1. Language Definition

The language L is formally defined as:

$L = \{w \in \Sigma^* \mid N_1(w) = 2 \cdot N_0(w)\}$ $\Sigma = \{0, 1\}$

- $\Sigma = \{0, 1\}$: The alphabet of the language.
- w: Any string belonging to $\{0, 1\}^*$. Including the empty string
- Nc(w): Function that returns the number of occurrences of character c in string w.

2. Classification and Recognition

The language (L) is classified as a Context-Free Language (CFL). It is not a Regular Language (as it is beyond the capability of a Finite Automaton), and critically, it is not a Deterministic Context-Free Language (DCFL). Consequently, the recognition of this language necessitates a Non-deterministic Pushdown Automaton (NPDA), because the Deterministic Pushdown Automaton (DPDA) cannot recognize it based on mathematical proofs.

3. Objective and Scope

The main goal of this work is to construct a Deterministic Pushdown Automaton (DPDA) that can recognize the language L which consists of all binary strings where the count of '1's is precisely double the count of '0's, including the empty string. This document serves to track the iterative methodology used to reach the **Final Formal DPDA** from its initial concept to its final implementation.

4. Proposed Solution

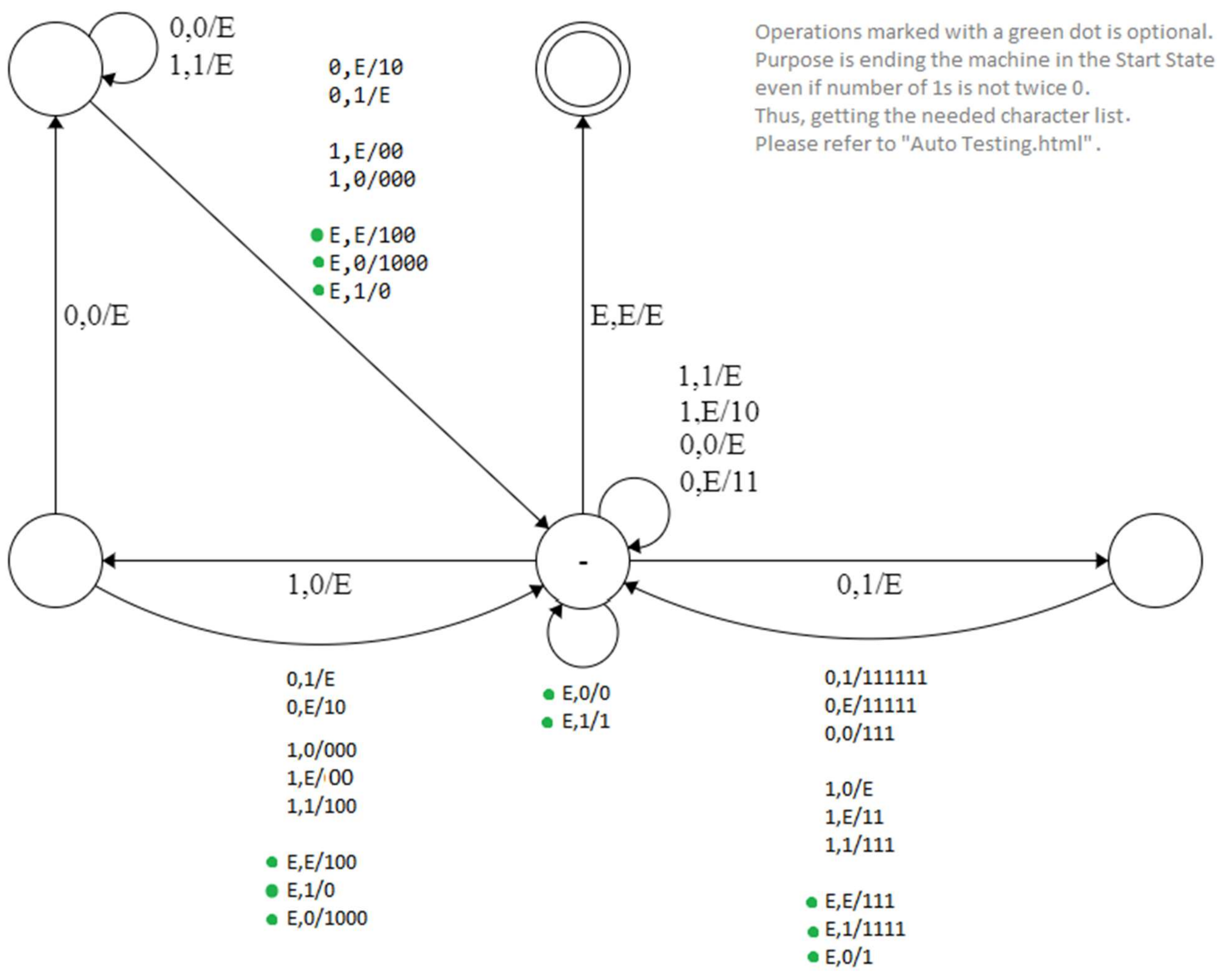
The approach followed to construct a Deterministic Pushdown Automaton (DPDA) for the language L is to use the stack to hold the "expected" characters for every character read from the input string. This technique is a very common method used in PDA designs.

