Automated Visualization for Flat and Hierarchical State Machines

Jasmine Lesner, Gabriel Hugh Elkaim

***Abstract*—Finite State Machines (FSMs) are crucial for event-driven control systems, enabling simplified decision-making through state transitions. However, the increasing complexity of FSMs, marked by the addition of states and events, significantly complicates debugging and feature integration. Traditional state diagram tools require manual inputs or source code annotations, making them susceptible to errors and inefficiencies. This paper introduces an innovative tool that automates the generation of accurate state diagrams from FSM source code. The tool leverages naming conventions and Abstract Syntax Tree (AST) patterns, utilizing a pipeline of XSLT transformations. It offers full automation for standard coding practices, while providing flexibility for non-standard conventions through customizable XSLT templates. This approach allows users to adapt the tool for different coding styles and enhances the process of designing, debugging, and updating FSMs, ensuring that the visual representations always align with the implemented code.**

***Index Terms*—Finite State Machines (FSMs), Automated Visualization, State Diagram Generation, Source Code Analysis, Abstract Syntax Tree (AST), XSLT Transformations, Event-Driven Control Systems, Debugging and Feature Integration, Coding Conventions, Software Tools for FSMs.**

I. Introduction

FINITE state machines (FSMs) are crucial for event-driven control systems, enabling simplified decision-making through state transitions. However, the increasing complexity of FSMs, marked by the addition of states and events, significantly complicates debugging and feature integration.

A state diagram can provide a high level map of how an FSM operates. Having a map to navigate FSM logic helps developers design, debug and update FSMs however creating and updating such a map manually is a tedious task and due to human errors and feature creep one can never be sure a state diagram matches the code that implements the FSM.

A. Diagram Tools

Diagram tools like Graphviz [1] (also Mermaid.JS [2], PlantUML [3], ...) require diagrams to be already described using their visualization language. Tools like Doxygen [4] require source code to be annotated for state diagram generation. Unified Modeling Language [5] IDE tools like "Enterprise Architect" [6] need manual intervention for FSM diagram creation. No tool found can automatically generate diagrams directly from source code.

Autonomous Systems Lab, University of California, Santa Cruz, CA 95064
https://asl.soe.ucsc.edu/home

B. Automatic Diagrams

FSM code typically involves a series of checks: current state, last event, event parameters, and guard conditions. Implementations can vary, using structures like switch-default or if-elseif-else statements, and the sequence of checks can differ. This variability poses a challenge: **How can we automatically generate accurate visual representations of FSMs from their source code?**

To address this, we developed a tool that extracts state diagrams from source code. It uses naming conventions and Abstract Syntax Tree (AST) patterns, employing a pipeline of XSLT [7]. This tool is fully automated when standard code conventions are followed. For non-standard conventions, it offers flexibility through modifiable XSLT templates. Users can adapt the tool to alternative naming conventions either by altering the XSLT directly or by preprocessing the source code. When encountering unfamiliar variable names and coding styles, the tool's AST pattern recognition can be expanded with new or updated XSLT templates. This approach ensures that any enhancements in the diagram generation process are immediately reflected across all diagrams, facilitating efficient and accurate visualization of FSM implementations.

II. Method

The tool operates in three stages:

1) The first stage reads source code and generates an abstract syntax tree AST
2) The second stage analyzes and annotates the AST with tags relevant for a state diagram.
3) The third stage uses the AST tags to generate a diagram description which is then rendered visually in various formats (PNG, SVG, PDF)

A. Stage One: AST Generation

1) Supported Inputs: We designed our tool to interpret FSMs in an embedded C variant for PIC32MX microcontrollers, a cost-effective 32-bit MCU family with versatile memory and integrated peripherals. This technology is used in UCSC classrooms [8] for developing robotic applications with Microchip's MPLAB X IDE [9] and MPLAB XC Compilers [10], ranging from basic movement to complex autonomous functions.

2) Keywords and Constructs: The embedded C variant for PIC32MX microcontrollers uses C language elements like `va_list`, `__attribute__`, and `__extension__`, which are not recognized by some parsers like PycParser [11]. These elements, unnecessary for our diagram generation, are eliminated

```
 1 find "$src_path" -type f -name '*.c' -print0 \
 2     | xargs -0 egrep -l nextState \
 3     | while read f ; do
 4         ff="`basename \"${f}\"`"
 5         b="`dirname \"${f}\"`"
 6         (
 7             cd "$b" \
 8             && echo "amalgamating '${f}'" \
 9             && cat "${ff}" \
10                 | dos2unix \
11                 | perl -p "${epath}" \
12                 | ( egrep -avi '^#define '  || true ) \
13                 > "${ff}.undef" \
14             && echo "
15                 cpp
16                 -I\"${course_include_path}\"
17                 -I\"${pic32mx_include_path}\" \
18                 $ilist $iconfig2 -I'${b}' -I. '${ff}.undef' \
19                 " \
20                 | bash \
21                 | perl -pe '
22                     s{zz0912819zz}{}g;
23                 ' \
24                 | dos2unix \
25                 > "${ff}.cp5" \
26             && rm -f "${ff}.undef"
27         ) 2>&1
28     done
29
30 find "$src_path" -name '*.c.cp5' \
31 | while read f ; do
32 echo "visualizing '$f'"
33 (
34     sx=saxonb-xslt
35     cat "$f" \
36         | tr -d '\r' \
37         | ( egrep -avi '^[[:blank:]]*$|^#|va_list|__attribute__' || true ) \
38         | perl -pe's{__extension__}{ }g; s{__}{}g; ' \
39         | python3 c_ast_xml.py \
40         | tee "${f}.xml" \
41         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00005_identity.xml \
42         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00100_declutter_attributes.xml \
43         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00200_add_bLine_eLine.xml \
44         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_CurrentStateTest.xml \
45         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_EventParamTest.xml \
46         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_EventTypeTest.xml \
47         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_NextStateLabel.xml \
48         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00400_add_CascadeElements.xml \
49         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00500_add_CascadeLabel.xml \
50         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00550_add_EventLabel.xml \
51         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00560_add_Guard_Element.xml \
52         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00570_add_Guard_Attributes.xml \
53         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00600_add_onEntry_onExit.xml \
54         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00600_add_onTransition2.xml \
55         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00620_drop_unwanted_code.xml \
56         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00800_gv_digraph4.xml \
57         | perl -pe 's/ && / &amp;&amp; /g;
58                     s/ < / &lt; /g;
59                     s/ > / &gt; /g;
60                     s/ <= / &lt;= /g;
61                     s/ >= / &gt;= /g;
62                 ' \
63         > "${f}.gv"
64
65         dot -Tpng "${f}.gv" -o "${f}.png"
66         dot -Tpdf "${f}.gv" -o "${f}.pdf"
67         dot -Tsvg "${f}.gv" -o "${f}.svg"
68 ) 2>&1
69 done
```

Fig. 1. Two principal commands power our tool. The first command (spanning lines 1-28) prepares C code for parsing. During preparation some macros are protected from expansion (line 11) and after cpp this protection is removed (lines 21-23). The second command (spanning lines 30-69) builds the AST (line 39), runs the annotation pipeline (lines 41-55) and generates diagrams (lines 56-67).

using regular expressions. Additionally, superfluous elements such as empty lines and comments are also removed.

