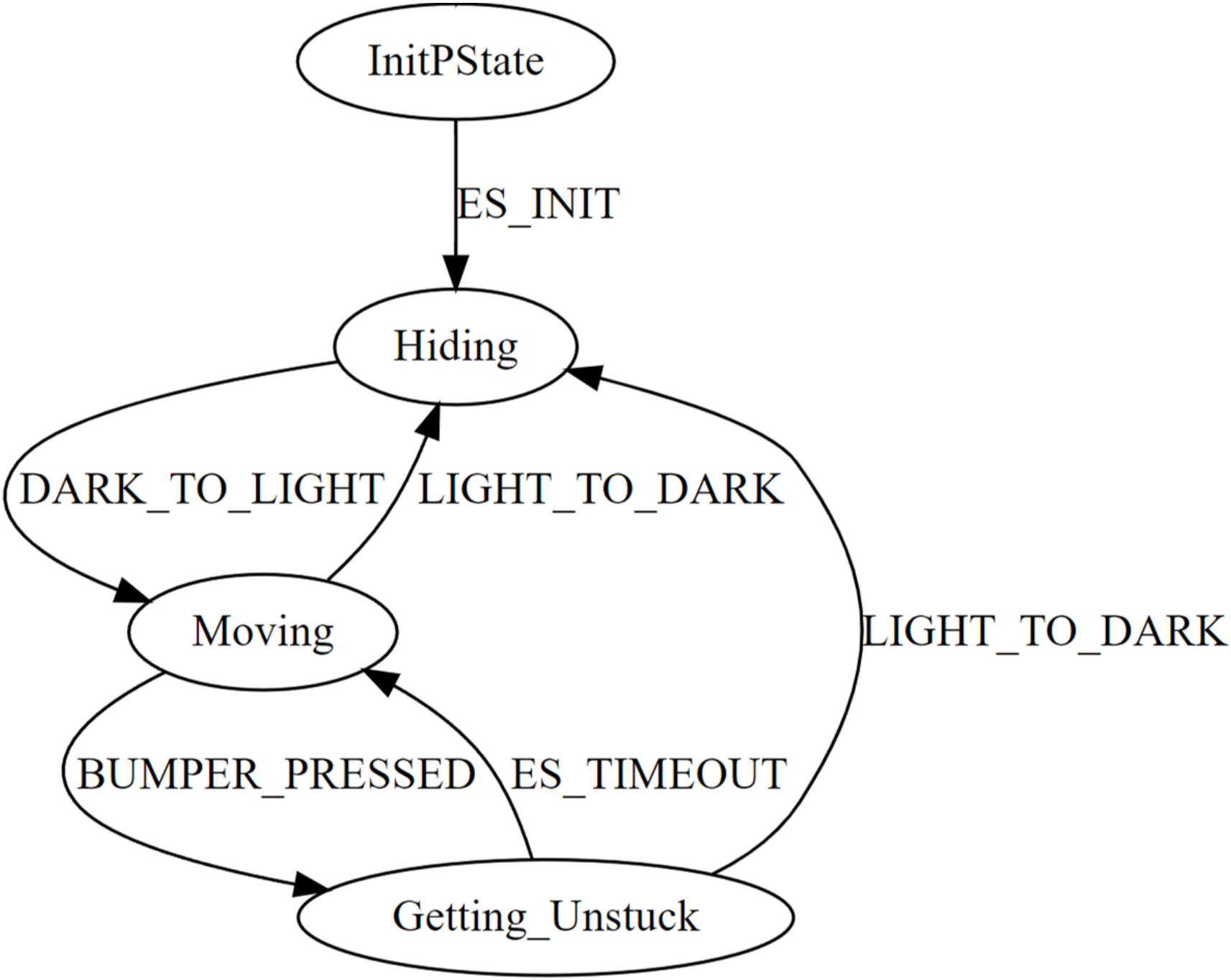
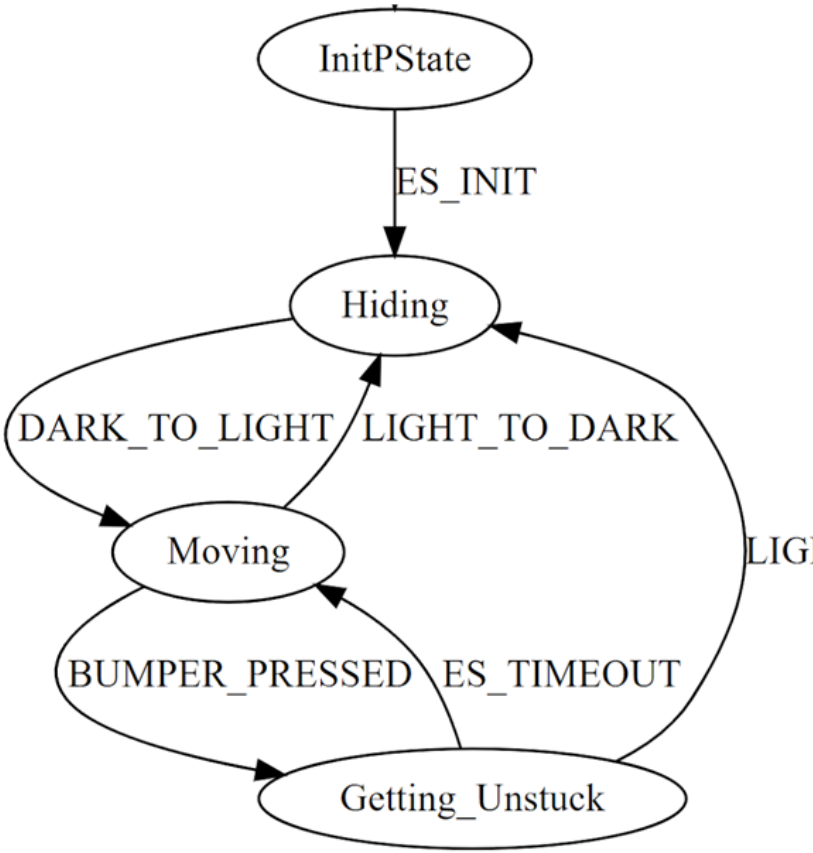
state machine visualizer


InitPState
ES_INIT
Hiding
DARK_TO_LIGHT
LIGHT_TO_DARK
Moving
LIGHT_TO_DARK
BUMPER_PRESSED
ES_TIMEOUT
Getting_Unstuck

https://dreampuf.github.io/GraphvizOnline/

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding [label = ES_INIT];
    Hiding -> Moving [label = DARK_TO_LIGHT];
    Moving -> Hiding [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving [label = ES_TIMEOUT];

}
```

Engine: dot Format: svg



```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

```
TemplateFSMState_t nextState;  ES_Tattle();

switch (CurrentState) {

    case InitPState:
        if (ThisEvent.EventType == ES_INIT) {
            nextState = Hiding; makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        break;

    case Hiding:
        switch (ThisEvent.EventType) {

            case ES_ENTRY: break;

            case ES_EXIT: break;

            case DARK_TO_LIGHT:
                goForward(MOVING_SPEED); nextState = Moving;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                break;

            default:
                break;
        }
        break;

    case Moving:
        if (ThisEvent.EventType == LIGHT_TO_DARK) {
            stopMovement(); nextState = Hiding;
            makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
```

```
TemplateFSMState_t nextState;  ES_Tattle();

switch (CurrentState) {

    case InitPState:
        if (ThisEvent.EventType == ES_INIT) {
            nextState = Hiding; makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        break;

    case Hiding:
        switch (ThisEvent.EventType) {

            case ES_ENTRY: break;

            case ES_EXIT: break;

            case DARK_TO_LIGHT:
                goForward(MOVING_SPEED); nextState = Moving;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                break;

            default:
                break;
        }
        break;

    case Moving:
        if (ThisEvent.EventType == LIGHT_TO_DARK) {
            stopMovement(); nextState = Hiding;
            makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
```

NOW?

NEXT?

```
TemplateFSMState_t nextState;  ES_Tattle();

switch (CurrentState) {

    case InitPState:
        if (ThisEvent.EventType == ES_INIT) {
            nextState = Hiding; makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        break;

    case Hiding:
        switch (ThisEvent.EventType) {

            case ES_ENTRY: break;

            case ES_EXIT: break;

            case DARK_TO_LIGHT:
                goForward(MOVING_SPEED); nextState = Moving;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                break;

            default:
                break;
        }
        break;

    case Moving:
        if (ThisEvent.EventType == LIGHT_TO_DARK) {
            stopMovement(); nextState = Hiding;
            makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
```

```
TemplateFSMState_t nextState;  ES_Tattle();

switch (CurrentState) {

    case InitPState:
        if (ThisEvent.EventType == ES_INIT) {
            nextState = Hiding; makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        break;

    case Hiding:
        switch (ThisEvent.EventType) {

            case ES_ENTRY: break;

            case ES_EXIT: break;

            case DARK_TO_LIGHT:
                goForward(MOVING_SPEED); nextState = Moving;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                break;

            default:
                break;
        }
        break;

    case Moving:
        if (ThisEvent.EventType == LIGHT_TO_DARK) {
            stopMovement(); nextState = Hiding;
            makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
```

WHY?

