Newlib strlen Proof

Shubham Sondhi

2017CS10381

1 Introduction

The equivalence tool was able to find the equivalence between my unoptimized function of strlen and the optimized function for strlen in newlib. This report shares the manual proof between the two codes and the proof generated by the equivalence tool, and also the comparison between these proofs.

2 Newlib and my strlen's codes

The function calls to the llvm.dbg.value has been ignored in all of the proofs, and also in the following LLVM IR codes.

2.1 C code of my strlen

```
5.  size_t strlen(const char* s){
6.      const char* a = s;
7.      for (;;){
8.          if(!(*s)){
9.                  return s-a;
10.         }
11.         s++;
12.     }
13. }
```

2.2 C code of Newlib's strlen

```
#define LBLOCKSIZE    (sizeof (long))
#define UNALIGNED(X) ((long)X & (LBLOCKSIZE - 1))

    #if LONG_MAX == 2147483647L
    #define DETECTNULL(X) (((X) - 0x01010101) & ~(X) & 0x80808080)
    #else
    #if LONG_MAX == 9223372036854775807L
    /* Nonzero if X (a long int) contains a NULL byte. */
    #define DETECTNULL(X)
    (((X) - 0x0101010101010101) & ~(X) & 0x8080808080808080)
    #else
    #error long int is not a 32bit or 64bit type.
    #endif
    #endif

    #ifndef DETECTNULL
    #error long int is not a 32bit or 64bit byte
    #endif

23.   size_t strlen (const char *str)
24.    {
25.      const char *start = str;
26.
27.    #if !defined(PREFER_SIZE_OVER_SPEED)
28.        && !defined(__OPTIMIZE_SIZE__)
29.      unsigned long *aligned_addr;
30.
31.      while (UNALIGNED (str))
32.        {
33.          if (!*str)
34.            return str - start;
```

```
35.           str++;
36.         }
37.
38.
39.
40.       aligned_addr = (unsigned long *)str;
41.       while (!DETECTNULL (*aligned_addr))
42.         aligned_addr++;
43.
44.
45.
46.       str = (char *) aligned_addr;
47.
48.     #endif /* not PREFER_SIZE_OVER_SPEED */
49.
50.       while (*str)
51.         str++;
52.       return str - start;
53.     }
```

2.3 LLVM IR code of my srtlen

```
define dso_local i32 @strlen(i8* %s) #0 !dbg !8 {
entry:
  call void @llvm.dbg.value(metadata i8* ...
  call void @llvm.dbg.value(metadata i8* ...
  br label %for.cond, !dbg !18

for.cond:                                         ; preds = %if.end, %entry
  %s.addr.0 = phi i8* [ %s, %entry ], [ %incdec.ptr, %if.end ]
  call void @llvm.dbg.value(metadata i8* ...
  %0 = load i8, i8* %s.addr.0, align 1, !dbg !19
  %tobool = icmp ne i8 %0, 0, !dbg !24
```

```
  br i1 %tobool, label %if.end, label %if.then, !dbg !25

if.then:                                          ; preds = %for.cond
  %sub.ptr.lhs.cast = ptrtoint i8* %s.addr.0 to i32, !dbg !26
  %sub.ptr.rhs.cast = ptrtoint i8* %s to i32, !dbg !26
  %sub.ptr.sub = sub i32 %sub.ptr.lhs.cast,
            %sub.ptr.rhs.cast, !dbg !26
  ret i32 %sub.ptr.sub, !dbg !28

if.end:                                           ; preds = %for.cond
  %incdec.ptr = getelementptr inbounds i8,
            i8* %s.addr.0, i32 1, !dbg !29
  call void @llvm.dbg.value(metadata i8*...
  br label %for.cond, !dbg !30, !llvm.loop !31
}
```

