Korean Automata

202246116 이재헌

목차

1. 과제 개요 및 목적

이 과제는 한글 입력 오토마타를 구현하는 것이 목적입니다. 사용자가 실시간으로 콘솔에 한글(자음, 모음, 숫자)을 입력하면 이를 조합해서 완성형 한글 문자로 만들어주는 프로그램을 만들어야 합니다

한글은 자음과 모음이 결합되어서 하나의 문자를 구성하는 구조적 특성을 가지고 있습니다. 예를 들어서 'ㄱㅗㅇ'을 순서대로 입력하면 '공'으로 하나의 문자로 완성되어야 합니다. 이를 위해서 사용자의 각 입력을 파악해서 적절히 조합해주어야 합니다.

2. 한글 구조와 조합 원리

한글은 자음과 모음이 결합하여 하나의 글자를 이룹니다. 한글의 기본적인 음절은 초성, 중성, 종성으로 이루어져 있습니다. 초성은 음절의 시작 자음을 의미하며, 중성은 음절의 모음을 의미합니다. 종성은 음절의 끝의 받침을 의미합니다. 또한, 종성은 생략이 가능합니다.

컴퓨터는 완성형 문자를 조합하는 것이 아닌, 유니코드 값을 계산하여 표현합니다. 유니코드 시작 값은 0xAC00이며 이를 기준으로 초성, 중성, 종성을 해당 문자의 유니코드를 결정할 수 있습니다.

`한글 유니코드 = 0xAC00 + (초성_index × 21 × 28) + (중성_index × 28) + 종성_index`

- 초성: 19개
- 중성: 21개
- 종성: 28개

한글에서는 두 개의 자음이 결합하여 하나의 받침이 될 수 있습니다. 이를 복자음이라고 합니다. 예를 들어서 ㄱ + ㅅ → ㄱㅅ으로 가능합니다. 이는 오토마타 구현시에 별도로 처리를 해주어야 합니다. 예를 들어서 ㄹ이고 종성인 상태에서 ㅂ이 들어오면 ㄹㅂ을 하나의 종성으로 판단해주어야 합니다.

또한, 두 개의 모음이 결합하여 하나의 모음이 될 수 있는데, 이를 복모음이라고 합니다. 예를 들어서 ㅡ와 ㅣ가 합쳐지면 ㅢ 가 됩니다.

3. 오토마타 설계

이 과제는 Finite State Machine (FSM: 유한 상태 머신) 의 개념을 기반으로 설계되었습니다. FSM은 유한 개의 상태에서 입력에 따라 상태 전이(Transition)을 수행합니다.

FSM 구성 요소 정의

유한 상태 집합

상태 이름	의미
START	아무것도 입력되지 않은 초기 상태
CHO	초성이 입력된 상태
JUNG	중성이 입력된 상태 (초성+중성까지 조합된 상태)
JONG	종성이 입력된 상태 (초성+중성+종성까지 조합된 상태)

