CS420

Introduction to the Theory of Computation

Lecture 17: Acceptance, emptiness and equality tests

Tiago Cogumbreiro

Today we will learn...

- Practical uses of Regular/Context-Free languages
- Acceptance tests
- Emptiness tests
- Equality tests

Section 4.1

Why do we need Regular Languages? Context Free Languages?

Use Case 1: DFA/NFA

Using a DFA/NFA to structure hardware usage

Use Case 1: DFA/NFA

Using a DFA/NFA to structure hardware usage

- Arduino is an open-source hardware to design **microcontrollers**
- Programming can be difficult, because it is highly concurrent
- Finite-state-machines structures the logical states of the hardware
- **Input:** a string of hardware events
- String acceptance is not interesting in this domain

Example

> The FSM represents the logical view of a micro-controller with a light switch

Use Case 1

Declare states

```
#include "Fsm.h"
// Connect functions to a state
State state_light_on(on_light_on_enter, NULL, &on_light_on_exit);
// Connect functions to a state
State state_light_off(on_light_off_enter, NULL, &on_light_off_exit);
// Initial state
Fsm fsm(&state_light_off);
```

Source: platformio.org/lib/show/664/arduino-fsm

Use Case 1

Declare transitions

```
// standard arduino functions
void setup() {
  Serial.begin(9600);

  fsm.add_transition(&state_light_on, &state_light_off,
                     FLIP_LIGHT_SWITCH,
                     &on_trans_light_on_light_off);
  fsm.add_transition(&state_light_off, &state_light_on,
                     FLIP_LIGHT_SWITCH,
                     &on_trans_light_off_light_on);
}
```

Source: platformio.org/lib/show/664/arduino-fsm

Use Case 1

Code that runs on before/after states

```
// Transition callback functions
void on_light_on_enter() {
  Serial.println("Entering LIGHT_ON");
}

void on_light_on_exit() {
  Serial.println("Exiting LIGHT_ON");
}

void on_light_off_enter() {
  Serial.println("Entering LIGHT_OFF");
}
// ...
```

Source: platformio.org/lib/show/664/arduino-fsm

Use Case 2

Regular Expressions: Input validation

Use Case 2

Regular Expressions: Input validation

HTML includes regular expressions to perform client-side form validation.

```
<input id="uname" name="uname" type="text"
       pattern="_([a-z]|[A-Z]|[0-9])+" minlength="4" maxlength="10">
```

- `_[a-zA-Z0-9]+`
- `[a-zA-Z0-9]` means any character beween a and z, or between `A` and `Z`, or between `0` and `9`
- `R+` means repeat `R` one or more times
- In this case, the username must start with an underscore `_`, and have one or more letters/numbers
- `minlength` and `maxlength` further restrict the string's length

Use Case 3

Regular Expressions: Text manipulation

Use Case 3

Regular Expressions: Text manipulation

Programming languages include regular expressions for fast and powerful text manipulation.

Example (JS)

```
let txt1 = "Hello World!";
let txt2 = txt1.replace(/[a-zA-Z]+/, "Bye"); // Replaces the first word by "Bye"
console.log(txt2);
// Bye World!
```

Use Case 4

Parsing JSON

Grammar for JSON

ANTLR is a **parser generator**.

- **Input:** a ***grammar***; **Output:** a parser, and data-structures that represent the parse tree (known as a Concrete Syntax Tree)
- The HTML DOM is an example of an ***Abstract*** Syntax Tree

```
json: value; // initial rule

obj: '{' pair (',' pair)* '}' | '{' '}'; // a sequence of comma-separated pairs

pair: STRING ':' value; // Example: "foo": 1

array: '[' value (',' value)* ']' | '[' ']'; // a sequence of comma-separated values

value: STRING | NUMBER | obj | array | 'true' | 'false' | 'null';
// ...
```

Source: raw.githubusercontent.com/antlr/grammars-v4/master/json/JSON.g4

A grammar for JSON integers

```
NUMBER: '-'? INT ('.' [0-9] +)? EXP?;

fragment INT: '0' | [1-9] [0-9]*; // fragment means do not generate code for this rule

fragment EXP : [Ee] [+\-]? INT;   // fragment means do not generate code for this rule
```

Source: raw.githubusercontent.com/antlr/grammars-v4/master/json/JSON.g4

A grammar for JSON

```
> ls *.java
JSONBaseListener.java JSONParser.java JSONVisitor.java
JSONBaseVisitor.java JSONLexer.java JSONListener.java
> cat JSONBaseListener.java
// Generated from ../JSON.g4 by ANTLR 4.7.2
import org.antlr.v4.runtime.tree.ParseTreeListener;

/**
 * This interface defines a complete listener for a parse tree produced by
 * {@link JSONParser}.
 */
public interface JSONListener extends ParseTreeListener {
        /**
         * Enter a parse tree produced by {@link JSONParser#json}.
         * @param ctx the parse tree
         */
        void enterJson(JSONParser.JsonContext ctx);
        /**
         * Exit a parse tree produced by {@link JSONParser#json}.
         * @param ctx the parse tree
         */
        void exitJson(JSONParser.JsonContext ctx);
```

Context-free languages

- Programming languages are context-free
- Context-free grammars are crucial for compilers/interpreters

Why do we need Turing Machines?

Why do we need Turing Machines?

