# CSE 355: Module 7 Machine Design (PDAs)

Dr. Jamison W. Weber

Due Tuesday, October 15, 2024, 11:59pm

In this individual assignment, you will formulate pushdown automata (PDAs) that recognize target languages using a Python programming interface. The programming interface enables you to formally define PDAs, check their validity, visualize them as state machines, and evaluate their computation on test strings. When you're satisfied with your PDAs, export and upload them to the corresponding Gradescope assignment for autograding. This will provide immediate, personalized feedback if there are any mistakes. You may revise and resubmit your code as many times as you'd like until the due date.

## 1 Target Languages

For each of the following problems, formulate a PDA that recognizes the target language. Throughout, assume all languages are defined over the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$.

### Problem 1

$L_1 = \{w \mid w \text{ contains at least three } \texttt{1}\text{s}\}$.

### Problem 2

$L_2 = \{w \mid w \text{ starts and ends with the same symbol}\}$.

*Note.* You may assume that $\varepsilon \notin L_2$ because it has neither a starting nor ending symbol.

### Problem 3

$L_3 = \{w \mid |w| \text{ is odd and its middle symbol is } \texttt{0}\}$.

### Problem 4

$L_4 = \{w \mid w = w^{\mathcal{R}};\ \text{i.e., } w \text{ is a palindrome}\}$.

*Note.* Recall that for a string $w = w_1 w_2 \cdots w_n$, its reverse is $w^{\mathcal{R}} = w_n w_{n-1} \cdots w_1$. You may assume $\varepsilon = \varepsilon^{\mathcal{R}}$.

### Problem 5

$L_5 = \{w\texttt{\#}x \mid w^{\mathcal{R}} \text{ is a substring of } x\}$.

*Note.* The # symbol that marks where $w$ ends and $x$ begins is an extra symbol being added to the alphabet for this language only. The strings $w$ and $x$ are both defined over $\Sigma = \{\texttt{0}, \texttt{1}\}$.

### Problem 6

$L_6 = \{\texttt{0}^m \texttt{1}^n \mid m > n \geq 0\}$.

### Problem 7

$L_7 = \{\texttt{0}^m \texttt{1}^{m-n} \texttt{0}^n \mid m \geq n \geq 0\}$.

## 2 Grading

The instructions in the following sections will show you how to use the programming interface, test your ideas, and ultimately export your solutions as a `.json` file. This file contains specifications of your PDAs, which Gradescope will instantiate and test to see if they correctly recognize the target languages.

Each problem (PDA) is worth 5 points. You get 1 point if the PDA is valid (i.e., it follows all the rules of a formal specification) and an additional 4 points if it recognizes the target language. If your PDA is invalid, Gradescope will tell you what the problem is. If your PDA recognizes a language other than the target one, Gradescope will output (1) example strings that your PDA recognizes but are outside the target language, and/or (2) example strings that are in the target language but your PDA does not recognize. This feedback should guide your resubmissions.

This assignment is worth 30 points, but with seven problems and 5 points per problem, there is the opportunity to earn up to 35 points total (up to 5 points of extra credit). Thus, you have the choice of either skipping one problem without penalty or trying to complete them all for the extra credit.

## 3 Installation

This assignment uses the same setup as the previous Machine Design assignments: `python 3.10` (or a later version) and the `cse355-machine-design` package. Detailed installation instructions can be found in the previous assignments. To ensure that you're running the latest version of our package, run:

```
> pip install cse355-machine-design --upgrade
```

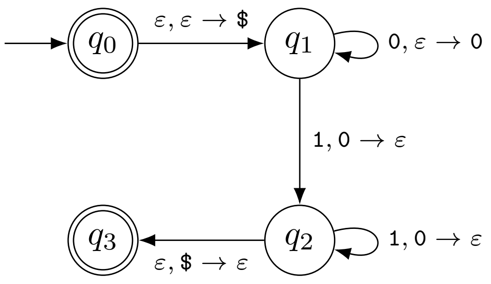
As before, complete details on the package and comprehensive documentation can be found at https://github.com/DaymudeLab/CSE355-MachineDesign.

## 4 Usage Guide

In this section, we demonstrate how to define and interact with PDAs using the `cse355-machine-design` package. Suppose that we are asked to formulate an PDA recognizing the language

$$L = \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}.$$

As you may recall from Lecture 1 of this module, the below PDA $P$ recognizes $L$:


$q_0$
$\varepsilon, \varepsilon \to \$$
$q_1$
$\mathtt{0}, \varepsilon \to \mathtt{0}$
$\mathtt{1}, \mathtt{0} \to \varepsilon$
$q_2$
$\mathtt{1}, \mathtt{0} \to \varepsilon$
$q_3$
$\varepsilon, \$ \to \varepsilon$

### 4.1 Defining a PDA

To start, download the template Python source file called `module7_machinedesign.py` from Canvas. The top of this file defines our package imports:

```python
from cse355_machine_design import PDA, registry
```

Next, we have functions for defining the seven PDAs for this assignment (one per problem). Copy/paste one of these functions to add a new function for our example PDA, $P$:

```
def example():
    """
    L = {0^n1^n | n >= 0}
    """

    Q = {}
    Sigma = {}
    Gamma = {}
    delta = {

    }
    q0 = ""
    F = {}

    return PDA(Q, Sigma, Gamma, delta, q0, F)
```

