Problem-solving Transfer among Programming Languages

Quanfeng Wu and John R. Anderson
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

June 4, 1990

For submission to: Human-Computer Interaction
Transfer among languages

Abstract

This study investigated knowledge transfer among three programming languages --

Languages

Problem-solving Transfer among

Problem-solving

Cambridge Mellon University, PA 15213.

Authors present address: Quanteng Wu and John R. Anderson, Department of Psychology,
REFERENCES

2. SUMMARY

6.2. Declarative versus Procedural Transfer
6.1. Three Levels of Transfer in Programming

6. GENERAL DISCUSSION

5.2. Results
5.1. Design

PASCAL

5. EXPERIMENT 3: TRANSFER BETWEEN PROGRAMMING IN LISP AND IN

4.3. Discussion
4.2. Results
4.1. Design

PROLOG - A FURTHER STUDY

4. EXPERIMENT 2: TRANSFER BETWEEN PROGRAMMING IN LISP AND IN

3.3. Discussion
3.2. Results
3.1. Design

PROLOG

3. EXPERIMENT 1: TRANSFER BETWEEN PROGRAMMING IN LISP AND IN

2.2. Two Examples of Programming Problems
2.1. A Comparison of LISP, PROLOG, and PASCAL

2. A COMPARISON OF THE THREE LANGUAGES

1. INTRODUCTION

CONTENTS
Transfer among Languages

called learning transfer. It is the goal in our teaching work. The transfer studied in schools is
pursue this goal. We feel we needed to know better what could be transferred in order to
transferred among the languages (Anderson, Corter, Fischman, Hoffman & Pellettier, 1989). In
this research project, we are also building a computer tutoring system to teach multiple programming languages with the goal that there be substantial
the issue that led us to this research project. We are also building a computer tutoring
between two languages we would be in a better position to teach for transfer. This is in fact
the knowledge representation that is involved. Perhaps if we knew that was in common
the question of the nature of language (e.g., Mandelkern & Lin, 1947) again raises the
to transfer skills in programming generally from one language we learn to a later
have to transfer skill in programming generally from one language to another, but we also
real world that we have to convert programs from one language to another, but we also
This theoretical question is of practical significance as well. Not only is it the case in the case in the
some proposals as to what is being transferred.

1. Introduction

But what is really meant by "the understanding of the program" in this paper we will offer
completely different than what must be transferred is the understanding of the program.
representation of a computer program. Since the codes in the two languages may be
Theoretically, this is an interesting phenomenon because it reflects on the mental
demonstrate the accuracy of that experience and explore the basis for this transfer.
will be much easier to write the program in a second language. This paper will
Most people have the experience that if they write a program in one language then it
Solving analogical problems for example, Slomin & Hayes (1976), and among isomorphic problems. For example, Simon & Newell (1967) and Kolenko & Fallside (1988) called it transfer. While Keane (1969) called it analogical problem solving. Generally, analogical problem solving is also such a kind of transfer: namely, transfer  

out of the lisp experimental results. Finally, some further discussions on transfer among the rest of the paper is structured as follows: in the next section a brief comparison of the three programming languages involved in this study will be presented; this is followed by demonstration that there are indeed commonalities shared by them. Then the three experiments will be reported and discussed; also, Katz's results will be compared with experiments in the lisp experimental results. While the third was an extension of the second experiment, the second was an extension of the first.

The last experiment also replicated an experiment in lisp and pascal. The first experiment was an extension of the second experiment, while the third was an extension of the second. The second experiment demonstrated the existence of transfer between writing lisp and pascal programs the second attempt to demonstrate that there are indeed commonalities shared by them. Then the three experiments will be presented; this is followed by demonstration that there are indeed commonalities shared by them. Then the three experiments will be presented; this is followed by demonstration that there are indeed commonalities shared by them.

There is a long history of research on transfer (e.g., Thomske & Woodworth, 1970). This kind of transfer may be called problem solving transfer, it involves people who have already know both languages and have to translate a solution from one language to the other. It is our belief that both types of transfer depend on commonalities in the representation of the problem solution; however, other relationships are also logically possible. As we only explore the second kind of transfer in this paper we will not be able to establish anything definite about this issue of whether the two types of transfer both depend on the same kinds of commonalities in program representation.
Concerned with transfer between LISP and PROLOG and between LISP and PASCAL, we note that PROLOG and PASCAL provide a purely descriptive language, even between PASCAL and LISP articles, there is neither purely PROLOG than between PASCAL and PROLOG. However, there are more commonalities between PASCAL and LISP and between LISP and PROLOG, standing in the middle. Following this view, it is also plausible that there is a continuum with LISP standing in the middle. Following this view, it is also plausible that the PASCAL, LISP, and PROLOG are seen as occupying overlapping ranges along the continuum from purely descriptive languages to purely descriptive languages. Taken that view, from purely descriptive languages to purely descriptive languages, there are also a number of commonalities among them. More specifically, we can take a general picture of programming languages: they fall into three different categories of programming languages -- namely, procedural, functional, and logic-based (Figure 1). However, the three languages will be shown in further illustrating the similarities and differences among the three languages. It is our belief that commonalities in knowledge representation and transfer will be proposed to account for the results from the experiments.
Transfer among languages

In Pascal or a function (in LISP) can call itself and/or other procedures/functions (in
order of their codes can be organized hierarchically in that a procedure/function
 executes sequentially according to the static
programs written in these two languages are executed sequentially. That is,

A major commonly shared by LISP and PROLOG are their control-flows. That is,

Programmers may also be very confused with the notion of predicate in programming.
Mathematically, there is a correspondence between the two; however, notice PROLOG.
Elementary constituents -- namely, the function in LISP and the predicate in PROLOG.
Another prominent commonality of LISP and PROLOG is their
correspondences. The basic operations facilitated by both languages have strikingly
does, the basic operations in LISP and PROLOG also have a great

Although PROLOG offers far fewer built-in operators for list processing than LISP
does, the basic operations in LISP and PROLOG also have a great

to data structures and list operations on them. LISP and PROLOG also have a great

recursiveness turns out to be an important commonality between LISP and PROLOG. In

backtracking also entails that programs are actually executed in a recursive way. Thus,

form of control-flow; correspondingly, in PROLOG the mechanism of matching and

language implementation. However, in LISP programming recursion is usually a major

unification procedure, or in other words, the matching and backtracking mechanisms of the

is not the case for PROLOG. The control-flow of a PROLOG program is embedded in the

flow of a LISP program is transparent from the static organization of the code; but this

An outstanding difference between LISP and PROLOG is their control-flows. LISP is

will only further explain some commonalities and differences of these two pairs.

Inset Figure 1 about here
Transfer among Languages

the complexity of the recursive algorithm is exponential: \( T_{\text{recursion}} = O(n^2) \).

Furthermore, the iterative solution is more efficient for execution than the recursive one. The

comparative complexity of the iterative algorithm is only linear: \( T_{\text{iteration}} = O(n) \); whereas,

Nevertheless, to a great extent, this example shows that there may be numerous

two examples of programming problems which were actually used as testing

2.2. Two Examples of Programming Problems

constructing any complex data structures. This type of data structures is very powerful and versatile in

structures is the list; however, this type of data structures usually have to be the same. While in LISP, the primary type of constructive data structures the Base type of

PASCAL, the basic type of constructive data structures are sets, arrays, records, and files.

between PASCAL and LISP. Some striking differences existing in

recursion is less efficient in PASCAL whereas recursion is usually stressed in LISP. To most

mechanisms normally used for implementing the two languages. That is, generally,

introduce PASCAL and LISP. However, this difference is simply incurred by the institutional materials which

usually emphasized in PASCAL whereas recursion is usually stressed in LISP. To most

Although both iteration and recursion can be implemented in both languages, iteration is

general, there are two types of repetition control-flow -- namely, iteration and recursion.

in the emphasis put on the repetition structures realized by these two languages. In

PASCAL (in LISP). However, usually, there is also a big difference between

PASCAL
3.1 Method

LISP to PROLOG or from PROLOG to LISP. Also, here transfer was studied bidirectionally; that is, either from PROLOG beforehand. Also, here transfer was studied bidirectionally; that is, either from transfer between programming in these languages, our subjects knew both LISP and programming in LISP and in PROLOG. As we were only investigating problem solving on this experiment, this experience was to demonstrate the existence of transfer between.

3. EXPERIMENT 1: TRANSFER BETWEEN LISP AND PROLOG

[Insert Figure 3 about here]

Programming for the problem. Different in appearance, there still may be some transfer in the understanding phase of the styles of programming in these languages. However, although these programs are very implementations can utilize the specific features provided by the language and conform to problem may appear to be quite different from one another. That is, each of the example, here we see that the programs written in the three languages for the same problem and some model programs are shown in Figure 3. Contrary to the previous representing books and then to search the database to count the books by a given author.

The second problem is to construct a simple database as to contain entries

[Insert Figure 2 about here]

Commonalities among the programs written in the three languages for the same problem.
transfer among languages

3. All the problems and models solutions to them can be obtained by writing to the authors.

SAT/GRE math scores and self-ratings of their proficiency in various programming

skills were already shown in Figure 2.3. The relevant information gathered from the questionnaire was

PROCEDURE. Before the experiment actually began, the subjects were requested to ill

POWERSEL, Simple Database Searching, and Simple Expression Parsing. Among them, two

MATERIALS. The four problems used in the experiment were Fibonacci, Function.