3) Macro Encoding: C programs use macros, e.g., `#include "stdio.h"` and `#define FRONT_BUMBER 0x42`. These are processed by the C Preprocessor (CPP) [12], which enables macro functions, file inclusion, and conditional compilation. The `#include` macros need merging, and `#define` macros replace text in the code. In diagrams, it's beneficial to display macro names like `FRONT_BUMBER` instead of their expanded forms (e.g., `0x42`). Therefore, our tool selectively suppresses some macro expansions during CPP processing. This is achieved by protecting them from expansion, and later removing this protection. The protection is added (figure 1 line 11) by an adhoc script the generation of which is shown in figure 2.

```
 1 epath="$src_path"/encode.pl
 2
 3 find \
 4     "${src_path}" \
 5     "${course_include_path}" \
 6     -type f \( -name '*.h' -o -name '*.hpp' -o -name '*.c' \) \
 7     | tr "\n" "\0" \
 8     | xargs -0 cat \
 9     | dos2unix \
10     | ( egrep -ai '^#define' || true ) \
11     | perl -pe 's/#define (\w)(\w+)[ \(].*$/s{\\b$1$2\\b}{$1zz0912819zz$2}g; #
       encode123 /g;' \
12     | ( grep encode123 || true ) \
13     | perl -pe 's/ # encode123//g;' \
14     | sort | uniq \
15     > "${epath}"
```

Fig. 2. Generation of adhoc encoding script to protect macros. Line 11 in figure 1 adds the macro protection and lines 21-23 remove it.

4) Apply CPP: After filtering out unsupported keywords and encoding macros, we use CPP to expand #include files. Post-CPP, the macro protections are removed, reverting them to their original names.

5) Construct AST: A Python script processes the CPP output, creating an XML with two sections: "code" and "ast". The "code" section lists the input C code with line numbers, useful for diagram annotations. The "ast" section contains the corresponding AST, as generated by PycParser.

B. Stage Two: AST Annotation

In this stage, we annotate the AST using a series of XSLT steps, facilitating independent inspection and development of each annotation phase.

1) XML Normalize: Initially, we normalize the XML AST to enhance readability and track changes more efficiently. This involves removing unnecessary whitespace and maintaining the integrity of all XML elements and attributes. Indentation is used for clear visualization of the AST's tree structure.

2) AST Declutter: We simplify the AST by removing redundant elements and attributes generated by PycParser that are not required for state diagrams. Attributes like `quals`, `align`, `storage`, `funcspec`, and `line` (when null) are omitted, along with any empty attributes, using targeted XSLT rules. This decluttering focuses on creating a cleaner, more navigable AST.

3) bLine / eLine: Each AST element is assigned `bLine` and `eLine` attributes, marking the start and end line numbers in the original C code, respectively. This facilitates linking AST elements to their corresponding source code lines, essential for illustrating logic in state diagrams.

4) CurrentStateTest: For `case` and `default` elements within `switch` statements checking `CurrentState`, we add a `CurrentStateTest` attribute, reflecting the state name represented by that case. This annotation is extendable to `if-elseif-else` patterns if encountered.

5) EventParamTest: We tag AST elements within conditional statements involving `EventParam` with an `EventParamTest` attribute, indicating the specific `EventParam` being tested.

6) EventTypeTest: Similar to `EventParamTest`, conditional statements involving `EventType` are tagged with an `EventTypeTest` attribute, specifying the `EventType` under consideration.

7) NextStateLabel: Elements indicating state changes (class `Assignment`, operation `=`, and `nextState` on the left side) receive a `NextStateLabel` attribute, denoting the new state as defined in the assignment's right-hand value.

8) CascadeElements: `Case` and `Default` elements following uninterrupted `Case` elements (without a `Break`) gain `CascadeElement` children, representing each cascading case value.

9) CascadeLabel: A `CascadeLabel` attribute is formed by merging the current case value with all `CascadeElement` values, separated by `" or "`. This label collectively represents switch branches that cascade together.

10) EventLabel: Elements with `NextStateLabel` are also tagged with an `EventLabel`, combining relevant `EventType` and `EventParam` values.

11) GuardElements: `If` statements leading to state transitions but not checking Event attributes are marked with a `guard` child element, encapsulating the condition's code. This highlights the triggering logic in diagrams.

12) GuardLabel: To uniquely identify guards, we use `CurrentStateTest` and `NextStateLabel` attributes, with the guard's line number serving as an identifier. The `EventLabel` differentiates true and false conditions.

13) onEntry / onExit: `onEntry` and `onExit` elements are added, populated with code executed upon entering and exiting states, respectively.

14) onTransition: The `onTransition` element, filled with code executed during state transitions, is added. This information is displayed alongside event labels in the state diagram.

15) Code Declutter: We remove code lines that are redundant or non-essential, such as references to `nextState`, `makeTransition`, and `ThisEvent.EventType`. This is because their actions are already represented diagrammatically.

C. Stage Three: Diagram Generation

Once AST annotations are applied they are used to generate a description of a diagram in the GraphViz [1] diagram description language. This is done in four steps by XSLT in figure 3:

1) Step: Diagram Setup: Output format is set to plain text, suitable for Graphviz format and the initial starting state for the diagram is identified.

2) Step: Loop over States: We loop through AST elements representing different states, excluding the initial state and guard conditions. These are formatted with matching styles and labels including `onEntry` and `onExit` code blocks.

3) Step: Loop over Guards: We loop through guard conditions associated with state transitions, adding them to the digraph with their specific style.

4) Step: Loop over Transitions: Last we loop through state transitions adding them to the diagram description with their `onTransition` code blocks.

III. Results

A. Input & Output Samples

Figure 6 displays the state diagram generated from FSM code in figure 5. This FSM, representing the primary level in a hierarchical state machine (HSM), controls a wheeled robot modeled after a cockroach (shown in figure 4). It exhibits behaviors like moving in darkness and freezing in light, with an added periodic 'jig dance'. While each top-level HSM state contains a nested FSM, these are omitted for brevity.

Figure 8 presents the FSM derived from the code in figure 7, which is a lower-level FSM in a multi-tiered HSM for a competition robot. This complex FSM includes labels like `if(barrierCount < BARRIER_COUNT)` and `if ((fieldSide == FIELD_LEFT)...` demonstrating our tool's capability to manage even chained state transition guard conditions. The diagram also exemplifies the labeling of state diagram elements with corresponding source code.

B. Tool Benchmarks

The tool underwent benchmarking on WSL2 Ubuntu Linux on top of a Windows 10 Pro host, powered by an Intel Core i7-8850H CPU. This setup features six physical cores, with twelve hyper-threaded virtual cores, operating between 800MHz and 4200MHz.

Table I lists the outcomes of four benchmark runs, each time processing identical code files to generate thirteen state diagrams. Two of these diagrams are shown in figures 6 and 8 generated from code in figures 5 and 7. The tests were conducted on a laptop plugged into AC power, using Windows 10 Pro default power profile settings. During the tests, three virtual cores were occupied with background tasks, leaving nine cores primarily for our benchmarking.

The benchmark results indicate that:

1) **Diagram Generation Time:** It takes less than ten seconds to generate one state diagram, with a 20-25% time variation between the fastest and slowest runs. This discrepancy is likely due to thermal throttling affecting CPU performance.
2) **CPU Utilization vs. Elapsed Time:** Contrary to expectations, higher CPU utilization did not correlate with shorter elapsed times. The longest processing times coincide with the highest CPU usages, suggesting that thermal throttling is slowing down the cores, increasing the overall time despite seemingly higher CPU % usage.
3) **Core Utilization Efficiency:** The tool utilizes eight of the nine available virtual cores, leaving limited scope for further parallelization on our test system. While servers with more cores might benefit from concurrent diagram generation, our users (UCSC students) are unlikely to see significant performance improvements on standard laptops or PCs from additional parallel processing capabilities.

