Technical Disclosure Commons

Defensive Publications Series

November 2020

Replacement of Goto Statements During Code Migration

Paneendra BA

Paul McReynolds

Abhay Garg

Arijit De

Alex Ginet

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
BA, Paneendra; McReynolds, Paul; Garg, Abhay; De, Arijit; and Ginet, Alex, "Replacement of Goto Statements During Code Migration", Technical Disclosure Commons, (November 12, 2020)
https://www.tdcommons.org/dpubs_series/3762

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Replacement of Goto Statements During Code Migration

ABSTRACT

This disclosure describes code conversion techniques for replacement of Goto statements in program code in a source language with equivalent constructs in a target language. Different types of equivalent code constructs are utilized to replace Goto statements in the source language, depending on the use of the Goto statement in the original code that is to be converted. Another option is to introduce a method-like syntax in the language. Backward jumps and forward jumps are handled by the addition of new do-while loops that start from or end at a label associated with the Goto statement. In some cases, code blocks are divided into closure methods such that every loop that includes a nested label is converted into a recursive method. On Error Goto statements are converted by utilizing try-catch blocks.

KEYWORDS

- Visual Basic
- JavaScript
- Goto statement
- Code migration
- Legacy code
- Recursion

BACKGROUND

During migration of legacy computer applications, code written in a particular programming language may have to be converted (translated) into code in a different programming language. Automatic tools are available for such code migration and can efficiently convert code from one language to another.

Some source programming languages support the use of Goto statements/keywords. Goto statements are jump statements used to jump from one point in the code to another. Some target programming languages do not support Goto statements, thereby posing a challenge for the direct translation of Goto statements when converting from a source language that supports such statements. Translation of Goto statements using standard constructs such as loops and closures that are provided in most programming languages can enable automatic migration of code that includes Goto statements.

DESCRIPTION

This disclosure describes techniques for automatic replacement of Goto statements in program code with equivalent constructs. The techniques can be utilized, for example, during migration of code from one programming language that supports Goto statements to another programming language that does not support Goto statements. An automated tool can implement the described techniques to convert code written in a particular language that supports a Goto statement, e.g., Basic, Visual Basic, etc. to a different programming language that may not support Goto statement use, e.g., JavaScript.

When a file that includes program code that is to be converted is received, certain conditions are verified to determine whether the file can actually be successfully converted. For example, it is verified that all ON statements in the code only include Error and Goto Keywords, that all label declarations occur after the Goto keywords, that all Goto calls have valid label names (e.g., all utilized labels are declared within a function where they are utilized), that label declarations are included at a top-level in a corresponding function, etc. If the conditions are not met and it is determined that the file is not convertible, a suitable message (e.g., “File not supported”) is provided to the user that attempts code conversion.

Different types of equivalent constructs are utilized to replace Goto statements, depending on the use of the Goto statement in the original code that is to be converted.

Syntactical support

Support for Goto statements can be added to a parser associated with a language to explicitly recognize Goto as a keyword. Another option is to introduce a method-like syntax in a language, e.g., Goto(label) which obviates the need for any parsing change, while providing support for Goto keywords in the language.

```
function test() {
  var a;
  while (true) {
    a = a + 1;
    if (a > 10) {
      goto label;
    }
  }
  label: return;
}
```

```
function test() {
  var a;
  while (true) {
    a = a + 1;
    if (a > 10) {
      goto(label);
    }
  }
  label: return;
}
```

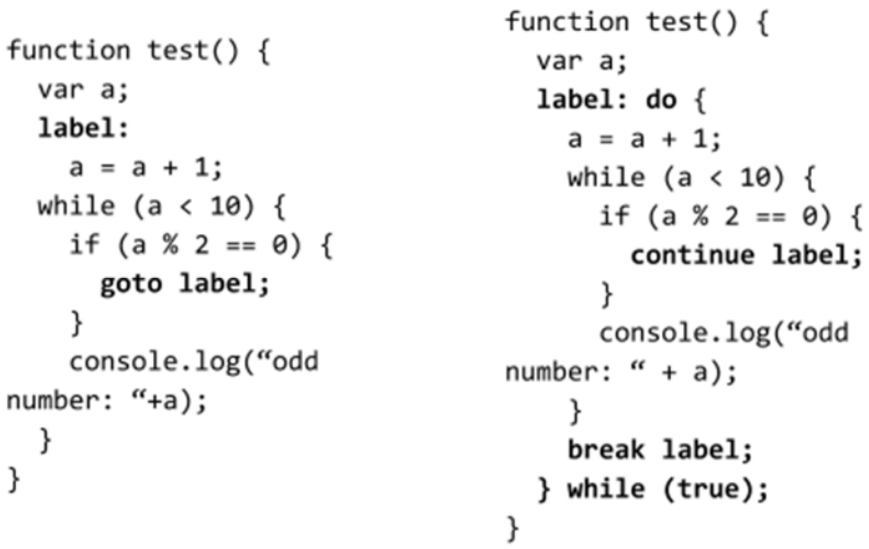
Fig. 1: Method-like syntax used to convert code (a) original code; (b) equivalent code

