Reconstructing "Scope Recovering using Bridge Parsing"

Benjamin Zill
Universität Koblenz-Landau
Software-Language-Engineering
http://softlang.wikidot.com/course:sle1516

Key references
Paper Title: Practical Scope Recovery Using Bridge Parsing
Authors: Emma Nilsson-Nymn, Torbjörn Ekman, Görel Hedin
http://dblp.uni-trier.de/db/conf/sle/sle2008.html

Building a Bridge Parser

A try, to reconstruct a Bridge Parser.
Based on the scientific research paper "Practical Scope Recovery Using Bridge Parsing"
from the SLE Confercence 2008.

1. **Introduction**
   - Goal
   - The Bridge Parser
2. **Given by the paper**
   - The Grammar
   - The Algorithms
3. **Let´s build a Bridge Parser**
   - The Tokenizer
   - The Bridge Builder
   - The Bridge Repairer

1. Introduction

So first of all, what we want to archieve with the "Bridge Parsing Technique"?

For instance, we have an incomplete source file of a programming language like java or c, espacially a language which seperates it´s scopes by braces, parenthesis and so on. In this source file is some kind of brace missing. Just like this example:

```
class c{
    void m(){
        int y;
    int x;

}
```

Now, with the Bridge Parser, we want to insert that missing brace at the right position of the source code, with respect to the indents. So the goal is to get a file like this:

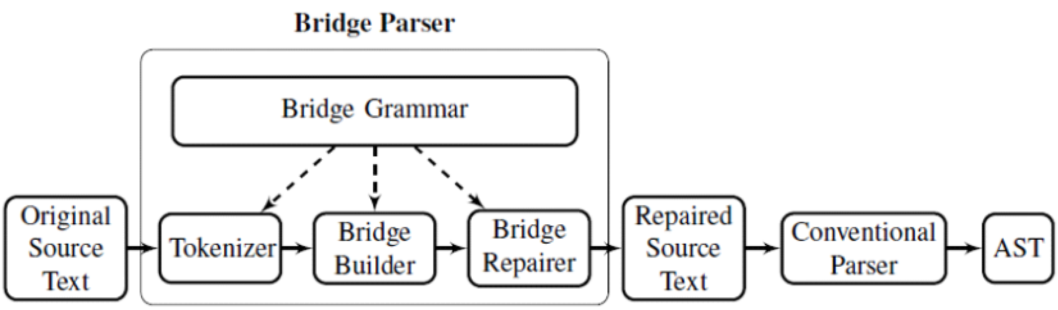
```
class c{
    void m(){
        int y;
    }
    int x;
}
```

The scope is recovered and the missing "right brace" is inserted, with respect to the indents of the file.

The Bridge Parser

The paper describes a parser, divided into three parts, givin by a Bridge Grammar:

1. Tokenizer: Creates a token list from the source text.
2. Bridge Builder: Creates Bridges between tokens on the same indent level. If he cannot build a bridge from a start token to an end token, the missing bridge is marked under a constructed bridge.
3. Bridge Repairer: Repairs the missing bridges. Insert the missing token at the right indent position.



pp. 100

There are also two algorithms:

1. One for the Bridge Builder, the Bridge Builder Algorithm, to iterate through the tokens and build the bridges or mark a missing bridge.
2. And for the Bridge Repairer, the Bridge Repairer Algorithm, to iterate through the tokens and inserts the missing token.

2. Givin by the paper

The Grammar

Lets have a look at the Bridge Grammar giving by the paper.

```
islands SOF, EOF, LBRACE, RBRACE, LPAREN, RPAREN
reefs INDENT(pos)

bridge from SOF to EOF
bridge from [a:INDENT LBRACE] to [b:INDENT RBRACE]
    when a.pos = b.pos {
    missing [RBRACE]
        find [c:INDENT] where (c.pos <= a.pos) insert after
    missing [LBRACE]
        find [c:INDENT] where (c.pos <= a.pos) insert after
}
bridge from [a:INDENT LPAREN] to [b:INDENT RPAREN]
    when a.pos = b.pos {
    missing [RPAREN]
        find [c:ISLAND] insert before
        find [c:INDENT] where (c.pos <= a.pos) insert after
    missing [LPAREN]
        find [c:ISLAND] insert before
        find [c:INDENT] where (b.pos <= c.pos) insert before
}
```

pp. 106

This grammar is divided into three parts. Tokenizer, Bridge Builder and Bridge Repairer definitions.

Tokenizer:

```
islands SOF, EOF, LBRACE, RBRACE, LPAREN, RPAREN
reefs INDENT(pos)
```

The tokenizer describes how to parse the input file. Specify the input into islands and water. For our example, we need to define braces and parenthesis as islands. Further we want to describe some kind of position handling for our indents. So that we can match bridges between our island tokens with the respect of the different indent levels given by the "incompletet" input file.

Now we are able to build a tokenlist from any "incomplete" input file.

