CSE 355 Fall 2013: Optional Project
Principles of Model Checking and Program Verification

Arizona State University - G. Fainekos

Introduction

Finite automata can be used to model terminating software programs or discrete event systems. We will refer to such an automaton as *model of the system*. Similarly, the expected behavior of the environment can be captured by an automaton whose "composition" with the model of the system can model the interaction of the environment with the system. In this project, we will assume that we have a model of the system interacting with its environment and we would like to verify whether the execution of the system satisfies a specification given as another automaton (*specification automaton*).

Translating the above in language theoretic terms, the system-environment automaton A accepts a language L(A) which consists of strings that generate paths on the automaton that represent the actual behavior of the system in its environment. Our goal is to find whether this language is a subset of the language L(S) that is accepted by the specification automaton S, i.e., $L(A) \subseteq L(S)$. Moreover, if it is not the case that $L(A) \subseteq L(S)$, then we must provide a string w which demonstrates that w belongs in L(A), but is not in L(S). This implies that w is a *counterexample* for the correctness of the system. In practice, the counterexample can provide feedback to the software engineer on how the system fails.

Details

In order to solve the above problem, you will have to implement the following:

1. Read the specification automaton S and model of the system A from a text file. Note that both automata A and S could be nondeterministic.
2. If automata A and S are nondeterministic, convert them to deterministic automata.
   Hint: See Gallier notes Ch. 2.6 for an improved algorithm.
3. Convert the automaton S into a new automaton S' that accepts the complement of L(S), i.e., $\overline{L(S)}$.
4. Construct a new automaton M which accepts the language $L(M) = L(A) \cap \overline{L(S)}$.

   Q: Can you do the construction on the fly so you don't have to create the whole state space?
5. Find a string w that belongs in L(M). This is done by finding a path from the initial state of M to a final state in M and returning a string that will generate this path.

Note that if $w \in L(M)$, then $w \in L(A)$ and $w \in \overline{L(S)}$. Thus, $w \in L(A)$ and $w \notin L(S)$, which is what we are looking for.

For full credit, you will have to use your implementation to verify the properties of a simplified model of the software of an infusion pump.

6. Model a simple infusion pump system and verify its properties.

What must be submitted

You can use any programming language as long as you can run the executable or interpret the code on a machine in the labs of the department or your own laptop. Note that C or C++ is preferred though.

i) You **must** demonstrate that your implementation works. Namely, I will provide you with an input file containing the alphabet and the automata and you should print out a counterexample or that the system satisfies the specification.
ii) A short paper describing each algorithm, theorem and/or result you used in your implementation. **This is an important part of the project (40% of the points that you will receive).** I expect to see a detailed professional document with citations, mathematical formulations and how each mathematical result/object has been converted into code, algorithm or data structure. Also, you must discuss the complexity of each part of your implementation. The latter is material that we will discuss in Chapter 7 of the book. The report must include the model of the infusion pump and the finite automata that correspond to the specifications.
iii) Of course, your code.

Rules

The programming projects are an opportunity for you to apply what you have learned in the course. These are aspects of the theory that I find to be of particular importance. As a general rule, the programs will be relatively short, and are not intended to be robust, product-quality programs, but demonstrations of a solution to a very specific problem. I'm looking for programs which demonstrate a clear understanding of the underlying theoretical concepts.

This project is **optional**.

Note that the grade is on the **functionality** of the program. In other words, **DOES** the program work? By definition, programs which don't work are useless. So, no partial credit will be given for programs that do not work. *Think that you are asked by your employer/client to write a program. You would not be paid for a partial implementation that does not work.*

The total project is worth 100pt which corresponds to the +6% increase of your grade. The project is due on the final exam date. Partial credit will be given for the following combinations of **working** implementation:

- **[20pt]** 1, 5: Meaning, given one automaton, return a string that belongs in its language.
    - Milestone 1: *This part is due by Midterm 2*
- **[40pt]** 1, 3, 5: Meaning, given a deterministic automaton (DFA), return a string that belongs in the complement of its language.
    - Milestone 2: *This part is due by 2013/12/03*
    - *This the minimum milestone required in order to waive the automatic failure requirement in the final exam.*
- **[60pt]** 1, 3, 4, 5: Meaning your implementation only works on DFA.
- **[80pt]** 1-5: Meaning everything besides the modeling and verification of the infusion pump.

- **[100pt]** 1-6: Meaning everything.

Some portion of your grade is on programming style (readability, maintainability, extensibility, and accessibility of the code). Few programs are ever the work of one person, and the most successful programs are often those that are modified and enhanced by others, both in industry and academia. Code which is unreadable or not understandable is ignored; write code which others would enjoy reading and enhancing (and I or the TA could grade).