TABLE I
BENCHMARK RESULTS

Run	Percent of CPU	Elapsed Time
Run 1	821%	1:21.22
Run 2	808%	1:17.26
Run 3	819%	1:35.38
Run 4	816%	1:16.48

```
  1 <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  2     <xsl:output method="text"/>
  3     <xsl:template match="/">
  4         <xsl:variable name="InitState" select="//ext[@class='Decl' and @name='
       CurrentState']/init/@name"/>
  5         <xsl:text>
  6 digraph fsm {
  7     </xsl:text>
  8     <xsl:value-of select="$InitState"/>
  9     <xsl:text>[shape = "point", color = "black",style="filled",width=.1,forcelabels
       =false];
 10 
 11     // states
 12     node [shape=plaintext]
 13     </xsl:text>
 14         <xsl:for-each select="
 15             //*[
 16                 @CurrentStateTest
 17                     and not(@CurrentStateTest = '')
 18                     and not(@CurrentStateTest = $InitState)
 19                     and not(guard)
 20                 ]">
 21             <xsl:text>
 22     </xsl:text>
 23             <xsl:value-of select="@CurrentStateTest"/>
 24 <xsl:text><![CDATA[ [label=<<TABLE BORDER="1" CELLBORDER="0" CELLSPACING="0" style
       ="rounded">
 25         <TR>
 26             <TD BORDER="1" SIDES="B">]]></xsl:text>
 27             <xsl:value-of select="@CurrentStateTest"/>
 28             <xsl:text><![CDATA[</TD>
 29         </TR>]]></xsl:text>
 30         <xsl:if test="stmts[@class='Assignment' and rvalue/args/exprs/@name='
       ThisEvent']/lvalue[@name='ThisEvent']">
 31             <xsl:text><![CDATA[
 32         <TR>
 33             <TD ALIGN="LEFT">]]></xsl:text>
 34         <xsl:value-of select="stmts[@class='Assignment']/rvalue/name/@name"/>
 35         <xsl:text><![CDATA[</TD>
 36         </TR>
 37             ]]></xsl:text>
 38         </xsl:if>
 39         <xsl:text><![CDATA[
 40         <TR>
 41             <TD ALIGN="LEFT"><B>/Entry: </B></TD>
 42         </TR>]]></xsl:text>
 43             <xsl:for-each select="onEntry/line">
 44                 <xsl:text><![CDATA[
 45         <TR><TD ALIGN="LEFT">]]></xsl:text>
 46                 <!-- <xsl:value-of select="normalize-space(.)"/> -->
 47                 <xsl:value-of select="."/>
 48                 <xsl:text><![CDATA[</TD></TR>]]></xsl:text>
 49             </xsl:for-each>
 50             <xsl:text><![CDATA[
 51         <TR>
 52             <TD ALIGN="LEFT"><B>/Exit: </B></TD>
 53         </TR>
 54         ]]></xsl:text>
 55             <xsl:for-each select="onExit/line">
 56                 <xsl:text><![CDATA[<TR><TD ALIGN="LEFT">]]></xsl:text>
 57                 <!-- <xsl:value-of select="normalize-space(.)"/> -->
 58                 <xsl:value-of select="."/>
 59                 <xsl:text><![CDATA[</TD></TR>
 60         ]]></xsl:text>
 61             </xsl:for-each>
 62             <xsl:text><![CDATA[</TABLE>>];
 63 ]]></xsl:text>
 64         </xsl:for-each>
 65     <xsl:text>
 66     // guards
 67 </xsl:text>
 68 
 69     <xsl:for-each select="
 70             //*[
 71                 @CurrentStateTest
 72                     and not(@CurrentStateTest = '')
 73                     and not(@CurrentStateTest = 'InitPSubState')
 74                     and guard
 75                 ]">
 76             <xsl:text>
 77     </xsl:text>
 78             <xsl:value-of select="@CurrentStateTest"/>
 79 <xsl:text><![CDATA[ [shape=point, xlabel="]]></xsl:text>
 80 
 81             <xsl:for-each select="guard/line">
 82                 <!-- <xsl:value-of select="normalize-space(.)"/> -->
 83                 <xsl:value-of select="."/>
 84                 <xsl:text>
 85 </xsl:text>
 86             </xsl:for-each>
 87             <xsl:text><![CDATA["];
 88 ]]></xsl:text>
 89         </xsl:for-each>
 90         <xsl:text>
 91 
 92     // transitions
 93 </xsl:text>
 94 
 95     <xsl:for-each select="//*[ @NextStateLabel ]">
 96         <xsl:value-of select="ancestor::*[@CurrentStateTest][1]/@CurrentStateTest"/
       >
 97         <xsl:text> -> </xsl:text>
 98         <xsl:value-of select="@NextStateLabel"/>
 99         <xsl:text><![CDATA[[label=<<TABLE BORDER="0" CELLBORDER="0">
100     <TR><TD BORDER="1" SIDES="B">]]></xsl:text>
101         <xsl:value-of select="@EventLabel"/>
102         <xsl:text><![CDATA[</TD></TR>]]></xsl:text>
103         <xsl:for-each select="onTransition/line">
104             <xsl:text><![CDATA[
105             <TR><TD ALIGN="LEFT">]]></xsl:text>
106             <xsl:value-of select="."/>
107             <xsl:text><![CDATA[</TD></TR>]]></xsl:text>
108         </xsl:for-each>
109         <xsl:text><![CDATA[
110     </TABLE>>];
111 ]]></xsl:text>
112     </xsl:for-each>
113         <xsl:text>
114 }</xsl:text>
115     </xsl:template>
116 
117 </xsl:stylesheet>
```

Fig. 3. XSLT to generate a GraphViz diagram description from an annotated AST. Lines 1-10 handle diagram setup. Lines 11-65 loop over states. Lines 66-91 loop over guards. Lines 92-177 loop over transitions

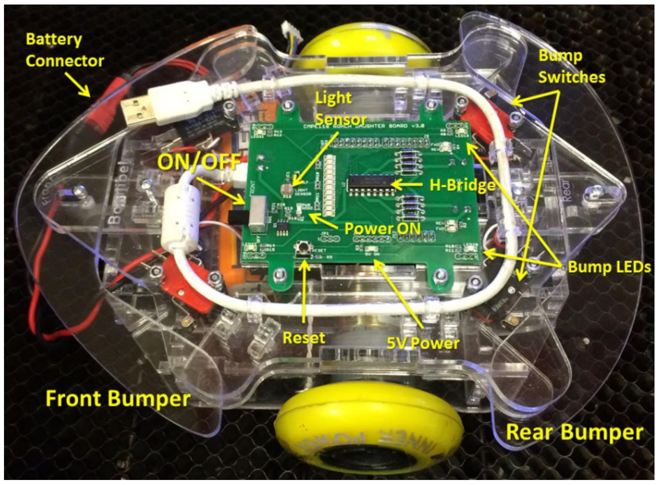


Fig. 4. A wheeled robot controlled by code in figure 5

IV. Discussion

A. AST XPATH

Initially, we employed regular expression patterns [13] for diagram generation data extraction. This method fell short as it treated source code linearly, struggling with nested structures like switch-default and if-elseif-else constructs.