- The process is not handled by a single state but also uses left and right states.
- While the solution path is simple and common, a paradox arises in the context of this specific problem, which contradicts the general logic of the solution. This paradox is resolved by an operation performed in the left state: the "cancellation of '0's between the input string and the stack."
- For resolving a paradox or arriving at a mathematically correct result, the simplification (cancellation) of zeros appears to be a property/operation that should not be overlooked. It may have a meaning or carry significance from a mathematical perspective.

5. Steps for Constructing the Final DPDA

This work initially began as a standalone effort with an experimental attempt in 2017. Although that proposed design appeared to be a close relative of the final design, it was later understood that the solution was incorrect. The first correct design was shared on my GitHub profile in April 2023. This section presents all steps, starting from the first correct PDA design and leading to the final DPDA and final Formal DPDA.

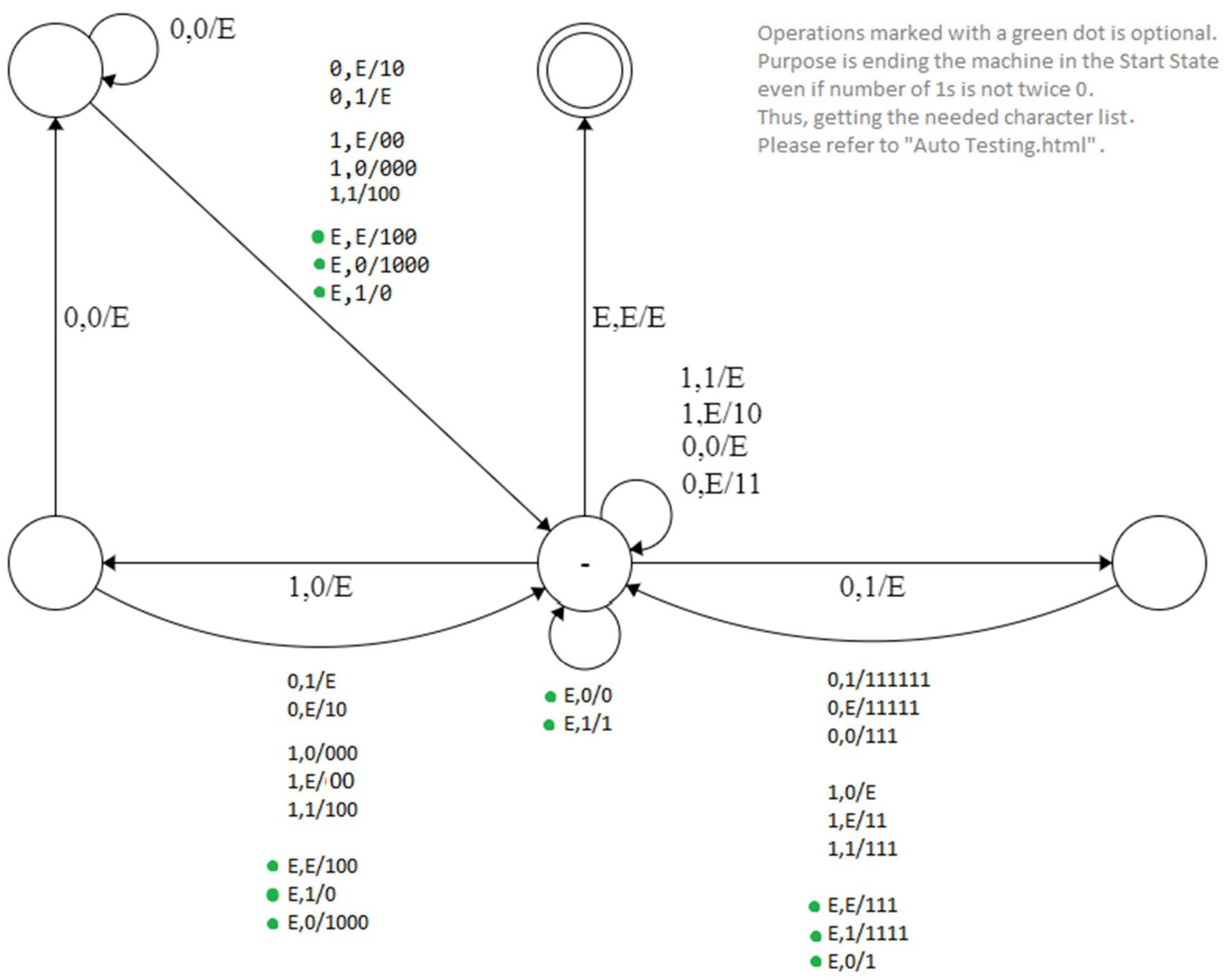
5.1 Original PDA


Operations marked with a green dot is optional.
Purpose is ending the machine in the Start State
even if number of 1s is not twice 0.
Thus, getting the needed character list.
Please refer to "Auto Testing.html".

The design presented above is the original construction. It starts in the **Center** state. If the input is the empty string, it is accepted. For non-empty inputs, the automaton writes onto the stack the characters that are *expected to be seen* for each input symbol processed.

The **Left** and **Right** states perform a lookahead operation by examining the next input symbol together with the current stack symbol, and then pushing the required symbols onto the stack accordingly.

5.2. Intermediate PDA



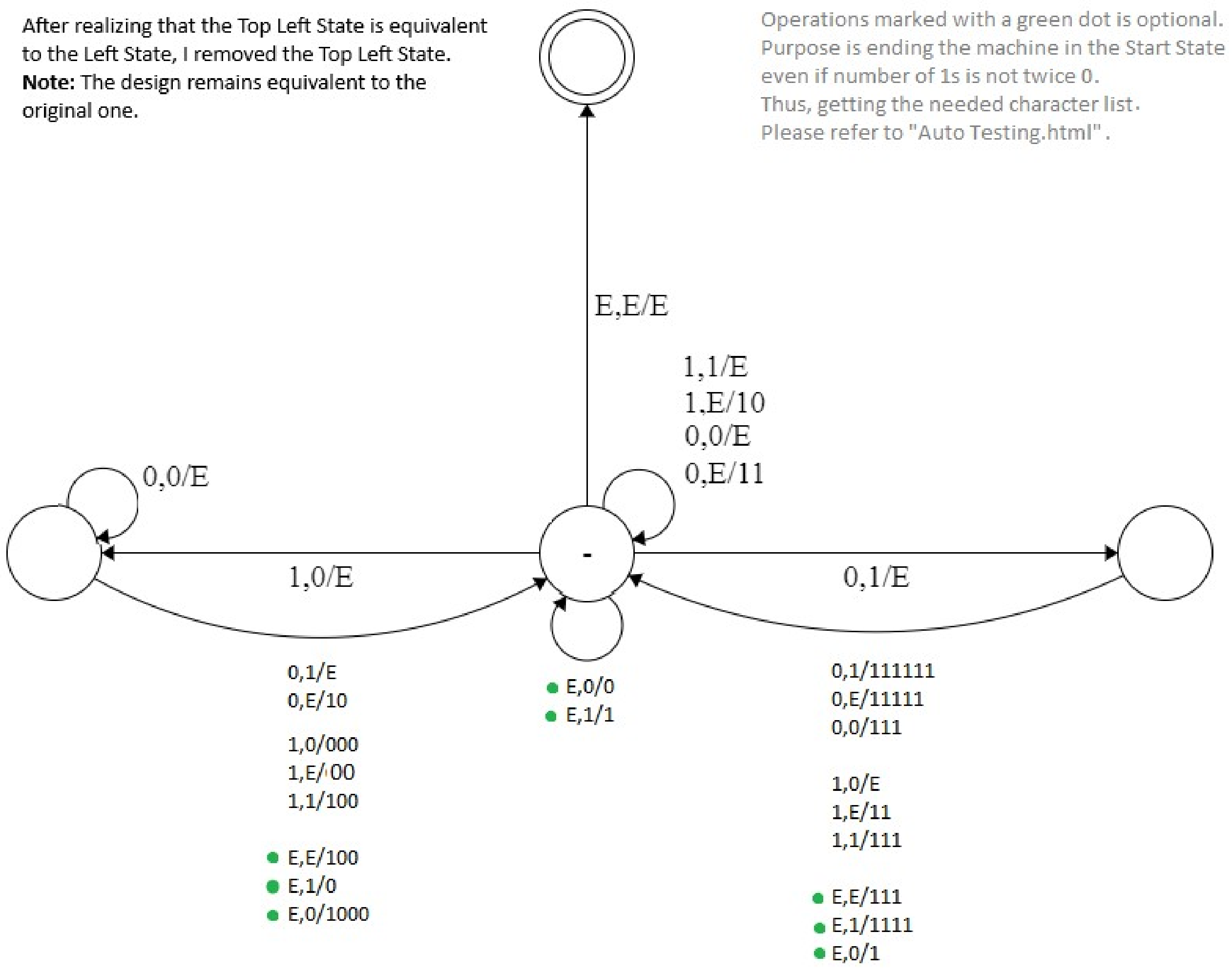
The design above represents the Intermediate PDA Design. The sole modification from the original was in the "top left state," where the transition 1, 1/E was changed to 1, 1/100 and redirected to the **start state** instead of looping back to **top left state**.