입력 집합: 자음, 모음, 기타

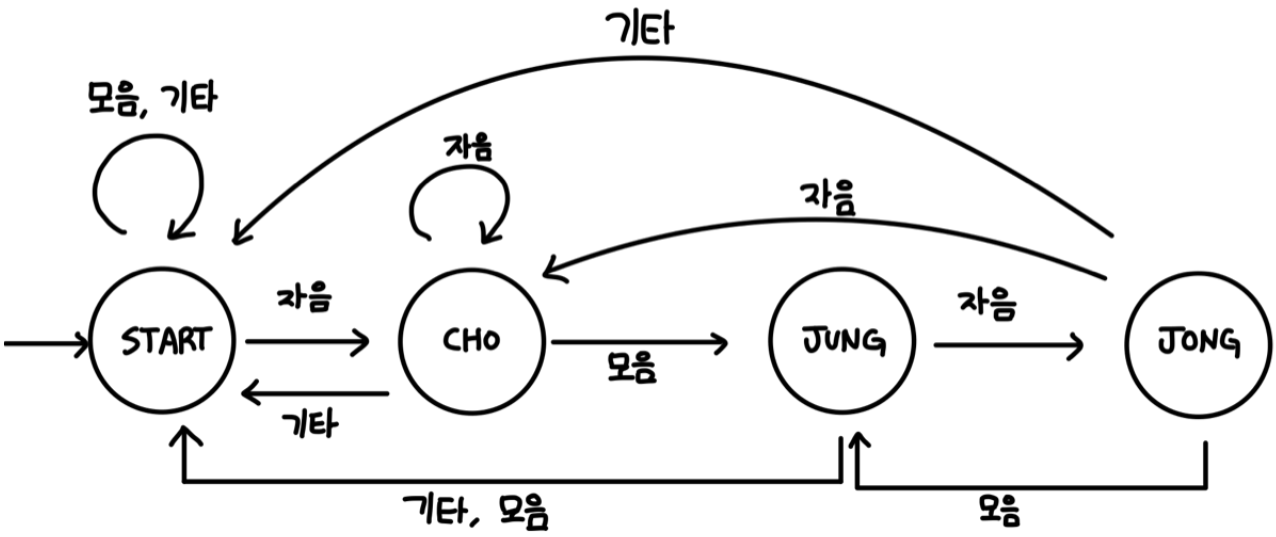
초기 상태: START

상태 전이 함수: 현재 상태와 입력에 따른 다음 상태 결정

현재 상태	입력	조건	다음 상태	동작
START	자음	X	CHO	초성 저장
START	모음	X	START	중성만 저장
START	기타	X	START	결과에 추가
CHO	모음	X	JUNG	중성 저장
CHO	자음	X	CHO	새 초성으로 저장
CHO	기타	X	START	결과에 추가
JUNG	자음	X	JONG	종성 후보 저장
JUNG	모음	X	START	결과에 추가
JUNG	기타	X	START	결과에 추가
JONG	모음	X	JUNG	이전 종성을 초성으로 재사용
JONG	자음	복자음 가능	CHO	buffer 저장 후 이를 새 초성으로 시작
JONG	자음	복자음 불가	CHO	새 초성 시작
JONG	기타	X	START	결과에 추가

최종 상태: Ctrl+C가 입력되기 전까지 무한 루프

Transition Table Diagram



4. 주요 기능 및 코드 설명

HangeulAutomata

이 클래스는 오토마타의 핵심 로직을 담당하며, 입력 처리 및 조합을 수행합니다.

- CHOSUNG, JUNGSUNG, JONGSUNG은 각 초성, 중성, 종성에 해당하는 자모 리스트입니다.
- 초기 상태를 START로 정의해 줬습니다.
- cho, jung, jong은 각각 현재 상태에서의 초성, 중성, 종성이 어떻게 구성되어 있는지를 파악하기 위한 변수입니다.
- buffer는 종성에서 복자음을 처리하기 위한 임시 저장 버퍼입니다.
- result는 결과를 출력하기 위한 문자열입니다.

combine(): 한글 자음 모음을 조합하는 함수입니다.

flush(): 현재까지 저장된 자모(cho, jung, jong)를 조합하여 완성형 문자를 result에 추가하고, 상태를 초기화하는 함수

```
class HangeulAutomata:
  def __init__(self):
    self.CHOSUNG = ['ㄱ', 'ㄲ', 'ㄴ', 'ㄷ', 'ㄸ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅃ', 'ㅅ',
'ㅆ', 'ㅇ',
                    'ㅈ', 'ㅉ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ']

    self.JUNGSUNG = ['ㅏ', 'ㅐ', 'ㅑ', 'ㅒ', 'ㅓ', 'ㅔ', 'ㅕ', 'ㅖ', 'ㅗ',
'ㅘ', 'ㅙ', 'ㅚ',
                     'ㅛ', 'ㅜ', 'ㅝ', 'ㅞ', 'ㅟ', 'ㅠ', 'ㅡ', 'ㅢ', 'ㅣ']

    self.JONGSUNG = [' ', 'ㄱ', 'ㄲ', 'ㄳ', 'ㄴ', 'ㄵ', 'ㄶ', 'ㄷ', 'ㄹ', 'ㄺ',
'ㄻ', 'ㄼ',
                     'ㄽ', 'ㄾ', 'ㄿ', 'ㅀ', 'ㅁ', 'ㅂ', 'ㅄ', 'ㅅ', 'ㅆ',
'ㅇ', 'ㅈ', 'ㅊ',
                     'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ']

    self.COMPLEX_CONSONANTS = {
      'ㄱㅅ': 'ㄳ', 'ㄴㅈ': 'ㄵ', 'ㄴㅎ': 'ㄶ', 'ㄹㄱ': 'ㄺ',
      'ㄹㅁ': 'ㄻ', 'ㄹㅂ': 'ㄼ', 'ㄹㅅ': 'ㄽ', 'ㄹㅌ': 'ㄾ',
      'ㄹㅍ': 'ㄿ', 'ㄹㅎ': 'ㅀ', 'ㅂㅅ': 'ㅄ'
```

```
    }

    # 상태 정의
    self.state = "START"
    self.cho = None
    self.jung = None
    self.jong = None
    self.buffer = None
    self.result = ""
```

process()

HangeulAutomata의 가장 중요한 역할을 차지하며, 문자를 상태별로 처리하는 함수입니다.