beginning of the string

small letter (s)

acceptable string length

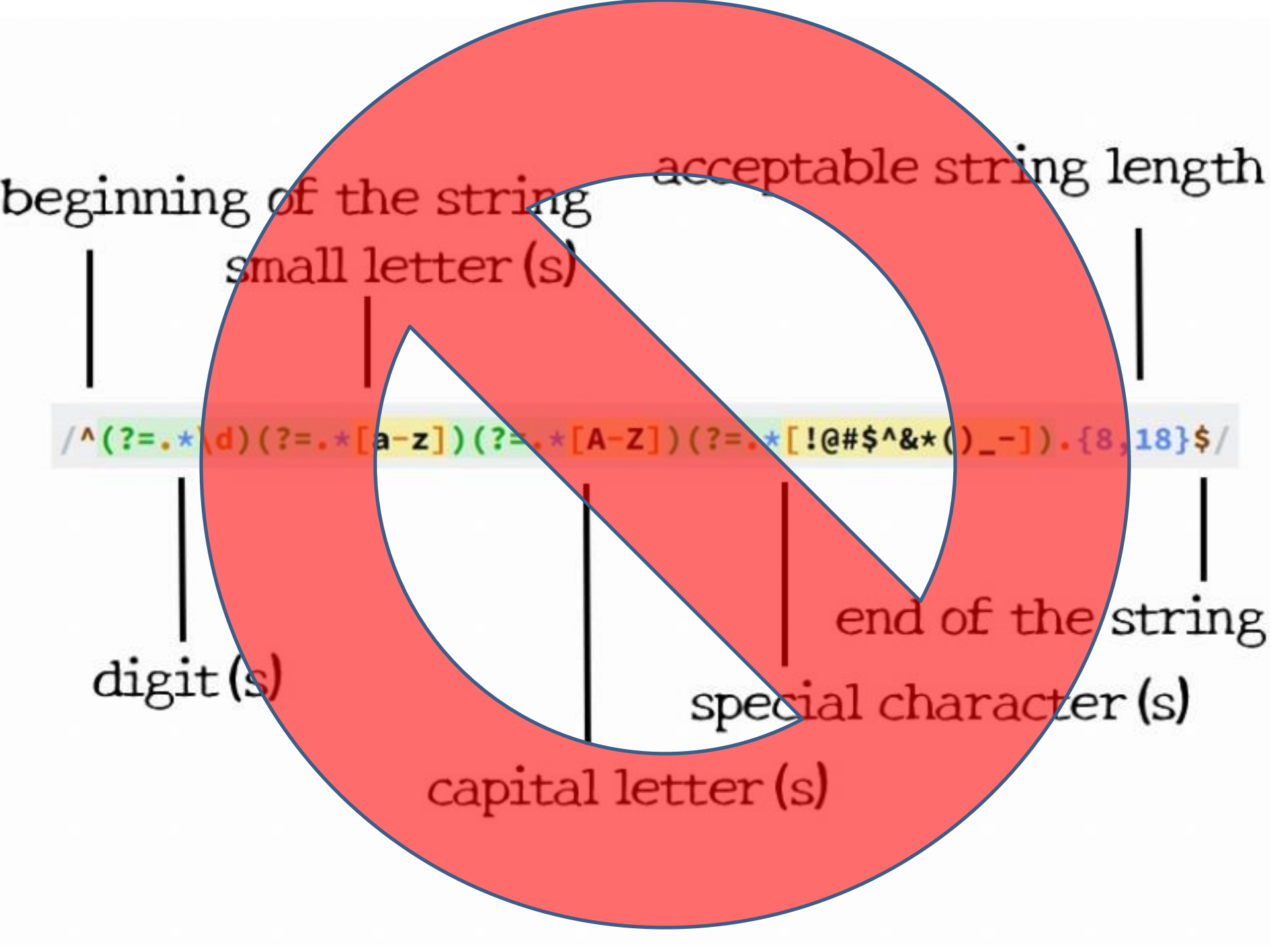
```
/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$^&*()_-]).{8,18}$/
```

digit (s)

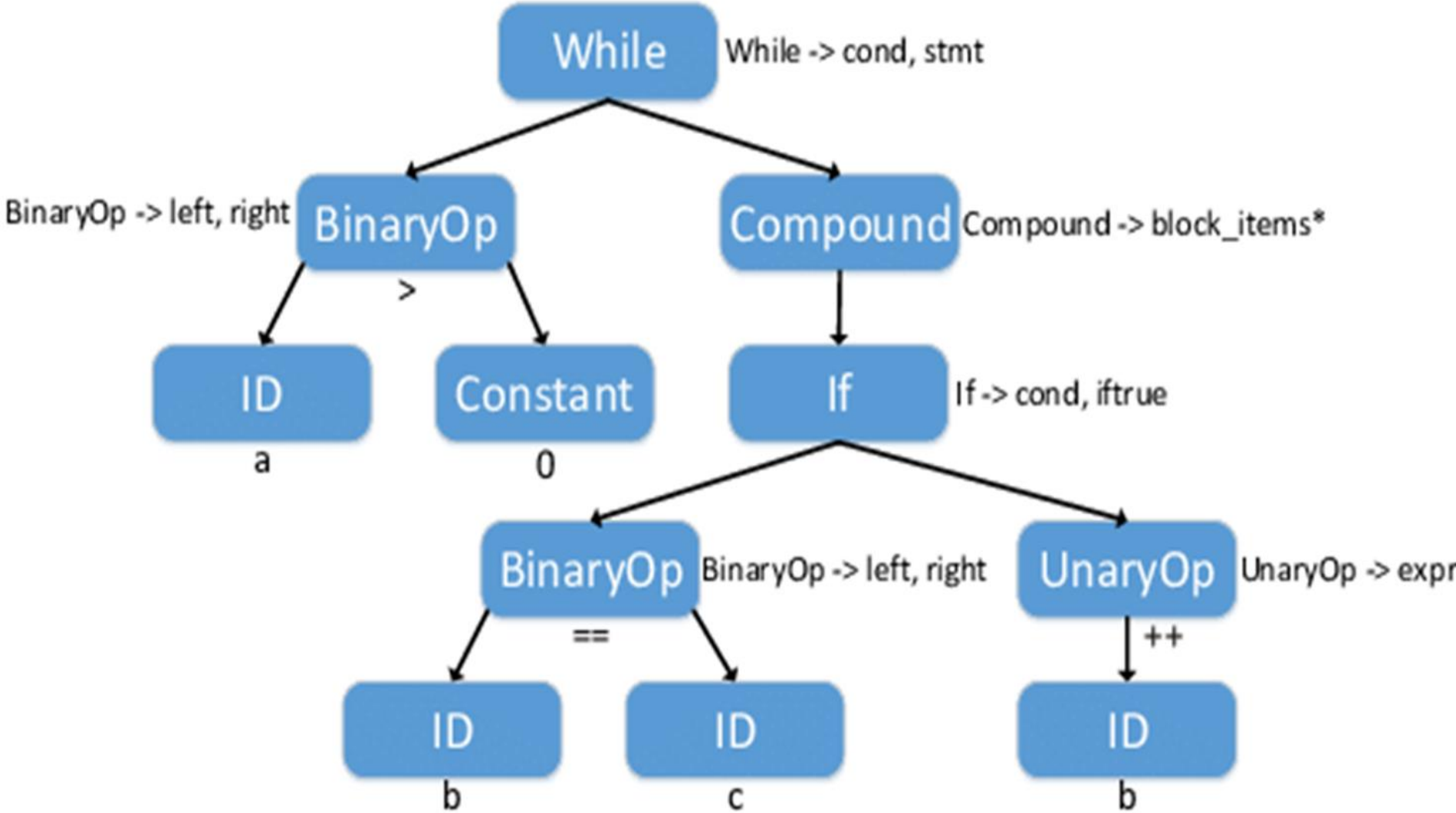
capital letter (s)

special character (s)

end of the string


beginning of the string
small letter (s)
acceptable string length
/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$^&*()_-]).{8,18}$/
digit (s)
capital letter (s)
special character (s)
end of the string

```
while(a>0){ if(b==c){b++;} };
```



https://github.com/eliben/pycparser

eliben / pycparser Public

Watch 96 | Fork 576 | Star 2.8k

Code | Issues 25 | Pull requests 1 | Actions | Projects | Wiki | Security | ...

master | Go to file | Add file | Code

eliben Update SECURITY.md ... ✓ 2 weeks ago 547

About

Complete C99 parser in pure Python

1.1 What is pycparser?

pycparser is a parser for the C language, written in pure Python. It is a module designed to be easily integrated into applications that need to parse C source code.

1.2 What is it good for?

Anything that needs C code to be parsed. The following are some uses for **pycparser**, taken from real user reports:

- C code obfuscator
- Front-end for various specialized C compilers
- Static code checker

1.3 Which version of C does pycparser support?

pycparser aims to support the full C99 language (according to the standard ISO/IEC 9899). Some features from C11 are also supported, and patches to support more are welcome.

pycparser supports very few GCC extensions, but it's fairly easy to set things up so that it parses code with a lot of GCC-isms successfully. See the FAQ for more details.

1.4 What grammar does pycparser follow?

pycparser very closely follows the C grammar provided in Annex A of the C99 standard (ISO/IEC 9899).

```python
def show(self, buf=sys.stdout
        , offset=0, attrnames=False
        , nodenames=False, showcoord=False, _my_node_name=None):
    lead = ' ' * offset
    if nodenames and _my_node_name is not None:
        buf.write(lead + self.__class__.__name__+ ' <' + _my_node_name + '>: ')
    else:
        buf.write(lead + self.__class__.__name__+ ': ')

    if self.attr_names:
        if attrnames:
            nvlist = [(n, getattr(self,n)) for n in self.attr_names]
            attrstr = ', '.join('%s=%s' % nv for nv in nvlist)
        else:
            vlist = [getattr(self, n) for n in self.attr_names]
            attrstr = ', '.join('%s' % v for v in vlist)
        buf.write(attrstr)

    if showcoord:
        buf.write(' (at %s)' % self.coord)
    buf.write('\n')

    for (child_name, child) in self.children():
        child.show(
            buf,
            offset=offset + 2,
            attrnames=attrnames,
            nodenames=nodenames,
            showcoord=showcoord,
            _my_node_name=child_name)
```

pycparser/pycparser/c_ast.py

```python
def myshow(node, xnode=ET.Element("root"), buf=sys.stderr, offset=0
        , attrnames=False, nodenames=False, showcoord=False, _my_node_name=None):
    lead = ' ' * offset
    xnode.set("class", node.__class__.__name__)

    if node.attr_names:
        if attrnames:
            nvlist = [(n, getattr(node, n)) for n in node.attr_names]
            attrstr = ', '.join('%s=%s' % nv for nv in nvlist)
            for nv in nvlist:
                if(nv[1] == None):
                    xnode.set(nv[0], "")
                else:
                    xnode.set(nv[0], str(nv[1]))
        else:
            vlist = [getattr(node, n) for n in node.attr_names]
            attrstr = ', '.join('%s' % v for v in vlist)

    for (child_name, child) in node.children():
        cxname = re.sub(r'\[.*\]', '', child_name)
        cxnode = ET.SubElement(xnode, cxname)

        myshow(child,
               cxnode,
               buf,
               offset=offset + 2,
               attrnames=attrnames,
               nodenames=nodenames,
               showcoord=showcoord,
               _my_node_name=child_name)

    return xnode
```