Bridge Builder:

```
bridge from SOF to EOF
bridge from [a:INDENT LBRACE] to [b:INDENT RBRACE]
    when a.pos = b.pos {
```

The bridge builder definitions describes where to build bridges. Here we have two examples. The first, a bridge from "start of file" token to the "end of file" token.
The second example is a bridge between a "left brace" token and a "right brace" token, with additional information to build up a bridge, only when they are on the same indent position (when a.pos = b.pos).

Bridge Repairer:

```
missing [RBRACE]
    find [c:INDENT] where (c.pos <= a.pos) insert after
missing [LBRACE]
    find [c:INDENT] where (c.pos <= a.pos) insert after
```

The last part are the bridge repairer definitions. They describe, what to do if one specific token is missing. For instance we have a missing right brace, so we need to find a indent token, where the position level is lower equal the position level of the given left brace token. Then insert the new right brace token after this position token.

The Algorithms

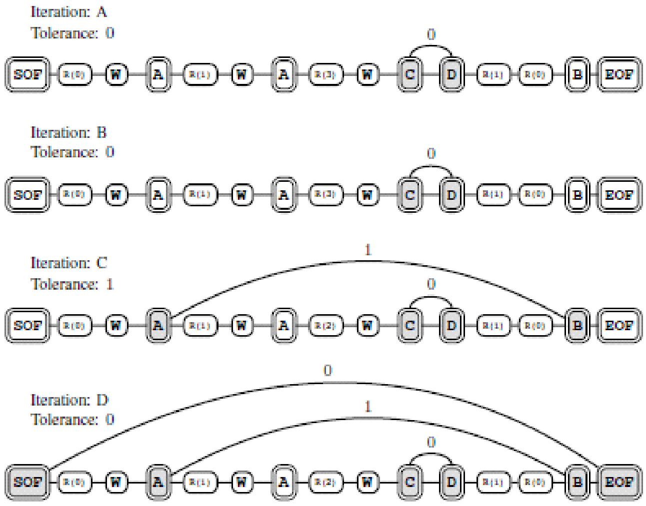
Bridge Builder Algorithm

```
BUILD-BRIDGES(sof)
tol ← 0
while ¬HAS-BRIDGE(sof)
   do start ← sof
      change ← FALSE
      while start ≠ NIL
          do end ← NEXT-UNMATCHED-ISLAND(start, tol)
             if BRIDGE-MATCH(start, end)
               then BUILD-BRIDGE(start, end)
                    change ← TRUE
                    start ← NEXT-UNMATCHED-START-ISLAND(end)
               else start ← NEXT-UNMATCHED-START-ISLAND(start)
      if ¬ change
        then tol ← tol + 1
        else if tol > 0
               then tol ← 0
```

The bridge builder algorithm will iterate through the whole token list and tries to match start islands with an end island. For instance a left brace with a right brace. If in one iteration (through the whole token list), there was no bridge built, the tolerance will increase and the next bridge that will be build over the unmatched token will have the increased tolerance. The algorithm will run until the start of file and end of file has a bridge.

pp. 104

Example from the book



pp. 103

Bridge Repairer Algorithm

```
BRIDGE-REPAIRER(bridge)
start ← START(bridge)
end ← END(bridge)
island ← NEXT-ISLAND(start)
while island ≠ end
    do if ¬HAS-BRIDGE(island)
          then if START-OF-BRIDGE(island)
                  then MEND-RIGHT(island, end)
                  else MEND-LEFT(start, island)
       bridge ← BRIDGE(island)
       BRIDGE-REPAIRER(bridge)
       island ← NEXT-ISLAND(END(bridge))
```

The bridge repairer algorithm get a bridge with a tolerance above zero, which means, that under that bridge, there is at least one broken bridge. It also iterates through the tokens until he finds the unmatched token with no bridge. If an end bridge token is missing, the MEND-RIGHT algorithm is called. If a start island is missing, the MEND-LEFT algorithm is called.

```
MEND-RIGHT(broken, end)
node ← NEXT(broken)
while node ≠ end
    do if HAS-BRIDGE(node)
          then node ← NEXT(BRIDGE-END(node))
          else if POSSIBLE-CONSTRUCTION-SITE(broken, node)
                  then CONSTRUCT-ISLAND-AND-BRIDGE(broken, node)
                       return
          else node ← NEXT(node)
CONSTRUCT-ISLAND-AND-BRIDGE(broken, PREVIOUS(end))
```

3. Let´s build a Bridge Parser

Tokenizer

For our Bridge Parser we use ANTLR to parse the incomplete input file to create the needed token list.