Inset Figure 4, About Here

analyze the transfer bidirectionally.

and between-subject designs: one advantage of this design is that it would allow us to
groups, as shown in the figure. This experimental design is a mixture of the within-subject

POWERLOG was allocated from one problem to the next and switched between the two

and PROLOG was allocated from the other in PROLOG. The order of programming in LISP

two programs, one in LISP and the other in PROLOG. The subjects were asked to solve one problem. For each problem, the subjects had to write

there were two sessions of the experiment, in each session, the subjects were partitioned into two groups. There were four sessions of the experiment. The 6 subjects were

DESIGN. Figure 4 shows the design of the experiment. The 6 subjects were

in average, the subjects were relatively more proficient in LISP than in PROLOG.

mean self-ratings of LISP skill was 3.63, while that for PROLOG skill was only 2.50. Thus,

information taken from Questionnaire, the subjects' mean GRE/SAT scores were 750; the

research assistants. They were reimbursed for participating in the experiment. From the

Mellon University (community) among them I undergraduate, 2 graduate students, and 3

Subjects. The six subjects involved in this experiment were from the CMU (Carnegie

Transfer among Languages

and the first-draft time is the rest-draft time -- namely, the time spending on revisions of the
debugging. So was the total problem solving time. The difference between the total time
in this experiment. The first draft time included the time spent on thinking, typing, and
time in this experiment. We only collected the total problem solving time and the first
time measures. We only collected the total problem solving time and the first draft
experiment: the time measures and the numbers of program errors produced by subjects.

Two kinds of quantitative measures were used here for analyzing the results of the

3.2 Results

sheets, but the final correctness of the programs was determined by the experimenter.

computer system. Test cases for the problems were selected on the problem statement

test cases for the problems were selected on the problem statement.

Throughout the whole experimential sessions, the experimenter was with the subjects.

------------------------------------------------------------

Insert Figure 5 about here

------------------------------------------------------------

EMACS.

actually used in the experiment were COMMONLISP and PROLOG: the editor was

the program under writing were saved into different files. The versions of the languages

on them editing and debugging again, until the programs worked correctly. Various drafts of

the program under writing were saved into different files. The versions of the languages

each problem, the subject went through the cycle of editing, debugging, if errors found

window in which LISP and PROLOG programs were debugged and tested. For
debugging window in which LISP and PROLOG programs were typed in and modified and a

created: an editing window in which programs were typed in and modified and a

The actual procedure of the experiment is shown in Figure 5. The subjects did
across the subjects within one group for individual problems are shown in graphical form. Figure 7 shows the data for the mean first-draft time. Here again, the data averaged

Insert Figure 6 About Here

the small numbers of degrees of freedom, the effects were typically highly significant. Despite the small numbers of degrees of freedom, the effects were typically highly significant. Main effect due to transfer: F(1, 2) = 36.71, p < 0.01. As can be seen, despite

programming skills as measured by their total problem-solving times. The results revealed

subjects as a macro subject; the subjects in the two groups were paired according to their

conditions in this design, for purposes of statistical convenience we treated each pair of

least the statistical significance of the data. Since each subject only realized half of the

language: LISP/PROLOG: problem 1 to 4; that is, 2 x 4 (pattern ANOVA) was used to

PROLOG), a three-way transfer x language x problem (transfer: first/second program:

program (one factor for this result is that one in this group had a relatively weaker skill in

Group 1; there was more time spent on the second PROLOG program than the initial LISP

in terms of cognitive expenditure. The only exception was the first problem for

solving a problem in one language to solving it in the other language, a kind of positive

the labeler form. From the graph we can see that there were time savings manifested from

and the data averaged across the two groups and the four problems are in

averaged across the subjects within one group for the four separate problems are in the

The data for the mean total problem-solving time are shown in Figure 6; the data

average.

In other words, this time measure was used to gauge the time spent on each rest draft in

\[
\text{Per-draft time} = \text{Rest-draft time / (No. of Drafts - 1).}
\]

we calculated the mean per-draft time for the subsequent drafts; that is, initial draft of problem; this rest-draft time was also used for data analysis. Furthermore,
some might trace down several small bugs in one pass or one bug in several passes.

The subjects usually attempted to fix one bug each day, although on some occasions
subsequent drafts were mainly manifested as fewer drafts rather than less time per draft.

Subsequent changes were mainly manifested as fewer drafts rather than less time per draft.