c_ast_xml_xslt.py

pycparser/pycparser/c_ast.py

```python
def show(self, buf=sys.stdout
        , offset=0, attrnames=False
        , nodenames=False, showcoord=False, _my_node_name=None):
    lead = ' ' * offset
    if nodenames and _my_node_name is not None:
        buf.write(lead + self.__class__.__name__+ ' <' + _my_node_name
    else:
        buf.write(lead + self.__class__.__name__+ ': ')

    if self.attr_names:
        if attrnames:
            nvlist = [(n, getattr(self,n)) for n in self.attr_names]
            attrstr = ', '.join('%s=%s' % nv for nv in nvlist)
        else:
            vlist = [getattr(self, n) for n in self.attr_names]
            attrstr = ', '.join('%s' % v for v in vlist)
        buf.write(attrstr)

    if showcoord:
        buf.write(' (at %s)' % self.coord)
    buf.write('\n')

    for (child_name, child) in self.children():
        child.show(
            buf,
            offset=offset + 2,
            attrnames=attrnames,
            nodenames=nodenames,
            showcoord=showcoord,
            _my_node_name=child_name)
```

c_ast_xml_xslt.py

```python
def myshow(node, xnode=ET.Element("root"), buf=sys.stderr
        , offset=0, attrnames=False
        , nodenames=False, showcoord=False, _my_node_name=None):
    lead = ' ' * offset
    xnode.set("class", node.__class__.__name__)

    if node.attr_names:
        if attrnames:
            nvlist = [(n, getattr(node, n)) for n in node.attr_names]
            attrstr = ', '.join('%s=%s' % nv for nv in nvlist)
            for nv in nvlist:
                if(nv[1] == None):
                    xnode.set(nv[0], "")
                else:
                    xnode.set(nv[0], str(nv[1]))
        else:
            vlist = [getattr(node, n) for n in node.attr_names]
            attrstr = ', '.join('%s' % v for v in vlist)

    for (child_name, child) in node.children():
        cxname = re.sub(r'\[.*\]', '', child_name)
        cxnode = ET.SubElement(xnode, cxname)

        myshow(child,
               cxnode,
               buf,
               offset=offset + 2,
               attrnames=attrnames,
               nodenames=nodenames,
               showcoord=showcoord,
               _my_node_name=child_name)

    return xnode
```

```python
import re
import sys
from pycparser import c_parser, c_ast, parse_file
from xml.dom import minidom
from lxml import etree as ET

def myshow(node, xnode=ET.Element("root"), buf=sys.stderr
    , offset=0, attrnames=False, nodenames=False
    , showcoord=False, _my_node_name=None):
    # ...

ast2 = parse_file("/dev/stdin", use_cpp=False)
xml2 = myshow(ast2, attrnames=True, nodenames=False)

if(len(sys.argv) < 2):
    print(prettify(ET.tostring(xml2, encoding='unicode')))
else:
    result = xml2
    for i in range(1,len(sys.argv)):
        xslt_doc = ET.parse(sys.argv[i])
        transformer = ET.XSLT(xslt_doc)
        result = transformer(result)
    print(result)
```

c_ast_xml_xslt.py

```python
import re
import sys
from pycparser import c_parser, c_ast, parse_file
from xml.dom import minidom
from lxml import etree as ET

def myshow(node, xnode=ET.Element("root"), buf=sys.stderr
    , offset=0, attrnames=False, nodenames=False
    , showcoord=False, _my_node_name=None):
    # ...

ast2 = parse_file("/dev/stdin", use_cpp=False)
xml2 = myshow(ast2, attrnames=True, nodenames=False)

if(len(sys.argv) < 2):
    print(prettify(ET.tostring(xml2, encoding='unicode')))
else:
    result = xml2
    for i in range(1,len(sys.argv)):
        xslt_doc = ET.parse(sys.argv[i])
        transformer = ET.XSLT(xslt_doc)
        result = transformer(result)
    print(result)
```

c_ast_xml_xslt.py

```python
import re
import sys
from pycparser import c_parser, c_ast, parse_file
from xml.dom import minidom
from lxml import etree as ET

def myshow(node, xnode=ET.Element("root"), buf=sys.stderr
    , offset=0, attrnames=False, nodenames=False
    , showcoord=False, _my_node_name=None):
    # ...

ast2 = parse_file("/dev/stdin", use_cpp=False)
xml2 = myshow(ast2, attrnames=True, nodenames=False)

if(len(sys.argv) < 2):
    print(prettify(ET.tostring(xml2, encoding='unicode')))
else:
    result = xml2
    for i in range(1,len(sys.argv)):
        xslt_doc = ET.parse(sys.argv[i])
        transformer = ET.XSLT(xslt_doc)
        result = transformer(result)
    print(result)
```

c_ast_xml_xslt.py

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType =
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
```

```
<ext class="FuncDef">
  <decl class="Decl" name="RunTemplateFSM">
    <type class="FuncDecl">
      <args class="ParamList">
        <params class="Decl" name="ThisEvent">
          <type class="TypeDecl" declname="ThisEvent">
            <type class="IdentifierType" names="['ES_Event']"/>
          </type>
        </params>
      </args>
      <type class="TypeDecl" declname="RunTemplateFSM">
        <type class="IdentifierType" names="['ES_Event']"/>
      </type>
    </type>
  </decl>
  <body class="Compound">
    <block_items class="Decl" name="makeTransition">
      <type class="TypeDecl" declname="makeTransition">
        <type class="IdentifierType" names="['uint8_t']"/>
      </type>
      <init class="ID" name="FALSE"/>
    </block_items>
    <block_items class="Decl" name="nextState">
      <type class="TypeDecl" declname="nextState">
        <type class="IdentifierType" names="['TemplateFSMState_t']"/>
      </type>
    </block_items>
    <block_items class="FuncCall">
      <name class="ID" name="ES_AddTattlePoint"/>
      <args class="ExprList">
        <exprs class="ID" name="__FUNCTION__"/>
        <exprs class="ArrayRef">
          <name class="ID" name="StateNames"/>
          <subscript class="ID" name="CurrentState"/>
        </exprs>
        <exprs class="ID" name="ThisEvent"/>
      </args>
    </block_items>
    <block_items class="Switch">
      <cond class="ID" name="CurrentState"/>
      <stmt class="Compound">
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_N
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EV
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME
                    nextState = Getting_Unstuck; makeTransition = TRU
                    ThisEvent.EventType = ES_NO_EVENT;
```

```
<block_items class="Switch">
  <cond class="ID" name="CurrentState"/>
  <stmt class="Compound">
    <block_items class="Case">
      <expr class="ID" name="InitPState"/>
      <stmts class="If">
        <cond class="BinaryOp" op="==">
          <left class="StructRef" type=".">
            <name class="ID" name="ThisEvent"/>
            <field class="ID" name="EventType"/>
          </left>
          <right class="ID" name="ES_INIT"/>
        </cond>
        <iftrue class="Compound">
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="nextState"/>
            <rvalue class="ID" name="Hiding"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="makeTransition"/>
            <rvalue class="ID" name="TRUE"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="StructRef" type=".">
              <name class="ID" name="ThisEvent"/>
              <field class="ID" name="EventType"/>
            </lvalue>
            <rvalue class="ID" name="ES_NO_EVENT"/>
          </block_items>
        </iftrue>
      </stmts>
      <stmts class="Break"/>
    </block_items>
```

gv_digraph.xslt

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

gv_digraph.xslt

```
<xsl:template match="/">
  <xsl:text>
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states</xsl:text>
      <!-- for every condition in the CurrentState switch statement -->
      <xsl:for-each select="
      //block_items[@class='Switch' and ./cond[@class='ID' and @name='CurrentState']]
      /stmt/block_items[@class='Case']
      /expr[@class='ID']
      ">
        <xsl:text>
    </xsl:text>
        <!-- output name of the condition label -->
        <xsl:value-of select="@name"/><xsl:text>;</xsl:text>
      </xsl:for-each>

    <xsl:text>

    // transitions
    init -> </xsl:text>
            <!-- locate right hand side value of CurrentState assignment statement inside the InitTemplateFSM() function -->
            <xsl:value-of select="
            //ext[@class='FuncDef' and ./decl[@class='Decl' and @name='InitTemplateFSM']]
            //block_items[@class='Assignment' and @op='=' and ./lvalue[@class='ID' and @name='CurrentState']]
            /rvalue/@name"
            />
    <xsl:text>[label = ES_INIT];
</xsl:text>

  <!-- for all = operations to nextState apply templates (BELOW) that are suitable -->
  <xsl:apply-templates select="//*[@op='=' and ./lvalue[@name='nextState'] ]"/>

  <xsl:text>
}</xsl:text>
</xsl:template>
```

gv_digraph.xslt

```
<xsl:template match="/">
  <xsl:text>
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states</xsl:text>
      <!-- for every condition in the CurrentState switch statement -->
      <xsl:for-each select="
      //block_items[@class='Switch' and ./cond[@class='ID' and @name='CurrentState']]
      /stmt/block_items[@class='Case']
      /expr[@class='ID']
      ">
        <xsl:text>
    </xsl:text>
        <!-- output name of the condition label -->
        <xsl:value-of select="@name"/><xsl:text>;</xsl:text>
      </xsl:for-each>

    <xsl:text>

    // transitions
    init -> </xsl:text>
            <!-- locate right hand side value of CurrentState assignment statement inside the InitTemplateFSM() function -->
            <xsl:value-of select="
            //ext[@class='FuncDef' and ./decl[@class='Decl' and @name='InitTemplateFSM']]
            //block_items[@class='Assignment' and @op='=' and ./lvalue[@class='ID' and @name='CurrentState']]
            /rvalue/@name"
            />
    <xsl:text>[label = ES_INIT];
</xsl:text>

  <!-- for all = operations to nextState apply templates (BELOW) that are suitable -->
  <xsl:apply-templates select="//*[@op='=' and ./lvalue[@name='nextState'] ]"/>

  <xsl:text>
}</xsl:text>
</xsl:template>
```

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

```
// states
InitPState;
Hiding;
Moving;
Getting_Unstuck;
```

```
// states</xsl:text>
  <!-- for every condition in the CurrentSta
  <xsl:for-each select="
  //block_items[
    @class='Switch'
    and ./cond[@class='ID'
    and @name='CurrentState']
  ]
  /stmt/block_items[@class='Case']
  /expr[@class='ID']
  ">
    <xsl:text>
</xsl:text>
    <!-- output name of the condition label
    <xsl:value-of select="@name"/><xsl:text>
  </xsl:for-each>
```