To overcome these limitations, we shifted to using a C parser and Abstract Syntax Trees (ASTs). ASTs represent the hierarchical nature of source code, enabling us to use XPATH [14], a pattern language designed for tree structures. This approach is more effective than regular expressions for parsing nested code patterns.

Consider this XPATH used in our tool:

`ancestor::*[@CurrentStateTest][1]/@CurrentStateTest`

This XPATH works as follows:

- `ancestor::*[@CurrentStateTest][1]`: It locates the nearest ancestor element with a
- `CurrentStateTest` attribute in the AST hierarchy. The process involves:
  - `ancestor::*` to select all ancestor elements.

```
ES_Event RunTemplateHSM(ES_Event ThisEvent) {
   uint8_t makeTransition = FALSE; TemplateHSMState_t nextState; ES_Tattle();

   switch (CurrentState) {
   case InitPState:
      if (ThisEvent.EventType == ES_INIT) {
         InitLightSubHSM(); InitDarkSubHSM(); InitJigSubHSM(); ES_Timer_SetTimer(
                ↪ JIG_TIMER, JIG_TIME); nextState = InDark; makeTransition = TRUE;
                ↪  ThisEvent.EventType = ES_NO_EVENT;
      }
      break;

   case InLight:
      ThisEvent = RunLightSubHSM(ThisEvent);
      switch (ThisEvent.EventType) {
      case ES_ENTRY: ES_Timer_InitTimer(JIG_TIMER, JIG_TIME); break;
      case ES_EXIT: ES_Timer_StopTimer(JIG_TIMER); break;
      case LIGHT_TO_DARK: nextState = InDark; makeTransition = TRUE; ThisEvent.
             ↪ EventType = ES_NO_EVENT; break;
      case ES_TIMEOUT: nextState = Jig; makeTransition = TRUE; ThisEvent.EventType =
             ↪  ES_NO_EVENT; ES_Timer_SetTimer(JIG_SPIN_TIMER, JIG_SPIN_TIME);
             ↪ break;
      }
      break;

   case InDark:
      ThisEvent = RunDarkSubHSM(ThisEvent);
      switch (ThisEvent.EventType) {
      case ES_ENTRY: StopMotors(); break;
      case DARK_TO_LIGHT: nextState = InLight; makeTransition = TRUE; ThisEvent.
             ↪ EventType = ES_NO_EVENT; break;
      }
      break;

   case Jig:
      ThisEvent = RunJigSubHSM(ThisEvent);
      switch (ThisEvent.EventType) {
      case JIG_FINISHED: nextState = InLight; makeTransition = TRUE; ThisEvent.
             ↪ EventType = ES_NO_EVENT; break;
      case LIGHT_TO_DARK: nextState = InDark; makeTransition = TRUE; ThisEvent.
             ↪ EventType = ES_NO_EVENT; break;
      }
      break;
   }

   if (makeTransition == TRUE) {
      RunTemplateHSM(EXIT_EVENT); CurrentState = nextState; RunTemplateHSM(
             ↪ ENTRY_EVENT);
   }

   ES_Tail(); return ThisEvent;
}
```

Fig. 5. This FSM, representing the primary level in a hierarchical state machine (HSM), controls a wheeled robot modeled after a cockroach (shown in figure 4). It exhibits behaviors like moving in darkness and freezing in light, with an added periodic 'jig dance'.

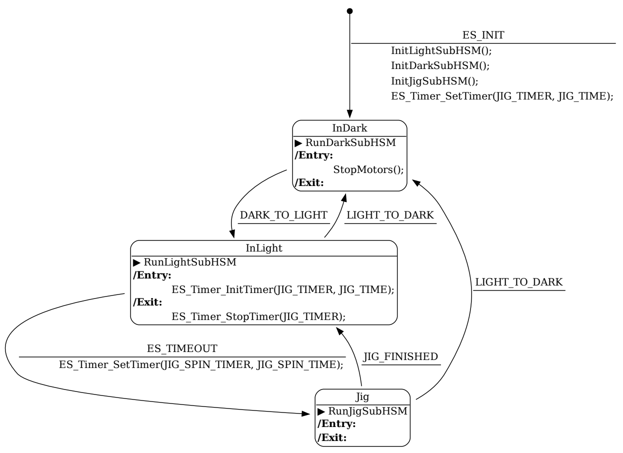


Fig. 6. State diagram generated from FSM code in figure 5

   – `[@CurrentStateTest]` to filter ancestors with the `CurrentStateTest` attribute.
   – `[1]` to pick the first element from this filtered set.
- `/@CurrentStateTest`: Retrieves the `CurrentStateTest` attribute's value from the selected ancestor.

To walk the AST and fetch information as above is impractical with regular expressions.

B. Annotation Pipeline

Our second prototype attempted to directly convert ASTs into state diagrams. This was acceptable for simple diagrams however it soon proved overly complex and unmanageable, when adding features like event parameters, transition logic, and guards.

To address this, we developed a third prototype featuring an annotation pipeline. This pipeline breaks down the diagram generation process into distinct steps, each handling a specific type of annotation. This modular approach allows for easier debugging and verification of each step. After the annotations are complete, the AST is ready for a straightforward transformation into a state diagram using a single XSLT step. This final step uses the pre-annotated AST and three loops to fill out a predefined diagram description template as shown in figure 3.

At present, our annotation pipeline comprises fifteen XSLT steps (lines 41-55 in figure 1). Additional steps can be incorporated as needed for new diagram features or to handle more AST patterns. An example of one such early annotation step is illustrated in Figure 9. This step determines the diagram label associated with the current state and adds it as an attribute named `CurrentStateTest`.

Figure 9 includes an XPATH pattern that targets `block_items` AST elements based on specific criteria:

- `@class='Case' or @class='Default'`: This selects `block_items` nodes either with a `class` attribute value of `Case` or `Default`.
- `../../../block_items[@class='Switch'] /cond[@class='ID' and @name='CurrentState']`: The process here is:
   – `../../..`: Ascends three levels in the AST from the current `block_items` node.
   – `/block_items[@class='Switch']`: Selects `block_items` nodes that are children of the node reached and have a `class` attribute of `Switch`.
   – `/cond[@class='ID' and @name='CurrentState']`: Then selects `cond` nodes that have a `class` attribute of `ID` and a `name` attribute of `CurrentState`.
- `and not(@CurrentStateTest)`: Excludes nodes already tagged with a `CurrentStateTest` attribute.

This XPATH pattern selects `block_items` nodes classified as either `Case` or `Default`, but only if they are hierarchically related to `block_items` nodes of class `Switch` with a child `cond` node meeting specific criteria (`class='ID'` and `name='CurrentState'`). These nodes must not already have a `CurrentStateTest` attribute. This ensures no overwriting if `CurrentStateTest` is already computed in another step.

