
```

Appendix B: Exercise Problems for Experiment 1

The following problems were those used for Experiment 1. In one version, the last two problems were presented in the order below. In the second version, the last two problems were presented in the opposite order.

For the following items, provide the ruby code that produces the following:

(1) The country name with id 5.
(2) The population of the city of Toronto.

Ruby has a remove method that allows an object to be deleted from an array of objects. For example if `word_list` has the contents `['cat', 'dog', 'cow', 'horse']`, then `word_list.remove('dog')` would delete the string 'dog' from the array. That is, the contents of `word_list` would then be `['cat', 'cow', 'horse']`.
For the next two questions, assume that country_list is an array of Country objects.

(3) Provide ruby code that uses the remove method to delete the country named 'France' from the country_list array.
(4) Provide ruby code that uses the remove method to delete the country whose language is "Norwegian" from the country_list array.

Appendix C: Exercise instructions and examples for Experiment 2

Database and Models Exercise

A Part model and a Manual model have been created with the following attributes:

Part
- name: string
- color: string
- tag: string
- weight: integer

Manual
- title: string
- pages: integer
- format: string
- level: string

Note that the Rails ActiveRecord framework automatically assigns an integer **id** to each record.

Below are some example objects belonging to the Part model:

id	name	color	tag	weight
201	hood	red	x45	12
202	wheel	black	y65	5
203	door	green	z56	14

Below are some example objects belonging to the Manual model:

id	title	pages	format	level
501	Building Models	12	pamphlet	novice
502	Model Painting	76	paperback	intermediate
503	Parts and Glue	365	hardback	advanced

The next page asks you to provide ruby expressions and statements. Below are some examples that you might find useful as a reference.

Rails ActiveRecord allows you to access an object and assign it to a variable:

```
hood_part = Part.find(201)
glue_manual = Manual.find_by_format('hardback')
```

Given the two assignments above, note what the following ruby expressions produce:

`glue_manual.pages` – produces the number of pages for the glue_manual object, which would be 365 using the examples in the above tables

`hood_part.weight` – produces the weight for the hood_part object

`Part.find_by_weight(14).tag` – produces the tag for part with weight of 14

`Manual.find(608).format` – produces the format for the Manual object with id 608

Appendix D: Exercise Problems for Experiment 2

The following problems were those used for Experiment 2. Both versions are presented.

D.1. *Version A*

For the following items, provide the ruby code that produces the following:

(1) The part weight with id 205.
(2) The name of the part whose tag is 'm108'.

Ruby has a remove method that allows an object to be deleted from an array of objects. For example if `word_list` has the contents `['cat', 'dog', 'cow', 'horse']`, then `word_list.remove('dog')` would delete the string 'dog' from the array. That is, the contents of `word_list` would then be `['cat', 'cow', 'horse']`.
For the next two questions, assume that parts_list is an array of Part objects and that manuals_list is an array of Manual objects.

(3) Provide ruby code that uses the remove method to delete the 'wheel' part from the parts_list array.
(4) Provide ruby code that uses the remove method to delete the manual whose title is "Model Painting" from the manuals_list array.

Assume that a framework developer has added a new ActiveRecord method called **display** that nicely displays the contents of a database object. It is invoked by adding it to any ActiveRecord object. For example, if hood_part is a Part object, its contents can be displayed with the following Ruby code:

```
hood_part.display
```

(5) Provide ruby code that uses the display method to display the 'green' part.
(6) Provide ruby code that uses the display method to display the manual whose level is "advanced".

D.2. *Version B*

For the following items, provide the ruby code that produces the following:

(1) The part name with id 205.
(2) The weight of the part whose tag is 'm108'.

Ruby has a remove method that allows an object to be deleted from an array of objects. For example if `word_list` has the contents `['cat', 'dog', 'cow', 'horse']`, then `word_list.remove('dog')` would delete the string 'dog' from the array. That is, the contents of `word_list` would then be `['cat', 'cow', 'horse']`.
For the next two questions, assume that country_list is an array of Country objects.

(3) Provide ruby code that uses the remove method to delete the 'green' part from the parts_list array.

(4) Provide ruby code that uses the remove method to delete the manual whose level is "advanced" from the manuals_list array.

Assume that a framework developer has added a new ActiveRecord method called **display** that nicely displays the contents of a database object. It is invoked by adding it to any ActiveRecord object. For example, if hood_part is a Part object, its contents can be displayed with the following Ruby code:

```
hood_part.display
```

(5) Provide ruby code that uses the display method to display the 'wheel' part.
(6) Provide ruby code that uses the display method to display the manual whose title is "Model Painting".