```
<block_items class="Switch">
  <cond class="ID" name="CurrentState"/>
  <stmt class="Compound">
    <block_items class="Case">
      <expr class="ID" name="InitPState"/>
      <stmts class="If">
        <cond class="BinaryOp" op="==">
          <left class="StructRef" type=".">
            <name class="ID" name="ThisEvent"/>
            <field class="ID" name="EventType"/>
          </left>
          <right class="ID" name="ES_INIT"/>
        </cond>
        <iftrue class="Compound">
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="nextState"/>
            <rvalue class="ID" name="Hiding"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="makeTransition"/>
            <rvalue class="ID" name="TRUE"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="StructRef" type=".">
              <name class="ID" name="ThisEvent"/>
              <field class="ID" name="EventType"/>
            </lvalue>
            <rvalue class="ID" name="ES_NO_EVENT"/>
          </block_items>
        </iftrue>
      </stmts>
      <stmts class="Break"/>
    </block_items>
    <block_items class="Case">
      <expr class="ID" name="Hiding"/>
      <stmts class="Switch">
        <cond class="StructRef" type=".">
          <name class="ID" name="ThisEvent"/>
          <field class="ID" name="EventType"/>
        </cond>
        <stmt class="Compound">
          <block_items class="Case">
            <expr class="ID" name="ES_ENTRY"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="ES_EXIT"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="DARK_TO_LIGHT"/>
```

```
// states
InitPState;
Hiding;
Moving;
Getting_Unstuck;
```

```
// states</xsl:text>
  <!-- for every condition in the CurrentSta
  <xsl:for-each select="
  //block_items[
    @class='Switch'
    and ./cond[@class='ID'
    and @name='CurrentState']
  ]
  /stmt/block_items[@class='Case']
  /expr[@class='ID']
  ">
    <xsl:text>
</xsl:text>
    <!-- output name of the condition label
    <xsl:value-of select="@name"/><xsl:text>
  </xsl:for-each>
```

```
<block_items class="Switch">
  <cond class="ID" name="CurrentState"/>
  <stmt class="Compound">
    <block_items class="Case">
      <expr class="ID" name="InitPState"/>
      <stmts class="If">
        <cond class="BinaryOp" op="==">
          <left class="StructRef" type=".">
            <name class="ID" name="ThisEvent"/>
            <field class="ID" name="EventType"/>
          </left>
          <right class="ID" name="ES_INIT"/>
        </cond>
        <iftrue class="Compound">
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="nextState"/>
            <rvalue class="ID" name="Hiding"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="makeTransition"/>
            <rvalue class="ID" name="TRUE"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="StructRef" type=".">
              <name class="ID" name="ThisEvent"/>
              <field class="ID" name="EventType"/>
            </lvalue>
            <rvalue class="ID" name="ES_NO_EVENT"/>
          </block_items>
        </iftrue>
      </stmts>
      <stmts class="Break"/>
    </block_items>
    <block_items class="Case">
      <expr class="ID" name="Hiding"/>
      <stmts class="Switch">
        <cond class="StructRef" type=".">
          <name class="ID" name="ThisEvent"/>
          <field class="ID" name="EventType"/>
        </cond>
        <stmt class="Compound">
          <block_items class="Case">
            <expr class="ID" name="ES_ENTRY"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="ES_EXIT"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="DARK_TO_LIGHT"/>
```

```
// states
InitPState;
Hiding;
Moving;
Getting_Unstuck;
```

```
// states</xsl:text>
  <!-- for every condition in the CurrentSta
  <xsl:for-each select="
  //block_items[
    @class='Switch'
    and ./cond[@class='ID'
    and @name='CurrentState']
  ]
  /stmt/block_items[@class='Case']
  /expr[@class='ID']
  ">
    <xsl:text>
</xsl:text>
    <!-- output name of the condition label
    <xsl:value-of select="@name"/><xsl:text>
  </xsl:for-each>
```

```
<block_items class="Switch">
  <cond class="ID" name="CurrentState"/>
  <stmt class="Compound">
    <block_items class="Case">
      <expr class="ID" name="InitPState"/>
      <stmts class="If">
        <cond class="BinaryOp" op="==">
          <left class="StructRef" type=".">
            <name class="ID" name="ThisEvent"/>
            <field class="ID" name="EventType"/>
          </left>
          <right class="ID" name="ES_INIT"/>
        </cond>
        <iftrue class="Compound">
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="nextState"/>
            <rvalue class="ID" name="Hiding"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="makeTransition"/>
            <rvalue class="ID" name="TRUE"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="StructRef" type=".">
              <name class="ID" name="ThisEvent"/>
              <field class="ID" name="EventType"/>
            </lvalue>
            <rvalue class="ID" name="ES_NO_EVENT"/>
          </block_items>
        </iftrue>
      </stmts>
      <stmts class="Break"/>
    </block_items>
    <block_items class="Case">
      <expr class="ID" name="Hiding"/>
      <stmts class="Switch">
        <cond class="StructRef" type=".">
          <name class="ID" name="ThisEvent"/>
          <field class="ID" name="EventType"/>
        </cond>
        <stmt class="Compound">
          <block_items class="Case">
            <expr class="ID" name="ES_ENTRY"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="ES_EXIT"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="DARK_TO_LIGHT"/>
```

```
// states
InitPState;
Hiding;
Moving;
Getting_Unstuck;
```

```
// states</xsl:text>
  <!-- for every condition in the CurrentSta
  <xsl:for-each select="
  //block_items[
    @class='Switch'
    and ./cond[@class='ID'
    and @name='CurrentState']
  ]
  /stmt/block_items[@class='Case']
  /expr[@class='ID']
  ">
    <xsl:text>
</xsl:text>
    <!-- output name of the condition label
    <xsl:value-of select="@name"/><xsl:text>
  </xsl:for-each>
```

```
<block_items class="Switch">
  <cond class="ID" name="CurrentState"/>
  <stmt class="Compound">
    <block_items class="Case">
      <expr class="ID" name="InitPState"/>
      <stmts class="If">
        <cond class="BinaryOp" op="==">
          <left class="StructRef" type=".">
            <name class="ID" name="ThisEvent"/>
            <field class="ID" name="EventType"/>
          </left>
          <right class="ID" name="ES_INIT"/>
        </cond>
        <iftrue class="Compound">
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="nextState"/>
            <rvalue class="ID" name="Hiding"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="makeTransition"/>
            <rvalue class="ID" name="TRUE"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="StructRef" type=".">
              <name class="ID" name="ThisEvent"/>
              <field class="ID" name="EventType"/>
            </lvalue>
            <rvalue class="ID" name="ES_NO_EVENT"/>
          </block_items>
        </iftrue>
      </stmts>
      <stmts class="Break"/>
    </block_items>
    <block_items class="Case">
      <expr class="ID" name="Hiding"/>
      <stmts class="Switch">
        <cond class="StructRef" type=".">
          <name class="ID" name="ThisEvent"/>
          <field class="ID" name="EventType"/>
        </cond>
        <stmt class="Compound">
          <block_items class="Case">
            <expr class="ID" name="ES_ENTRY"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="ES_EXIT"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="DARK_TO_LIGHT"/>
```

```
// states
InitPState;
Hiding;
Moving;
Getting_Unstuck;
```

```
// states</xsl:text>
  <!-- for every condition in the CurrentSta
  <xsl:for-each select="
  //block_items[
    @class='Switch'
    and ./cond[@class='ID'
    and @name='CurrentState']
  ]
  /stmt/block_items[@class='Case']
  /expr[@class='ID']
  ">
    <xsl:text>
</xsl:text>
    <!-- output name of the condition label
    <xsl:value-of select="@name"/><xsl:text>
  </xsl:for-each>
```

```
<block_items class="Switch">
  <cond class="ID" name="CurrentState"/>
  <stmt class="Compound">
    <block_items class="Case">
      <expr class="ID" name="InitPState"/>
      <stmts class="If">
        <cond class="BinaryOp" op="==">
          <left class="StructRef" type=".">
            <name class="ID" name="ThisEvent"/>
            <field class="ID" name="EventType"/>
          </left>
          <right class="ID" name="ES_INIT"/>
        </cond>
        <iftrue class="Compound">
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="nextState"/>
            <rvalue class="ID" name="Hiding"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="ID" name="makeTransition"/>
            <rvalue class="ID" name="TRUE"/>
          </block_items>
          <block_items class="Assignment" op="=">
            <lvalue class="StructRef" type=".">
              <name class="ID" name="ThisEvent"/>
              <field class="ID" name="EventType"/>
            </lvalue>
            <rvalue class="ID" name="ES_NO_EVENT"/>
          </block_items>
        </iftrue>
      </stmts>
      <stmts class="Break"/>
    </block_items>
    <block_items class="Case">
      <expr class="ID" name="Hiding"/>
      <stmts class="Switch">
        <cond class="StructRef" type=".">
          <name class="ID" name="ThisEvent"/>
          <field class="ID" name="EventType"/>
        </cond>
        <stmt class="Compound">
          <block_items class="Case">
            <expr class="ID" name="ES_ENTRY"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="ES_EXIT"/>
            <stmts class="Break"/>
          </block_items>
          <block_items class="Case">
            <expr class="ID" name="DARK_TO_LIGHT"/>
```

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

```
// transitions
init -> InitPState[label = ES_INIT];
```

```
// transitions
init -> InitPState[label = ES_INIT];
```

```
uint8_t InitTemplateFSM(uint8_t Priority)
{
    MyPriority = Priority;
    // put us into the Initial PseudoState
    CurrentState = InitPState;
    // post the initial transition event
    if(ES_PostToService(MyPriority, INIT_EVENT) == TRUE)
    {
        return TRUE;
    } else {
        return FALSE;
    }
}
```

```xml
<ext class="FuncDef">
  <decl class="Decl" name="InitTemplateFSM">
    <type class="FuncDecl">
      <args class="ParamList">
        <params class="Decl" name="Priority">
          <type class="TypeDecl" declname="Priority">
            <type class="IdentifierType" names="['uint8_t']"/>
          </type>
        </params>
      </args>
      <type class="TypeDecl" declname="InitTemplateFSM">
        <type class="IdentifierType" names="['uint8_t']"/>
      </type>
    </type>
  </decl>
  <body class="Compound">
    <block_items class="Assignment" op="=">
      <lvalue class="ID" name="MyPriority"/>
      <rvalue class="ID" name="Priority"/>
    </block_items>
    <block_items class="Assignment" op="=">
      <lvalue class="ID" name="CurrentState"/>
      <rvalue class="ID" name="InitPState"/>
    </block_items>
    <block_items class="If">
      <cond class="BinaryOp" op="==">
        <left class="FuncCall">
          <name class="ID" name="ES_PostToService"/>
          <args class="ExprList">
            <exprs class="ID" name="MyPriority"/>
            <exprs class="CompoundLiteral">
              <type class="Typename" name="">
                <type class="TypeDecl" declname="">
                  <type class="IdentifierType" names="['ES_Event']"/>
                </type>
              </type>
              <init class="InitList">
                <exprs class="ID" name="ES_INIT"/>
                <exprs class="Constant" type="int" value="0x0000"/>
              </init>
            </exprs>
          </args>
        </left>
        <right class="ID" name="TRUE"/>
      </cond>
      <iftrue class="Compound">
        <block_items class="Return">
          <expr class="ID" name="TRUE"/>
        </block_items>
      </iftrue>
      <iffalse class="Compound">
        <block_items class="Return">
          <expr class="ID" name="FALSE"/>
        </block_items>
      </iffalse>
    </block_items>
  </body>
</ext>
```


