Converting FSA to Code

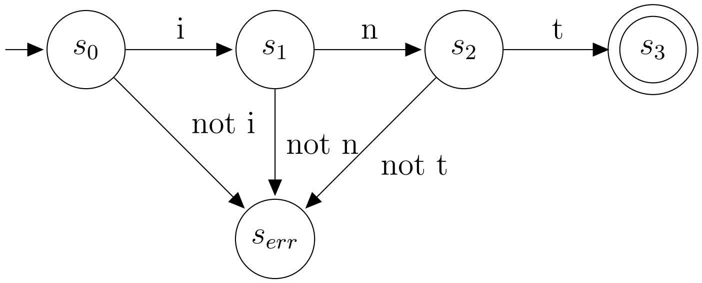
CS236 - Discrete Structures
Brett Decker and Michael Goodrich
Winter 2021

Introduction

It may not be obvious how to convert a finite-state automaton into code. This document will give some possible examples of how to implement basic finite-state automata in C++.

Encoding States and Transitions

One way to implement a finite-state automaton in C++ is to have a function per state and a function call per transition.
Consider the finite-state automaton that recognizes the C++ keyword “int”:



This could be implemented as follows (assuming `accept` and `index` are a member variable accessible by all methods):

```
public bool Start(std::string& input) {
   accept = false;
   index = 0;
   S0(input);
   return accept;
}

void S0(std::string& input) {
  if (input.at(index) == ’i’) {
    index++;
    S1(input);
  }
  else {
    Serr(input);
  }
}

void S1(std::string& input) {
  if (input.at(index) == 'n') {
    index++;
    S2(input);
  }
  else {
    Serr(input);
  }
}

void S2(std::string& input) {
  if (input.at(index) == 't') {
    index++;
    S3(input);
  }
  else {
    Serr(input);
  }
}

void S3(std::string& input) {
  accept = true;
}

void Serr() {
  // accept is already false
}
```

When applying this to a lexer using the Parallel and Max approach (described in the Project 1 Guide), instead of returning a true/false value for whether or not the input was accepting, we simply return an integer value indicating the number of characters read from the input. A value of 0 indicates the input was rejected (no character could be read). A value greater than 0 indicates the input was accepted and the number of characters read that led to the accepting state. Note that the s_{err} state now needs to set `read` to zero. The implementation now looks like:

```
public int Start(std::string& input) {
   read = 0;
   S0(input);
   return read;
}

void S0(std::string& input) {
  if (input.at(read) == ’i’) {
    read++;
    S1(input);
  }
  else {
    Serr();
  }
}

void S1(std::string& input) {
  if (input.at(read) == ’n’) {
    read++;
    S2(input);
  }
  else {
    Serr();
  }
}

void S2(std::string& input) {
  if (input.at(read) == ’t’) {
    read++;
    // No need to transition to the
    //   accepting state, since ‘read‘
    //   is already correct
  }
}

void Serr() {
  read = 0;
}
```