Fig. 1 depicts an example of conversion of code that includes a Goto keyword in a language that supports a Goto keyword to another language by utilizing a method-like syntax. Fig. 1(a) depicts code in an original programming language, while Fig. 1(b) depicts the equivalent translated code.

Backward Jump

A backward jump refers to a Goto statement in code where the label appears before the Goto statement. Conversion of instances of a backward jump is handled by the addition of a new

do-while loop starting from a label associated with the Goto statement that extends till the end of a current block.



```
function test() {
  var a;
  label:
    a = a + 1;
  while (a < 10) {
    if (a % 2 == 0) {
      goto label;
    }
    console.log("odd
number: "+a);
  }
}
```

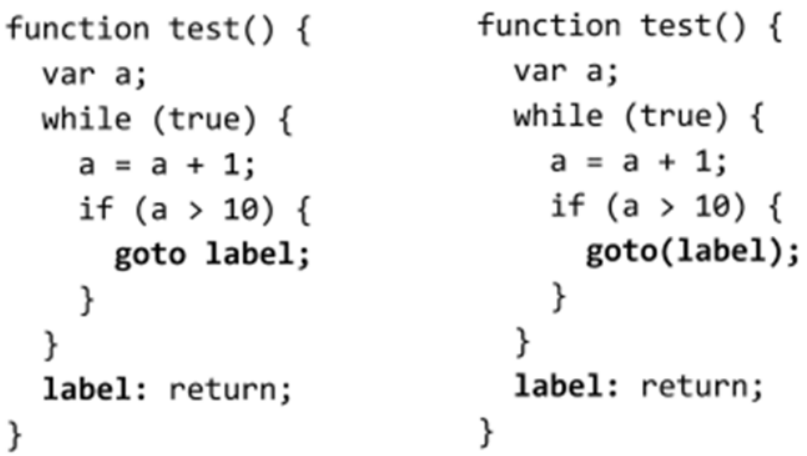
```
function test() {
  var a;
  label: do {
    a = a + 1;
    while (a < 10) {
      if (a % 2 == 0) {
        continue label;
      }
      console.log("odd
number: " + a);
    }
    break label;
  } while (true);
}
```

Fig. 2: Backward jump Goto statement; (a) original code; (b) equivalent code

Fig. 2 depicts an example of code conversion that includes a Goto statement that implements a backward jump. Fig. 2(a) depicts code in an original programming language, while Fig. 2(b) depicts equivalent translated code.

Forward Jump

Forward jump is when the Goto statement appears before the label in the method. A simple forward jump can be supported using a while loop that ends at the label and starts at the start of the current block.



```
function test() {
  var a;
  while (true) {
    a = a + 1;
    if (a > 10) {
      goto label;
    }
  }
  label: return;
}
```

```
function test() {
  var a;
  while (true) {
    a = a + 1;
    if (a > 10) {
      goto(label);
    }
  }
  label: return;
}
```

Fig. 3: Forward jump Goto statement; (a) original code; (b) equivalent code

Fig. 3 depicts an example of code conversion that includes a Goto statement that implements a forward jump. Fig. 3(a) depicts code in an original programming language, while Fig. 3(b) depicts equivalent translated code.

