AUTOMATED VISUALIZATION FOR FLAT AND HIERARCHICAL STATE MACHINES

Autonomous Systems Lab @ UCSC
Jasmine Lesner | Gabriel Elkaim
{jlesner,elkaim}@ucsc.edu

INTRODUCTION

Our research aims to **automate state machine visualization directly from source code**. Automation of code visualization is valuable because it enables up-to-date diagrams with attendant verification, documentation, and understanding.

Existing visualization tools require extensive user input and manipulation for satisfactory results; no tool found runs fully automated. In contrast, our work uses **variable name conventions** and XSLT matching **Abstract Syntax Tree (AST)** patterns to generate state diagrams directly from source code.

METHOD & TOOLS

Our **prototype** uses the following tools:

- **CPP**: for macro expansion and file inclusion.
- **PycParser**: for Lexical Analysis, Syntax Analysis, and Abstract Syntax Tree (AST) generation from C code.
- **XSLT**: to express ASTs patterns to annotate ASTs represented in XML and describe diagrams.
- **GraphViz**: to create images from our generated diagram descriptions

To **generate a state diagram** from a *.c file:

CPP

Pull in #include files to generate code that has **no dependencies** on other files.

PycParser

Lexical Analysis to break down the C-code into a series of tokens representing basic language elements.

Syntax Analysis to check if the tokens follow the grammatical structure of C and identify syntax errors.

AST Generation which is saved as XML together with line number annotated code.