In the first language group, the transfer on programming in the first language to the second, in other words, the transfer on
problems. From the results, we see that this pre-draft time was fairly stable from
also presents the mean pre-draft time for subsequent drafts averaged across both groups
the test showed significant main effect due to transfer: F(1, 2) = 25.41, p < 0.05. Figure 9
prents the data in terms of this measure; another three-way ANOVA was performed and
a second three-way ANOVA was used to analyze these results; the results of
mismatches were largely semantic errors, in other words, algorithmic errors. Figure 9
experienced that there were mismatches in the subjects in the syntactic checking of their programs. These
corresponds to the numbers of mismatches subjects made in programming. Because the
by subjects in the course of programming: Roughly speaking, this measure also
The second measure used for data analysis was the number of program drafts made

---

Inser Figure 8 About Here

---

Inser Figure 7 About Here

---

Indicated significant main effect due to transfer: F(1, 2) = 187.03, p < 0.01. This
Inser Figure 7 About Here

---

Indicated significant main effect due to transfer: F(1, 2) = 330.69, p < 0.01. As
Inser Figure 8 About Here
Typing in the codes on the computer, solutions into components and translate them into the second language before actually
solutions to her solutions in the first language; she then worked on paper to break the
her access to her solutions in the first language and since she found a great deal of difficulty in these two problems we allowed
solutions and since she found a great deal of difficulty in these two problems we allowed
in both LISP and PROLOG among the subjects. She asked for print-outs of her
solution for parse-expression from PROLOG to LISP. She was the weakest programmer
solution for parse-expression from PROLOG to LISP. She was the weakest programmer
mechanical translations of her solution for parse-expression from PROLOG to LISP.
and her
mechanical translations of her solution for parse-expression from PROLOG to LISP.
their own solutions in the first language. Only one subject seemed to make such
their own solutions in the first language. Only one subject seemed to make such
representation. However, this seems unlikely as our subjects usually did not access to the
representation. However, this seems unlikely as our subjects usually did not access to the
one language into the code of the other without reference to a deeper algorithmic
one language into the code of the other without reference to a deeper algorithmic
It is also conceivable that subjects might be just mechanically translating the code of
It is also conceivable that subjects might be just mechanically translating the code of
We will refer this type of transfer as algorithmic transfer.
We will refer this type of transfer as algorithmic transfer.
The algorithmic knowledge gained from programming in the first language to the second,
The algorithmic knowledge gained from programming in the first language to the second,
the pattern of the first draft times in fact seems to indicate that subjects did transfer a lot of
the pattern of the first draft times in fact seems to indicate that subjects did transfer a lot of
Although in this experiment we did not have a very accurate measure of the planning time,
Although in this experiment we did not have a very accurate measure of the planning time,
both total problem solving time and the time for finishing the first drafts of the programs.
both total problem solving time and the time for finishing the first drafts of the programs.
The time saving is consistent for
The time saving is consistent for
From the quantitative results shown above, we clearly see that there is substantial
From the quantitative results shown above, we clearly see that there is substantial

3.3. Discussion

Insert figure 9 about here.

Program.
Thus, we can grossly identify the number of drifts with the number of semantic bugs in the
Transfer among languages

In this second experiment we attempted to decompose the data down into smaller units to
experiment we collected only rather gross measures of time saving in writing a program.
This is a further experiment on transfer between LISP and PROLOG. In the first

A FURTHER STUDY

4. EXPERIMENT 2: TRANSFER BETWEEN LISP AND PROLOG --

At the syntactic level, due to switching back and forth between programming in LISP
and in PROLOG, it appeared that subjects did get some syntactic interference between
programming in these two languages, in both coding and debugging processes. For
example, they would forget to use the correct form of list structure representation -- that is,
semantically.

minor syntactic interference was overwhelmed by the positive transfer at the more
problems, they were more likely to produce such syntactic interference. Overall, such
less interference seemed to occur when some of them were getting bored in later
is [a, b, c] in PROLOG. However, it appeared that the more attentive subjects were, the
instead of (a, b, c) in LISP, they might use (a, b, c). (The correct list structure representation -- that is,
example, they would forget to use the correct form of list structure representation -- that is,
Accompanying this level as problem-level or pre-algorithmic transfer:
syntactic algorithms in different languages and yet still get positive transfer! and we will refer transfer
problems, In later experiments, we will again witness such cases of using different
understanding. In fact, two subjects in this experiment changed their algorithms on four
the two languages and yet they may still transfer something which is the result of problem
4 types of transfer. For this type of transfer, subjects may use rather different algorithms in
occur before algorithmic planning! In other words, its level is higher than the above two
Another possibility of transfer other than the above two is a kind of transfer which may
of Transfer.

collected fine-grained measures of time saving and to attempt a detailed analysis of the locus
of the timing editor (see the two shadowed boxes in Figure 5). This editor enabled us to
when the editors the editing window to go to the debugging window he was asked to turn
on to edit or display a program he was requested to turn on the timing editor, and each time
onto to edit or display a program he was requested to turn on the timing editor, and each time
of interactions between the subject and the computer. Each time when the subject began
designed a special editor based on EMACS, this editor was used to time-stamp each step
experiment (see Figure 5). However, here instead of using the standard EMACS editor we

Procedure. Almost the same procedure as used in Experiment 1 was followed in this

in the order of actual presentation.