Use of Closures

Some sections of complex code may include placement of Goto statements that cannot be converted using do-while loops. In such situations, closure may be utilized to replace Goto keywords. Code blocks are divided into closure methods and chained up such that at the end of a block, a function representing the next block is called. This enables entry into any code block by calling a function, irrespective of the nesting level of the block. Every loop that includes a nested label is converted into a recursive method.

```
function complexGotos() {
  var a = 0;
  while (a < 10) {
    console.log("entered while
loop");
    insideLoop:
      a++;
    if (a < 5) {
      goto loopAgain;
    }
    console.log(a);
  }
  goto exitFunc;
  loopAgain: goto insideLoop;
  exitFunc:
console.log("exiting");
}
```

```
function complexGotos() {
  function whileLoop() {
    if (a < 10) {
      console.log("entered
while loop");
      insideLoop();
    }
  }
  function insideLoop() {
    a++;
    if (a < 5) {
      loopAgain();
      return;
    }
    console.log(a);
    whileLoop();
  }
  function loopAgain() {
    insideLoop();
  }
  function exitFunc() {
    console.log("exiting");
  }
  var a = 0;
  whileLoop();
  exitFunc();
}
```

Fig. 4: Utilization of closure methods; (a) original code; (b) equivalent code

Fig. 4 depicts an example of utilization of closure methods for replacement of Goto statements during code conversion. Fig. 4(a) depicts code in an original programming language, while Fig. 4(b) depicts equivalent translated code.

On Error Goto statements

Some programming languages such as Visual Basic for Applications (VBA), provide for exception handling during program execution by utilizing On Error statements that are

commonly followed by a "Goto label", "Goto 0”, etc. This statement specifies action(s) to be performed when further program statements cause an exception/error. In some implementations, On Error statements can be converted by utilizing *try-catch* blocks that are supported by some target programming languages. A *try* statement is utilized to define a block of code to be tested for errors while it is being executed, while a *catch* statement enables definition of a block of code to be executed, upon encountering an error in the *try* block.

For conversion of code that includes On Error Goto statements, a block of statements of the function are included into a *try* clause. A new variable (“err_handler”) is declared outside the *try* clause and assigned to an empty string. Upon encountering a label, the *try* clause is closed and a *catch* clause is invoked. The On Error goto statement is converted to a err_handler assignment statement. A label declaration is translated into a corresponding if statement and a corresponding block of statements is included as consequent statements within the if block.

```
Sub Func()
  Dim A as Integer
  On Error GoTo errHandler
  ...
  If <cond> Then
    ...
   On Error GoTo 0
  End If
  ...
  On Error GoTo errHandler
  ...
errHandler:
  <error handling code>
End Sub
```

```
function Func() {
var a;
var onErrorLabel = null;
try {
  onErrorLabel =
'errHandler';
  ...
  If (<cond>) {
    ...
    onErrorLabel = null;
  }
  ...
  onErrorLabel =
'errHandler';
  ...
} catch (err) {
  If (onErrorLabel == null)
{
    throw err;
  }  // else will always
continue to next statement
which is equivalent to
jumping to label
}
<error handling code>
}
```

Fig. 5: Handling of On Error Goto statements; (a) original code; (b) equivalent code

Fig. 5 depicts an example of handling of On Error Goto statements during code conversion. Fig. 5(a) depicts code in an original programming language that utilizes On Error statements, while Fig. 5(b) depicts equivalent translated code that utilizes try-catch blocks.

CONCLUSION

This disclosure describes code conversion techniques for replacement of Goto statements in program code in a source language with equivalent constructs in a target language. Different

types of equivalent code constructs are utilized to replace Goto statements in the source language, depending on the use of the Goto statement in the original code that is to be converted. Another option is to introduce a method-like syntax in the language. Backward jumps and forward jumps are handled by the addition of new do-while loops that start from or end at a label associated with the Goto statement. In some cases, code blocks are divided into closure methods such that every loop that includes a nested label is converted into a recursive method. On Error Goto statements are converted by utilizing try-catch blocks.

REFERENCES

1. Ceccato, Mariano & Tonella, Paolo & Matteotti, Christina. (2008). Goto Elimination Strategies in the Migration of Legacy Code to Java. *Proceedings of the European Conference on Software Maintenance and Reengineering*, CSMR. 53-62.10.1109/CSMR.2008.4493300, https://selab.fbk.eu/ceccato/papers/2008/csmr2008.pdf, last accessed 24 October 2020.
2. The Ideal Language has "goto," https://mortoray.com/2011/10/23/the-ideal-language-has-goto, last accessed 24 October 2020.
3. GOTO Elimination Algorithm, https://dzone.com/articles/goto-elimination-algorithm last accessed 24 October 2020.
4. Exploring version 1.10 - Structured Exception Handling https://www.vbmigration.com/Blog/post/2008/11/06/Exploring-version-110-Structured-Exception-Handling.aspx last accessed 24 October 2020.