Reviving Pacbase COBOL-Generated Code

Jean-Philippe Bernardy
jp@raincode.com

February 14, 2002

1 Introduction

VisualAge Pacbase[1] is an application development offering developed and marketed by IBM. Interestingly enough, Pacbase generates COBOL code.

Some people wish to get rid of Pacbase, and work further with plain COBOL. One could think of using that COBOL output as is, but Pacbase-generated code is not supposed to be maintained, and is thus awfully crippled. It never uses structured statements, but prefers using lots of NEXT SENTENCE and GOTOs.

2 Approach

Our method is to apply generic[1] reformating techniques to the generated COBOL code to improve its maintainability to a satisfying level.

The reformating takes place as a suite of passes on the source, among which: cosmetic improvements (indentation, adding END-IFs, etc.), removal of dead code, detection of loops (PERFORMs), detection of IF-like branches, removal of superfluous labels, detection of IF-ELSE branches, detection of EVALUATE-like branches and detection of VARYING-like loop increments.

Each step takes (or could take) advantage of the result of the other steps, slowly rising the "abstraction level" of the code.

3 Technology and techniques

In order to perform those transformations of the source, we use "compiler-like" techniques, that is:

- syntactic analysis of input code
- semantic analysis
- matching of patterns, based on both syntax and semantics

The specificity lies in the fact that we need to *patch the source*. A compiler merely deals with its internal structures, while we need a human maintainable output.

All these basic features were pre-existing in our enabling technology, RainCode [2], which allowed us to focus on the actual patterns to recognize, and patches to apply. The application layer above RainCode for this specific purpose is concize enough, less than 2000 lines.

The following example shows what we actually do:

[1] they can be applied to any COBOL code

```
IF A = 2 THEN
  MOVE 'Y' TO STATUS
END-IF
IF A NOT = 2 THEN
  MOVE 'N' TO STATUS
END-IF.
```

becomes

```
IF A = 2 THEN
  MOVE 'Y' TO STATUS
ELSE
  MOVE 'N' TO STATUS
END-IF
```

To perform such a patch, we have to recognize: two consecutive IFs, having opposite conditions and having no ELSE clause. We also check that the body of first IF does not have side effects on its condition.

Note that, while a COBOL programmer would never write such code, it arises rather commonly in our samples. This is because it is easier to write such code in Pacbase (due to its syntax). Moreover, applying the previous passes reveals structures that might have been hidden in the original Pacbase source.

A similar technique is described in [3]. Our contribution lies in pattern recognition that relies on side effect analysis. Our approach is also more general, because it addresses a wider range of abstractions, like EVALUATEs or VARYING clauses of PERFORMs.

4 End to end example

Here is an example of the result of the process, on a non-trivial piece of code.

```
F05DC.
    MOVE        1                       TO ICATR.          (1)
    GO TO     F05DC-B.
F05DC-A.
    ADD         1                       TO ICATR.          (2)
F05DC-B.
    IF          10                      <  ICATR THEN      (3)
      GO TO     F05DC-FN
    END-IF.
    IF    CATX(ICATR) = '0' THEN                           (4)
      NEXT SENTENCE
    ELSE
      GOTO F05DC-C
    END-IF.
    MOVE 'X' TO CATM(ICATR).                               (5)
F05DC-C.
    IF    CATX(ICATR) NOT = '0' THEN                       (6)
       NEXT SENTENCE
    ELSE
      GOTO F05DC-D
    END-IF.
    MOVE 'Y' TO CATM(ICATR).                               (7)
```

```
F05DC-D.
    GO TO F05DC-A.                                          (8)
F05DC-FN.
    EXIT.
```

We discover a PERFORM structure, conditionned by the test 10 < ICATR. We find out that the loop counter is ICATR, starting at 1 and stepping by 1.

```
          PERFORM WITH TEST BEFORE                          (8)
            VARYING ICATR FROM 1                            (1)
                          BY 1                              (2)
            UNTIL 10 < ICATR                                (3)
            IF CATX(ICATR) = '0' THEN                       (4)
              MOVE 'X' TO CATM(ICATR)                       (5)
            ELSE                                            (6)
              MOVE 'Y' TO CATM(ICATR)                       (7)
            END-IF
          END-PERFORM.
```

Notice that every GOTO has been removed, and that the total number of statements has fallen from 15 to 4.

5 Results and metrics

Our output COBOL code is rather good. It shows most of the original Pacbase structures correctly translated into their COBOL counterparts.

Basically, as a measurable metric of COBOL code improvement, we count the number of GOTO statements.

For batch programs, we remove from 80 to 95 percents of GOTO statements.

The remaining GOTOs are due to too complex control flow. They cannot be removed without either duplicating code or introducing temporary flag-variables, which we have chosen to avoid.

In online programs, the control flow is much more intricate that in batch ones. We hence obtain a less satisfying rate of about 60 percents of removed GOTOs.

6 Ongoing work

We are currently working on the detection of duplicated code, which can come from Pacbase standard generation process, to factor them in copybooks.

We are also looking for a way to restructure online programs, in order to simplify their control flow and thus enhance the impact of our method.

References

[1] http://www-4.ibm.com/software/ad/vapacbase/.

[2] http://www.raincode.com.

[3] Mark van den Brand, Alex Sellink, and Chris Verhoef. Control flow normalization for COBOL/CICS legacy systems. In *2nd Euromicro Working Conference on Software Maintenance and Reengineering.*