```
// transitions
init -> InitPState[label = ES_INIT];
```

```
uint8_t InitTemplateFSM(uint8_t Priority)
{
    MyPriority = Priority;
    // put us into the Initial PseudoState
    CurrentState = InitPState;
    // post the initial transition event
    if(ES_PostToService(MyPriority, INIT_EVENT) == TRUE)
    {
        return TRUE;
    } else {
        return FALSE;
    }
}
```

```xml
<ext class="FuncDef">
  <decl class="Decl" name="InitTemplateFSM">
    <type class="FuncDecl">
      <args class="ParamList">
        <params class="Decl" name="Priority">
          <type class="TypeDecl" declname="Priority">
            <type class="IdentifierType" names="['uint8_t']"/>
          </type>
        </params>
      </args>
      <type class="TypeDecl" declname="InitTemplateFSM">
        <type class="IdentifierType" names="['uint8_t']"/>
      </type>
    </type>
  </decl>
  <body class="Compound">
    <block_items class="Assignment" op="=">
      <lvalue class="ID" name="MyPriority"/>
      <rvalue class="ID" name="Priority"/>
    </block_items>
    <block_items class="Assignment" op="=">
      <lvalue class="ID" name="CurrentState"/>
      <rvalue class="ID" name="InitPState"/>
    </block_items>
    <block_items class="If">
      <cond class="BinaryOp" op="==">
        <left class="FuncCall">
          <name class="ID" name="ES_PostToService"/>
          <args class="ExprList">
            <exprs class="ID" name="MyPriority"/>
            <exprs class="CompoundLiteral">
              <type class="Typename" name="">
                <type class="TypeDecl" declname="">
                  <type class="IdentifierType" names="['ES_Event']"/>
                </type>
              </type>
              <init class="InitList">
                <exprs class="ID" name="ES_INIT"/>
                <exprs class="Constant" type="int" value="0x0000"/>
              </init>
            </exprs>
          </args>
        </left>
        <right class="ID" name="TRUE"/>
      </cond>
      <iftrue class="Compound">
        <block_items class="Return">
          <expr class="ID" name="TRUE"/>
        </block_items>
      </iftrue>
      <iffalse class="Compound">
        <block_items class="Return">
          <expr class="ID" name="FALSE"/>
        </block_items>
      </iffalse>
    </block_items>
  </body>
</ext>
```

```
// transitions
init -> InitPState[label = ES_INIT];
```

```
// transitions
init -> </xsl:text>
        <!-- locate right hand side value of CurrentState assignment statement inside the InitTemplateFSM() fun
        <xsl:value-of select="
        //ext[@class='FuncDef' and ./decl[@class='Decl' and @name='InitTemplateFSM']]
        //block_items[@class='Assignment' and @op='=' and ./lvalue[@class='ID' and @name='CurrentState']]
        /rvalue/@name"
        />
<xsl:text>[label = ES_INIT];
```

```
uint8_t InitTemplateFSM(uint8_t Priority)
{
    MyPriority = Priority;
    // put us into the Initial PseudoState
    CurrentState = InitPState;
    // post the initial transition event
    if(ES_PostToService(MyPriority, INIT_EVENT) == TRUE)
    {
        return TRUE;
    } else {
        return FALSE;
    }
}
```

gv_digraph.xslt

```
<xsl:template match="/">
  <xsl:text>
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states</xsl:text>
      <!-- for every condition in the CurrentState switch statement -->
      <xsl:for-each select="
      //block_items[@class='Switch' and ./cond[@class='ID' and @name='CurrentState']]
      /stmt/block_items[@class='Case']
      /expr[@class='ID']
      ">
        <xsl:text>
    </xsl:text>
        <!-- output name of the condition label -->
        <xsl:value-of select="@name"/><xsl:text>;</xsl:text>
      </xsl:for-each>

    <xsl:text>

    // transitions
    init -> </xsl:text>
            <!-- locate right hand side value of CurrentState assignment statement inside the InitTemplateFSM() function -->
            <xsl:value-of select="
            //ext[@class='FuncDef' and ./decl[@class='Decl' and @name='InitTemplateFSM']]
            //block_items[@class='Assignment' and @op='=' and ./lvalue[@class='ID' and @name='CurrentState']]
            /rvalue/@name"
            />
    <xsl:text>[label = ES_INIT];
</xsl:text>

  <!-- for all = operations to nextState apply templates (BELOW) that are suitable -->
  <xsl:apply-templates select="//*[@op='=' and ./lvalue[@name='nextState'] ]"/>

  <xsl:text>
}</xsl:text>
</xsl:template>
```

gv_digraph.xslt

```
<xsl:template match="/">
  <xsl:text>
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states</xsl:text>
      <!-- for every condition in the CurrentState switch statement -->
      <xsl:for-each select="
      //block_items[@class='Switch' and ./cond[@class='ID' and @name='CurrentState']]
      /stmt/block_items[@class='Case']
      /expr[@class='ID']
      ">
        <xsl:text>
    </xsl:text>
        <!-- output name of the condition label -->
        <xsl:value-of select="@name"/><xsl:text>;</xsl:text>
      </xsl:for-each>

    <xsl:text>

    // transitions
    init -> </xsl:text>
            <!-- locate right hand side value of CurrentState assignment statement inside the InitTemplateFSM() function -->
            <xsl:value-of select="
            //ext[@class='FuncDef' and ./decl[@class='Decl' and @name='InitTemplateFSM']]
            //block_items[@class='Assignment' and @op='=' and ./lvalue[@class='ID' and @name='CurrentState']]
            /rvalue/@name"
            />
    <xsl:text>[label = ES_INIT];
</xsl:text>

  <!-- for all = operations to nextState apply templates (BELOW) that are suitable -->
  <xsl:apply-templates select="//*[@op='=' and ./lvalue[@name='nextState'] ]"/>

  <xsl:text>
}</xsl:text>
</xsl:template>
```

```
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside top switch statement this template handles assignments to nextState guarded by switch statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 2=count(ancestor::block_items[@class='Case'])]">
            <xsl:text>    </xsl:text>
            <!-- top switch statement labels the current state -->
            <xsl:value-of select="
                ancestor::block_items[@class='Case'][2]/expr[@class='ID']/@name
            "/>
            <xsl:text> -> </xsl:text>
            <!-- right hand side of assignment supplies the next state -->
            <xsl:value-of select="rvalue/@name"/>
            <xsl:text>  [label = </xsl:text>
            <!-- take value of EventType from inner case statement id name  -->
            <xsl:value-of select="
              ../expr[@class='ID']/@name
             "/>
             <xsl:text>]; // switch switch
</xsl:text>
  </xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside top switch statement this template handles assignments to nextState guarded by switch statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 2=count(ancestor::block_items[@class='Case'])]">
            <xsl:text>    </xsl:text>
            <!-- top switch statement labels the current state -->
            <xsl:value-of select="
                ancestor::block_items[@class='Case'][2]/expr[@class='ID']/@name
            "/>
            <xsl:text> -> </xsl:text>
            <!-- right hand side of assignment supplies the next state -->
            <xsl:value-of select="rvalue/@name"/>
            <xsl:text>  [label = </xsl:text>
            <!-- take value of EventType from inner case statement id name  -->
            <xsl:value-of select="
              ../expr[@class='ID']/@name
             "/>
             <xsl:text>]; // switch switch
</xsl:text>
  </xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside top switch statement this template handles assignments to nextState guarded by switch statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 2=count(ancestor::block_items[@class='Case'])]">
            <xsl:text>    </xsl:text>
            <!-- top switch statement labels the current state -->
            <xsl:value-of select="
                ancestor::block_items[@class='Case'][2]/expr[@class='ID']/@name
            "/>
            <xsl:text> -> </xsl:text>
            <!-- right hand side of assignment supplies the next state -->
            <xsl:value-of select="rvalue/@name"/>
            <xsl:text>  [label = </xsl:text>
            <!-- take value of EventType from inner case statement id name  -->
            <xsl:value-of select="
              ../expr[@class='ID']/@name
             "/>
             <xsl:text>]; // switch switch
</xsl:text>
  </xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside top switch statement this template handles assignments to nextState guarded by switch statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 2=count(ancestor::block_items[@class='Case'])]">
            <xsl:text>    </xsl:text>
            <!-- top switch statement labels the current state -->
            <xsl:value-of select="
                ancestor::block_items[@class='Case'][2]/expr[@class='ID']/@name
            "/>
            <xsl:text> -> </xsl:text>
            <!-- right hand side of assignment supplies the next state -->
            <xsl:value-of select="rvalue/@name"/>
            <xsl:text>  [label = </xsl:text>
            <!-- take value of EventType from inner case statement id name  -->
            <xsl:value-of select="
              ../expr[@class='ID']/@name
             "/>
             <xsl:text>]; // switch switch
</xsl:text>
  </xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside top switch statement this template handles assignments to nextState guarded by switch statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 2=count(ancestor::block_items[@class='Case'])]">
            <xsl:text>    </xsl:text>
            <!-- top switch statement labels the current state -->
            <xsl:value-of select="
                ancestor::block_items[@class='Case'][2]/expr[@class='ID']/@name
            "/>
            <xsl:text> -> </xsl:text>
            <!-- right hand side of assignment supplies the next state -->
            <xsl:value-of select="rvalue/@name"/>
            <xsl:text>  [label = </xsl:text>
            <!-- take value of EventType from inner case statement id name  -->
            <xsl:value-of select="
              ../expr[@class='ID']/@name
             "/>
             <xsl:text>]; // switch switch
</xsl:text>
  </xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside top switch statement this template handles assignments to nextState guarded by switch statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 2=count(ancestor::block_items[@class='Case'])]">
            <xsl:text>    </xsl:text>
            <!-- top switch statement labels the current state -->
            <xsl:value-of select="
                ancestor::block_items[@class='Case'][2]/expr[@class='ID']/@name
            "/>
            <xsl:text> -> </xsl:text>
            <!-- right hand side of assignment supplies the next state -->
            <xsl:value-of select="rvalue/@name"/>
            <xsl:text>  [label = </xsl:text>
            <!-- take value of EventType from inner case statement id name  -->
            <xsl:value-of select="
                ../expr[@class='ID']/@name
              "/>
              <xsl:text>]; // switch switch
</xsl:text>
  </xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside the top switch statement this template handles assignments to nextState guarded by if statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 1=count(ancestor::block_items[@class='Case'])]">
        <xsl:text>    </xsl:text>
        <!-- top level switch statement labels the current state -->
        <xsl:value-of select="
            ancestor::block_items[@class='Case'][1]/expr[@class='ID']/@name
        "/>
        <xsl:text> -> </xsl:text>
        <!-- right hand side of assignment supplies the next state -->
        <xsl:value-of select="rvalue/@name"/>
        <xsl:text>  [label = </xsl:text>