The outcome of this XSLT is tagging all branches of switch statements conditional on the variable `CurrentState` with a `CurrentStateTest` attribute. This attribute holds the XPATH value referencing the label of the current switch branch: `./expr[@class='ID']/@name`

The `CurrentStateTest` attribute's purpose is to track the current state label, allowing subsequent pipeline logic to

```c
ES_Event RunHSM_Top_Orienting(ES_Event ThisEvent) {

uint8_t makeTransition=FALSE; HSM_Top_OrientingState_t nextState; ES_Event postEvent
       ↪ ; ES_Tattle(); uint8_t nextFromTrack; uint8_t nextFromTape;

switch (CurrentState) {
case InitPSubState:
   if (ThisEvent.EventType==ES_INIT) {
      wallHit=FALSE; barrierCount=0; barrierTrack=BARRIER_NULL; barrierTape=
            ↪ BARRIER_NULL; fieldSide=FIELD_UNKNOWN; centerTimerTime=
            ↪ TIMER_TICKS_CENTER_BUMP;
      turningTimerTime=DCMOTOR_TIME_TURN_90DEG; nextState=Find; makeTransition=TRUE;
            ↪  ThisEvent.EventType=ES_NO_EVENT;
   }
   break;

case Find:
   switch (ThisEvent.EventType) {
   case ES_ENTRY: DCMotor_Drive(DCMOTOR_DRIVE_SPEED, FORWARDS); break;
   case ES_EXIT: DCMotor_Stop(); break;
   case BUMPER_PRESSED: wallHit=TRUE; centerTimerTime=TIMER_TICKS_CENTER_BUMP;
         ↪ nextState=Align; makeTransition=TRUE; ThisEvent.EventType=ES_NO_EVENT;
         ↪  break;
   case TRACK_ENTERED: barrierTrack=barrierCount; centerTimerTime=
         ↪ TIMER_TICKS_CENTER_TRACK; nextState=Center; makeTransition=TRUE;
         ↪ ThisEvent.EventType=ES_NO_EVENT; break;
   case TAPE_ENTERED: barrierTape=barrierCount; centerTimerTime=
         ↪ TIMER_TICKS_CENTER_TAPE; nextState=Center; makeTransition=TRUE;
         ↪ ThisEvent.EventType=ES_NO_EVENT; break;
   }
   break;

case Align:
   switch (ThisEvent.EventType) {
   case ES_ENTRY: ES_Timer_InitTimer(TIMER_TOP_ORIENTING, TIMER_TICKS_ALIGN);
         ↪ DCMotor_Turn(DCMOTOR_DRIVE_SPEED, FORWARDS, LEFT); break;
   case ES_EXIT: DCMotor_Stop(); break;
   case ES_TIMEOUT:
      if (ThisEvent.EventParam==TIMER_TOP_ORIENTING) {
         nextState=Center; makeTransition=TRUE; ThisEvent.EventType=ES_NO_EVENT;
      }
      break;
   }
   break;

case Center:
   switch (ThisEvent.EventType) {
   case ES_ENTRY:
      ES_Timer_InitTimer(TIMER_TOP_ORIENTING, centerTimerTime); DCMotor_Drive(
            ↪ DCMOTOR_DRIVE_SPEED, BACKWARDS);
      if (wallHit==TRUE) barrierCount++; break;
   case ES_EXIT: DCMotor_Stop(); break;
   case ES_TIMEOUT:
      if (ThisEvent.EventParam==TIMER_TOP_ORIENTING) {
         if (barrierCount < BARRIER_COUNT) {
            nextState=Rotate;
         } else {
            if (barrierTrack==(BARRIER_COUNT - 1)) {
               nextFromTrack=0;
            } else if (barrierTrack==BARRIER_NULL) {
               nextFromTrack=BARRIER_NULL;
            } else { nextFromTrack=barrierTrack + 1; }
            if (barrierTape==(BARRIER_COUNT - 1)) {
               nextFromTape=0;
            } else if (barrierTape==BARRIER_NULL) {
               nextFromTape=BARRIER_NULL;
            } else { nextFromTape=barrierTape + 1; }
            if (barrierTrack==BARRIER_NULL) {
               fieldSide=FIELD_UNKNOWN;
            } else if (barrierTape==BARRIER_NULL) {
               fieldSide=FIELD_UNKNOWABLE;
            } else if (nextFromTrack==barrierTape) {
               fieldSide=FIELD_LEFT;
            } else if (nextFromTape==barrierTrack) {
               fieldSide=FIELD_RIGHT;
            }
            if ((fieldSide==FIELD_LEFT) || (fieldSide==FIELD_RIGHT) || (fieldSide==
                  ↪ FIELD_UNKNOWABLE)) {
               nextState=Turning_Beacon;
               turningTimerTime=DCMOTOR_TIME_TURN_90DEG * (barrierTrack + 1);
            } else {
               nextState=Turning_OtherSide; turningTimerTime=DCMOTOR_TIME_TURN_90DEG
                     ↪  * (barrierTape + 1); wallHit=FALSE; barrierCount=0;
                     ↪ barrierTrack=BARRIER_NULL; barrierTape=BARRIER_NULL;
                     ↪ fieldSide=FIELD_UNKNOWN; centerTimerTime=
                     ↪ TIMER_TICKS_CENTER_BUMP; turningTimerTime=
                     ↪ DCMOTOR_TIME_TURN_90DEG;
            }
         }
         makeTransition=TRUE; ThisEvent.EventType=ES_NO_EVENT;
      }
      break;
   }
   break;

case Rotate:
   switch (ThisEvent.EventType) {
   case ES_ENTRY: ES_Timer_InitTimer(TIMER_TOP_ORIENTING, TIMER_TICKS_ROTATE);
         ↪ DCMotor_TankTurn(DCMOTOR_TURN_SPEED, RIGHT); break;
   case ES_EXIT: DCMotor_Stop(); break;
   case ES_TIMEOUT:
      if (ThisEvent.EventParam==TIMER_TOP_ORIENTING) {
         nextState=Find; makeTransition=TRUE; ThisEvent.EventType=ES_NO_EVENT;
      }
      break;
   }
   break;

case Turning_Beacon:
   switch (ThisEvent.EventType) {
   case ES_ENTRY: ES_Timer_InitTimer(TIMER_TOP_ORIENTING, turningTimerTime);
         ↪ DCMotor_TankTurn(DCMOTOR_TURN_SPEED, RIGHT); break;
   case ES_EXIT: DCMotor_Stop(); break;
   case ES_TIMEOUT:
      if (ThisEvent.EventParam==TIMER_TOP_ORIENTING) {
         postEvent.EventType=ORIENTED; postEvent.EventParam=fieldSide; PostHSM_Top(
               ↪ postEvent); nextState=InitPSubState; makeTransition=TRUE;
               ↪ ThisEvent.EventType=ES_NO_EVENT;
      }
      break;
   }
   break;

case Turning_OtherSide:
   switch (ThisEvent.EventType) {
   case ES_ENTRY: ES_Timer_InitTimer(TIMER_TOP_ORIENTING, turningTimerTime);
         ↪ DCMotor_TankTurn(DCMOTOR_TURN_SPEED, RIGHT); break;
   case ES_EXIT: DCMotor_Stop(); break;
   case ES_TIMEOUT:
      if (ThisEvent.EventParam==TIMER_TOP_ORIENTING) {
         nextState=Driving_OtherSide; makeTransition=TRUE; ThisEvent.EventType=
               ↪ ES_NO_EVENT;
      }
      break;
   }
   break;

case Driving_OtherSide:
   switch (ThisEvent.EventType) {
   case ES_ENTRY: DCMotor_Drive(DCMOTOR_DRIVE_SPEED, FORWARDS); break;
   case ES_EXIT: DCMotor_Stop(); break;
   case TAPE_EXITED: nextState=Find; makeTransition=TRUE; ThisEvent.EventType=
         ↪ ES_NO_EVENT; break;
   }
   break;
}

if (makeTransition==TRUE) {
   RunHSM_Top_Orienting(EXIT_EVENT); CurrentState=nextState; RunHSM_Top_Orienting(
         ↪ ENTRY_EVENT);
}

ES_Tail(); return ThisEvent;
}
```

Fig. 7. A lower-level FSM in a multi-tiered HSM for a UCSC competition [8] robot.

reference this label without recalculating. If the current state is determined differently, like through if-elseif-else constructs instead of a switch statement, another template can handle that scenario. Hence, the downstream logic needing the current state label does not depend on the specific logic computing the `CurrentStateTest` attribute.

