PDA

Stefan Hauser

April 24, 2020

Contents

1 Einleitung

1.1 Was ist PDA?

PDA ist eine C++-Konsolenapplikation, welche als tabellenbasierter Kellerautomat fungiert und somit die Gültigkeit eines Eingabewortes evaluiert. Die Struktur des Kellerautomaten wird dabei aus einer TOML-Datei geladen.

1.2 Was ist TOML?

Das für den Datenaustausch erstellte Dateformat TOML legt vor allem Wert auf gute Lesbarkeit und soll sich auch leicht parsen lassen. TOML bietet dabei die Möglichkeit, sich in eine Zuordnungstabelle, auch assoziatives Datenfeld genannt, umwandeln zu lassen. Das heißt, dass die Schlüssel für die Zuordnung der Elemente nicht numerisch sein muss, der Schlüssel kann beispielsweise auch ein String sein.
Zum parsen der TOML-Dateien habe ich die C++ - Bibliothek TOML++ verwendet. Durch sie kann ich das File einlesen und bekomme als Ergebnis ein Objekt des Datentyps tomltable zurück, aus welchem ich die Elemente meines Automaten extrahieren kann.

1.3 Wie muss ein TOML-File für DBA aufgebaut sein?

Damit der Automat richtig eingelesen werden kann, müssen im TOML-File der Titel des Automaten sowie eine Tabelle mit den Definitionen des Automaten existieren. Weiters wird eine Tabelle von Transitionstabellen benötigt. Ein TOML-File könnte etwa so aussehen:

```
title = "Kellerautomat"

[definitions]
z0 = "A"
E = ["0", "1"]
F = ["B"]
K = ["_", "0", "1"]
Z = ["A", "B"]
k0 = "_"

[[table.transition]]
z = "A"
k0 = "_"
```

```
e = "0"
z_new = "B"
k_new = "_"
```

2 Implementierung

2.1 Logging

Die Logger sollen in main.cpp aktiviert werden können und im Automaten selbst die Schritte tracken. Dies habe ich so gelöst, dass ich ein eigenes Logger-Modul geschrieben habe, in welchem sich shared pointer auf Logger-Objekte befinden.
Die einzelnen Logger sind in Logger.cpp definiert:

```
const std::shared_ptr<spdlog::logger> Logger::logger =
        spdlog::stdout_color_st("atomaton_logger");
const std::shared_ptr<spdlog::logger> Logger::debug_logger =
        spdlog::stderr_color_st("atomaton_debug");
```

2.2 Auswerten der Kommandozeilenparameter

Die Eingaben des Nutzers werden mit Hilfe der C++ - Bibliothek CLI11 in main.cpp abgearbeitet. Das eigentliche Eingabewort ist dabei als "positional option" definiert und wird in einem String gespeichert.
Durch die Option -f kann ein eigenes TOML-File angegeben werden, standartmäßig wird automaton.toml geladen.
Durch die Angabe von -d werden die einzelnen Schritte des Automaten für Debug-Zwecke mitprotokolliert. Diese Meldungen reichen vom Parsen des TOML-Files bis hin zu Zwischenschritten bei den Transaktionen.
Durch die Angabe von -v werden die Transitions ausgegeben, um so zu beobachten, wie der Automat zu seinem Ergebnis kommt.

2.3 Parsen des TOML-File und befüllen des Automaten

Das Einlesen und Parsen der TOML-Datei sowie das Befüllen der Attribute des Automaten erfolgt in der Funktion load, welche sich die TOML-Datei beziehungsweise den Pfad der TOML-Datei in Form eines Strings als Parameter erwartet.
Wird die Datei gefunden, wird ihr Inhalt als toml table in der Variablen

config gespeichert. Des Weiteren wird ein Automaten-Objekt erstellt, dessen Attribute werden im nächsten Schritt aus config befüllt.
Auf die Elemente des Table, in TOML++ genannt Knoten, kann direkt zugegriffen werden. Der Inhalt des Tables definitions wird in einer eigenen Variable gespeichert, auch auf dessen Inhalt kann wieder direkt zugegriffen werden. Somit können die Werte aus der TOML-Datei gelesen, überprüft und - falls korrekt - in den Automaten geladen werden.
Auschnitt wie durch die Transitions iteriert wird:

```
auto transition_node = (*table)["transition"];
auto transitions = transition_node.as_array();

for (const auto& t : *transitions) {

        auto transition_table = t.as_table();

        auto z_node = (*transition_table)["z"];
        ....
}
```

Etwas komplizierter ist es bei den Transitions, da diese vereinfacht gesagt Table in einem Table sind und nicht direkt auf diese zugegriffen werden kann. Daher musste ich den Knoten zu einem Array machen und durch diesen iterieren. Ist das Parsen der Transition fertig wird ihr Index im Transition Table festgelegt und sie wird unter diesem darin gespeichert.
Die Berrechnung des Transition-Index erfolgt so:

```
auto state_index_it{std::find(states.begin(),
        states.end(), current_state)};
auto state_index{std::distance(states.begin(),
        state_index_it)};

auto stack_character_index_it{std::find(stack_alphabet.begin(),
        stack_alphabet.end(), stack_token)};
auto stack_character_index{std::distance(stack_alphabet.begin(),
        stack_character_index_it)};

auto input_character_index_it{std::find(input_alphabet.begin(),
        input_alphabet.end(), input_token)};
auto input_character_index = std::distance(input_alphabet.begin(),
        input_character_index_it);
```

```
size_t row{state_index * stack_alphabet.size() +
        stack_character_index};
size_t index{row * (input_alphabet.size() + 1) +
        input_character_index};
```

Sollte bei einem Element ein Fehler auftreten wird ein Fehler geworfen, andernfalls wird der fertige Automat zurückgeliefert.

2.4 Evaluation der Eingabe

Ob die Eingabe akzeptiert wird oder nicht wird in der Funktion check bestimmt, welche als Parameter das Eingabewort als String erhält. Sie geht jedes Zeichen des Eingabewortes durch und übergibt dieses der Funktion next, welche durch die Funktion getTransition prüft, ob es für das aktuelle Eingabezeichen und den aktuellen Zustand zusammen mit dem obersten Element des Stack eine Transition gibt. Durch die Funktion transitionTo wird zuletzt der aktuelle Zustand gewechselt und das Eingabezeichen wird auf den Stack gelegt.
Die Check-Funktion:

```
bool PD_Automaton::check(const std::string& word) {
    bool accepted{true};
    for (const auto& token : word) {
        if (!(accepted = next(token)))
            break;
    }

     while (std::find(accepted_states.begin(), accepted_states.end(),
        == accepted_states.end() && accepted)
                accepted = next(0);

    accepted = accepted && (std::find(accepted_states.begin(),
    accepted_states.end(), current_state) != accepted_states.end());

    return accepted;
}
```

Sollte es keine entsprechende Transition geben, wird eine Transition mit einem empty character versucht. Wenn gar keine Transition möglich ist wird false zurückgeliefert.

3 Anwendung

Der Aufruf des Programms erfolgt mittels
./dba <Eingabewort> [-d] [-v] [-f <TOMLDATEI>]

Das Eingabewort ist zwingend, die anderen Optionen sind optional. Wird

durch -f kein anderes File angegeben, wird standartmäßig "./../automaton.toml" geladen.