Analysis conducted on the original design revealed that the following transitions were never visited (or utilized) by the machine:

- In the Left State: 0,1/E | 1,1/100 | E,1/0
- In the Top Left State: 0,1/E | 1,1/E | E,1/0

It was theorized that if the transition 1,1/E in the "top left state" is **never visited**, it could be rewritten as 1,1/100 and redirected to the start state (instead of looping), making the "top left state" and the "left state" functionally identical. This rearrangement, implemented in this design, enabled its transformation into the **PDA Design** introduced in Section 5.3.

5.3. Final PDA

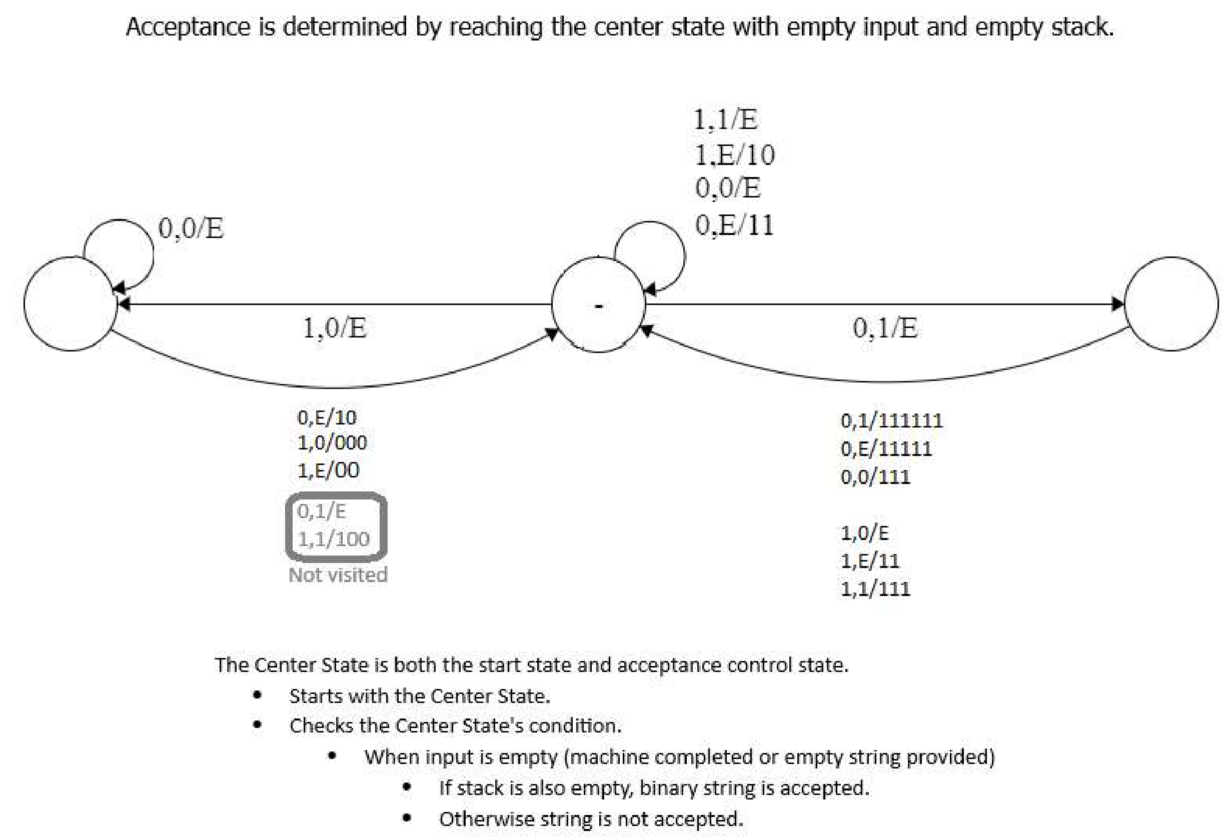

After realizing that the Top Left State is equivalent to the Left State, I removed the Top Left State.
Note: The design remains equivalent to the original one.
Operations marked with a green dot is optional.
Purpose is ending the machine in the Start State even if number of 1s is not twice 0.
Thus, getting the needed character list.
Please refer to "Auto Testing.html".
E,E/E
1,1/E
1,E/10
0,0/E
0,E/11
0,0/E
-
1,0/E
0,1/E
0,1/E
0,E/10
1,0/000
1,E/100
1,1/100
● E,E/100
● E,1/0
● E,0/1000
● E,0/0
● E,1/1
0,1/111111
0,E/11111
0,0/111
1,0/E
1,E/11
1,1/111
● E,E/111
● E,1/1111
● E,0/1

The Final PDA design presented above was obtained after refining the intermediate design and recognizing that the **Top Left** and **Left** states were identical. Consequently, the **Top Left** state was completely removed from the construction.

An analysis was conducted on the original design. During this analysis, transitions that were never visited were identified. It was observed that a transition in the **Top Left** state was never reached and could therefore be modified without affecting the behavior of the automaton. After applying the corresponding adjustment, the **Top Left** state was found to be identical to the **Left** state. Consequently, it was concluded that the **Top Left** state could be eliminated from the design.

All designs presented so far are equivalent to one another.

5.4. DPDA



Acceptance is determined by reaching the center state with empty input and empty stack.

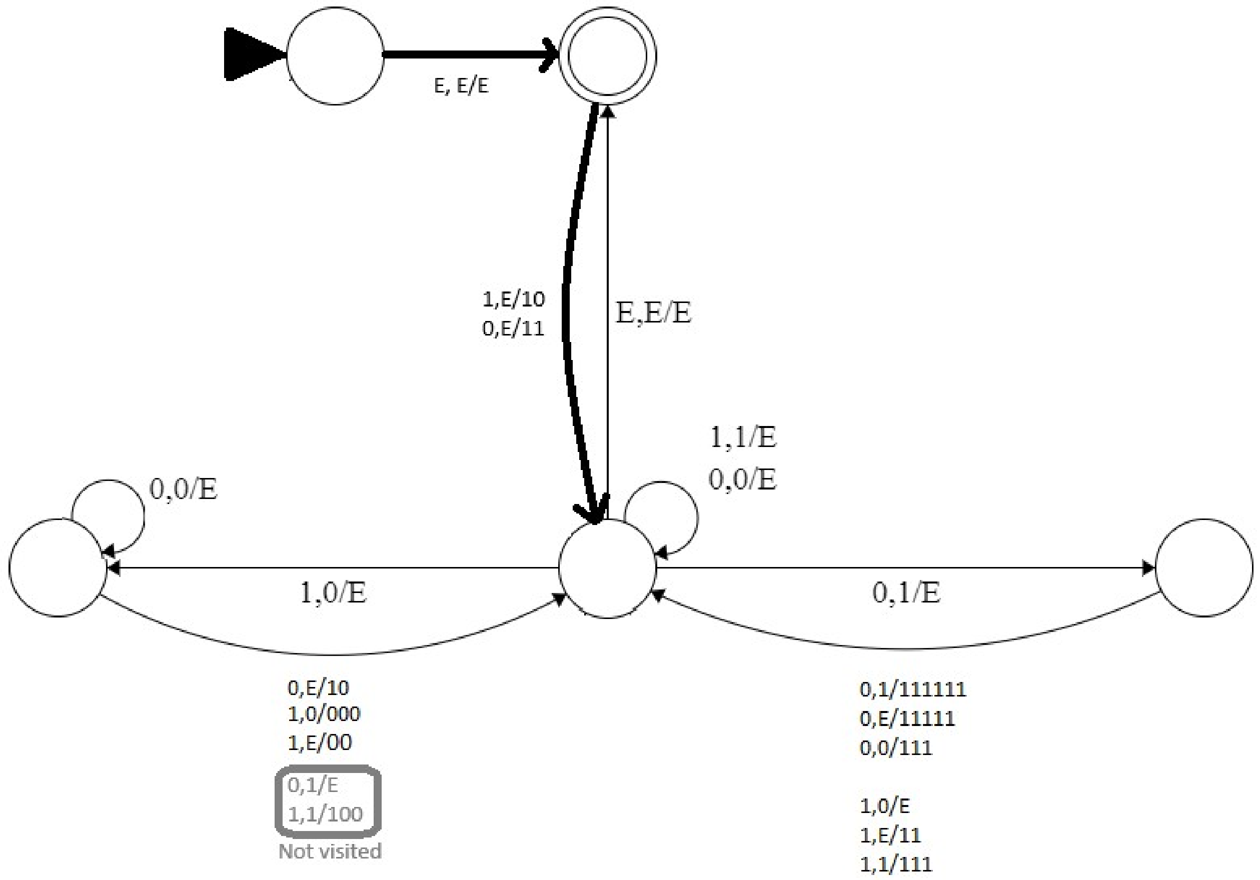
The Center State is both the start state and acceptance control state.

- Starts with the Center State.
- Checks the Center State's condition.
    - When input is empty (machine completed or empty string provided)
        - If stack is also empty, binary string is accepted.
        - Otherwise string is not accepted.

The E transitions that is violating the determinism in the final PDA design were removed. In addition, the E transition from the center state to the final state, which was another factor violating determinism, was also eliminated.

From the perspective of determinism, it is possible to define a machine's acceptance criterion for only one state as "accepts on empty input and empty stack," even if the design does not explicitly utilize a designated final state. Therefore, this acceptance criterion was added to the design, and the **Deterministic Pushdown Automaton (DPDA)** was successfully achieved.

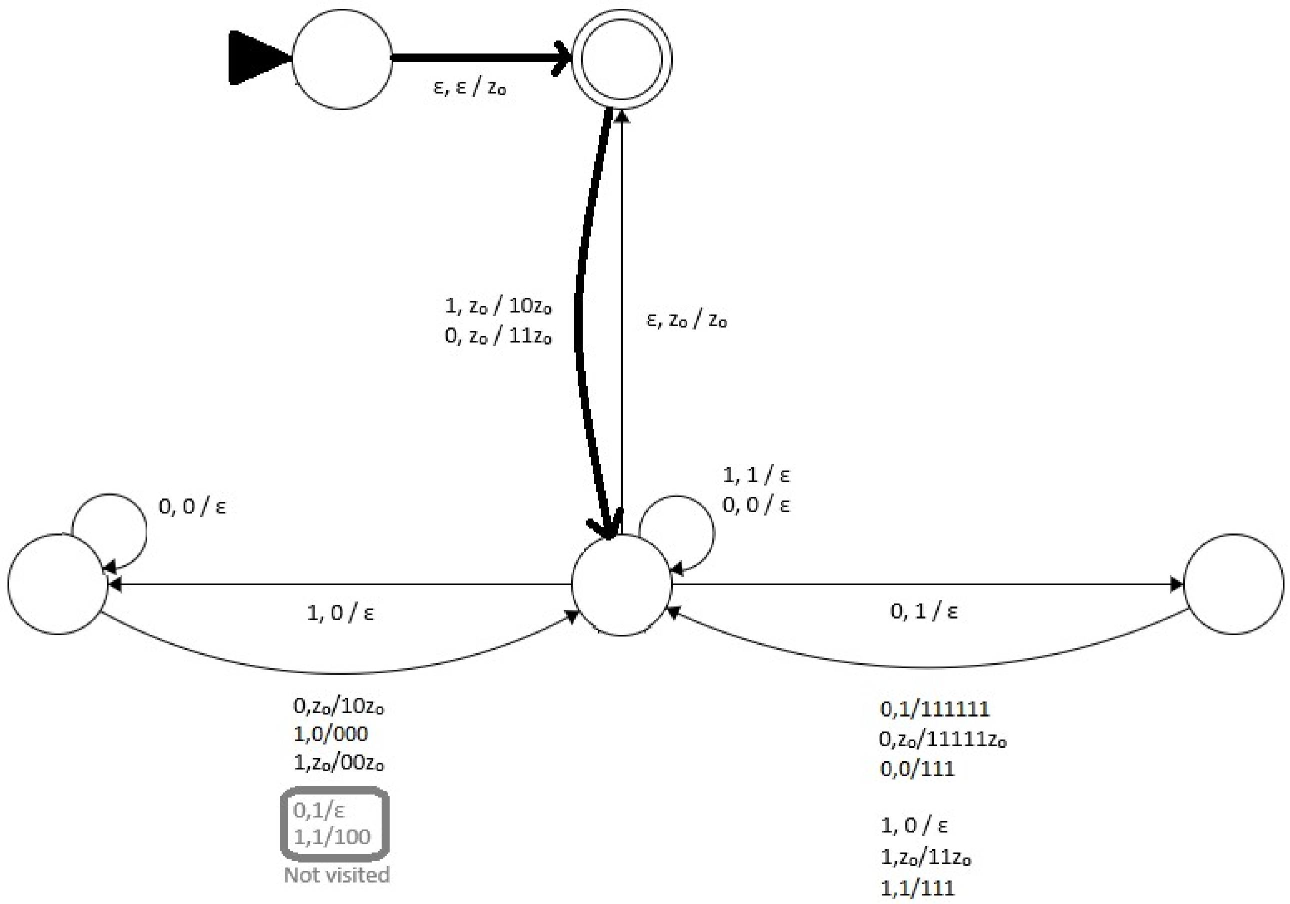
5.5. Final DPDA



Studies continued on the DPDA presented in the previous section. The objective of these studies was to define a **final state** and implement a transition to this final state from the center state upon the condition of an **"empty input, empty stack"** rule, all while maintaining determinism.

The final state was successfully defined, the transitions were completed, and the **Final DPDA** was achieved.

5.6. Final Formal DPDA



The work also continued on the Final DPDA. The aim was to create a formal DPDA using the key concepts of formal DPDA definition. These efforts were also completed, resulting in the creation of the Final Formal DPDA.

6. Summary

1. What is the general logic followed by the solution?

The machine checks whether the number of '1's in the input binary string is exactly twice the number of '0's (including the empty string). To achieve this, when the machine reads an input character, it writes symbols to the stack that "represent the input characters it must still see to satisfy the condition that the count of '1's is twice the count of '0's." This method is quite simple and widely used.

2. What is the purpose of the optional E transitions marked in green?

The E-transitions marked in green for the left and right states serve the following purposes:

- **Case 1 (For Empty Stack Check):** If the input string is empty, the stack is not empty, and the PDA is in the center state, the current stack contents indicate the characters that need to be appended to the original input string for the count of '1's to be twice the count of '0's.
- **Case 2 (Final Transition to Center State):** If the input string has been completely consumed, and the automaton is **not** in the center state (which also implies that the count of '1's is not equal to twice the count of '0's), these transitions allow a final move to the center state. In this transition, the characters pushed onto the stack serve to store the characters that are needed for the string to finally satisfy the condition (i.e., for the count of '1's to be twice the count of '0's) in the stack itself while in the center state.

3. Why does the search in stack in the left and right states conclude in two steps?

This is a question that would naturally arise. The general logic of the solution is that when an input character arrives, the characters required to see to make language "count of 1s is twice of 0s" are stored on the stack, and they are deleted when a match occurs. However, sometimes stack needs to be searched because there won't be a one-to-one match. Despite this, as seen in the design, when a search operation is required, only the first two characters of the stack are searched; the entire stack is not searched.

The reason for this can be seen when looking at the design. The following rule was adhered to during all stack writing operations: **"If '1' or '0' is present in the set of characters to be written to the stack, it is either the first or the second character."** Since all stack writing operations was performed according to this rule, the stack search operation only examines the first two characters of the stack. In short, looking at the first two characters of the stack is equivalent to searching the entire stack to determine if a '0' or '1' is present.