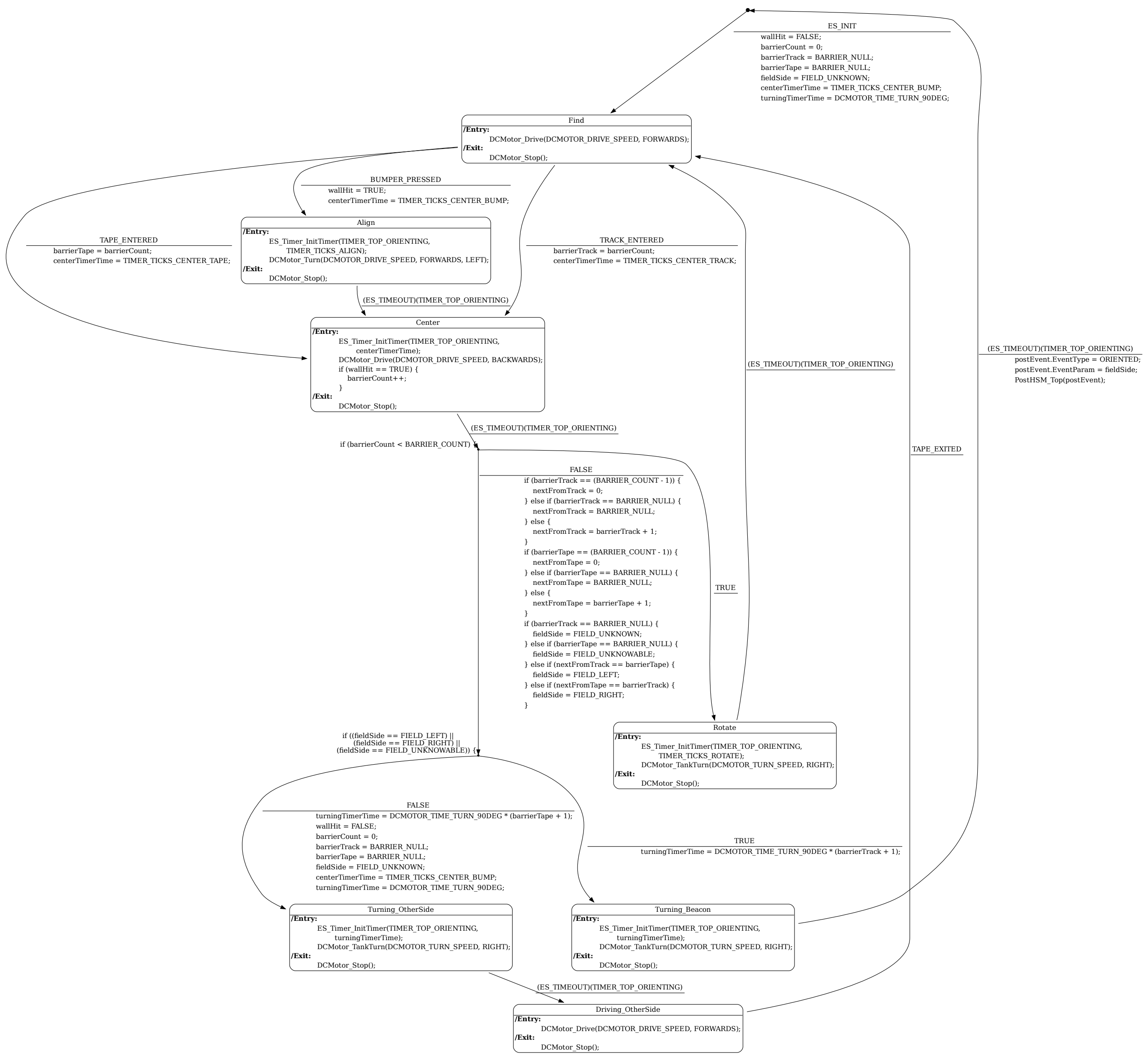
C. Limitations and Challenges

Some limitations and challenges associated with our tool include:

a) CPP Includes: In Section II-A4, we discuss the application of CPP to generate a C code stream independent of other files. The success of CPP hinges on accessing all necessary project and library include files. Although our tool includes standard files, version mismatches with users' code may necessitate manual updates to the CPP launch command. To facilitate this, our tool outputs each CPP command, allowing users to modify the CPP launch command as needed if the default setting fails.

b) AST Understanding: The AST's complexity compared to the original source code is evident in Figure 11, which depicts the AST for the first branch of a CurrentState switch statement from Figure 10. The AST's verbosity and size—often expanding a few hundred lines of code into thousands—pose significant navigational challenges.