Materials. The six problems used in this experiment were Problems/Fibonacci equation,

except that two additional problems were used here.

design. Basically, the design was the same as that in Experiment 1 (see Figure 4)

the subjects were relatively more proficient in LISP than in PROLOG.

while that for PROLOG skill was only 3.4. Thus, again as in Experiment 1, on average,
supposed mean GRE/SAT scores were 760. The mean self-ratings of LISP skill was 4.14,
subjects gathered from Questionnaire, the
taking part in the experiment. From the information gathered from Questionnaire, the
among them 3 were undergraduate and 5 were graduates. They were reimbursed for.

Subjects. 8 subjects were involved in this experiment. They were all from CMU.

4.1. Method

In this experiment to give further observations of transfer, allow greater localization of the transfer. Also, two additional problems were incorporated.
Within it, although some components of algorithm selection and planning may also be involved, the first-draft thinking time is largely reflects the process of problem understanding. This measure is only the first portion of what we refer to as pre-programming time. Moreover, in addition, we also could further decompose the thinking, key-striking and debugging time into first-draft (initial-draft) time and revision (subsequent-draft) time, this could be done by considering the transitions made between the two windows. Furthermore, as discussed by Singley & Anderson (1986) and Katz (1988), we further decomposed the programming time into thinking time and key-striking time (execution time). From Singley & Anderson (1986) and Katz (1988), we further decomposed the time spent either in the editing window or in the debugging window. Taking the method for the time measures, we dissected the problem solving time for each problem into two types of measures were used in analyzing the data from this experiment -- the time measures and the numbers of program changes made by the subjects in programming.

4.2. Results
as much time on the second program.

Programs that showed the greatest transfer benefit were subjects only spending 23% of their time on the second program. While the absolute times were small, in terms of relative gains these per-

21.03, p < 0.02. The results revealed significant main effects due to transfer: F(1, 3) = 10.21, p < 0.05. The main effects of transfer were marginally significant: F(1, 3) = 5.27, p < 0.06 for the first-draft thinking time, the main effect due to transfer was not significant: F(1, 3) = 3.47, p < 0.1. However, for the first-draft keyshinking time, the main effect due to transfer was显著: F(1, 3) = 144.3, p < 0.01. For the solving time, the main effect due to transfer was significant: F(1, 3) = 472.12.

As in Experiment 1, transfer X language X problem (2 x 2 x 6) pattern ANOVAs were performed on the separated data for the six individual problems. For the total problem solving time, the mean times at various levels of localization -- namely, the various levels of decomposition of the problem solving time. The data presented were averaged over both the programs and the two groups. From the figure we can see that thinking and keyshinking, but no significant transfer manifested in the first-draft keyshinking.

Figure 10 presents the mean times at various levels of localization -- namely, the
We will discuss these cases in detail in the GENERAL DISCUSSION section. Again, these cases demonstrated the high-level transfer as discussed in Experiment 1. Algorithms and/or styles in LISP and PROLOG programming were for the same problems. Also, as in Experiment 1, there were several cases of the same subjects using different

design phases. Subjects took longer in designing the algorithm (the first draft time) and had less complete algorithms (the number of drafts) in the first language than in the second.

This is consistent with our identification of the transfer with the algorithm revisions that are necessary. The effect is in the planning of the first draft and in the number of subsequent revisions that saving (again about 37% total time saving). Furthermore, this experiment indicated that this experiment further indicated this transfer was positive and substantial in terms of time and performance demonstrated transfer between LISP and PROLOG programming.

4.3. DISCUSSION

Inset Figure 11 about here

Of the fewer total drafts, the results are also presented in Figure 11. The results indicated that there was no transfer for the percent time; that is, the transfer that occurred on the rest drafts was mainly the result of the more drafts. Figure 11 presents the results of this measure. The results revealed significant main effects due to transfer (F(1, 3) = 35.33, p < .01). As in Experiment 1, we also calculated the mean percent time for the rest drafts. Again, a three-way transfer X language X problem ANOVA was performed on the data of more drafts made the more rest-draft time. Figure 11 presents the results of this measure. Related to the numbers of drafts subjects made in the course of programming, this distribution was was distributed over the thinking, keying, and debugging times. This distribution was For the rest-draft times, there was also substantial transfer manifested; this transfer
used on the ANDREW system were COMMONLISP and UNIX-PASCAL. The actual versions of the languages also illustrated in Figure 5 were in the debugging window LISP and PASCAL (instead of Prolog). Almost the same as in Experiment 2, the procedure of this experiment is

Procedure.

Except that Prolog was substituted by PASCAL and that only five problems were used in this experiment. The design of this experiment was almost the same as that in Experiment 2.