    <!-- locate first enclosing context that tests EventType (such as some if statement) -->
    <xsl:for-each select="
            ancestor::*[
              .//*[@op='==' and .//field[@name='EventType']]
            ][1]
    ">
        <!-- inside the first enclosing context that tests EventType take the value it is tested against -->
        <xsl:value-of select="
            .//*[@op='==' and .//field[@name='EventType']]
            /right/@name
          "/>
    </xsl:for-each>
    <xsl:text>]; // switch if
</xsl:text>
</xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside the top switch statement this template handles assignments to nextState guarded by if statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 1=count(ancestor::block_items[@class='Case'])]">
        <xsl:text>    </xsl:text>
        <!-- top level switch statement labels the current state -->
        <xsl:value-of select="
            ancestor::block_items[@class='Case'][1]/expr[@class='ID']/@name
        "/>
        <xsl:text> -> </xsl:text>
        <!-- right hand side of assignment supplies the next state -->
        <xsl:value-of select="rvalue/@name"/>
        <xsl:text>  [label = </xsl:text>

      <!-- locate first enclosing context that tests EventType (such as some if statement) -->
      <xsl:for-each select="
            ancestor::*[
              .//*[@op='==' and .//field[@name='EventType']]
            ][1]
      ">
        <!-- inside the first enclosing context that tests EventType take the value it is tested against -->
        <xsl:value-of select="
            .//*[@op='==' and .//field[@name='EventType']]
            /right/@name
          "/>
      </xsl:for-each>
      <xsl:text>]; // switch if
</xsl:text>
</xsl:template>
```

```
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```
<!--
  Inside the top switch statement this template handles assignments to nextState guarded by if statements
-->
<xsl:template match="*[@op='=' and ./lvalue[@name='nextState'] and 1=count(ancestor::block_items[@class='Case'])]">
        <xsl:text>    </xsl:text>
        <!-- top level switch statement labels the current state -->
        <xsl:value-of select="
            ancestor::block_items[@class='Case'][1]/expr[@class='ID']/@name
        "/>
        <xsl:text> -> </xsl:text>
        <!-- right hand side of assignment supplies the next state -->
        <xsl:value-of select="rvalue/@name"/>
        <xsl:text>  [label = </xsl:text>

      <!-- locate first enclosing context that tests EventType (such as some if statement) -->
      <xsl:for-each select="
            ancestor::*[
              .//*[@op='==' and .//field[@name='EventType']]
            ][1]
      ">
        <!-- inside the first enclosing context that tests EventType take the value it is tested against -->
        <xsl:value-of select="
            .//*[@op='==' and .//field[@name='EventType']]
            /right/@name
          "/>
      </xsl:for-each>
      <xsl:text>]; // switch if
</xsl:text>
</xsl:template>
```

c_ast_xml_xslt.py

```python
import re
import sys
from pycparser import c_parser, c_ast, parse_file
from xml.dom import minidom
from lxml import etree as ET

def myshow(node, xnode=ET.Element("root"), buf=sys.stderr
    , offset=0, attrnames=False, nodenames=False
    , showcoord=False, _my_node_name=None):
    # ...

ast2 = parse_file("/dev/stdin", use_cpp=False)
xml2 = myshow(ast2, attrnames=True, nodenames=False)

if(len(sys.argv) < 2):
    print(prettify(ET.tostring(xml2, encoding='unicode')))
else:
    result = xml2
    for i in range(1,len(sys.argv)):
        xslt_doc = ET.parse(sys.argv[i])
        transformer = ET.XSLT(xslt_doc)
        result = transformer(result)
    print(result)
```

c_ast_xml_xslt.py

```python
import re
import sys
from pycparser import c_parser, c_ast, parse_file
from xml.dom import minidom
from lxml import etree as ET

def myshow(node, xnode=ET.Element("root"), buf=sys.stderr
    , offset=0, attrnames=False, nodenames=False
    , showcoord=False, _my_node_name=None):
    # ...

ast2 = parse_file("/dev/stdin", use_cpp=False)
xml2 = myshow(ast2, attrnames=True, nodenames=False)

if(len(sys.argv) < 2):
    print(prettify(ET.tostring(xml2, encoding='unicode')))
else:
    result = xml2
    for i in range(1,len(sys.argv)):
        xslt_doc = ET.parse(sys.argv[i])
        transformer = ET.XSLT(xslt_doc)
        result = transformer(result)
    print(result)
```

c_ast_xml_xslt.py

usage

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(smv) # wc -l sample/*
    310 sample/step000_c_code_containing_state_machine.c
   1294 sample/step001_c_code_after_cpp.c
   1137 sample/step002_c_code_supported_by_pycparser.c
   5873 sample/step003_abstract_syntax_tree.xml
     26 sample/step004_gv_digraph.gv
     23 sample/step005_gv_digraph_unique.gv
(smv) #
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(smv) # wc -l sample/*
   310 sample/step000_c_code_containing_state_machine.c
  1294 sample/step001_c_code_after_cpp.c
  1137 sample/step002_c_code_supported_by_pycparser.c
  5873 sample/step003_abstract_syntax_tree.xml
    26 sample/step004_gv_digraph.gv
    23 sample/step005_gv_digraph_unique.gv
(smv) #
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(smv) # wc -l sample/*
   310 sample/step000_c_code_containing_state_machine.c
  1294 sample/step001_c_code_after_cpp.c
  1137 sample/step002_c_code_supported_by_pycparser.c
  5873 sample/step003_abstract_syntax_tree.xml
    26 sample/step004_gv_digraph.gv
    23 sample/step005_gv_digraph_unique.gv
(smv) #
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(smv) # wc -l sample/*
   310 sample/step000_c_code_containing_state_machine.c
  1294 sample/step001_c_code_after_cpp.c
  1137 sample/step002_c_code_supported_by_pycparser.c
  5873 sample/step003_abstract_syntax_tree.xml
    26 sample/step004_gv_digraph.gv
    23 sample/step005_gv_digraph_unique.gv
(smv) #
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(smv) # wc -l sample/*
   310 sample/step000_c_code_containing_state_machine.c
  1294 sample/step001_c_code_after_cpp.c
  1137 sample/step002_c_code_supported_by_pycparser.c
  5873 sample/step003_abstract_syntax_tree.xml
    26 sample/step004_gv_digraph.gv
    23 sample/step005_gv_digraph_unique.gv
(smv) #
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
(smv) # wc -l sample/*
   310 sample/step000_c_code_containing_state_machine.c
  1294 sample/step001_c_code_after_cpp.c
  1137 sample/step002_c_code_supported_by_pycparser.c
  5873 sample/step003_abstract_syntax_tree.xml
    26 sample/step004_gv_digraph.gv
    23 sample/step005_gv_digraph_unique.gv
(smv) #
```

details

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```python
from pycparser import c_parser, c_ast, parse_file

# ...

ast2 = parse_file("/dev/stdin", use_cpp=False)

# ...
```

c_ast_xml_xslt.py

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
964- typedef __builtin_va_list __isoc_va_list;
965- # 145 "c:\\program files\\microchip\\xc32\\v4.10\\pic32mx\\include\\musl\\bits
966  typedef int ssize_t;
967- # 336 "c:\\program files\\microchip\\xc32\\v4.10\\pic32mx\\include\\musl\\bits
968  typedef long long off_t;
969- # 506 "c:\\program files\\microchip\\xc32\\v4.10\\pic32mx\\include\\musl\\bits
970  typedef struct _IO_FILE FILE;
971- # 23 "c:\\program files\\microchip\\xc32\\v4.10\\pic32mx\\include\\musl\\stdio
972- # 50 "c:\\program files\\microchip\\xc32\\v4.10\\pic32mx\\include\\musl\\stdio
973  typedef union _G_fpos64_t {
974   char __opaque[16];
975   double __align;
976  } fpos_t;
977
978- extern __attribute__((section(".data._stdout"))) FILE *const *_stdout;
979- extern __attribute__((section(".data._stderr"))) FILE *const *_stderr;
980- extern __attribute__((section(".data._stdin"))) FILE *const *_stdin;
981
```

```
832  typedef int ssize_t;
833  typedef long long off_t;
834  typedef struct _IO_FILE FILE;
835  typedef union _G_fpos64_t {
836   char __opaque[16];
837   double __align;
838  } fpos_t;
839
840
```

```
| egrep -avi '^#|va_list|__attribute__' \
| perl -pe's{__extension__}{ }g;' \
```

```diff
422
423- __extension__ typedef unsigned long long UINT64;
424
425
426  typedef union
427  {
428      UINT8 Val;
429      struct
430      {
431-         __extension__ UINT8 b0:1;
432-         __extension__ UINT8 b1:1;
433-         __extension__ UINT8 b2:1;
434-         __extension__ UINT8 b3:1;
435-         __extension__ UINT8 b4:1;
436-         __extension__ UINT8 b5:1;
437-         __extension__ UINT8 b6:1;
438-         __extension__ UINT8 b7:1;
439      } bits;
440  } UINT8_VAL, UINT8_BITS;
```

```diff
314
315+  typedef unsigned long long UINT64;
316
317
318  typedef union
319  {
320      UINT8 Val;
321      struct
322      {
323+         UINT8 b0:1;
324+         UINT8 b1:1;
325+         UINT8 b2:1;
326+         UINT8 b3:1;
327+         UINT8 b4:1;
328+         UINT8 b5:1;
329+         UINT8 b6:1;
330+         UINT8 b7:1;
331      } bits;
332  } UINT8_VAL, UINT8_BITS;
```

```diff
1028 int snprintf(char *restrict, size_t, const char *restrict, ...);
1029
1030- int vprintf(const char *restrict, __isoc_va_list);
1031- int vfprintf(FILE *restrict, const char *restrict, __isoc_va_list);
1032- int vsprintf(char *restrict, const char *restrict, __isoc_va_list);
1033- int vsnprintf(char *restrict, size_t, const char *restrict, __isoc_va_list);
1034
1035 int scanf(const char *restrict, ...);
1036 int fscanf(FILE *restrict, const char *restrict, ...);
1037 int sscanf(const char *restrict, const char *restrict, ...);
1038- int vscanf(const char *restrict, __isoc_va_list);
1039- int vfscanf(FILE *restrict, const char *restrict, __isoc_va_list);
1040- int vsscanf(const char *restrict, const char *restrict, __isoc_va_list);
1041
1042 void perror(const char *);
```

```diff
887  int snprintf(char *restrict, size_t, const char
888
889
890  int scanf(const char *restrict, ...);
891  int fscanf(FILE *restrict, const char *restrict,
892  int sscanf(const char *restrict, const char *res
893
894  void perror(const char *);
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

usage