Your code must be accompanied by a report. A template will be provided. The report should contain the following sections.

1. Introduction
    a. What is model checking?
    b. Where is it used? I.e., provide an application domain
2. Finite Automata
    a. What is the formal definition of an automaton?
    b. What data structure did you use to represent the automaton?
3. Operations on Automata
    a. What is the algorithm that you used to compute the complement and intersection?
    b. What is the computational complexity of the algorithms?
4. Searching for counterexamples
    a. How do you search an automaton for a string?
    b. What is the computational complexity of the search algorithm?
5. What were the lessons learnt when applying your tool to the infusion pump example?
6. Conclusions
7. References

Infusion Pumps

Infusion pumps are used in order to deliver a medicine to a patient's circulatory system. The drug delivery may be continuous at a constant rate, periodic, or a one-time bolus dosage.

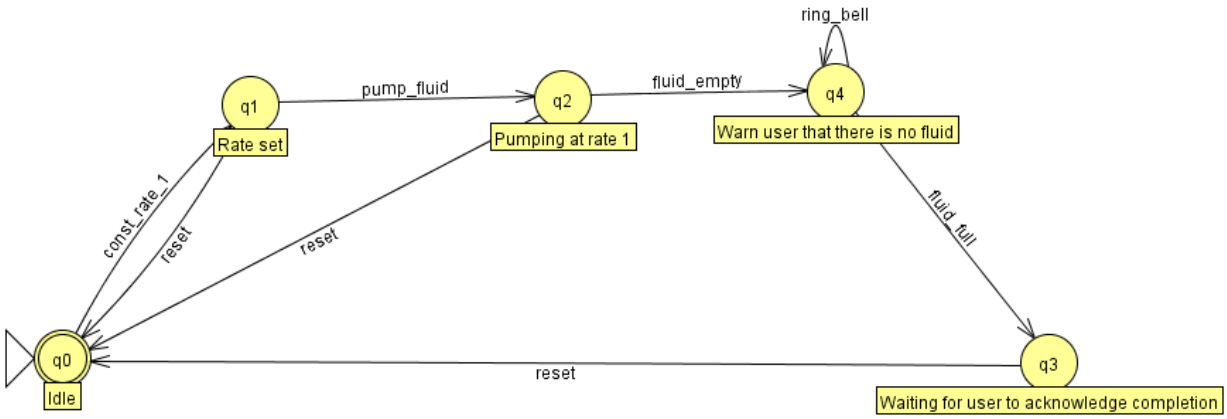
Fig. 1: JMS Infusion pump (OT-701)

These options are controlled through the pump user interface as in Fig.1. Improper use of infusion pumps can lead to improper drug dosage with adverse effects to the patient's health.

For a better understanding of the general context you may read the following two reports posted on Blackboard > Content > Project:

1. Arney et. al.: Formal Methods Based Development of a PCA Infusion Pump Reference Model
2. Weinstock and Goodenough: Towards an Assurance Case Practice for Medical Devices

We will assume a simplified user interface and system functionality. In a realistic modeling framework, the inputs to and the outputs of the system should be separated, but in this case study, the inputs and outputs will simply be treated as symbols to be processed by a finite state machine. The behavior of the system is specified by the finite automaton P in the following figure.



The acceptable behaviors of P, i.e., the language of P, are all the strings that bring the model back to the idle state.

Brief description of the simple model:

1. When the pump is idle, the user can choose only one constant rate for fluid flow.
2. After the flow rate is set, the pump starts operating.
3. If the pump runs out of fluid, then a warning should be issued until the fluid tank has a sufficient level of fluid again.
4. If the pump runs out of fluid, then the fluid must be refilled and the system must be reset by the user.
5. If the tank has a sufficient level of fluid, then the user should be able to reset the pump.

Remark: Formal system models can also be used as a means of communication between engineering teams in order to reduce errors due to misunderstanding of system requirements and specifications when expressed in a natural language setting.

A requirement for the pump is:

- If the pump runs out of fluid, then a warning sound should be issued at least once.

Such a requirement can be formally captured using a regular expression (and, thus, a finite automaton):

$$S_1 = R_1^*(\varepsilon \cup \langle fluid\_empty \rangle \langle ring\_bell \rangle R_2^*)$$

where

$$R_1 = (\langle const\_rate\_1 \rangle \cup \langle pump\_fluid \rangle \cup \langle reset \rangle \cup \langle ring\_bell \rangle \cup \langle fluid\_full \rangle)$$

$$R_2 = (\langle const\_rate\_1 \rangle \cup \langle pump\_fluid \rangle \cup \langle reset \rangle \cup \langle ring\_bell \rangle \cup \langle fluid\_full \rangle \cup \langle fluid\_empty \rangle)$$

The requirement S_1 captures the fact that any string that does not contain <fluid_empty> is acceptable, while if <fluid_empty> occurs in the string, then it must be immediately followed by a <ring_bell> and then any other sequence of symbols is acceptable.

