Examples of Variant Translators

Ralph E. Griswold

Department of Computer Science, The University of Arizona

Variant translators [1] provide a powerful method for transforming Icon programs. They can be used to add features, overload operators, or even translate Icon programs into an entirely different syntax. Some examples follow.

1. Modeling String Scanning

String scanning can be modeled by procedures using the following transformation:

$expr_1$? $expr_2$ $\rightarrow$ Escan(Bscan($expr_1$),$expr_2$)

Supplying the appropriate procedures is useful in understanding how scanning environments are maintained.

A variant translator to transform scanning expressions for this purpose consists of the following variant definitions:

```
Bques(x,y,z)         "Escan(Bscan("   x         "),"       z         ")"

Baugques(x,y,z)      x          " := Escan(Bscan("         x         "),"       z         ")"
```

Example Input

```
while line := read() do
   if line ? pattern1() then
      line ?:= pattern2() ? pattern3()
```

Output

```
while line := read() do if Escan(Bscan(line),pattern1())

then Escan(Bscan(line := Escan(Bscan(line),pattern2())),pattern3());
```

(This output has been edited to add a newline for formatting purposes.)

2. EBCDIC Digraphs

The variant translator given by the following definitions replaces braces and brackets in Icon programs by their EBCDIC digraph equivalents:

Arglist3(x)	x	"$<$>"				
Case(u,v,w,x,y,z)	"case "	v	" of $(\n"		y	"\n$)"
Brace(x,y,z)	"$(\n"	y	"\n$)"			
Brack(x,y,z)	"$<"	y	"$>"			
Pdco0(x,y,z)	x	"$("	"$)"			
Pdco1(w,x,y,z)	w	"$("	y	"$)"		
Section(u,v,w,x,y,z)	u	"$<"	w	x	y	"$>"
Subscript(w,x,y,z)	w	"$<"	y	"$>"		

Example Input

```
text := []

while line := read() do {

    write(line)

    put(text,line)

    }
```

Output

```
text := $<$>;

while line := read() do $(

write(line);

put(text,line)

$);
```

3. Postfix Operations

Casting Icon expressions in postfix form is helpful in understanding the order of evaluation and backtracking to resume suspended generators (see pages pp 81-82 of the second edition of the *The Icon Programming Language*). The following variant translator produces output that is useful in preparing tutorial examples:

<bop>(x,y,z)	"("	x	","	z	")<bop>"
<aop>(x,y,z)	"("	x	","	z	")<aop>"
<bcs>(x,y,z)	"("	x	","	z	")<bcs>"
<uop>(x,y)	"("	y	")<uop>"		
<ucs>(x,y)	"("	y	")<ucs>"		
Apply(x,y,z)	"("	x	","	z	")!"
Field(x,y,z)	"("	x	","	z	")."
Invoke(w,x,y,z)	"("	y	")"	w	

Example Input

```
every write(find(s1,s2) = find(s3,s4))
```

Output

```
every (((s1,s2)find,(s3,s4)find)=)write;
```

4. Evaluation Sandwiches

So-called ‘‘evaluation sandwiches’’ are used to put wrappers around expressions. A wrapper can record the evaluation of the enclosed expression, its failure, the results it produces, its resumption, and so forth. Wrappers usually are implemented by procedures, which are linked with the transformed program. The follow variant definitions insert calls to I__(x,y) and X__(x,y) around every expression, where x and y and the column and lines numbers of the enclosed expression:

```
%{

#define In(x) cat(3,q("2(I__("),Locer(x),q("),"))

#define Out(x) cat(3,q(",X__("),Locer(x),q("))"))

%}
```