The six variables you need to define (`Q`, `Sigma`, `Gamma`, `delta`, `q0`, and `F`) are the six components in a PDA's formal definition $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. PDA $P$ has states $Q = \{q_0, q_1, q_2, q_3\}$, so we write these states as strings in our Python function:

```
    Q = {"q_0", "q_1", "q_2", "q_3"}
```

Next, we define our input alphabet. The input strings are defined over $\{\texttt{0}, \texttt{1}\}^*$, so:

```
    Sigma = {"0", "1"}
```

We define our stack alphabet $\Gamma = \{\texttt{0}, \$\}$ analogously:

```
    Gamma = {"0", "$"}
```

Recall that the transition function for a PDA has the form $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$; i.e., $\delta$ takes the current state, an input symbol to read (which may be $\varepsilon$), and a stack symbol to pop (which may be $\varepsilon$) and maps them to a next state and a stack symbol to push (which may be $\varepsilon$). Thus, our `delta` dictionary has keys that are state-input-pop triples and values that are sets of state-push pairs. As for NFAs, we use the underscore "_" symbol to represent $\varepsilon$:

```
    delta = {
        ("q_0", "_", "_"): {("q_1", "$")},
        ("q_1", "0", "_"): {("q_1", "0")},
        ("q_1", "1", "0"): {("q_2", "_")},
        ("q_2", "1", "0"): {("q_2", "_")},
        ("q_2", "_", "$"): {("q_3", "_")}
    }
```

The start state of $P$ is $q_0$, so:

```
    q0 = "q_0"
```

Finally, our set of accept states is $\{q_0, q_3\}$, so:

```
    F = {"q_0", "q_3"}
```

The completed function is:

```
def example():
    """
    L = {0^n1^n | n >= 0}
    """
    Q = {"q_0", "q_1", "q_2", "q_3"}
    Sigma = {"0", "1"}
    Gamma = {"0", "$"}
    delta = {
        ("q_0", "_", "_"): {("q_1", "$")},
        ("q_1", "0", "_"): {("q_1", "0")},
        ("q_1", "1", "0"): {("q_2", "_")},
        ("q_2", "1", "0"): {("q_2", "_")},
        ("q_2", "_", "$"): {("q_3", "_")}
    }
    q0 = "q_0"
    F = {"q_0", "q_3"}

    return PDA(Q, Sigma, Gamma, delta, q0, F)
```

## 4.2 Interacting with Your PDA

Now that we've defined the PDA $P$, we can put it to work. In a terminal, navigate to the directory containing your `module7_machinedesign.py` file and then run the `python` command. This will start an interactive session:

```
> python
Python 3.10.13
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Typing

```
>>> from module7_machinedesign import *
```

will import all definitions from your file into this interactive session. You can then set up one of your PDAs, like the example machine $P$ that we defined:

```
>>> example_pda = example()
```

Our programming interface provides tools for investigating and testing your PDAs. First, you can open a state diagram of your PDA in your web browser using:

```
>>> example_pda.display_state_diagram()
```

Second, you can evaluate whether an input string is accepted (`True`) or rejected (`False`) by passing it to the `evaluate` function:

```
>>> example_pda.evaluate("000111")
True
>>> example_pda.evaluate("100")
False
```

PDAs, like NFAs, are nondeterministic. However, because of a PDA's stack, its branching computation can be very complicated to follow. So instead of writing out all the branches when you add the `enable_trace=2` parameter, we instead show one possible accepting computation if your PDA accepts the input string:

```
>>> example_pda.evaluate("000111", enable_trace=2)
State: "q_0"    Input: ".000111"       Stack: []
Read: "_"       Pop: "_"       Push: "$"
State: "q_1"    Input: ".000111"       Stack: ['$']
Read: "0"       Pop: "_"       Push: "0"
State: "q_1"    Input: "0.00111"       Stack: ['0', '$']
```

```
Read: "0"       Pop: "_"        Push: "0"
State: "q_1"    Input: "00.0111"        Stack: ['0', '0', '$']
Read: "0"       Pop: "_"        Push: "0"
State: "q_1"    Input: "000.111"        Stack: ['0', '0', '0', '$']
Read: "1"       Pop: "0"        Push: "_"
State: "q_2"    Input: "0001.11"        Stack: ['0', '0', '$']
Read: "1"       Pop: "0"        Push: "_"
State: "q_2"    Input: "00011.1"        Stack: ['0', '$']
Read: "1"       Pop: "0"        Push: "_"
State: "q_2"    Input: "000111."        Stack: ['$']
Read: "_"       Pop: "$"        Push: "_"
State: "q_3"    Input: "000111."        Stack: []
Accepting configuration reached, accepting string
True
```

Otherwise, if the string is not accepted, you will simply see a short failure message.

## 4.3 Submitting Your PDAs

At the bottom of the template file, you will see the following code:

```
if __name__ == "__main__":
    problem1().submit_as_answer(1)
    problem2().submit_as_answer(2)
    problem3().submit_as_answer(3)
    problem4().submit_as_answer(4)
    problem5().submit_as_answer(5)
    problem6().submit_as_answer(6)
    problem7().submit_as_answer(7)
    registry.export_submissions()
```

*Don't change this code!* Each `problem#()` function is building and returning a PDA based on how you filled out those functions above. The `submit_as_answer` function is recording those PDAs as your answers to the corresponding problems. It is *very important* that the parameter to the `submit_as_answer` function matches the problem number! If these differ, Gradescope won't be able to match up your answers to its grading code and you won't receive any points.

When you're ready to submit your answers, run `python module7_machinedesign.py` on the command line in the directory of your assignment file. This will create a `submissions.json` file that you upload to Gradescope to be graded.