Question to be answered for the project: Does the pump P satisfy the specification S_1? Use your implementation to justify your answer. If the pump model does not satisfy the specification, then modify the model and prove that it is correct.

Question to be answered for the project: Extend your model to allow one more fluid flow rate (<const_rate_2>). Give a regular expression or a finite automaton for the requirement: "After initialization of the pump, if a <const_rate_1> or <const_rate_2> occurs, then, eventually, a <reset> occurs".

Notes:

1. You don't have to explicitly store the long names <const_rate_1> in your implementation. You can replace <const_rate_1> with a letter, for example c.
2. You can convert the regular expressions into automata using JFLAP.

References for additional reading

1. Baier and Katoen, Principles of Model Checking, The MIT Press.
    a. Online access through ASU Library: http://site.ebrary.com.ezproxy1.lib.asu.edu/lib/asulib/docDetail.action?docID=10223883
    b. Recommended readings:
        i. Chapter 1: System Verification
        ii. Chapter 4: Regular properties (Sections 4.1 and 4.2)
        iii. Appendix
2. Stephan Merz, Model Checking: A Tutorial Overview
    a. Posted on Blackboard
    b. Recommended readings:
        i. Section 1: Introduction
        ii. Section 2: Analysis of a Cryptographic Protocol

Example

The following is an example input file (no ε-transitions need to be considered):

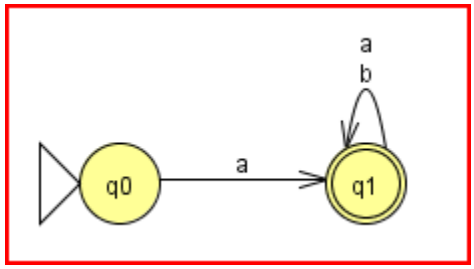
```
% Input alphabet
a
b
% Specification automaton
1 a
2
2 a
2
2 b
2
% Initial state
1
% Final states
2
% System automaton
1 a
1
1 b
1
2
2 b
2
% Initial state
1
% Final states
2
```

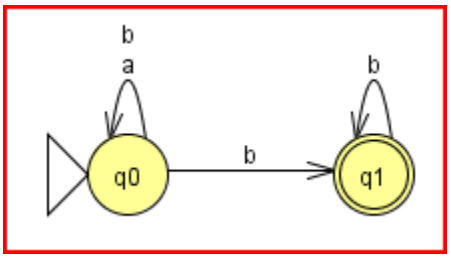
The input file is self-explanatory.

The specification automaton defines as acceptable behavior all the strings that start with ‘a’. The model of the system accepts strings that finish with ‘b’. The specification obviously does not hold on the system, however, this example demonstrates all the steps of the project.

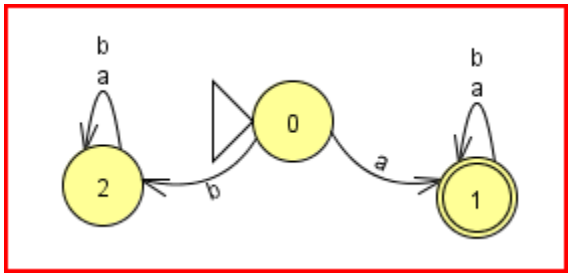
The shortest path from the initial state of the automaton to the final state is: 0i, 2ii. The corresponding string w = b is the counterexample that demonstrates that the specification does not hold on the automaton M.

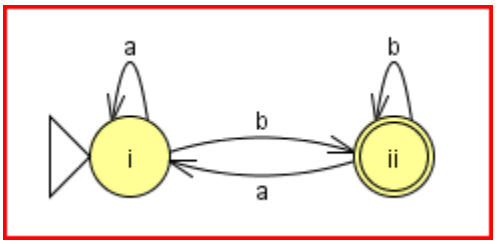
Inputs:



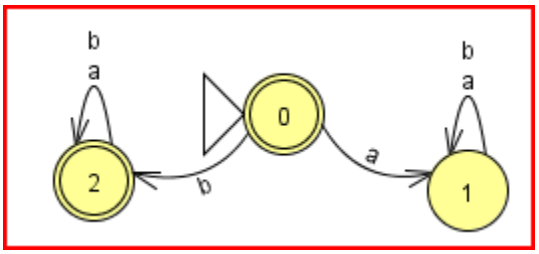


Conversion to DFA:





Complementation of specification:



Product automaton M: