Appeared in *Proceedings of VL 2000: IEEE International Symposium on Visual Languages*. Seattle, WA: IEEE Computer Society, September 10-13 2000, pp. 157-164.

Tabular and Textual Methods for Selecting Objects from a Group

John F. Pane
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 8078
pane+vl2000@cs.cmu.edu

Brad A. Myers
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 5150
bam+@cs.cmu.edu

Abstract

The accurate formulation of boolean expressions is a notorious problem in programming languages and database query tools. This paper studies the ways that untrained users naturally express and interpret queries, revealing some of the underlying reasons why this task is so difficult. Among the study's findings are: people interpret the word AND to mean either conjunction or disjunction depending on context, the scope to which they attribute the word NOT depends on whether the subsequent operator is AND or OR, and they often ignore parenthesis. Therefore, relying on these words and symbols for query formulation will result in poor usability. A tabular query form is proposed that avoids the need to name the operators, provides a clear distinction between conjunction and disjunction, and makes grouping more explicit. Comparing the tabular language with textual boolean expressions, the study finds that untrained users perform better when they express their queries in the tabular language, and about equally well when interpreting queries written in either language. We conclude that systems may benefit by adopting a tabular notation for query formulation.

1. Introduction

We are applying human-computer interaction techniques to the design of new programming language features. Design decisions are resolved by looking to prior research for warnings about potential problems, suggestions for design alternatives, and guidance in selecting among various potential solutions. When necessary, we perform new user studies to investigate questions that are not fully addressed by prior research.

We are currently using this method to design a primarily textual programming system for children to use in creating interactive simulations and games. One of the challenges of this effort is to craft the features of the system to address the problems observed in the prior research. While this is straightforward in some cases, it is quite difficult in others.

One of the more notorious problem areas in programming languages is the accurate specification of boolean expressions [1]. In addition to programming languages, this same problem also appears in the task of formulating queries for common end-user activities such as web searching, library catalog searching, and other database retrieval tasks [2]. Despite the great difficulty that users have demonstrated with using the boolean operators *AND*, *OR*, and *NOT* to construct these expressions, no universally better alternatives have been discovered, so most programming languages continue to rely on them, including many visual and forms-based languages (e.g., [3, 4]). Early web search engines also used these operators, although many have turned to less expressive query languages (for example, the plus and minus unary operators for inclusion and exclusion). *Newsweek* reports that even with these simplifications, most web users are dissatisfied with search engines, and less than 6% manage to use these operators in their searches [5].

The problems with boolean queries are exemplified in studies of non-programmers writing solutions to programming problems in their own words [6, 7]. For example, in these studies it was very common for participants to use the word *AND* where the word *OR* is the correct boolean operator. Instead of saying something like “count the cars with license plates from Georgia **or** Louisiana” they would say “count the cars with license plates from Georgia **and** Louisiana.” The latter version refers to an empty set of license plates when interpreted according to boolean logic, but in English it is usually interpreted to mean the union of the two states’ license plates.[1] It was also noted that the words *OR* and *NOT* rarely appeared, suggesting that boolean expressions are not a natural way to formulate these statements. The participants often used other words and sentence structures to specify their queries accurately. For example, rather than saying “if I get up late and I’m not very hungry I skip breakfast,” they might say “if I get up late I skip breakfast unless I’m very hungry.” This latter construction avoids both the *AND* and *NOT* operators.

In the new study reported here, we investigated several of these alternative formulations to see whether they were more accurate than traditional boolean expressions. In addi-

1. This ambiguity in how to interpret the word that means “and” also appears in many other natural languages, according to our informal poll.

tion, because prior research suggests that non-textual query languages may be more effective than textual syntaxes [8], the study compared these textual alternatives against a proposed new query language that uses tabular forms that would integrate well into a primarily textual language.

The study used a grid of nine colored shapes, where a subset of the shapes could be marked. The participants were given two kinds of problems: *code generation* problems, where some shapes were already marked and they had to formulate a query to select them; and *code interpretation* problems, where they were shown a query and had to mark the shapes selected by the query. They solved all of these problems twice, once using purely textual queries, and once using the proposed tabular forms.

The results suggest that a tabular language for specifying boolean expressions can improve the usability of a programming or query language. On code generation tasks, the participants performed significantly better using the tabular form, while on code interpretation tasks they performed about equally in the textual and tabular conditions. The study also uncovered systematic patterns in the ways participants interpreted boolean expressions, which contradict the typical rules of evaluation used by programming languages. These observations help to explain some of the underlying reasons why boolean expressions are so difficult for people to use accurately, and suggest that refining the vocabulary and rules of evaluation might improve the learnability and usability of textual query languages. A general awareness of these contradictions can help designers of future query systems adhere to one of the basic human-computer interaction principles: to speak the user's language [9].

2. Related work

There is a large body of prior research that identifies usability issues in the design of programming languages, and can provide a basis for making design decisions. For the language we are currently designing for children, a survey of the prior research that applies to beginners is especially relevant [10].

Many researchers have noted that boolean query languages using the *AND*, *OR*, and *NOT* operators are not very effective in programming languages or database retrieval (e.g. [1, 2]). Several researchers have noted that the common usage of these operators in natural language causes errors in queries, such as the substitution of *AND* for *OR* [11, 12]. It has also been noted that the intended scope of the *NOT* operator is ambiguous in natural language [13].

The difficulties of boolean expressions are intensified when several operators must be combined to form the query [14]. Parenthesis improved performance in that study, but other studies have shown that beginners have difficulty with parentheses, especially if they are nested [11, 12].

Replacing the boolean query language with a different subset of natural language, using other words for the operators, is still likely to be inadequate [15]. Many systems that permit unrestricted natural language queries have been shown to be effective for information retrieval tasks (e.g. [16]), but studies of use of natural language for programming are less common (e.g. [17, 18]).

These problems have led researchers to develop graphical interfaces for queries. For example, truth tables and Venn diagrams have been shown to be effective for specifying simple queries [12, 19, 20]. Another system used tiles in a two-dimensional grid, where one dimension represented union and the other represented intersection, although these implicit semantics were found to be confusing [21]. A system that used the graphical metaphor of water flowing through filters was found to be superior to boolean expressions [8], however the screen space required for this tool might limit its effectiveness in a larger context such as a programming language.

3. Design alternatives for Boolean queries

Our new programming language for children will be primarily textual. In reviewing the prior research we found that although the problem of boolean queries is notorious, there are few prescriptions for how solve it effectively. For example, the prior work suggests that we should avoid using the words *AND*, *OR*, and *NOT*, but there is no hard evidence that a different textual query language would be any better.

Earlier studies have analyzed the natural language solutions that non-programmers provided to solve programming problems, and identified some common trends in the ways that boolean queries were expressed [6, 7]. The vocabulary and syntax of these solutions were unconstrained, so they provide insight into how people prefer to express their answers. We speculated that a programming language that closely matches these natural preferences would be more usable than one that requires users to translate their natural solutions into a less natural form. With this in mind, we proposed several alternate ways to express textual queries and compared them in this study. In addition, we also proposed a tabular format for queries.

3.1. Tabular query forms

Although some graphical query methods had been shown to be more effective than boolean expressions, many of them were limited to expressing very simple queries. We wanted a solution that is fully expressive. Also, many of the graphical systems would not integrate well into a programming language, where the entire computer screen cannot be devoted to this one subtask of the programming process. We required a format that is compact and readable in the context of a larger program. With these points in mind, we designed a tabular form that is fully expressive and compatible with the programming language we are developing.

Since our new programming language will represent data on *cards* containing attribute-value pairs, we designed the query form to also use a card metaphor. For the purposes of this study, we simplified the forms by leaving out the attribute names, and limiting the number of terms to three. We called these *match forms* (see Figure 1). Criteria are placed into the slots, one term per slot. All of the terms on a single form implicitly form a conjunction. Negation is specified by prefacing a term with the *NOT* operator. Disjunction is specified by including an additional match form adjacent to the first one.

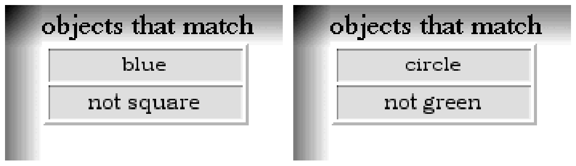

Figure 1. Match forms expressing the query: (blue and not square) or (circle and not green)

This two-dimensional layout is similar to the grid of tiles described by Anick et al. [21] – one dimension implements intersection and the other implements union. However, match forms provide cues to help users remember which operator uses each dimension, such as the text in the form heading and the visual grouping. In addition, the scope of the NOT operator is made explicit by confining it to a single term. This proposed query language can express arbitrarily complex queries, although some queries have to be formulated in a less concise way than pure boolean expressions would allow. To relieve this somewhat, the forms in our proposed language will also allow an entire form to be negated ("objects that do *not* match ..."), but that feature is not used in this study.

4. Hypotheses

The study tests nine hypotheses. The first seven hypotheses examine various textual alternatives to traditional boolean expressions, and the last two hypotheses examine the tabular design alternative.

4.1. AND vs. nested IF

Hypothesis 1: Users will interpret nested IF statements more accurately than a boolean expression using AND.

In the prior studies, people frequently nested an *IF* statement inside another *IF* statement. Instead of saying, "if a and b then ... ," they would say, "if a then if b then ... ." The use of nested *IF*s may be easier to use and understand because it avoids using the confusing *AND* operator for conjunction, and keeps the boolean expression simpler.

4.2. NOT vs. Unless

Hypothesis 2: Users will interpret an Unless clause more accurately than a boolean expression that uses AND and NOT.

In the prior studies, people often wrote a simple conditional statement and then stated an exception at the end. For example, they would write, "if a then ... unless b" This is an alternative to "if a and not b then ... ." In addition to avoiding the *AND* operator, the *Unless* clause permits the user to express a negated term without using the *NOT* operator.

4.3. Location of Unless

Hypothesis 3: Users will interpret an Unless clause more accurately when it appears at the very end of the statement.

Although in English it may be natural to say "if a then ... unless b," in programming languages those ellipsis (...) may be filled with a large block of code. If the *Unless* clause will appear at the very end of the *IF* statement, it will be far removed from the part of the query that is specified in the *IF* clause. Because this violates the principle of locality [22], it may reduce usability. While the principle of locality could not be tested directly with our simple stimuli, we wanted to investigate whether the *Unless* clause is sensitive to its placement within the query.

4.4. Context-dependent interpretation of AND

Hypothesis 4: Users will interpret AND as boolean conjunction in some contexts but not in other contexts.

People often use *AND* in places where the correct boolean operator is *OR*. This may be because interpretation of *AND* in the English language depends on its context. In some cases it is interpreted to be a further restriction on a query (boolean conjunction or set intersection), while in other cases it is interpreted to expand the query (boolean disjunction or set union). For example, these two statements are usually interpreted differently: "pick up the boxes that are blue and green" vs. "pick up the boxes that are blue and the boxes that are green." We attempted to demonstrate this context-sensitive interpretation of the *AND* operator.

4.5. Verbose AND vs. OR

Hypothesis 5: Users will interpret a verbose AND expression as boolean disjunction more accurately than an OR expression.

If Hypothesis 4 is confirmed, it would be useful to characterize the contexts in which *AND* is interpreted as a boolean disjunction instead of conjunction. If certain constructions consistently lead to disjunctive interpretations, perhaps they can reliably replace the rarely-used *OR* operator. We hypothesized that a more verbose expression that restates part of the query is more likely to induce a disjunctive interpretation (see the example in Section 4.4.).

4.6. Operator precedence of NOT

Hypothesis 6: Users will interpret the NOT operator with lower precedence than the other boolean operators.

People often interpret the *NOT* operator with lower precedence than the other boolean operators. This is opposite to the rules of interpretation in most programming languages, where *NOT* has higher precedence than the other boolean operators. That is, in "not a and b," programming languages associate the *NOT* tightly with the "a", while we expect people to first interpret the expression “a and b” and then apply the *NOT* operator to the result.

4.7. Parenthesis for expression grouping

Hypothesis 7: Users will misinterpret parenthesized expressions.

Regardless of the precedences chosen for the boolean operators, a mechanism is required for the user to clarify or override them. Programming languages typically use parenthesis to explicitly group sub-expressions, but research has shown that beginners have difficulty with parenthesis.

4.8. Tabular vs. textual

Hypothesis 8: Users will interpret queries that use match forms more accurately than equivalent textual queries.

Hypothesis 9: Users will generate more accurate queries using match forms than they generate using text.

We investigated the relative usability of match forms compared with text on both interpretation and generation of queries. We expected match forms to be effective because they eliminate many of the problems with text that are discussed above. By avoiding the words *AND* and *OR*, any confusion with the meaning of these words in English is avoided. Also, the precedence or grouping of the operators becomes less ambiguous.

5. Method

Before beginning the study, participants filled out a questionnaire that collected basic demographic information. Then they answered a set of problems that were divided into four sections. In two of the sections, the *writing* sections, the participants generated queries to match a result that we supplied. In the other two sections, the *reading* sections, participants interpreted queries that we supplied. We label the two writing sections *WT* (writing text) and *WF* (writing forms), and the two reading sections *RT* (reading text) and *RF* (reading forms).