Materials. The five problems used in this experiment were Fibonacci function, add-

Design. The design of this experiment was 3.71.

PASCAL was 3.72. The mean self-ratings of LISP proficiency was 3.29, while the mean score of the

Subjects. Again, 8 CMU students -- 3 undergraduates and 5 graduates -- were involved in this experiment. They were reimbursed for participating in the experiment.

Method

Language were replicated. Some of Kart's (1988) results on transfer between these two

This experiment was basically an extension of the second experiment to transfer

5. EXPERIMENT 3: TRANSFER BETWEEN LISP AND PASCAL
Comparing language whereas LISp is interactive (that relatively more faciliilates program data pattern conform) to the transfer occurred on rest-datas. However, as PASCAL is a.

Figure 13 shows the results of this measure on the five individual problems. Generally, this refers to the savings of the numbers of program changes made in transfer programming.

Also as in Experiment 2, we would expect that the transfer occurred on rest-datas was

\[ p > 0.01. \]

The pre-programming time, the main effect due to transfer was significant: \( F(1, 3) = 43.76 \), for time, the main effect due to transfer was marginal significant: \( F(1, 3) = 20.15, p > 0.05 \). For the first-draft debugging transfer was marginal significant: \( F(1, 3) = 8.84, p > 0.06 \). For the first-draft keyshifting time, the main effect due to transfer was also significant: \( F(1, 3) = 12.49, p > 0.01 \). However, for the first-draft thinking time, the main effect due to transfer was also significant: \( F(1, 3) = 74.04, p > 0.01 \).

As in the previous experiment, transfer x language x problem (2 x 2 x 5) pattern ANOVAs were performed on the separated data for the five individual problems. For the inserted Figure 12 about here.


data analysis. The averaged times at various levels of localizations are presented in Figure 12. As in the situation of transfer between LISP and PHOLOG, here we also see that.

\[ \text{Results} \]

5.2.
6.1. Three Levels of Transfer in Programming

In explaining knowledge transfer, the theory of identical elements has been proposed very early in the century (Thurstone & Woodworth, 1917). Basically, the theory posits that if the common elements shared by two domains of knowledge that enables the

6. GENERAL DISCUSSION

...
The algorithmic level refers to the knowledge that is domain-language-dependent in that

of instances of transfer at this level.

and function names from one language to another; Katz (1966) also documents a number
between list constructions in PROLOG and LISP. Subjects also transferred variable names
However, there was some transfer at this level. We have already noted the interference
lowest level and played a relatively minor role, positive or negative, in our transfer results.
The syntactic level is the algorithmic level, the algorithmic level, and the problem level. The syntactic level is the
these three levels may be called the
access programming in different languages. These three levels may be called the
To account for the above results, we propose that there are three levels of transfer

0.01.

problem, the analyses revealed significant main effect due to transfer: F(1, 9) = 43.10, p
language ANOVA (2 x 2) was performed on the data of total problem solving time for this
Nevertheless, on this problem the subjects still displayed a 27% time savings. A transfer X
languages, namely, all used recursive solutions in LISP and iterative solutions in PASCAL.
 languages; named, all used different algorithms to implement database-search problem (see Figure 3) in the two
used different algorithms to implement database-search problem from one language to another. For instance, in Experiment 3 all the subjects
programming from one language to another. For instance, in Experiment 3 all the transfer
structures, and programming styles, there was still quite a lot transfer manifested in
different languages by the same subjects appeared to be very different in algorithms, data
However, as noted earlier, although sometimes the programs written in different

PROLOG.

witnessed a great deal of transfer between PASCAL and LISP and between LISP and
light of the identical-element theory, it is not surprising that in the three experiments we
are indeed a number of commonalities shared by the three languages. Therefore, in the
similarities and differences among PASCAL, LISP, and PROLOG, we have seen that there
knowledge acquired in one domain to transfer to the other. In our earlier analyses on the
6.2. Declarative versus Procedural Transfer

and algorithm to a different language and algorithm. An algorithm for LISP or PASCAL, and this constraint could be transferred from one language to another. This problem constraint was implemented to designing the multiple authors of the same book. This problem constraint was important to designing the multiple authors of the same book. By the same author, but not the same discipline, subjects would realize in the first program that they would not have to have a system that represented multiple books by the same author but not the same discipline.

Database-Search problem (Figure 3) subjects would realize in the first program. For instance, in the transfer level, goes beyond reading the problem statement. For instance, in the transfer level, the declarative, subjects had to spend little time re-reading the problem statement. However, declarative in the pre-programming time are one manifestation of transfer at this problem level. The dramatic responsibility for this transfer is language-independence but problem-relied. The dramatic responsibility for this transfer is language-independence but problem-relied. The dramatic responsibility for this transfer is language-independence but problem-relied. The dramatic responsibility for this transfer is language-independence but problem-relied.