c) Annotation Development: Understanding the effects of annotation steps requires examining the AST before and after each step. Figure 12 demonstrates the use of `tee` commands for capturing AST states around the



```
ES_INIT
    wallHit = FALSE;
    barrierCount = 0;
    barrierTrack = BARRIER_NULL;
    barrierTape = BARRIER_NULL;
    fieldSide = FIELD_UNKNOWN;
    centerTimerTime = TIMER_TICKS_CENTER_BUMP;
    turningTimerTime = DCMOTOR_TIME_TURN_90DEG;
```

```
Find
/Entry:
        DCMotor_Drive(DCMOTOR_DRIVE_SPEED, FORWARDS);
/Exit:
        DCMotor_Stop();
```

```
BUMPER_PRESSED
    wallHit = TRUE;
    centerTimerTime = TIMER_TICKS_CENTER_BUMP;
```

```
Align
/Entry:
        ES_Timer_InitTimer(TIMER_TOP_ORIENTING,
             TIMER_TICKS_ALIGN);
        DCMotor_Turn(DCMOTOR_DRIVE_SPEED, FORWARDS, LEFT);
/Exit:
        DCMotor_Stop();
```

```
TAPE_ENTERED
    barrierTape = barrierCount;
    centerTimerTime = TIMER_TICKS_CENTER_TAPE;
```

```
TRACK_ENTERED
    barrierTrack = barrierCount;
    centerTimerTime = TIMER_TICKS_CENTER_TRACK;
```

```
Center
/Entry:
        ES_Timer_InitTimer(TIMER_TOP_ORIENTING,
             centerTimerTime);
        DCMotor_Drive(DCMOTOR_DRIVE_SPEED, BACKWARDS);
        if (wallHit == TRUE) {
           barrierCount++;
        }
/Exit:
        DCMotor_Stop();
```

```
(ES_TIMEOUT)(TIMER_TOP_ORIENTING)
    postEvent.EventType = ORIENTED;
    postEvent.EventParam = fieldSide;
    PostHSM_Top(postEvent);
```

```
if (barrierCount < BARRIER_COUNT) {
```

```
FALSE
    if (barrierTrack == (BARRIER_COUNT - 1)) {
       nextFromTrack = 0;
    } else if (barrierTrack == BARRIER_NULL) {
       nextFromTrack = BARRIER_NULL;
    } else {
       nextFromTrack = barrierTrack + 1;
    }
    if (barrierTape == (BARRIER_COUNT - 1)) {
       nextFromTape = 0;
    } else if (barrierTape == BARRIER_NULL) {
       nextFromTape = BARRIER_NULL;
    } else {
       nextFromTape = barrierTape + 1;
    }
    if (barrierTrack == BARRIER_NULL) {
       fieldSide = FIELD_UNKNOWN;
    } else if (barrierTape == BARRIER_NULL) {
       fieldSide = FIELD_UNKNOWABLE;
    } else if (nextFromTrack == barrierTape) {
       fieldSide = FIELD_LEFT;
    } else if (nextFromTape == barrierTrack) {
       fieldSide = FIELD_RIGHT;
    }
```

```
Rotate
/Entry:
        ES_Timer_InitTimer(TIMER_TOP_ORIENTING,
             TIMER_TICKS_ROTATE);
        DCMotor_TankTurn(DCMOTOR_TURN_SPEED, RIGHT);
/Exit:
        DCMotor_Stop();
```

```
if ((fieldSide == FIELD_LEFT) ||
    (fieldSide == FIELD_RIGHT) ||
    (fieldSide == FIELD_UNKNOWABLE)) {
```

```
FALSE
    turningTimerTime = DCMOTOR_TIME_TURN_90DEG * (barrierTape + 1);
    wallHit = FALSE;
    barrierCount = 0;
    barrierTrack = BARRIER_NULL;
    barrierTape = BARRIER_NULL;
    fieldSide = FIELD_UNKNOWN;
    centerTimerTime = TIMER_TICKS_CENTER_BUMP;
    turningTimerTime = DCMOTOR_TIME_TURN_90DEG;
```

```
TRUE
    turningTimerTime = DCMOTOR_TIME_TURN_90DEG * (barrierTrack + 1);
```

```
Turning_OtherSide
/Entry:
        ES_Timer_InitTimer(TIMER_TOP_ORIENTING,
             turningTimerTime);
        DCMotor_TankTurn(DCMOTOR_TURN_SPEED, RIGHT);
/Exit:
        DCMotor_Stop();
```

```
Turning_Beacon
/Entry:
        ES_Timer_InitTimer(TIMER_TOP_ORIENTING,
             turningTimerTime);
        DCMotor_TankTurn(DCMOTOR_TURN_SPEED, RIGHT);
/Exit:
        DCMotor_Stop();
```

```
Driving_OtherSide
/Entry:
        DCMotor_Drive(DCMOTOR_DRIVE_SPEED, FORWARDS);
/Exit:
        DCMotor_Stop();
```

Fig. 8. State diagram generated from FSM code in figure 7

`s00400_add_CascadeElements.xml` annotation step. Differences can be highlighted using `diff -u before.xml after.xml` or an IDE's equivalent function.

D. Features Supported

1) Automatic Labeling:

- **Current, Next State, and Transition Event Labels:** Automatically labels states and transitions, enhancing the clarity of state progressions and events triggering these transitions.
- **Initial State Elements:** Clearly marks the starting state of each FSM, providing an immediate understanding of the FSM's entry point.
- **Switch Cascade Labels:** Simplifies diagrams by merging labels when multiple conditions in a switch statement lead to the same next state, aiding in reducing diagram complexity.
- **Event Parameter Labels:** Adds context to events by displaying associated parameters, such as timer IDs in timeout events, facilitating a deeper understanding of event-specific behaviors.
- **Entry/Exit Logic Labels:** Marks repetitive logic executed upon entering or exiting states, crucial for understanding state-dependent behaviors.
- **Transition Logic Labels:** Indicates logic executed during transitions, essential for tracking changes in behavior in response to events.

```
    <xsl:template match="
      block_items[
          (@class='Case'
              or @class='Default')
          and (
              ../../..
              /block_items[@class='Switch']
              /cond[@class='ID' and @name='CurrentState']
          )
          and not(@CurrentStateTest)
      ]">
      <xsl:copy>
          <xsl:apply-templates select="@*"/>
          <xsl:attribute name="CurrentStateTest">
              <xsl:value-of select="./expr[@class='ID']/@name"/>
          </xsl:attribute>
          <xsl:apply-templates select="node()"/>
      </xsl:copy>
  </xsl:template>
```

Fig. 9. Example annotation that adds `CurrentStateTest` attribute

```
switch (CurrentState) {
   case InitPSubState:
      if (ThisEvent.EventType == ES_INIT)
      {
         ES_Timer_StopTimer(TIMER_TOP_RELOADING);
         trackCrossings = 0;
         nextState = Turning;
         makeTransition = TRUE;
         ThisEvent.EventType = ES_NO_EVENT;
      }
      break;
      ...
```

Fig. 10. Sample code snip showing just the first case in a switch statement

- **Macro Expansion Suppression:** Represents constants (e.g., `TURN_RIGHT_ENUM` instead of `0x45`) with their defined labels, improving readability and comprehension.

2) Advanced Features:

- **Transition Guards:** Displays conditions that control state transitions, instrumental for visualizing decision-making within the FSM.
- **Hierarchical State Machines:** Supports nested state machines, providing abstraction and modularity, and encapsulating complex logic within states.

3) Ease of Use:

- **Automatic Discovery of FSMs:** Identifies and processes FSMs in `*.c` files automatically, streamlining the diagram generation process for entire projects. No need to generate diagrams one at a time.
- **Isolated Installation and Runtime:** Uses Linux containers for a single-command, isolated setup and operation, ensuring compatibility across different systems including WSL2 for Windows and Docker Desktop for MacOS.

E. Future Work

To foster collaborative development and wider adoption, the complete tool is available under an Open Source license (AGPLv3) and can be accessed free of charge at: https://github.com/jlesner/smv2.

Possible future work includes:

- **More Code Patterns:** As UCSC students use our tool, supporting a wider range of FSM code patterns is our primary focus.
- **More Inputs and Outputs:** Extending support to FSMs in Java, Python, JavaScript, etc. Generation of diagrams not just using GraphViz but also using Mermaid.js, PlantUML, etc. Introduction of new diagram types such as Harel Statecharts and Activity Diagrams.

```
<block_items class="Switch" line="602">
   <cond class="ID" line="602" name="CurrentState"/>
   <stmt class="Compound" line="602">
      <block_items class="Case" line="603">
         <expr class="ID" line="603" name="InitPSubState"/>
         <stmts class="If" line="604">
            <cond class="BinaryOp" line="604" op="==">
               <left class="StructRef" line="604" type=".">
                  <name class="ID" line="604" name="ThisEvent"/>
                  <field class="ID" line="604" name="EventType"/>
               </left>
               <right class="ID" line="604" name="ES_INIT"/>
            </cond>
            <iftrue class="Compound" line="605">
               <block_items class="FuncCall" line="606">
                  <name class="ID" line="606" name="ES_Timer_StopTimer"/>
                  <args class="ExprList" line="606">
                     <exprs class="ID" line="606" name="TIMER_TOP_RELOADING"/>
                  </args>
               </block_items>
               <block_items class="Assignment" line="607" op="=">
                  <lvalue class="ID" line="607" name="trackCrossings"/>
                  <rvalue class="Constant" line="607" type="int" value="0"/>
               </block_items>
               <block_items class="Assignment" line="608" op="=">
                  <lvalue class="ID" line="608" name="nextState"/>
                  <rvalue class="ID" line="608" name="Turning"/>
               </block_items>
               <block_items class="Assignment" line="609" op="=">
                  <lvalue class="ID" line="609" name="makeTransition"/>
                  <rvalue class="ID" line="609" name="TRUE"/>
               </block_items>
               <block_items class="Assignment" line="610" op="=">
                  <lvalue class="StructRef" line="610" type=".">
                     <name class="ID" line="610" name="ThisEvent"/>
                     <field class="ID" line="610" name="EventType"/>
                  </lvalue>
                  <rvalue class="ID" line="610" name="ES_NO_EVENT"/>
               </block_items>
            </iftrue>
         </stmts>
         <stmts class="Break" line="612"/>
      </block_items>
      ...
```