There were five *WT* questions and five identical *WF* questions. Comparing the performance across these two conditions is the basis for testing hypothesis 9. By random assignment, half of the participants answered the WT questions first, and the other half answered the WF questions first, to control for any effect of presentation order. All of the writing questions were presented before any of the reading questions, so that the queries that were displayed in the reading sections would not bias their responses in the writing sections.

There were eleven *RT* questions, forming the basis for testing hypotheses 1-7. The first five hypotheses can be evaluated by comparing relative student accuracy across a pair of questions. Hypotheses 6 and 7 can be evaluated by examining which of two interpretations the participants used in answering a single question. To control for any effect of presentation order, participants were randomly assigned to a path through the questions. The paths were constructed so that, for every pair of questions we intended to compare, the number of times that either question appeared first was balanced.

The eleven *RF* questions were constructed by translating the *RT* questions into the tabular language. So, each participant answered the same question twice, once with text and again with match forms. Comparing the performance in these two conditions is the basis for testing hypothesis 8. By random assignment, half the participants solved the *RF* questions first, and the other half solved the *RT* questions first, to control for any effect of presentation order.

There were a total of 32 questions in the four reading and writing sections. After this, participants answered a survey of seven preference questions. Each of these showed a query result along with two or more queries that would correctly generate the result. The participants were asked to select the one they liked the best.

5.1. Participants

In addition to examining these hypotheses with children who are the target audience of our programming language, we were interested in how the results would generalize to other ages. So, we recruited both children and adults to participate in the study.

33 volunteers participated, 13 children (ages 10-14), and 20 adults (ages 18-46). 14 were male and 19 were female. All but two were native speakers of English. 7 participants reported that they had written computer programs (4 adults, 3 children). 27 reported that they had some experience with web search engines, and 18 had used advanced searching features (such as AND, OR, NOT, +, -, etc.). Two adults were experienced with the SQL database query language.

5.2. Materials

The 32 problems were presented in a web browser, one problem per web page. Each of the problem groups was preceded by an instruction page explaining how the query language or query forms work and introducing the format of the exercises. The *WT* and *WF* instruction pages were constructed to be as similar to each other as possible, as were the *RT* and *RF* pages. The web server managed the random assignment of participants to a path through the problems, the presentation of the problems in the order determined by that path, and the collection of the data anonymously. Figure 2 contains an example problem from each of the four problem groups.

5.3. Procedure

Participants began on the demographic questionnaire page and proceeded at their own pace through the materials. They were instructed to be as accurate as possible and were told that there was no time limit. When they submitted an answer, the server recorded it and presented them with the next page in the sequence. The server performed some