<aop>(x,y,z)	In(y)	x	" <aop> "		z	Out(y)		
<bcs>(x,y,z)	In(y)	x	" <bcs> "		z	Out(y)		
<bop>(x,y,z)	In(y)	x	" <bop> "		z	Out(y)		
<ucs>(x,y)	In(x)	"<ucs>"	y	Out(x)				
<uop>(x,y)	In(x)	"<uop>"	y	Out(x)				
Apply(x,y,z)	In(x)	x	"!"	z	Out(x)			
Brace(x,y,z)	In(x)	"{\n"	y	"\n}"	Out(x)			
Brack(x,y,z)	In(x)	"["	y	"]"	Out(x)			
Break(x,y)	In(x)	"break "	y	Out(x)				
Case(u,v,w,x,y,z)	In(u)	"case "	v	" of {\n"	y	"\n}"	Out(u)	
Cclause0(x,y,z)	"default:"		In(y)	z	Out(y)			
Cclause1(x,y,z)	In(x)	x	Out(x)	":"	In(z)	z	Out(z)	
Create(x,y)	In(x)	"create "	y	Out(x)				
Every0(x,y)	In(x)	"every "	y	Out(x)				
Every1(w,x,y,z)	In(w)	"every "	x	" do "	z	Out(w)		
Fail(x)	In(x)	"fail"	Out(x)					
Field(x,y,z)	In(y)	x	"."	z	Out(y)			
If0(w,x,y,z)	In(w)	"if "	x	" then "	z	Out(w)		
If1(u,v,w,x,y,z)	In(u)	"if "	v	" then "	x	" else "	z	Out(u)
Invoke(w,x,y,z)	In(x)	w	"("	y	")"	Out(x)		
Keyword(x,y)	In(x)	"&"	y	Out(x)				
Kfail(x,y)	In(x)	"&fail"	Out(x)					
Next(x)	In(x)	"next "	Out(x)					
Paren(x,y,z)	In(x)	"("	y	")"	Out(x)			
Pdco0(x,y,z)	In(y)	x	"{"	"}"	Out(y)			
Pdco1(w,x,y,z)	In(x)	w	"{"	y	"}"	Out(x)		
Repeat(x,y)	In(x)	"repeat "		y	Out(x)			

Return(x,y)	In(x)	"return "	y	Out(x)				
Section(u,v,w,x,y,z)	In(v)	u	"["	w	x	y	"]"	Out(v)
Subscript(w,x,y,z)	In(x)	w	"["	y	"]"	Out(x)		
Suspend0(x,y)	In(x)	"suspend "		y	Out(x)			
Suspend1(w,x,y,z)	In(w)	"suspend "		x	" do "	z	Out(w)	
To0(x,y,z)	In(y)	x	" to "	z	Out(y)			
To1(v,w,x,y,z)	In(w)	v	" to "	x	" by "	z	Out(w)	
Unot(x,y)	In(x)	"not "	y	Out(x)				
Until0(x,y)	In(x)	"until "	y	Out(x)				
Until1(w,x,y,z)	In(w)	"until "	x	" do "	z	Out(w)		
While0(x,y)	In(x)	"while "	y	Out(x)				
While1(w,x,y,z)	In(w)	"while "	x	" do "	z	Out(w)		

Note the use of macros to simplify the textual specification.

This variant translator requires a C function, given in variant.c, for producing the column and line numbers:

```
#include "h/config.h"
#include "itran/tree.h"
#include "itran/trans.h"
#include "itran/tproto.h"

nodeptr cat(), q();

nodeptr Locer(x)
nodeptr x;
   {
    /* 100 characters of free space should be plenty */
    if (stre – strf <= 100)
                    tsyserr("out of free space");
     sprintf(strf,"%d,%d",Col(x),Line(x));
     return q(putid(strlen(strf)+1));
   }
```

Example Input

```
procedure main()
   every i := 1 to 10 do {
      write("f(i)=", f(i))
      }
end
```

Output

```
procedure main()
2(I__(4,2),every 2(I__(12,2),i := 2(I__(17,2),1 to 10,X__(17,2)),X__(12,2)) do 2(I__(26,2),{
2(I__(12,3),write("f(i)=",2(I__(23,3),f(i),X__(23,3))),X__(12,3))
},X__(26,2)),X__(4,2));
end
```

References

1. R. E. Griswold and K. Walker, *Variant Translators for Version 8 of Icon*, The Univ. of Arizona Tech. Rep. 90-4, 1990.