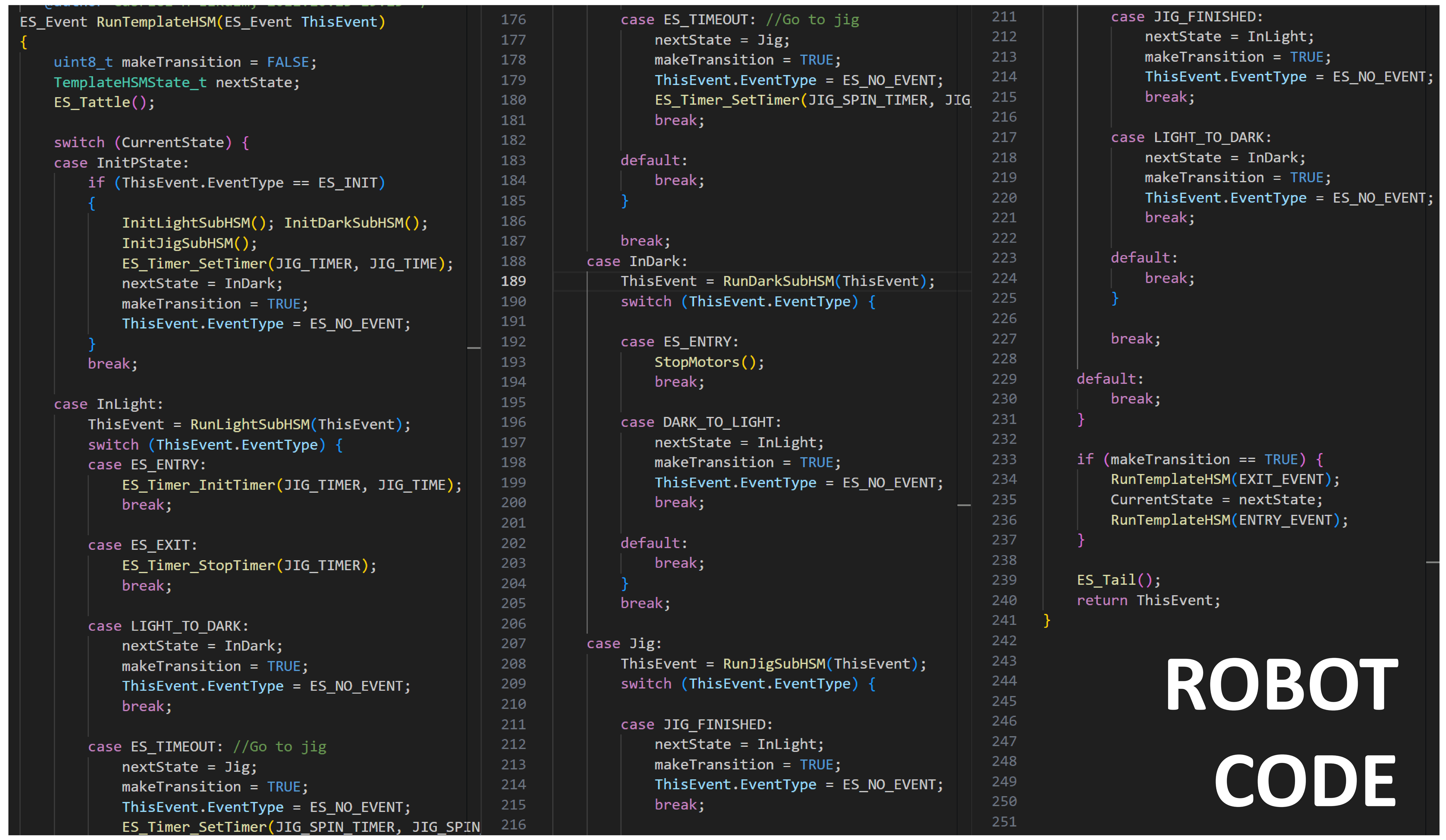
XSLT

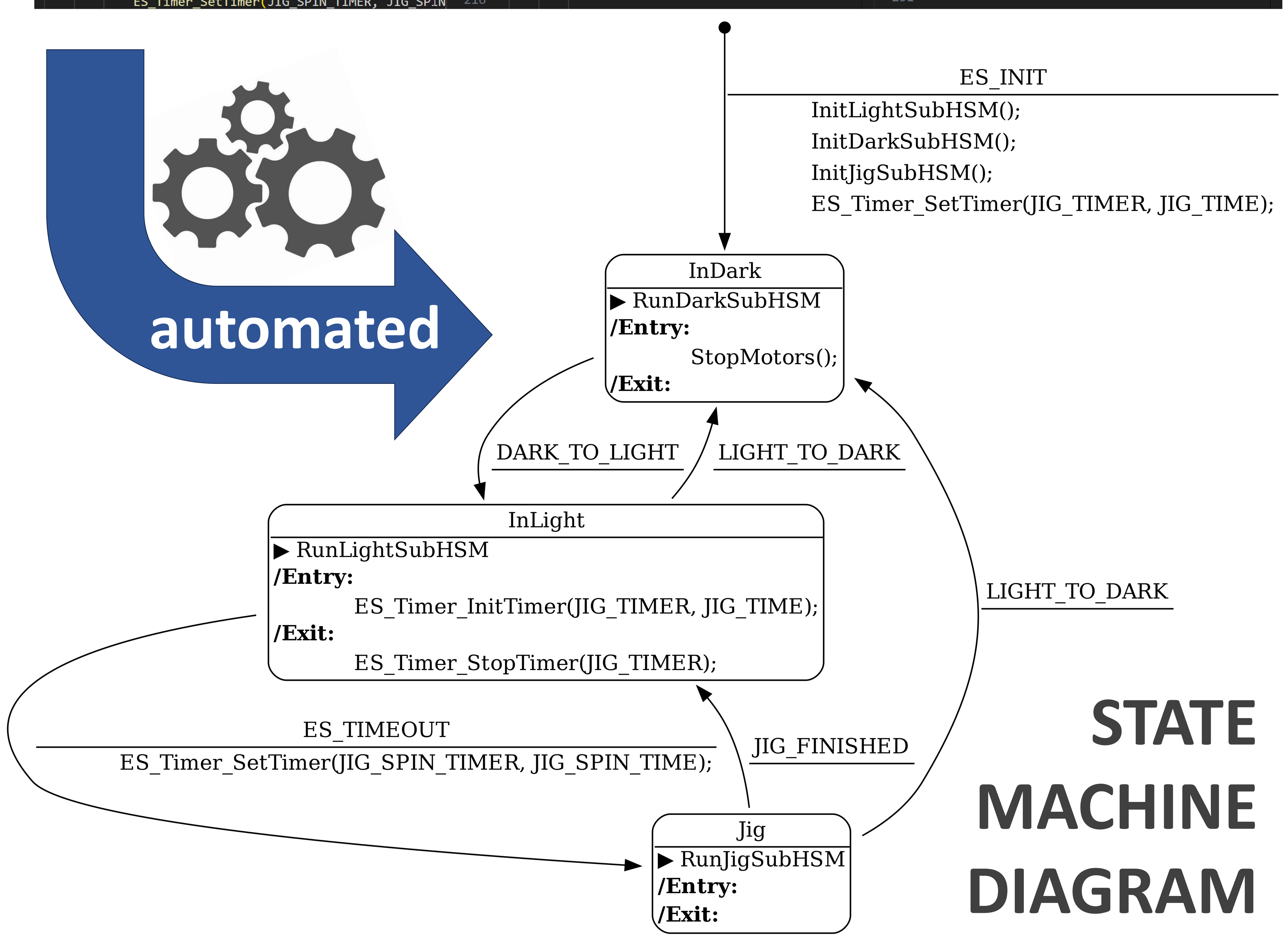
Initial XSLT templates **annotate the AST** to label states, events, transition logic, guards and other features.

Final XSLT template uses AST annotations to generate a *.gv **diagram description**.

GraphViz

Reads generated *.gv diagram description then **draws the diagram** and saves it as a *.png file.





RESULTS

Given the myriad implementations of state machines in C-code, our prototype is implemented on top of **Abstract Syntax Trees (ASTs)** and supports:

- State machines implemented with **if-then-else** statements and **switch-and-default** statements
- **Case Cascades** inside switch statements: when fall through is to the same next state we simplify the diagram by combining transition labels together.
- **Event Parameters** so that the same event with a different parameter is treated as its own transition
- **onEntry and onExit** elements which show logic executed immediately on state entry/exit
- **onTransition** elements which show the logic executed for a specific transition
- **Guard Conditions** which show the logic that decides if a transition should occur.
- **Hierarchical State Machines (HSMs)** which allow new state machine logic to be isolated within a state

CONCLUSION

Our prototype generates state diagrams directly from code using naming conventions for variables and AST code patterns defined using XSLT. Moving forward, UCSC's "ECE118: Introduction to Mechatronics" students will use our prototype so that broader usage will help us identify edge cases that require updates to our XSLT.

ACKNOWLEDGEMENTS

This project was initiated and funded by **CAHSI Undergraduates Program** and is supported by **National Science Foundation** Grants #2034030 and #1834620. **Bailen Lawson** supplied many ECE118 code samples enabling our testing. **One such code sample and its generated diagram are shown.**