```
12    // transitions
13    init -> InitPState[label = ES_INIT];
14    InitPState -> Hiding  [label = ES_INIT];
15    Hiding -> Moving  [label = DARK_TO_LIGHT];
16    Moving -> Hiding  [label = LIGHT_TO_DARK];
17    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
18    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
19    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
20    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
21    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
22    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];
```

→

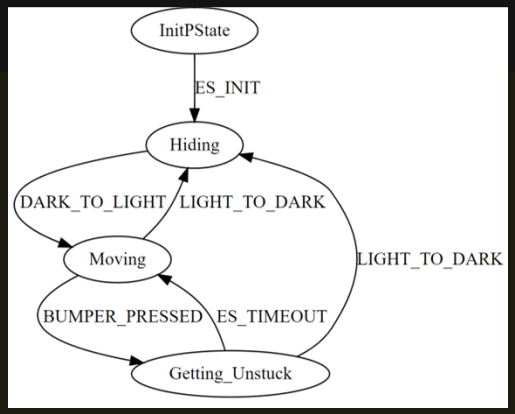
```
12    // transitions
13    init -> InitPState[label = ES_INIT];
14    InitPState -> Hiding  [label = ES_INIT];
15    Hiding -> Moving  [label = DARK_TO_LIGHT];
16    Moving -> Hiding  [label = LIGHT_TO_DARK];
17    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
18    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
19    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];
```

```
switch (CurrentState) {
    // ...
    case Moving:
        if (ThisEvent.EventType == LIGHT_TO_DARK) {
            stopMovement();
            nextState = Hiding;
            makeTransition = TRUE;
            ThisEvent.EventType = ES_NO_EVENT;
        }
        if (ThisEvent.EventType == BUMPER_PRESSED) {
        if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(90);
                ES_Timer_InitTimer(0,3000);
                nextState = Getting_Unstuck;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
        }
        else if(ThisEvent.EventParam == 2) {
                previousBumpers = 2; backUpLeft(90);
                ES_Timer_InitTimer(0,3000);
                nextState = Getting_Unstuck;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
        }
        else if(ThisEvent.EventParam == 4) {
                turnLeft(90);
                ES_Timer_InitTimer(0,3000);
                nextState = Getting_Unstuck;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
        }
        else if(ThisEvent.EventParam == 8) {
                turnRight(90);
                ES_Timer_InitTimer(0,3000);
                nextState = Getting_Unstuck;
                makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
        }
        }
        break;
```

```
// transitions
init -> InitPState[label = ES_INIT];
InitPState -> Hiding  [label = ES_INIT];
Hiding -> Moving  [label = DARK_TO_LIGHT];
Moving -> Hiding  [label = LIGHT_TO_DARK];
Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
Getting_Unstuck -> Moving  [label = ES_TIMEOUT];
```

```c
ES_Event RunTemplateFSM(ES_Event ThisEvent)
{
    uint8_t makeTransition = FALSE;
    TemplateFSMState_t nextState;  ES_Tattle();

    switch (CurrentState) {

        case InitPState:
            if (ThisEvent.EventType == ES_INIT)
            {
                nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }
            break;

        case Hiding:
            switch (ThisEvent.EventType) {

                case ES_ENTRY: break;

                case ES_EXIT: break;

                case DARK_TO_LIGHT:
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                    break;

                default:
                    break;
            }
            break;

        case Moving:
            if (ThisEvent.EventType == LIGHT_TO_DARK) {
                stopMovement(); nextState = Hiding;
                makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == BUMPER_PRESSED) {
            if(ThisEvent.EventParam == 1) {
                previousBumpers = 1; backUpRight(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            else if(ThisEvent.EventParam == 2) {
                    previousBumpers = 2; backUpLeft(TURNING_SPEED);
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                    nextState = Getting_Unstuck; makeTransition = TRUE;
                    ThisEvent.EventType = ES_NO_EVENT;
            }
            // ...
            }
            break;

        case Getting_Unstuck:
            if (ThisEvent.EventType == LIGHT_TO_DARK)  {
                stopMovement(); nextState = Hiding; makeTransition = TRUE;
                ThisEvent.EventType = ES_NO_EVENT;
            }

            if (ThisEvent.EventType == ES_TIMEOUT
                && ThisEvent.EventParam == SIMPLE_SERVICE_TIMER) {
                if(Roach_ReadBumpers() == 0) {
                    goForward(MOVING_SPEED); nextState = Moving;
                    makeTransition = TRUE; ThisEvent.EventType = ES_NO_EVENT;
                } else {
                    ES_Timer_InitTimer(SIMPLE_SERVICE_TIMER,TURN_TIME);
                }
            }
            break;

        default:
            break;
    }

    if (makeTransition == TRUE) {
        RunTemplateFSM(EXIT_EVENT); CurrentState = nextState;
        RunTemplateFSM(ENTRY_EVENT);
    }
    ES_Tail();
    return ThisEvent;
}
```

```graphviz
digraph fsm {

    init [shape = "point", color = "black"
      ,style="filled",width=.1,forcelabels=false];

    // states
    InitPState;
    Hiding;
    Moving;
    Getting_Unstuck;

    // transitions
    init -> InitPState[label = ES_INIT];
    InitPState -> Hiding  [label = ES_INIT];
    Hiding -> Moving  [label = DARK_TO_LIGHT];
    Moving -> Hiding  [label = LIGHT_TO_DARK];
    Moving -> Getting_Unstuck  [label = BUMPER_PRESSED];
    Getting_Unstuck -> Hiding  [label = LIGHT_TO_DARK];
    Getting_Unstuck -> Moving  [label = ES_TIMEOUT];

}
```



extras

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

```
sf=(
        "step000_c_code_containing_state_machine.c"
        "step001_c_code_after_cpp.c"
        "step002_c_code_supported_by_pycparser.c"
        "step003_abstract_syntax_tree.xml"
        "step004_gv_digraph.gv"
        "step005_gv_digraph_unique.gv" )
(
    target=TemplateFSM.c
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && cat $target > "${sp}/${sf[0]}" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        $target \
) \
    | tee "${sp}/${sf[1]}" \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | tee "${sp}/${sf[2]}" \
    | tee >( python3 c_ast_xml_xslt.py > "${sp}/${sf[3]}" ) \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | tee "${sp}/${sf[4]}" \
    | uniq \
    | tee "${sp}/${sf[5]}" \
    > gv_digraph.gv
```

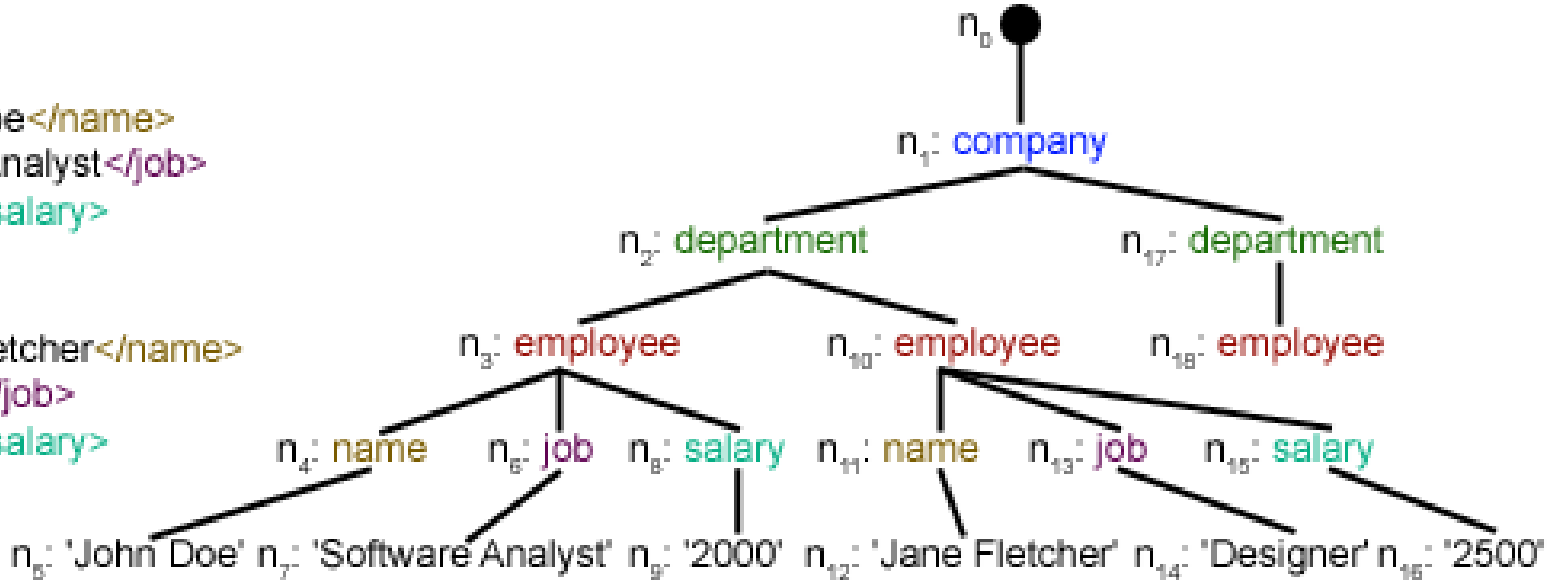
```
(smv) # wc -l sample/*
   310 sample/step000_c_code_containing_state_machine.c
  1294 sample/step001_c_code_after_cpp.c
  1137 sample/step002_c_code_supported_by_pycparser.c
  5873 sample/step003_abstract_syntax_tree.xml
    26 sample/step004_gv_digraph.gv
    23 sample/step005_gv_digraph_unique.gv
(smv) #
```