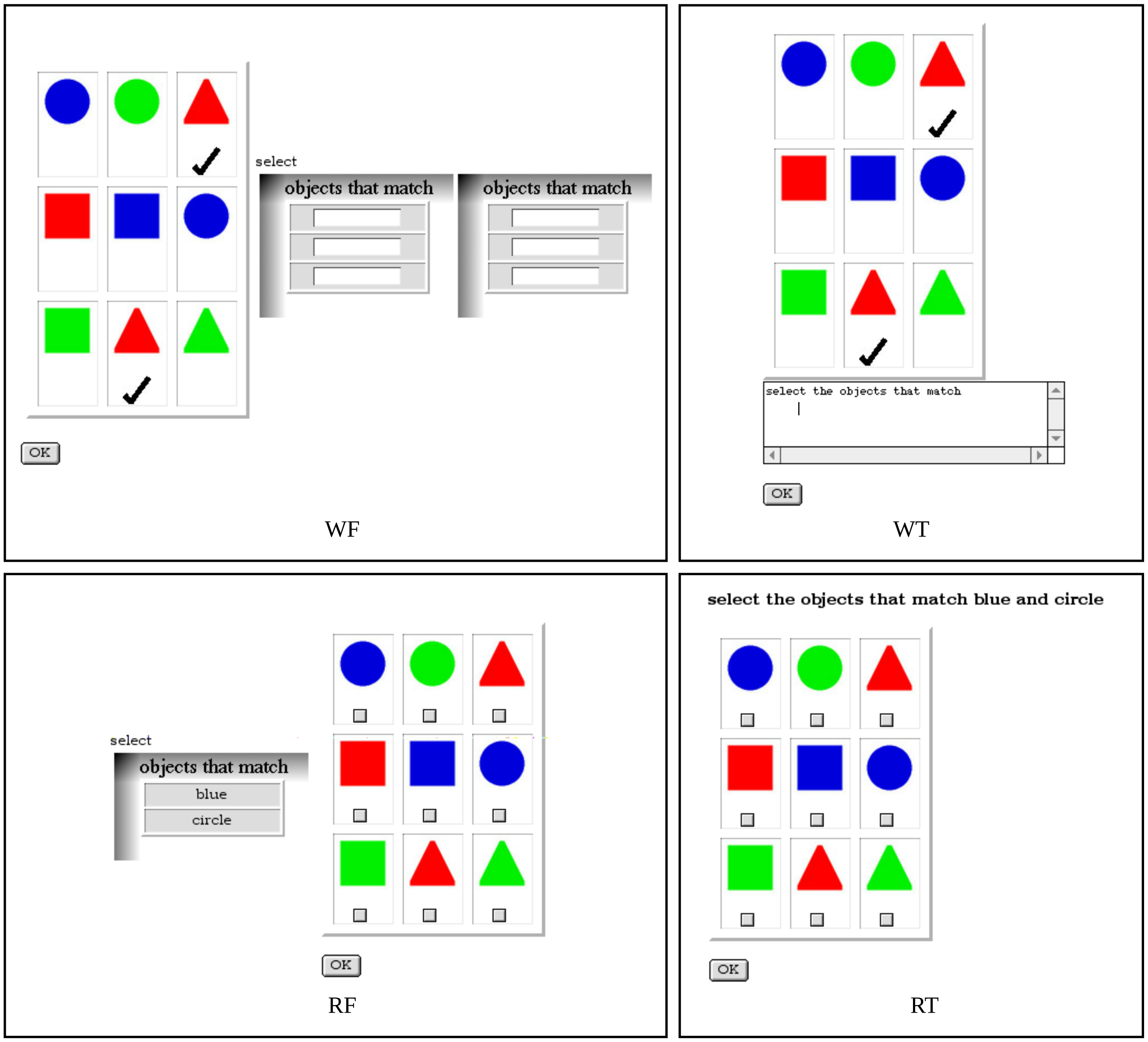

Figure 2. An example problem from each of the four problem groups (*WF, WT, RF, and RT*) before the answers are filled in. The color of each object is red, green or blue on the computer screen.

basic syntactic checks (for example, it made sure the user provided a boolean operator on the WT tasks, and that they didn't put multiple criteria into a single slot in the WF tasks). If this check failed, an error message asked the participant to go back and fix the answer. Any time participants returned to a previous page to revise an answer, we recorded all of the answers but only used the final one for the results presented here.

Each participant's answer was scored as correct or incorrect according to the following policy. Spelling errors were tolerated, as were additional words such as *an*, *a*, or *the*. Plural and singular forms of all words were accepted. Consistent use of an incorrect color name that did *not* actually appear in the study (e.g. orange for red) was tolerated. But, any incorrect replacement of one color or shape with another one that *did* appear in the study (e.g. blue for green) was marked as incorrect. Except where otherwise noted, textual answers were interpreted the way a programming language would interpret them. Invented shorthand notations were marked as incorrect. Redundant or overly complex answers were scored as correct if they resulted in the correct selection. Finally, on the problems where we gave special instructions, answers that did not follow the instructions were marked as incorrect even if they resulted in a correct selection (e.g. one of the questions in *WF* and *WT* asked the users include the word *NOT* in their answers).

Because we simplified the match forms for this study, some of the problems became more complex when they were translated from *RT* to *RF*. For example, the lack of a way to negate a whole match form causes the expression “not (a and b)” to be translated into the tabular equivalent of “(not a) or (not b).” These question pairs were discarded from the comparison of *RT* to *RF* in testing hypothesis 8.

6. Results

No significant differences were detected between children and adults, between males and females, or between programmers and non-programmers, so the results are aggregated across all of the participants. The numbers shown are percentages.

In evaluating hypotheses 1-5, we performed within subject comparisons on pairs of questions from the *RT* problem group. Statistical significance in these comparisons was evaluated with a non-parametric sign test. To test hypotheses 6 and 7, we examined which of two interpretations participants used in answering a single question. Statistical significance in these comparisons was evaluated with a binomial test. In evaluating hypotheses 8 and 9, we compared pairs of questions between *RT* & *RF* and between *WT* & *WF*, respectively. These comparisons were within subject, and statistical significance was evaluated with a non-parametric sign test. In all of the statistical tests, p<.05 was used as the threshold for significance.

Hypothesis 1 is not confirmed. Users will interpret nested IF statements more accurately than a boolean expression using AND.	% correct
Nested IF *select the objects that match red, if the objects match triangle*	94
AND *select the objects that match blue and circle*	85

not significant

Hypothesis 2 is not confirmed. Users will interpret an Unless clause more accurately than a boolean expression that uses AND and NOT.	%correct
Unless *select the objects that match blue, unless the objects match square*	97
AND NOT *select the objects that match square and not red*	94

not significant

Hypothesis 3 is confirmed. Users will interpret an Unless clause more accurately when it appears at the very end of the statement.	% correct
Unless at end *select the objects that match blue, unless the objects match square*	97
Unless earlier *unless the objects match green, select the objects that match circle*	76

p<.05

Hypothesis 4 is confirmed. Users will interpret AND as boolean conjunction in some contexts but not in other contexts.	% conjunction
select the objects that match blue and circle	85
select the objects that match blue and the objects that match circle (55% of the participants interpreted this as boolean disjunction)	36

p<.0001

Hypothesis 5 is disconfirmed. Users will interpret a verbose AND expression as boolean disjunction more accurately than an OR expression.	% disjunction
select the objects that match blue and the objects that match circle	55
select the objects that match square or green	82

p<.05

Hypothesis 6 is disconfirmed for NOT with AND. Users will interpret the NOT operator with lower precedence than the other boolean operators. *select the objects that match not red and square*	%
precedence of NOT is higher than AND interpreted as: (not red) and square	64
precedence of NOT is lower than AND interpreted as: not (red and square)	9

p<.001

Hypothesis 6 is confirmed for NOT with OR. Users will interpret the NOT operator with lower precedence than the other boolean operators. *select the objects that match not triangle or green*	%
precedence of NOT is higher than OR interpreted as: (not triangle) or green	9
precedence of NOT is lower than OR interpreted as: not (triangle or green)	67

p<.001

Hypothesis 7 is confirmed. Users will misinterpret parenthesized expressions. *select the objects that match (not circle) or blue*	%
ignore parenthesis, NOT has low precedence interpreted as: not (circle or blue)	39
observe parenthesis interpreted as: (not circle) or blue	12

p<.05

As mentioned above, three of the RF problems were not well-matched to the corresponding RT problems, so these pairs were discarded in analyzing hypothesis 8. This left eight pairs of reading problems to test hypothesis 8. All five pairs of writing problems were used to test hypothesis 9.

Hypothesis 8 is not confirmed. Users will interpret queries that use match forms more accurately than equivalent textual queries.	% correct
Match Forms (RF)	71
Text (*RT*)	74

not significant

Hypothesis 9 is confirmed. Users will generate more accurate queries using match forms than they generate using text.	% correct
Match Forms (*WF*)	94
Text (*WT*)	85

p<.0001

The following table breaks down the individual problems in *WF* vs. *WT*, showing the percent correct. The problems are labeled with our canonical text solutions.

	red and triangle	square and not red	(blue and circle) or (red and triangle)	circle or blue	square and not red[a]
Match Forms (*WF*)	94	73	91	42	33
Text (*WT*)	94	64	12	18	21
	n.s.	n.s.	p<.0001	p<.01	n.s.

a. The word *NOT* was required in the solution to this problem

7. Discussion

Although the results help to explain some of the reasons why boolean queries using *AND*, *OR*, and *NOT* are so difficult, the textual alternatives that we proposed did not improve performance. On the other hand, the proposed tabular query forms did improve performance on writing tasks, while performing about the same on reading tasks.

7.1. Textual query variations

Hypothesis 1 was not confirmed. The participants performed about the same using nested *IF* statements as they did using a boolean expression with the *AND* operator. On the preferences survey, the majority of the participants preferred the boolean expression.

Hypothesis 2 was not confirmed. The participants performed about the same using an *Unless* clause as they did using a boolean expression with the *AND* and *NOT* operators. On the preferences survey, the majority of the participants preferred the boolean expression.

Hypothesis 3 was confirmed. The participants performed significantly better with the *Unless* clause at the end than they did with the *Unless* clause earlier in the statement. However, given the result of Hypothesis 2, the importance of this result is questionable. Also, the very simple problems used in this study did not provide a good way to test the situation where we speculated that the *Unless* would violate the principle of locality. On the preferences survey, most of the participants preferred the *Unless* at the end.