Tokens like open or close a scope are represented as islands. Also left and right Parenthesis. Everything else is "Water" and not interesting for us.

```
islands SOF, EOF, LBRACE, RBRACE, LPAREN, RPAREN
reefs INDENT(pos)
```

So our ANTLR grammar can look like this:

```
prog:              token* EOF;

token:          water|island|ref|newline;
ref:        TAB;
island:             lbrace
                    | rbrace
                    | lparenthesis
                    | rparenthesis
                    //| semicolon
                    //| ref
                   // | newline
                    ;

water:              ID | INT | OTHER| semicolon;

lbrace:          '{';
rbrace:         '}';
lparenthesis:    '(';
rparenthesis:   ')';


OTHER:              '=';
ID:                 [a-zA-Z]+;

INT:                [0-9]+;

semicolon:          ';';
newline:  '\r'? '\n';
TAB:                 [ \t]+;
//WS:   ' ' -> skip;
```

Our start rule is "prog". It defines that our input file is build from tokens and an "End of File" token. "token" can be "water" or "islands".

Our "island" tokens are the braces and parenthesis and "water" tokens are Identifiers, like strings, or integers.

Further more, some maps are declared in the grammar, and will build up in the Parser file. This will give information, what bridges can be build and what island is an start island.

```
@parser::members{

 public static HashMap<String, String> islandMap = new HashMap<String, String>(){{
        put("SOF", "EOF");
        put("{", "}");
        put("(", ")");
    }};


 public static HashMap<String, String> refMap = new HashMap<String, String>(){{
    put("TAB", "TAB");
 }};
```

BridgeToken

A token class is created, where different informations are stored.

- Is the token an island
- Is the token a start island
- Text of the token
- Position of the token
- And for later bridge construction, has the token a bridge and a bridge object

The Tokenizer

The Tokenizer class uses the ANLTR Visitor to visit the parse tree of the input text file. For every visit of the island and water rule a token is constructed and added to a double linked list.

For example, the visitLbrace method:

```
@Override
public Void visitIsland(BridgeParser.IslandContext ctx) {
    tokenList.addLast(new BridgeToken(ctx.getText(), position));
    return null;
}
```

The Bridge Builder

The Bridges

To create the bridges, for the bridge builder algorithm, a Bridge class is constructed. A algorithm will iterates threw the nodes and checks, if there is any possibility to build a bridge.

The Bridge Builder class gets match methods from match functions from the parser. Which we declare in the grammar.

```
bridge from SOF to EOF
bridge from [a:INDENT LBRACE] to [b:INDENT RBRACE]
    when a.pos = b.pos {
```

```
public static boolean matchBrace(String a, String b, int apos, int bpos){ return (a.equals(b) && apos==bpos); }
public static boolean machtParenthesis(String a, String b,int apos, int bpos){ return (a.equals(b) && apos==bpos);}
```

The buildBridges method iterates threw the tokenlist. The bridgeMatch method tries to match the islands, by looking up the rule map and the defined matching methods.
If it matches a bridge will be build.

```
public void buildBridges(DoubleLinkedList.Node sof){
    int tolerance = 0;
    boolean change = false;
    DoubleLinkedList.Node start = null;
    DoubleLinkedList.Node end = null;
    //BridgeToken startOfFile = (BridgeToken) sof.element;
    while(!((BridgeToken) sof.element).hasBridge){
        start = sof;
        change = false;
        while(start!=null){
            end = nextUnmatchedIsland(start, tolerance);
            if(bridgeMatch(start, end)){
                buildBridge(start, end);
                change = true;
                start = nextUnmatchedStartIsland(end);
            }
            else
                start = nextUnmatchedStartIsland(start);
        }
        if(!change){
            tolerance = tolerance +1;

        }
        else if(tolerance > 0){
            tolerance = 0;
        }

    }
}
```

The Bridge Repairer

The bridgeRepairer algorithm now, iterates threw the tokens under a bridge. If a new bridge occures with a complete bridge, the method is called again with the new occured bridge.
If an island token occures and has no bridge, the mendRight algorithm is called if the island is an start island, else the mendLeft algorithm is called.

```
public static void bridgeRepairer(Bridge bridge){
    DoubleLinkedList.Node start = bridge.start;
    DoubleLinkedList.Node end = bridge.end;

    DoubleLinkedList.Node island = nextIsland(start);

    if(island == null) return;
    while(island != end){
        if(!((BridgeToken)island.element).hasBridge ){
            if(((BridgeToken) island.element).isStartIsland){
                mendRight(island, end);
            }
            else mendLeft(start, island);
        }

        bridge = ((BridgeToken) island.element).bridge;
        bridgeRepairer(bridge);
        island = nextIsland(bridge.end);
    }
}
```

The mendRight algorithm becomes the island which has no bridge and the end of the bridge which is above this broken/unmatched island. It iterates until the end of this bridge and tries to find the right position to insert the missing island.

```
public static void mendRight(DoubleLinkedList.Node broken, DoubleLinkedList.Node end){
    DoubleLinkedList.Node node = next(broken);

    while(node != end){
        if(possibleConstructionSite(broken, node)){
            constructIslandAndBridge(broken, node);
            return;
        }
        else node = next(node);
    }
   constructIslandAndBridge(broken,end.prev);

}
```