2.4 LLVM IR code of Newlib's strlen

```
define dso_local i32 @strlen(i8* nonnull %str) #0 !dbg !14 {
entry:
  call void @llvm.dbg.value(metadata i8*...
  call void @llvm.dbg.value(metadata i8*...
  br label %while.cond, !dbg !24

while.cond:                                       ; preds = %if.end, %entry
  %str.addr.0 = phi i8* [ %str, %entry ],
                              [ %incdec.ptr, %if.end ]
  call void @llvm.dbg.value(metadata i8*...
  %0 = ptrtoint i8* %str.addr.0 to i32, !dbg !25
  %and = and i32 %0, 3, !dbg !25
  %tobool = icmp ne i32 %and, 0, !dbg !24
  br i1 %tobool, label %while.body, label %while.end, !dbg !24
```

```
while.body:                       ; preds = %while.cond
  %1 = load i8, i8* %str.addr.0, align 1, !dbg !26
  %tobool1 = icmp ne i8 %1, 0, !dbg !26
  br i1 %tobool1, label %if.end, label %if.then, !dbg !29

if.then:                          ; preds = %while.body
  %sub.ptr.lhs.cast = ptrtoint i8* %str.addr.0 to i32, !dbg !30
  %sub.ptr.rhs.cast = ptrtoint i8* %str to i32, !dbg !30
  %sub.ptr.sub = sub i32 %sub.ptr.lhs.cast,
                                  %sub.ptr.rhs.cast, !dbg !30
  br label %return, !dbg !31

if.end:                           ; preds = %while.body
  %incdec.ptr = getelementptr inbounds i8,
                      i8* %str.addr.0, i32 1, !dbg !32
  call void @llvm.dbg.value(metadata i8*...
  br label %while.cond, !dbg !24, !llvm.loop !33

while.end:                        ; preds = %while.cond
  %2 = bitcast i8* %str.addr.0 to i32*, !dbg !35
  call void @llvm.dbg.value(metadata i8*...
  br label %while.cond2, !dbg !37

while.cond2:                      ; preds = %while.body6, %while.end
  %aligned_addr.0 = phi i32* [ %2, %while.end ],
                             [ %incdec.ptr7, %while.body6 ], !dbg !22
  call void @llvm.dbg.value(metadata i8*...
  %3 = load i32, i32* %aligned_addr.0, align 4, !dbg !38
  %sub = sub i32 %3, 16843009, !dbg !38
  %4 = load i32, i32* %aligned_addr.0, align 4, !dbg !38
  %neg = xor i32 %4, -1, !dbg !38
  %and3 = and i32 %sub, %neg, !dbg !38
  %and4 = and i32 %and3, -2139062144, !dbg !38
```

```
  %tobool5 = icmp ne i32 %and4, 0, !dbg !39
  %lnot = xor i1 %tobool5, true, !dbg !39
  br i1 %lnot, label %while.body6, label %while.end8, !dbg !37

while.body6:                      ; preds = %while.cond2
  %incdec.ptr7 = getelementptr inbounds i32,
                          i32* %aligned_addr.0, i32 1, !dbg !40
  call void @llvm.dbg.value(metadata i8*...
  br label %while.cond2, !dbg !37, !llvm.loop !41

while.end8:                       ; preds = %while.cond2
  %5 = bitcast i32* %aligned_addr.0 to i8*, !dbg !42
  call void @llvm.dbg.value(metadata i8*...
  br label %while.cond9, !dbg !43

while.cond9:                      ; preds = %while.body11, %while.end8
  %str.addr.1 = phi i8* [ %5, %while.end8 ],
                    [ %incdec.ptr12, %while.body11 ], !dbg !22
  call void @llvm.dbg.value(metadata i8*...
  %6 = load i8, i8* %str.addr.1, align 1, !dbg !44
  %tobool10 = icmp ne i8 %6, 0, !dbg !43
  br i1 %tobool10, label %while.body11,
                              label %while.end13, !dbg !43

while.body11:                     ; preds = %while.cond9
  %incdec.ptr12 = getelementptr inbounds i8,
                              i8* %str.addr.1, i32 1, !dbg !45
  call void @llvm.dbg.value(metadata i8*...
  br label %while.cond9, !dbg !43, !llvm.loop !46

while.end13:                      ; preds = %while.cond9
  %sub.ptr.lhs.cast14 = ptrtoint i8*
                              %str.addr.1 to i32, !dbg !47
```

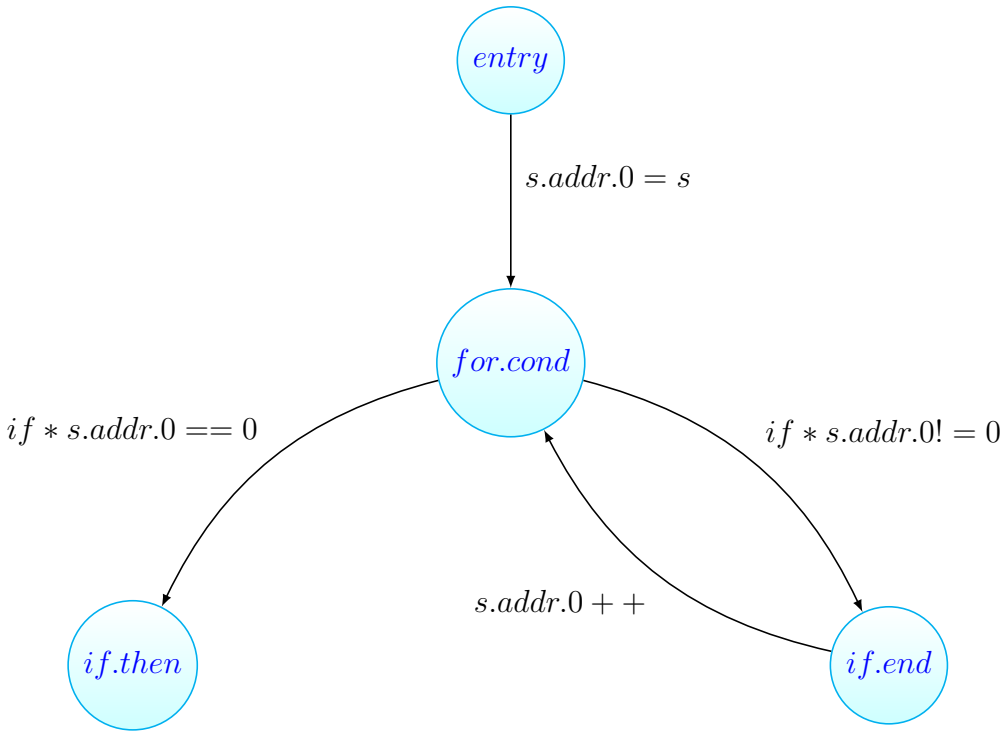
```
  %sub.ptr.rhs.cast15 = ptrtoint i8*
                              %str to i32, !dbg !47
  %sub.ptr.sub16 = sub i32 %sub.ptr.lhs.cast14,
                              %sub.ptr.rhs.cast15, !dbg !47
   br label %return, !dbg !48

return:                       ; preds = %while.end13, %if.then
  %retval.0 = phi i32 [ %sub.ptr.sub, %if.then ],
                  [ %sub.ptr.sub16, %while.end13 ], !dbg !22
   ret i32 %retval.0, !dbg !49
}
```

3 Control Flow Graphs

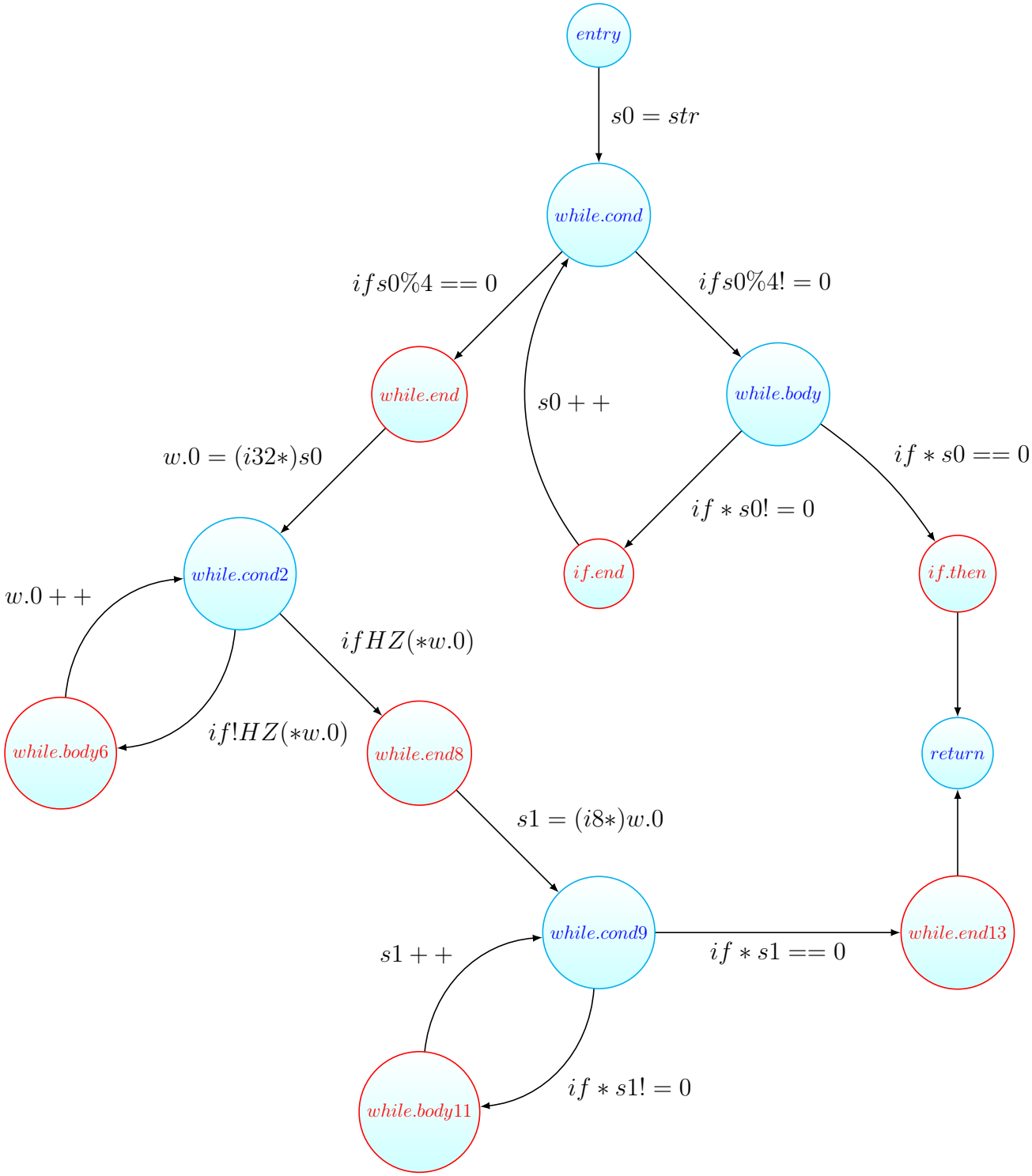
3.1 CFG of my strlen's LLVM IR

Figure 1: CFG of my strlen's LLVM IR



3.2 CFG of newlib's LLVM IR

Figure 2: CFG of newlib's LLVM IR



In the above figure CFG

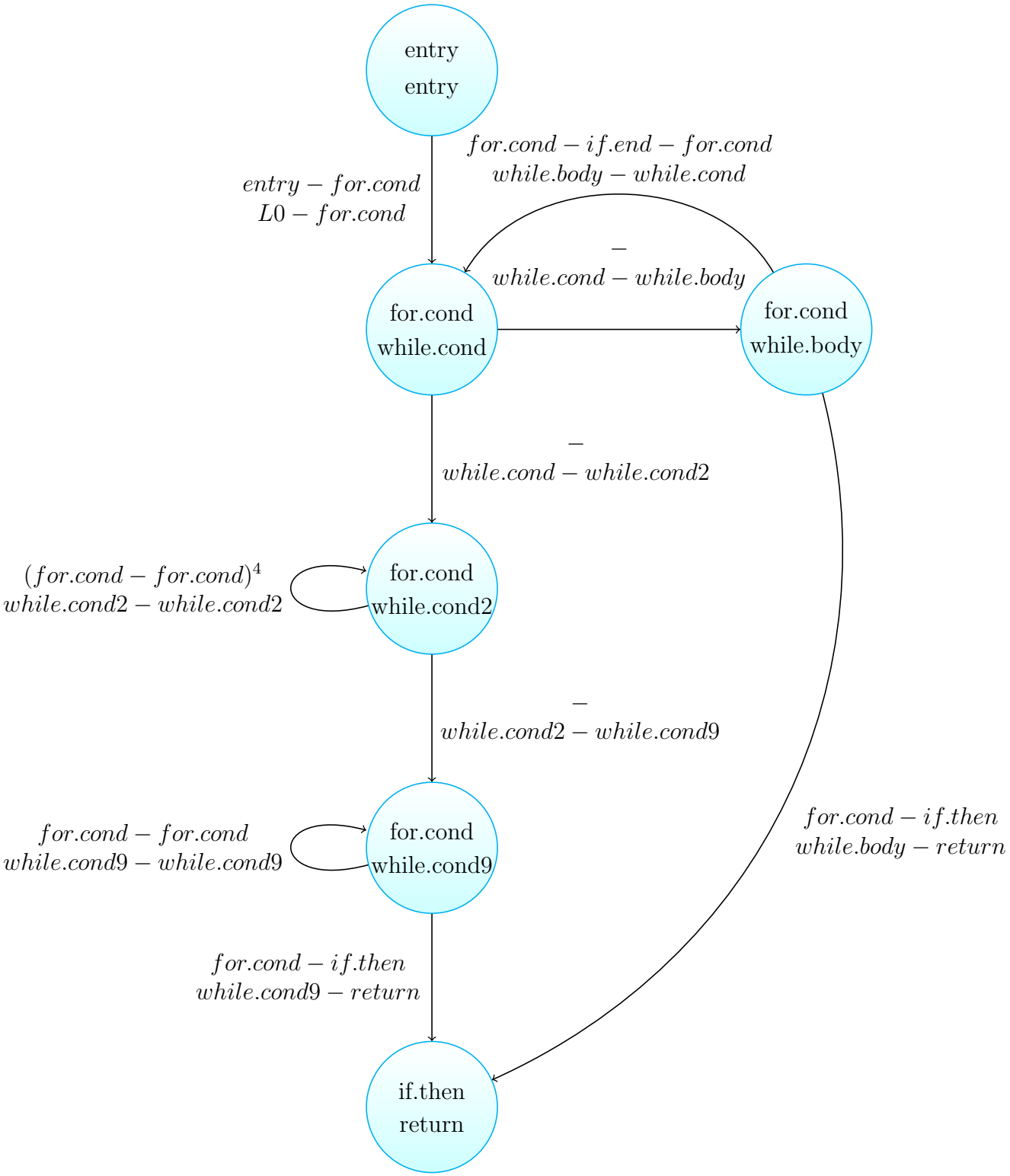
- s0 corresponds to the str.addr.0 register in the LLVM code.

- s1 corresponds to the str.addr.1 register in the LLVM code.
- w0 corresponds to the aligned_addr.0 register in the LLVM code.
- HZ is Haszero expression which checks if there's a 0 byte present in 4 consecutive bytes. It returns 0 if no 0 byte present otherwise a non zero value
- The nodes "while.end", "while.body6", "while.end8", "while.body11", "If.then", "while.end13" and "if.end", the ones that have been marked red in the above graph, have a single entry edge and a single exit edge. In the product-CFG, I removed them and connected their predecessor directly to their successor.

4 Proof of equivalence in LLVM IR

4.1 Manual Proof

Figure 3: Product CFG of the two codes in LLVM IR



4.1.1 Invariants at node(entry, entry)

(%s_C, H_C) = (%str_A, H_A)

s_C and str_A are the parameters in my strlen and newlib's strlen respectively, so they are same initially.

4.1.2 Invariants at node(for.cond, while.cond)

(%indec.ptr_C,%s.addr.0_C,%s_C, H_C) = (%indec.ptr_A%str.addr.0_A%str_A, H_A)

Value of s_C and s_A does not change throughout the program so they stay equal at each program point.

Neither a new memory location is allocated nor a memory location is changed so the heap also stays same.

No value is changed in this node. So the the values of these variables are still equal, given they were equal before entering this node, which will be proved for each node.

4.1.3 Invariants at node(for.cond, while.body)

(%indec.ptr_C,%s.addr.0_C,%s_C, H_C) = (%indec.ptr_A,%str.addr.0_A%str_A, H_A)

No value is changed in this node. So the the values of these variables are still equal, given they were equal before entering this node, which they are as proved in the other nodes.

4.1.4 Invariants at node(for.cond, while.cond2)

(%indec.ptr_C,%s.addr.0_C,%s_C, H_C) = (%indec.$ptr7_A$,%aligned_addr.0_A%str_A, H_A)

The value of %s.addr.0_A is assigned to %aligned_addr.0_A, if the incoming edge is from (for.cond, while.cond) so %aligned_addr.0_A = %s.addr.0_C. So ,initially they are same.

For the other incoming edge, which is a loop around the same node, %aligned_addr.0_A points to memory location of 4 bytes and %s.addr.0_C points to memory location of 1 byte. For every loop %indec.$ptr7_A$ is assigned the value of %aligned_addr.0_A + 1(increased the address by 4 bytes) and is assigned back to %aligned_addr.0_A

Block in C iterates 4 times for one iteration of A. In every iteration, %indec.ptr_C is assigned the value of %s.addr.0_C + 1(increased by 1 byte) and is assigned back to %s.addr.0_C.

Because this loop iterates four times, the address is increased by 4 bytes in the end, so %aligned_addr.0_A = %s.addr.0_C and %indec.$ptr7_A$ = %indec.ptr_C in this node.

4.1.5 Invariants at node(for.cond, while.cond9)

(%indec.ptr$_C$,%s.addr.0$_C$,%s$_C$, H$_C$) = (%indec.ptr12$_A$,%str.addr.1$_A$,%str$_A$, H$_A$)

The value of %aligned_addr.0$_A$ is assigned to %str.addr.1$_A$, if the incoming edge is from (for.cond, while.cond2) so %s.addr.0$_C$ = %str.addr.1$_A$. So ,initially they are same.

For the other incoming edge, which is a loop around the same node, %s.addr.0$_C$ and %str.addr.1$_A$ points to memory location of 1 byte.

In one iteration of A, %indec.ptr12$_A$ is assigned the value of %str.addr.1$_A$ + 1(increased the address by 1 byte) and is assigned back to %str.addr.1$_A$

In one iteration of C, %indec.ptr$_C$ is assigned the value of %s.addr.0$_C$ + 1(increased by 1 byte) and is assigned back to %s.addr.0$_C$.

So the values stay same after the loop

%str.addr.1$_A$ = %s.addr.0$_C$ and %indec.ptr12$_A$ = %indec.ptr$_C$ in this node.

4.1.6 At node(if.then, return)

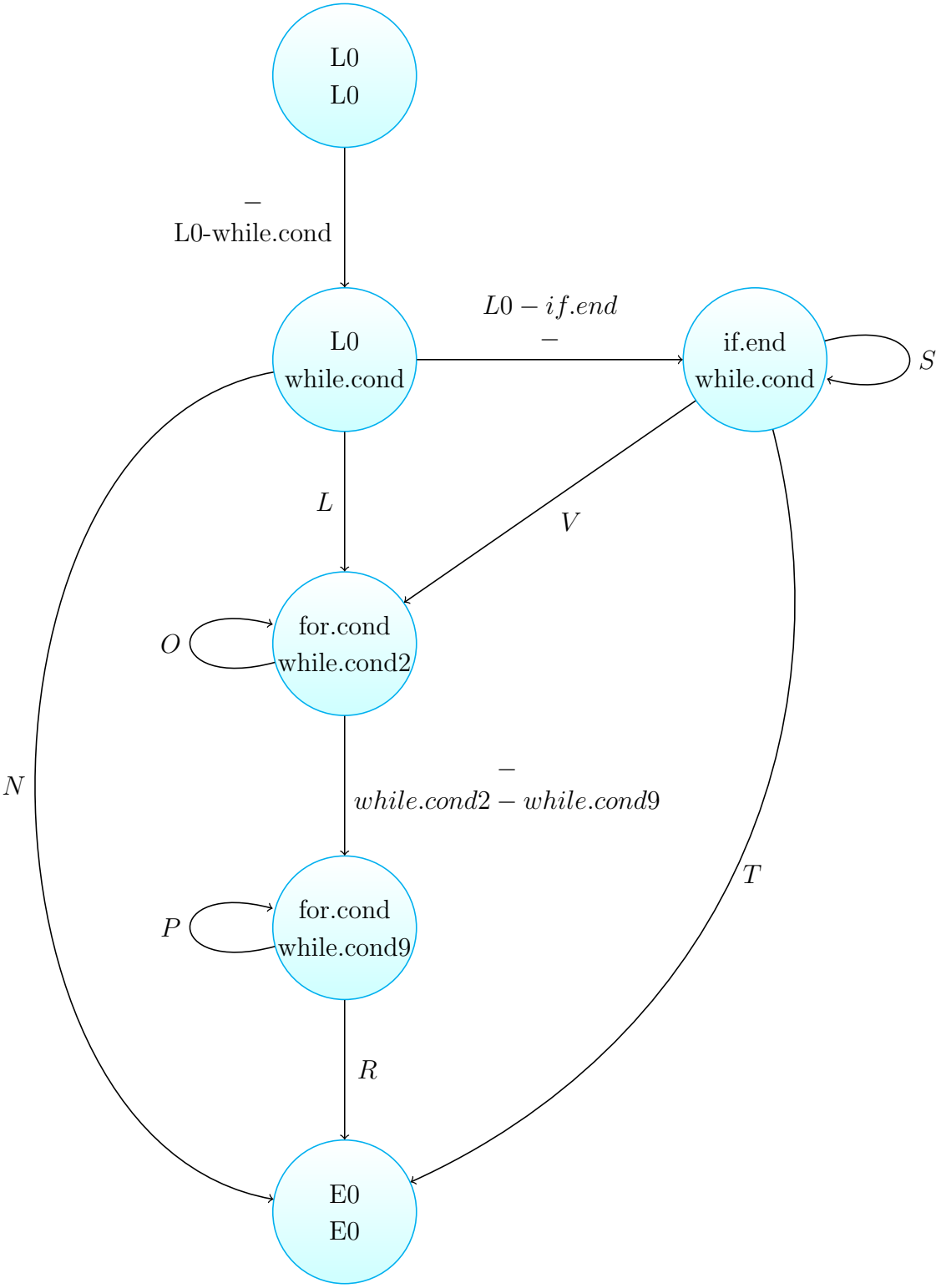
(%sub.ptr.sub$_C$,%s$_C$, H$_C$) = (%retval.0$_A$,%str$_A$, H$_A$)

There are two incoming edges to this node (for.cond, while.body) and for.cond,while.cond9)

For the edge that's coming from (for.cond, while.body), %retval.0$_A$ takes the value of %str.addr.0$_A$ - %str$_A$ and %sub.ptr.sub$_C$ takes the value of %s.addr.0$_C$ - %s$_C$, so they are same in this case. For the edge that's coming from (for.cond, while.cond9), %retval.0$_A$ takes the value of %str.addr.1$_A$ - %str$_A$ and %sub.ptr.sub$_C$ takes the value of %s.addr.0$_C$ - %s$_C$, so they are same in this case too.

4.2 Proof by equivalence tool

Figure 4: Product CFG of the two codes in LLVM IR



These are the complete paths found by equivalence tool:

L:

src:= (L0 - for.cond)

dst:= (while.cond - while.end)*(while.end - while.cond2)

N:

src:= (L0 - for.cond)*((((for.cond - if.end)*((if.end - for.cond)*(((for.cond - if.end)*(if.end - for.cond)*(for.cond - if.end)*((if.end - for.cond)*((for.cond - if.then)))+(for.cond - if.then))+(for.cond - if.then))))+(for.cond - if.then))*(if.then - E0))

dst:= (while.cond - while.body)*(while.body - if.then)*(if.then - return)*(return - E0)

O:

src:= ((for.cond - if.end)*(if.end - for.cond))^4

dst:= (while.cond2 - while.body)*(while.body6 - while.cond2)

P:

src:= (for.cond - if.end)*(if.end - for.cond)

dst:= (while.cond9 - while.body11)*(while.body11 - while.cond9)

R:

src:= (((for.cond - if.end)*((if.end - for.cond)*(((for.cond - if.end)*(if.end - for.cond)*(((for.cond - if.end)*(if.end - for.cond)*(for.cond - if.then))+(for.cond - if.then)))+(for.cond - if.then))))+(for.cond - if.then))*(if.then - E0)

dst:= (while.cond9 - while.end13)*(while.end13 - return)*(return - E0)

T:

src:= (if.end - for.cond)*(((for.cond - if.end)*((if.end - for.cond)*(((for.cond - if.end)*((if.end - for.cond)*(((for.cond - if.end)*(if.end - for.cond)*(for.cond - if.then))+(for.cond - if.then))))+(for.cond - if.then))))+(for.cond - if.then))*(if.then - E0)

dst:= (while.cond - while.body)*(while.body - if.then)*(if.then - return)*(return - E0)

V:

src:= (if.end - for.cond)*(for.cond - if.end)*(if.end - for.cond)

dst:= (while.cond - while.body)*(while.body - if.end)*(if.end - while.cond)*(while.cond - while.end)*(while.end - while.cond2)

S:

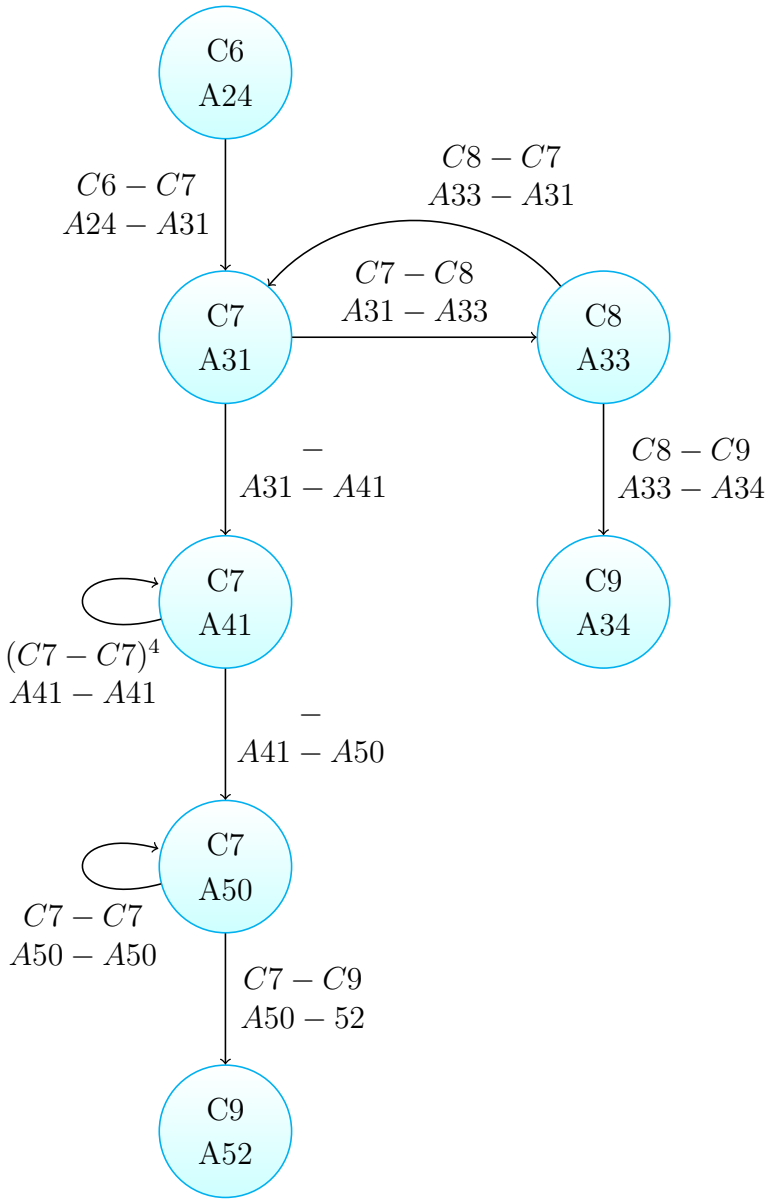
src:= (if.end - for.cond)*(for.cond - if.end)

dst:= (while.cond - while.body)*(while.body - if.end)*(if.end - while.cond)

5 Proof of equivalence in C

5.1 Manual Proof

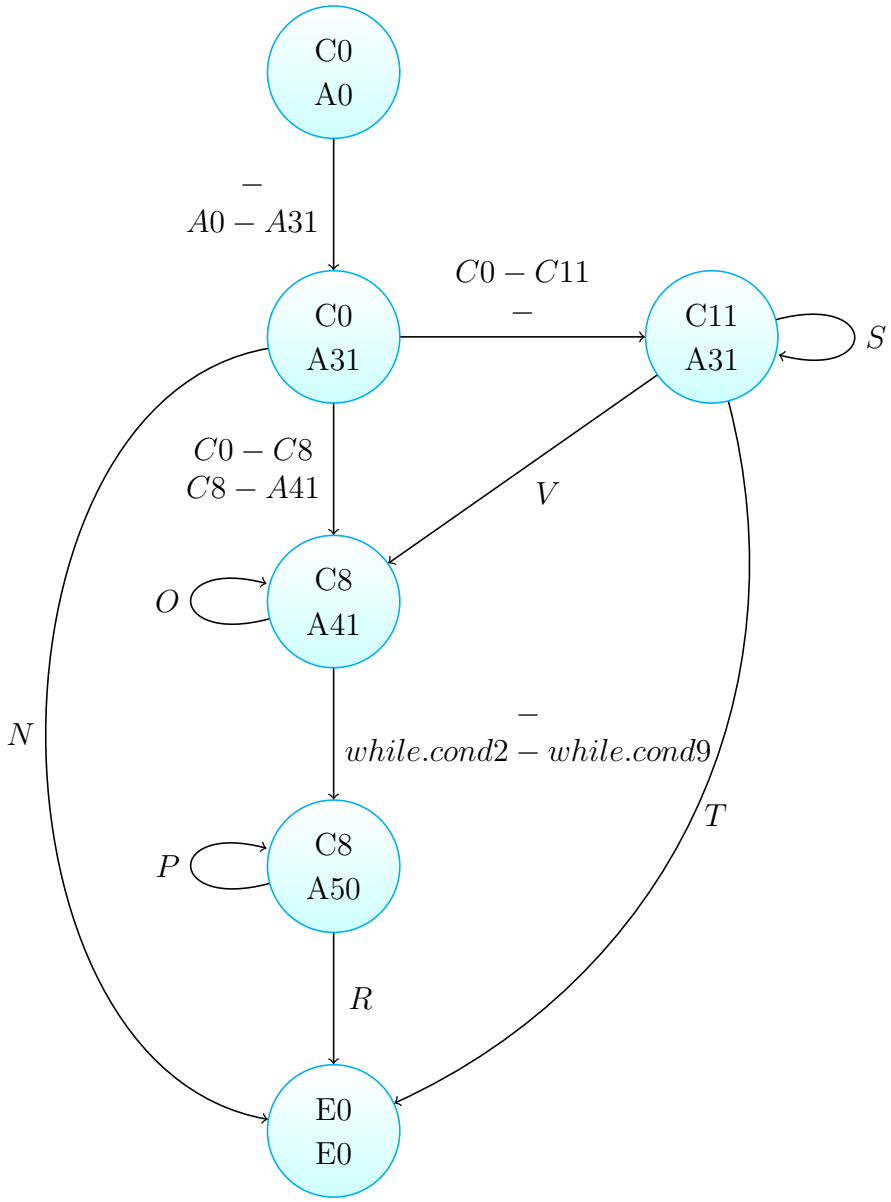
Figure 5: Product CFG of the two codes in C



C followed by a number n represents the nth statement in my C code of strlen, and A followed by a number m represents the mth statement in the newlib's strlen code.

5.2 Proof by equivalence tool

Figure 6: Product CFG of the two codes in LLVM IR



These are the complete paths found by equivalence tool:

N:

src:= (C0 - C8)*((((C8 - C11)*((C11 - C8)*(((C8 - C11)*(C11 - C8)*(C8 - C11)*((C11 - C8)*((C8 - C9)))+(C8 - C9))+(C8 - C9))))+(C8 - C9))*(C9 - E0)))

dst:= (A31 - A32)*(A32 - A33)*(A33 - A34)*(A34 - E0)

O:

src:= ((C8 - C11)*(C11 - C8))^4

dst:= (A41 - A42)*(A42 - A41)

P:

src:= (C8 - C11)*(C11 - C8)

dst:= (A50 - A51)*(A51 - A50)

R:

src:= (((C8 - C11)*((C11 - C8)*(((C8 - C11)*(C11 - C8)*(((C8 - C11)*(C11 - C8)*(C8 - if.then))+(C8 - C9)))+(C8 - C9))))+(C8 - C9))*(C9 - E0)

dst:= (A50 - A51)*(A51 - A52)*(A52 - E0)

T:

src:= (C11 - C8)*(((C8 - C11)*((C11 - C8)*(((C8 - C11)*((C11 - C8)*(((C8 - C11)*(C11 - C8)*(C8 - C9))+(C8 - C9))))+(C8 - C9))))+(C8 - C9))*(C9 - E0)

dst:= (A31 - A32)*(A32 - A33)*(A33 - A34)*(A34 - E0)

V:

src:= (C11 - C8)*(C8 - C11)*(C11 - C8)

dst:= (A31 - A32)*(A32 - 35)*(A35 - A31)*(A31 - A36)*(A36 - A41)

S:

src:= (C11 - C8)*(C8 - C11)

dst:= (A31 - A32)*(A32 - A35)*(A35 - A31)*(A31 - A36)*(A36 - A41)