Hypothesis 4 was confirmed. Two slightly different queries using *AND* resulted in significantly different interpretations. 85% of the participants interpreted the *AND* in “select the objects that match blue and circle” as a conjunction operator. But only 36% of them interpreted it that way in “select the objects that match blue and the objects that match circle.” Instead, 55% of them interpreted the *AND* in the second statement as a disjunction operator. This result helps to explain the frequently observed error where users incorrectly use *AND* instead of *OR*.

Hypothesis 5 was disconfirmed. Despite the fact that the majority of the participants interpreted *AND* as a disjunction operator in “select the objects that match blue and circle,” they are significantly more accurate in interpreting disjunction if the *OR* operator is used. On the preferences survey, the majority of the participants preferred *OR* over a verbose *AND* statement to express disjunction.

In the surprising results of hypothesis 6, we measured reliable effects in opposite directions depending on context. The hypothesis was disconfirmed when comparing the precedence of *NOT* with *AND*. 64% of the participants treated *NOT* with **higher** precedence than *AND*, matching the common usage in programming languages. However, the hypothesis was confirmed when comparing the precedence of *NOT* with *OR*. In this case, 67% of the participants treated *NOT* with **lower** precedence than *OR*. Since consistency is an important human-computer interaction principle [9], this reversal in the natural interpretation of precedence suggests that it is unwise to rely on implicit precedence rules.

Hypothesis 7 was confirmed. Users ignored parenthesis significantly more often than they observed them. The query was, “select the objects that match (not circle) or blue.” The results on hypothesis 6 suggest that without the parenthesis, most participants would have applied the *NOT* operator to the expression “circle or blue.” The parenthesis were not able to override this tendency.

7.2. Match forms vs. text

Hypothesis 8 was not confirmed. On reading tasks, the participants performed about as well with match forms as they did with text. However, hypothesis 9 was confirmed. On writing tasks, the participants performed significantly better with the match forms than they did with text. This disparity, where a positive effect is stronger on generation tasks than interpretation tasks, has also been observed in other systems (e.g. [23]). On the preferences survey, which was a reading task, the participants’ choices were about equally divided between text and match forms.

Match forms were not superior for code interpretation, but they did not have a detrimental impact on that task. Thus the overall effect of using match forms should be positive due to the strong gains on generation tasks, despite the lack of an effect on interpretation tasks.

The breakdown of individual questions in the query generation task shows that the participants performed about the same in the two conditions when the queries were simpler, but as the queries became more complex, the differences in favor of the match forms increased. While the trend in favor of match forms was present in all cases, only the queries that involved disjunction revealed significant differences between match forms and text. As expected, the most common error on these problems was the substitution of *AND* where *OR* was required.

The three problems that were excluded from the reading comparison were among the more complex queries. Since the advantage of query forms is stronger on more complex queries, excluding this data may have reduced any positive effect of match forms on the reading task, making it less likely that our study would be able to confirm hypothesis 8. Further research into this question is warranted.

The strong effect of match forms came with very little training. It is unlikely that the participants had used an equivalent tabular query language before, and they only viewed a brief instruction page with a few examples before beginning to solve the problems. While the instructions for the textual problems were similarly brief, the participants brought knowledge from a lifetime using the words *AND*, *OR*, and *NOT* in English. This may have interfered with the programming language interpretation, or made them less careful in reading the instructions.

8. Conclusions

Based on the results of this study, we can make the following recommendations to designers of programming languages, scripting tools and search engines that incorporate query mechanisms:

- Do not use the word *AND*.
- Do not rely on parenthesis for grouping.
- Do not rely on implicit operator precedence rules.
- Consider using tabular query forms instead of pure text.

This study of what is natural for untrained users provides a scientific basis for choosing among design alternatives in query tools for beginners. We will use these results in the design of a new programming system for children, expecting that this strategy will yield a language that is easier to learn and use than other languages. In addition to these specific recommendations, the strategy employed here can be used by developers to assist in the design of other kinds of tools.

Acknowledgments

We would like to thank Albert Corbett, Bernita Myers, Barbara Pane, and the anonymous participants. This research is funded in part by the National Science Foundation under Grant No. IRI-9900452. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] J.-M. Hoc, "Do We Really Have Conditional Statements in Our Brains?," in *Studying the Novice Programmer*, E. Soloway and J. C. Spohrer, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1989, pp. 179-90.

[2] C. Hildreth, "Intelligent Interfaces and Retrieval methods for Subject Search in Bibliographic Retrieval Systems," in *Research, Education, Analysis & Design*. Springfield, IL, 1988.

[3] *Prograph CPX User Guide*. Halifax, Nova Scotia: Pictorius Incorporated, 1996.