Fig. 11. Sample AST snip matching code in figure 10

```
 1 find "$src_path" -name '*.c.cp5' \
 2 | while read f ; do
 3     echo "visualizing '$f'"
 4     (
 5         sx=saxonb-xslt
 6         cat "$f" \
 7         | tr -d '\r' \
 8         | ( egrep -avi '^[[:blank:]]*$|^#|va_list|__attribute__' || true ) \
 9         | perl -pe's{__extension__}{ }g; s{__}{}g; ' \
10         | python3 c_ast_xml.py \
11         | tee "${f}.xml" \
12         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00005_identity.xml \
13         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00100_declutter_attributes.xml \
14         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00200_add_bLine_eLine.xml \
15         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_CurrentStateTest.xml \
16         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_EventParamTest.xml \
17         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_EventTypeTest.xml \
18         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00300_add_NextStateLabel.xml \
19         | tee before.xml  \
20         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00400_add_CascadeElements.xml \
21         | tee after.xml  \
22         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00500_add_CascadeLabel.xml \
23         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00550_add_EventLabel.xml \
24         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00560_add_Guard_Element.xml \
25         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00570_add_Guard_Attributes.xml \
26         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00600_add_onEntry_onExit.xml \
27         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00600_add_onTransition2.xml \
28         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00620_drop_unwanted_code.xml \
29         | $sx -s:/dev/stdin -o:/dev/stdout -xsl:s00800_gv_digraph4.xml \
30         | perl -pe 's/ && / &amp;&amp; /g;
31                     s/ < / &lt; /g;
32                     s/ > / &gt; /g;
33                     s/ <= / &lt;= /g;
34                     s/ >= / &gt;= /g;
35                 ' \
36         > "${f}.gv"
37
38         dot -Tpng "${f}.gv" -o "${f}.png"
39         dot -Tpdf "${f}.gv" -o "${f}.pdf"
40         dot -Tsvg "${f}.gv" -o "${f}.svg"
41
42     ) 2>&1
43 done
```

Fig. 12. Lines 19 and 21 show how annotation AST captures are done

- **More Intelligence:** Analysis to identify FSM programming errors, like states with incomplete transitions or potential deadlocks, where the FSM could freeze without any viable transitions.

V. Conclusion

We have described a new tool for automatically creating visualizations of Finite State Machines (FSMs), which is particularly useful in software engineering and robotics. The tool simplifies the creation of state diagrams, which is usually complex and error-prone, especially for intricate FSMs. It uses naming conventions, Abstract Syntax Tree (AST) patterns, and XSLT transformations to generate accurate FSM visuals from the source code, accommodating various coding patterns. This not only saves time and reduces errors but also helps in understanding FSM structures, proving especially beneficial in educational settings like UCSC's mechatronics courses [8].

The tool's ability to handle different FSM code patterns, including hierarchical state machines and transition guards, shows its versatility. It is being used in education to help students learn and implement FSMs in robotics. Although it currently works in a specific programming environment and with certain naming conventions, there's potential for expanding its capabilities to more programming languages, diagram types, and FSM verification diagnostics.

In summary, this tool marks a significant advancement in automating state diagram generation, improving the design and debugging of FSMs in various applications, especially in education.

Acknowledgments

This project was initiated and funded by CAHSI Undergraduates Program and supported by National Science Foundation Grants #2034030 and #1834620. Christopher Lesner helped with UNIX commands. Bailen Lawson supplied the FSM code samples used for AST pipeline development and testing. ChatGPT assisted with LaTeX and polished our paragraphs. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of CAHSI or the National Science Foundation.

Appendix A
Setup and Usage Guide

The State Machine Visualizer (SMV) is a tool for visualizing the structure and behavior of state machines in your code. Follow these steps to set up and use the tool.

Step-by-Step Instructions

STEP 1: Download the Script: First, download the `smv.bash` script using the following command:

```
wget https://raw.githubusercontent.com/jlesner/smv2/main/smv.bash
```

STEP 2: Inspect the Script:

- **Inspect Changes:** Review the `smv.bash` script to understand the changes it will make. It installs necessary tools like git, curl, and podman if they are not already present on your system.
- **Password Prompt:** The script uses `sudo apt-get`, which might prompt you for your password to install missing tools.
- **First-Time Setup:** On its initial run, `smv.bash` will download the latest version of the State Machine Visualizer and install required dependencies.
- **System Requirements:** The script is designed for Linux systems with the `apt` package manager, such as Ubuntu. Windows users can use Ubuntu/WSL2, and MacOS users might need to run Ubuntu in a VM.
- **Containerization:** To create a suitable environment, `smv.bash` builds a Linux container, installing additional dependencies (Python, Java, etc.) and executes the SMV code within this container. Note that this container requires approximately 900MB of space.
- **Cleanup:** At the end of the script, instructions are provided to remove the installations made by `smv.bash`. These instructions are for when you are done using SMV and want to remove it. Leaving things installed allows `smv.bash` to run faster.

STEP 3: Run the Script: To run the State Machine Visualizer, use the following command, replacing `${path_to_code}` with the path to your state machine files:

```
bash smv.bash ${path_to_code}
```

STEP 4: View the Results: After running the script, you can find the files it generated using this command:

```
find ${path_to_code} -name '*.cp5*'
```

References

[1] Graphviz. [Online]. Available: https://graphviz.org/

[2] Mermaid. [Online]. Available: https://mermaid.js.org/

[3] Plantuml. [Online]. Available: https://plantuml.com/

[4] D. van Heesch. Doxygen. [Online]. Available: https://www.doxygen.nl/

[5] O. M. Group. Unified modeling language specification. [Online]. Available: https://www.omg.org/spec/UML/

[6] S. Systems. Enterprise architect. [Online]. Available: https://sparxsystems.com/products/ea/index.html

[7] W. W. W. Consortium. Extensible stylesheet language transformations (xslt) version 3.0. [Online]. Available: https://www.w3.org/TR/xslt-30/

[8] G. H. Elkaim, "A hole in one: A project-based class on mechatronics," in *2011 IEEE International Conference on Microelectronic Systems Education*, 2011, pp. 35–38.

[9] Mplab® x integrated development environment (ide). [Online]. Available: https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide

[10] Mplab® xc compilers. [Online]. Available: https://www.microchip.com/en-us/tools-resources/develop/mplab-xc-compilers

[11] E. Bendersky and contributors. pycparser. [Online]. Available: https://github.com/eliben/pycparser

[12] B. Stroustrup, *The C++ Programming Language*, 4th ed. Addison-Wesley Professional, 2013, comprehensive guide on C++ by its original creator.

[13] Perl regular expressions. [Online]. Available: https://perldoc.perl.org/perlre

[14] W. W. W. Consortium. Xml path language (xpath) specification. [Online]. Available: https://www.w3.org/TR/xpath/