We believe this involves understanding the problem itself and the knowledge sometimes transfer was displayed even when a different algorithm was used in the two languages. However, as we noted all transfer could not be at the algorithmic level since there was less need for revision. Second language, and since they already had a worked out algorithm in one language, use the algorithm they had practiced in the first language to cut down the planning in the memory located in the first part of the planning time and number of drafts. Subjects were able to complete algorithms (not code) from one language to another. This is why the transfer was sometimes subjects used highly similar algorithms in the two languages; that is, subjects occasionally selected algorithms for LISP and PASCAL whereas iterative algorithms for PROLOG. However, we suspect much of our transfer was at this level. Most of the choosing recursive algorithms for LISP and PROLOG whereas iterative algorithms for

the particular algorithm chosen may be just appropriate for only certain languages (e.g.,
Transfer among Languages

However, the amount of transfer might be related to the closeness of the two languages. Programming languages as there usually would be some commonalities shared by them. We hypothesize that this can be generalized to transfer between any two PASCAL, we hypothesize that this can be generalized to transfer between LISP and PROLOG and between LISP and different languages can be drawn from this study.

The following conclusions concerning knowledge transfer across programming in

2. SUMMARY

The time and number of revisions but not in the execution time. The next solution, this hypothesis is consistent with the transfer being largely in the initial subjects are transferring their mental representation of one solution to seems declarative. Subjects are transferring their knowledge transferred productions does not seem to be applicable here. Rather, the knowledge transferred procedural form of programming skill. Thus, the explanation of transfer in terms of declarative procedural form of programming skill, it is unlikely that the locus of transfer can be in programmers in the three languages. It is unlikely that the locus of transfer can be in However, since our subjects in this study were already relatively competent


Successfully applied to transfer in the domain of computer-based text-editing (Singley & succeeds in applying to transfer in the domain of computer-based text-editing. This explanation of transfer has been a general form of representing procedural knowledge. This explanation of transfer has been a instantiation of the common elements shared by two skills in the form of production rules - a ACT explanation of transfer is an elaboration of the identical-elements theory. In fact, it recently extended the theory to encompass knowledge transfer as well. Essentially, the memory processes, problem solving, and learning. Singley & Anderson (1988) have knowledge. The theory has been successful in explaining a large set of empirical data on
of this work.

students, in the Department of Psychology at CMU, for their comments on various portions
of this work. We are grateful to Drs. Herbert Simon and Kurt Venlauf, and to many graduate

Acknowledgement. This research was supported by grant MDA 90 39K KO190 from

levels.

proposed that most of the transfer we observed occurred at the algorithmic and problem

different languages -- namely, the syntactic, algorithmic, and problem levels. It is also

3. We propose that there are three levels of transfer between programming in two

required for initializing that draft.

amount of time for constructing an initial draft of program and the number of revisions

2. The positive transfer among programming languages was mainly localized in the
REFERENCES
Thomlike, E. L. & Woodworth, R. S. (1901). The influence of improvement in one mental
Dissertation, Department of Psychology, Carnegie Melittion University.
Harvard University Press.
<table>
<thead>
<tr>
<th>Phyx Representation</th>
<th>Procedural Evaluation</th>
<th>Difference</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code> * <code>x</code></td>
<td>(intersect <code>x</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x</code> + <code>x</code></td>
<td>(union <code>x</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>record <code>x</code></td>
<td>(call <code>x</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>array <code>x</code> [n]</td>
<td>(call <code>x</code>)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- No allowed: `x` = [R, C, (R, G)]
- Allowed: `x` = (R, C, G)
- `x` = (R, C, Blue)
- Allowed: `x` = (R, C, Blue)
- Allowed: `x` = (R, C, Blue)
- Allowed: `x` = (R, C, Blue)
- Allowed: `x` = (R, C, Blue)

---

<table>
<thead>
<tr>
<th>Phyx Representation</th>
<th>Procedural Evaluation</th>
<th>Difference</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1. A comparison of PROLOG, LISP, and PASCAL -- major communities and differences among them.**
The Fibonacci function is defined as follows:

\[ f(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ f(n-1) + f(n-2) & \text{if } n > 1. \end{cases} \]

Figure 2. First example of programing problems for illustrating the similarities and differences of the three languages -- programs in PASCAL, LISP, and PROLOG for evaluating the Fibonacci function.
Suppose there is the following database:

- McClelland, Parallel Distributed Processing, 1986.

Write programs in PASCAL, LISP, and PROLOG to count the number of books by a given author.