[4] J. G. Hays and M. M. Burnett, "A Guided Tour of Forms/3," , Oregon State University, Dept. of Computer Science Technical Report 95-60-6, June 1995.

[5] J. Tanaka, "The Perfect Search," in *Newsweek*, vol. 134, 1999, pp. 71.

[6] L. A. Miller, "Natural Language Programming: Styles, Strategies, and Contrasts," *IBM Systems Journal*, vol. 20, pp. 184-215, 1981.

[7] J. F. Pane, C. A. Ratanamahatana, and B. A. Myers, "Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems," *International Journal on Human-Computer Studies, to appear*, 2000.

[8] D. Young and B. Shneiderman, "A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation," *Journal of American Society for Information Science*, vol. 44, pp. 327-339, 1993.

[9] J. Nielsen, "Heuristic Evaluation," in *Usability Inspection Methods*, J. Nielsen and R. L. Mack, Eds. New York: John Wiley & Sons, 1994, pp. 25-62.

[10] J. F. Pane and B. A. Myers, "Usability Issues in the Design of Novice Programming Systems," Carnegie Mellon University, Pittsburgh, PA, School of Computer Science Technical Report CMU-CS-96-132, August 1996.

[11] S. L. Greene, S. J. Devlin, P. E. Cannata, and L. M. Gomez, "No IFs, ANDs, or ORs: A Study of Database Querying," *International Journal of Man-Machine Studies*, vol. 32, pp. 303-326, 1990.

[12] A. Michard, "Graphical Presentation of Boolean Expressions in a Database Query Language: Design Notes and an Ergonomic Evaluation," *Behaviour and Information Technology*, vol. 1, pp. 279-288, 1982.

[13] A. McQuire and C. M. Eastman, "Ambiguity of Negation in Natural Language Queries," in *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Posters: Abstracts*, 1995, pp. 373.

[14] P. J. M. D. Essens, C. A. McCann, and M. A. Hartevelt, "An Experimental Study of the Interpretation of Logical Operators in Database Querying," in *Cognitive Ergonomics: Contributions from Experimental Psychology, Database Interrogation*, G. C. v. d. Veer, S. Bagnara, and G. A. M. Kempen, Eds. Amsterdam: North-Holland, Elsevier Science Publishers, 1992, pp. 201-225.

[15] A. Kohl and W. Rupietta, "The Natural Language Metaphor: An Approach to Avoid Misleading Expectations," in *Proceedings of IFIP INTERACT'87: Human-Computer Interaction*, 1987, pp. 555-560.

[16] H. Turtle, "Natural Language vs. Boolean Query Evaluation: A Comparison of Retrieval Performance," in *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Performance Evaluation*, 1994, pp. 212-220.

[17] A. W. Biermann, B. W. Ballard, and A. H. Sigmon, "An Experimental Study of Natural Language Programming," *International Journal of Man-Machine Studies*, vol. 18, pp. 71-87, 1983.

[18] A. Bruckman and E. Edwards, "Should We Leverage Natural-Language Knowledge? An Analysis of User Errors in a Natural-Language-Style Programming Language," in *Proceedings of the 1999 Conference on Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press, 1999, pp. 207-214.

[19] J. Thomas and J. Gould, "A Psychological Study of Query by Example," in *National Computer Conference*, vol. 44. Anaheim, CA: AFIPS, 1975.

[20] S. Jones, "Graphical Query Specification and Dynamic Result Previews for a Digital Library," in *Proceedings of the ACM Symposium on User Interface Software and Technology, Enabling Architectures*, 1998, pp. 143-151.

[21] P. G. Anick, J. D. Brennan, R. A. Flynn, D. R. Hanssen, B. Alvey, and J. M. Robbins, "A Direct Manipulation Interface for Boolean Information Retrieval via Natural Language Query," in *Proceedings of the Thirteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, User Interfaces*. Brussels, Belgium, 1990, pp. 135-150.

[22] J. R. Cordy, "Hints on the Design of User Interface Language Features – Lessons from the Design of Turing," in *Languages for Developing User Interfaces*, B. A. Myers, Ed. Boston: Jones and Bartlett, 1992, pp. 329-340.

[23] F. Modugno, A. T. Corbett, and B. A. Myers, "Evaluating Program Representation in a Visual Shell," in *Empirical Studies of Programmers: Sixth Workshop*, W. D. Gray and D. A. Boehm-Davis, Eds. Norwood, NJ: Ablex Publishing Corporation, 1996, pp. 131-146.