- Turing Machines are Computers!
- Turing Machines are Programs!
- We will study mathematically the limits of what is possible

Turing Recognizable

Decision problems

> **Disclaimer:** Henceforth, when we say a program, we are restricting ourselves to **decision problems**.

- Example: DFA accepting/rejecting a string
- Example: PDA accepting/rejecting a string
- Example: functions that return a boolean
- Example: programs run until they print yes/no
- Example: computers that run until they turn on a red/green light

Turing Recognizable

Recognized language of TM

Notation $L(M)$ is the set of strings that M **accepts**, its recognized language.

Definition 3.5: Turing-Recognizable language

A language is Turing-recognizable if some TM recognizes it.

What is a decidable Turing Machine?

What is a decidable Turing Machine?

For all inputs: REJECT $\vee$ ACCEPT

(No loops!)

Turing Decidable

A TM that for all inputs either accepts or rejects, and does not loop forever.

Definition 3.6: Decidable language

A language is decidable if some Turing-decidable machine recognizes it.

How do I know if a Turing Machine is decidable?

Turing Decidable

A TM that for all inputs either accepts or rejects, and does not loop forever.

Definition 3.6: Decidable language

A language is decidable if some Turing-decidable machine recognizes it.

How do I know if a Turing Machine is decidable?

We prove it!

Recap

- A decidable TM REJECT/ACCEPT any input
- A decidable ***language*** is one that is recognized by a decidable TM

Decidable algorithms

- Algorithms are equivalent to TMs
- An algorithm that returns REJECT/ACCEPT (eg, a boolean) for all inputs
- A decidable algorithm is always **total**†
  - A total function is defined (ie, returns a value) for all inputs. Looping implies that no value is being returned.
  - In some programming languages (eg, Coq, Agda, Idris) you can write total functions (mechanically proven by the language).

Decidable algorithms

- Algorithms are equivalent to TMs
- An algorithm that returns REJECT/ACCEPT (eg, a boolean) for all inputs
- A decidable algorithm is always **total**†
  - A total function is defined (ie, returns a value) for all inputs. Looping implies that no value is being returned.
  - In some programming languages (eg, Coq, Agda, Idris) you can write total functions (mechanically proven by the language).

Proving decidability

requires a proof that the function **terminates**!

(along with **correctness**)

Decidable algorithms

Example

Our algorithm that implements DFA acceptance

```
def dfa_accepts(dfa, inputs):
  st = dfa.start
  for i in inputs:
    st = dfa.state_transition(st, i)
  return st in dfa.accepted_states
```

Termination proof. The function loops over `len(inputs)`-steps (which is a natural number) and then returns a boolean.

A_X: Acceptance tests

Decidable algorithms on acceptance

(Will X accept this input?)

A_{DFA}: DFA Acceptance

> The language of all DFAs that accept a given string w

$$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Theorem 4.1. A_{DFA} is a decidable language.

Proof.

A_{DFA}: DFA Acceptance

> The language of all DFAs that accept a given string w

$$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Theorem 4.1. A_{DFA} is a decidable language.

Proof.

We already showed that function `dfa_accepts` is correct and that it terminates, thus there exists a TM that encodes it and that TM is decidable.

A_{NFA}: NFA Acceptance

> The language of all NFAs that accept a given string w

$$A_{\text{NFA}} = \{\langle N, w\rangle \mid N \text{ is an NFA that accepts input string } w\}$$

Theorem 4.2. A_{NFA} is a decidable language.

Proof.

A_{NFA}: NFA Acceptance

> The language of all NFAs that accept a given string w

$$A_{\text{NFA}} = \{\langle N, w\rangle \mid N \text{ is an NFA that accepts input string } w\}$$

Theorem 4.2. A_{NFA} is a decidable language.

Proof.

If we assume that the function that converts a DFA into an NFA is total, then the following algorithm is total and correct:

```
def nfa_accepts(nfa, input):
  return dfa_accepts(nfa_to_dfa(nfa), input)
```

And therefore, the TM that implements it is decidable, and so is A_{NFA}.

A_{REX}: Regular Expression Acceptance

UMASS BOSTON

> The language of all regex that accept a given string w

$$A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is an regular expression that accepts input string } w\}$$

Theorem 4.3. A_{REX} is a decidable language.

Proof.

A_{REX}: Regular Expression Acceptance

> The language of all regex that accept a given string w

$$A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is an regular expression that accepts input string } w\}$$

Theorem 4.3. A_{REX} is a decidable language.

Proof.

Similarly, if we assume that the function that converts a regular expression into an NFA is total, then the following algorithm is total and correct:

```
def rex_accepts(rex, input):
  return nfa_accepts(rex_to_nfa(nfa), input)
```

And therefore, the TM that implements it is decidable, and so is A_{REX}.

A_{CFG}: Context-Free-Grammar Acceptance

UMASS BOSTON

> The language of all context-free grammars that accept a given string w

$$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a context-free grammar that accepts input string } w\}$$

Theorem 4.7. A_{CFG} is a decidable language.

Proof.

A_{CFG}: Context-Free-Grammar Acceptance

UMASS BOSTON

> The language of all context-free grammars that accept a given string w

$$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a context-free grammar that accepts input string } w\}$$

Theorem 4.7. A_{CFG} is a decidable language.

Proof.

We studied the CYK algorithm that is decidable and given a CFG can test the acceptance of a CFG. Additionally, we also studied a decidable acceptance algorithm for PDAs, so we could convert the CFG to a PDA (which is a total function).