```
(
    cd "/mnt/c/Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X/" \
    && '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' \
        -I'C:\Program Files/Microchip/xc32/v4.10/pic32mx/include/' \
        -I'C:\Users/chris/OneDrive/Desktop/ECE118/include' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/FSM_Roach.X' \
        -I'C:\Users/chris/OneDrive/Desktop/MPLAB_Projects/EventChecking.X/' \
        TemplateFSM.c \
) \
    | egrep -avi '^#|va_list|__attribute__' \
    | perl -pe's{__extension__}{ }g;' \
    | python3 c_ast_xml_xslt.py gv_digraph.xslt \
    | uniq \
    > gv_digraph.gv
```

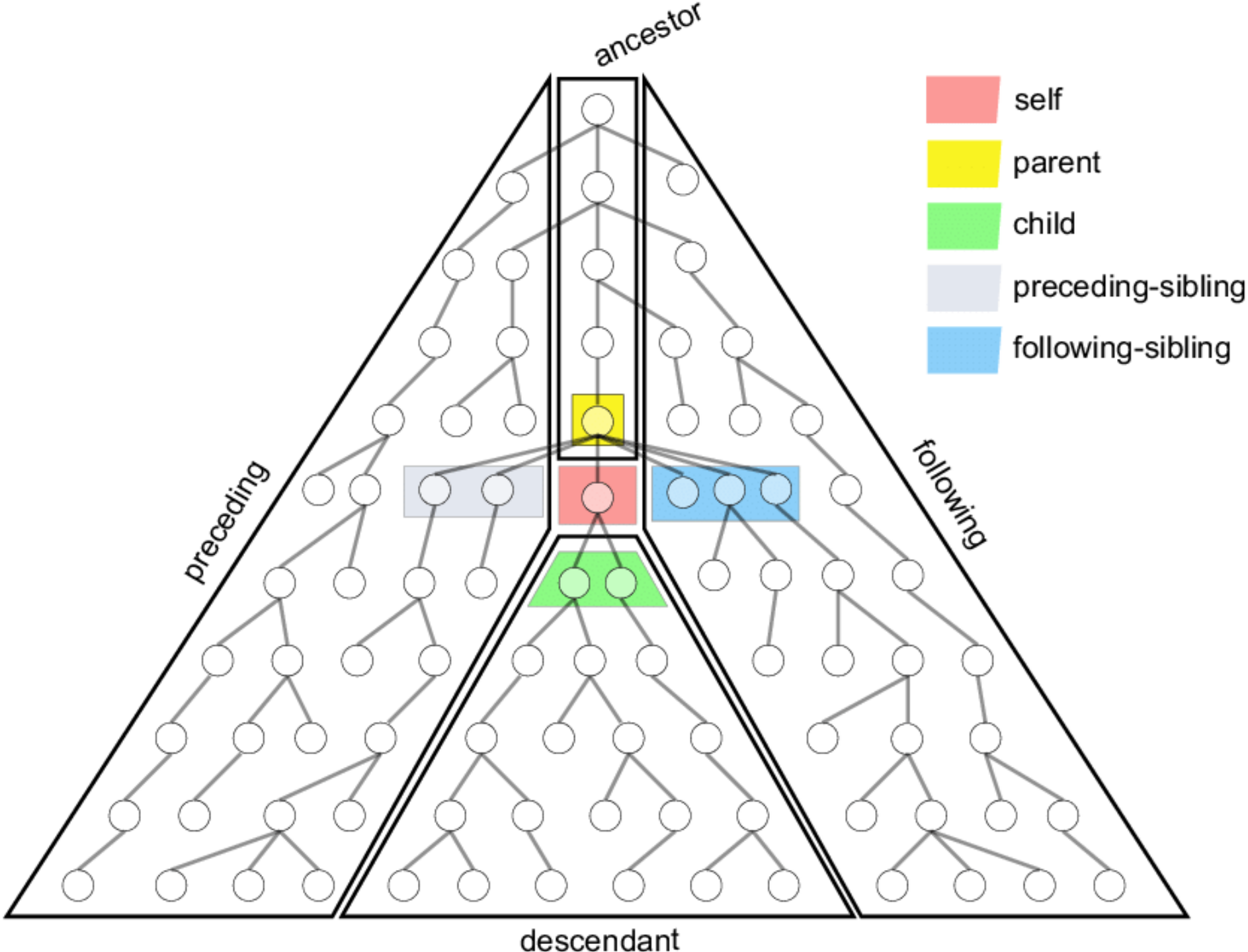
usage

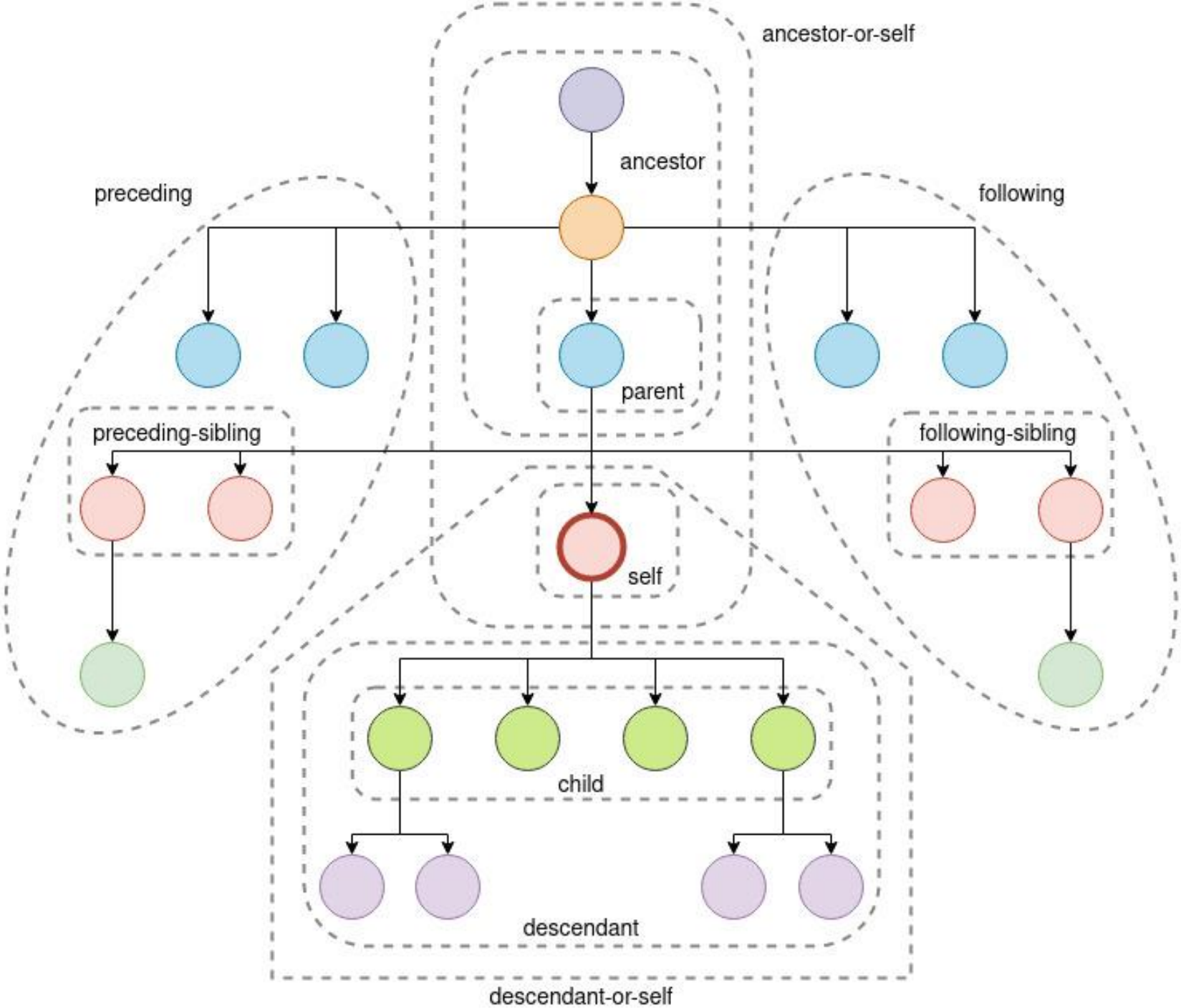
```
$strings '/mnt/c/Program Files/Microchip/xc32/v4.10/bin/xc32-cpp.exe' | head -1
!This program cannot be run in DOS mode.

$strings `which python3` | head -1
/lib64/ld-linux-x86-64.so.2

$wsl.exe -l -v
  NAME      STATE           VERSION
* Ubuntu    Running         2
$
```

```
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```



Pick elements from anywhere in the DOM with a specific HTML tag	//tagName
Pick an element by ID	//tagName[@id="idValue"]
Pick an element by class	//tagName[@class="classValue"]
Pick the child of an element	//parentName/childName
Pick the first child of an element	//parentName/childName[1]
Pick the parent of an element	//childName/..


ancestor
self
parent
child
preceding-sibling
following-sibling
preceding
following
descendant


ancestor-or-self
ancestor
preceding
following
parent
preceding-sibling
following-sibling
self
child
descendant
descendant-or-self

1. `paper/publisher/parent::*/author`
2. `/bib//address[ancestor::book]`
3. `/bib//author/ancestor::*//zip`

1. The first XPath expression is using the `parent::` axis with an element wildcard node test. From the `publisher` element, it "jumps up" to it's parent, the `paper` element, and then selects the `author` child element. It only selects that `author` element if the `publisher` sibling is present. It could be written more compactly and avoid looking up the node tree with this XPath using a predicate filter on the `paper` element: `paper[publisher]/author`.
2. That XPath selects all of the `address` elements that are descendants of the `bib` element and are also descendants of a `book` element(which could be the ancestor or descendant of the `bib` ancestor). The square braces are a predicate, which filters the nodes that pass the test for the expression. In this case, it must be a descendant of a `book` element in order to "pass" the test and be selected.
3. This XPath looks for any `zip` element that is the descendant of an element that is the ancestor of an `author` element that is the descendant of a `bib` element.

https://stackoverflow.com/questions/43462759/xpath-concept-confusion