**Model PASCAL programs:**

**Iterative solution — type**

```pascal
string = array[1..10] of char;
book_type = record
  name: string,
  title: string,
  year: integer
end;

var
  database: file of book_type;

begin
  n := 0;
  read(database, a_book);
  while not eof(database) do
  begin
    if a_book.name = author then
      n := n + 1;
    end;
  end;
  writeln('No. of books:', n);
end;
```

**Recursive solution — type**

```pascal
string = array[1..10] of char;
book_type = record
  name: string,
  title: string,
  year: integer
end;

var
  database: file of book_type;

begin
  n := 0;
  read(database, a_book);
  if a_book.name = author then
    n := n + 1;
  end;
  read(database, a_book);
  while not eof(database) do
  begin
    n := n + 1;
    read(database, a_book);
  end;
  writeln('No. of books:', n);
end;
```

**Model LISP programs:**

**Iterative solution — Using association list**

```lisp
(defvar books (list '(*author* "J.K. Rowling")
  '(*title* "Harry Potter and the Half Blood Prince")
  '(*year* 2005))
(defvar countbooks (fn lambda
  (author)
  (length
   (filter
    (fn lambda
      (eq
       *author*
       author)
    books))))
(defvar n (countbooks "J.K. Rowling"))
(n)
```

**Recursive solution — Using association list**

```lisp
(defvar books (list '(*author* "J.K. Rowling")
  '(*title* "Harry Potter and the Half Blood Prince")
  '(*year* 2005))
(defvar countbooks (fn lambda
  (author)
  (if
   (eq
    *author*
    author)
   (1)
   (count-books
    author
    *books*))))
(defvar n (count-books "J.K. Rowling"))
(n)
```

**Model PROLOG programs:**

**Declarative solution**

```prolog
setof(X, books(Book), books_by(Author, L, N)).
no_of_books(Author, N).
length(L, N).
```

**Recursive solution**

```prolog
books_by_Author(T, books(Author, L), N). books_by_Author(T, books(Author, L), N).
books_by_Author(T, books(Author, L), N).
```

**Iterative solution**

```prolog
books_by_Author(Author, books, N).
books_by_Author(Author, books, N).
books_by_Author(Author, books, N).
```

**NOTE:** in some COMMONLISP versions, this function can be written more compactly:
<table>
<thead>
<tr>
<th>Group</th>
<th>Prob-1</th>
<th>Prob-2</th>
<th>Prob-3</th>
<th>Prob-4</th>
<th>Prob-5</th>
<th>Prob-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LISP→PROLOG**</td>
<td>P→L</td>
<td>L→P</td>
<td>L→P</td>
<td>P→L</td>
<td>L→P</td>
</tr>
<tr>
<td>2</td>
<td>PROLOG→LISP**</td>
<td>L→P</td>
<td>P→L</td>
<td>P→L</td>
<td>L→P</td>
<td>L→P</td>
</tr>
</tbody>
</table>

NOTE: This figure presents the design which was common to the three experiments. However, 4 problems were used in Experiment 1; 6 in Experiment 2; and 5 in Experiment 3. **: Instead of PROLOG, it was PASCAL in Experiment 3.
NOTE: This figure presents the procedure which was common to all the three experiments. However, the timing-editor (as shown in the two shadowed boxes) was only used in Experiment 2 and 3; in Experiment 1, standard EMACS editor was used.
Table 1: Mean Total Problem Solving Time (in minutes)

<table>
<thead>
<tr>
<th>First Program</th>
<th>Lisp Programming</th>
<th>Prolog Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.4</td>
<td>26.7</td>
<td>18.8</td>
</tr>
<tr>
<td>13.4</td>
<td>24.4</td>
<td></td>
</tr>
</tbody>
</table>

Across two groups and four problems.

Figure 6: The mean problem solving time by two Groups of subjects in Experiment 1.
Figure 7. The mean first-draft time by two groups of subjects in Experiment 1, averaged across subjects within group and across two groups and four problems.
Figure 8. The mean rest-draft time, as the difference between the total time and the first-draft time, by two groups of subjects in Experiment I, averaged across subjects within group and across two rows and four problems.
Figure 9. Results of Experiment I on transfer between Prolog and in LISP and in Prolog -- No. of drafts made by Group-2 closely corresponding to semantic mistakes made in Programming.

Subjects in Group 2.

Subjects in Group 1.
Figure II. Results of Experiment 2 on transfer between programming in PROLOG and in LISP.
PASCAL -- LISP or LISP -- PASCAL

The data presented here are averaged across both problems and the two groups. The diagonals represent the conditions as either First or Second.

Figure 12. Results of Experiment 3 Time measures (in seconds) at various levels of localization in transfer between programming in PASCAL and in LISP.
Figure 13. Results of Experiment 3 on transfer between programming in LISP and in PASCAL.
The data presented here are averaged across both programs and the two groups. The diagonals represent the conditions as either PASCAL -- LISP or LISP -- PASCAL.

Figure 12. Results of Experiment 3 -- time measures (in seconds) at various levels of localization in transfer between programming in PASCAL and in LISP.