1. START 상태일 때

- 자음 → CHO로 전이
- 모음 → 초성 없이 모음만 입력된 것 ⇒ jung에 저장 및 flush()를 통해서 바로 출력
- 기타 → 그대로 결과 문자열에 추가

```
  def process(self, char):
    print(
      f"[입력: {char}] 상태: {self.state}, 초성: {self.cho}, 중성:
{self.jung}, 종성: {self.jong}")
    if self.state == "START":
      if self.is_consonant(char):
        # 자음이 입력되면 초성으로 상태 전이
        self.cho = char
        self.state = "CHO"
      elif self.is_vowel(char):
        # 초성이 없는 모음 단독 입력은 바로 중성으로 처리
        self.jung = char
        self.flush()
      else:
        # 특수 문자라면 그대로 출력
        self.result += char
```

2. CHO 상태일 때

- 자음 → 계속해서 CHO 상태를 유지해야 함 ⇒ 이전의 초성을 flush() 해주고, 새 자음을 초성으로 설정.
- 모음 → JUNG상태로 전이
- 기타 → flush() 해준 뒤 현재 문자를 결과에 추가

```
# 초성은 자음이 들어오면 그대로 저장하고, 모음이 들어오면 중성으로 전이
    elif self.state == "CHO":
      if self.is_vowel(char):
        self.jung = char
        self.state = "JUNG"
      elif self.is_consonant(char):
        self.flush()
        self.cho = char
      else:
        self.flush()
        self.result += char
        self.state = "START"
```

3. JUNG

- 자음 → JONG 상태로 전이
- 모음 → 모음이 다시 들어왔기 때문에 flush()로 상태를 초기화하고 현재 문자를 다시 process() 재귀 호출을 통해서 START 상태에서 처리할 수 있도록 처리
- 기타 → flush() 해준 뒤 현재 문자를 결과에 추가

```
# 중성 처리
   elif self.state == "JUNG":
     if self.is_vowel(char):
       self.flush()
       self.cho = None
       self.jung = None
       self.jong = None
       self.state = "START"
       # 재귀 호출을 통해서 입력된 문자를 새 시작으로 보도록 함
       # 즉, START에서 처리하도록 함
       self.process(char)

     elif self.is_consonant(char):
       # 자음이 들어왔을 때 종성 후보로 처리 가능한지 확인
       self.jong = char
       self.state = "JONG"
     else:
       self.flush()
       self.result += char
       self.state = "START"
```

4. JONG

- 자음

    - 복자음이 가능한 경우 ⇒ 현재 자음을 buffer에 저장한 뒤, 이전 글자를 flush() 하고 뒷 자음을 새로운 초성으로 간주하여 CHO 상태로 전이
    - 복자음이 불가능한 경우 ⇒ 기존 글자를 flush() 해준 뒤 현재 자음을 새로운 초성으로 간주하여 CHO 상태로 전이
- 모음 → 기존 종성이 새로운 글자의 초성이 되고, 현재 모음은 중성으로 전이해야 함. 따라서 flush() 후, JUNG 상태로 전이
- 기타 → flush() 해준 뒤 현재 문자를 결과에 추가

```
    elif self.state == "JONG":
      if self.is_vowel(char):
        # 종성이 있는 상태에서 모음 입력 -> 새로운 글자
        temp = self.jong
        self.jong = None
        self.flush()
        self.cho = temp  # 이전 종성을 새로운 초성으로 사용해서
        self.jung = char  # 현재 들어온 모음을 중성으로 사용해서
        self.state = "JUNG"  # 중성 상태로 전이

      elif self.is_consonant(char):
        combined = self.jong + char

        if combined in self.COMPLEX_CONSONANTS:
          # 복합 자음이 가능한 경우에는 앞과 뒤를 나눠서 처리해야함 (예: ㄹㅂ)
          self.buffer = char
          self.flush()  # 이전 글자를 완성
          self.cho = self.buffer  # 복합 자음의 뒷 부분을 초성으로 설정
          self.state = "CHO"
        else:
          # 복합 자음이 아니면 그냥 새 글자 시작
          self.flush()
          self.cho = char
          self.state = "CHO"

      else:
        self.flush()
        self.result += char
        self.state = "START"
```

5. 예외 및 고려사항

복자음 처리

받침으로 복자음이 들어오는 경우 복자음이 가능해야 합니다

또한 이후에 모음이 들어오는 경우 복자음이 분리되어야 합니다.

예를 들어서 '갉 | '가 아닌 ㄱ, ㅏ, ㄹ, ㄱ, | 가 있다면 '갈기'가 되어야 합니다.

한글의 구조적 특성

초성만 있고 중성이 없는 경우에는 자음만 출력해야 합니다.

중성만 있는 경우 모음만 출력해야 합니다.

중성 다음에 모음이 또 오는 경우에는 새로운 글자로 되어야 합니다.

입력을 처리하기 위한 음절 분리

한국어를 입력할 때 ㄱㅏㅇㅜㅣ 로 순서대로 입력되는 것이 아닌, 가위로 완성되어서 입력이 됩니다

하지만 오토마타는 조합을 해야 하기 때문에 완성된 문자를 분리해주어야 합니다.

예를 들어서 괆을 입력했다면 ㄱ,ㅘ,ㄹㅁ으로 분리가 되어야 합니다. 즉, 초성, 중성, 종성으로 분리되어야 합니다.

유실이 없이 프로그램 종료

중간에 입력하다가 종료(Ctrl+C)가 일어나는 경우

현재 진행 중인 것이 유실되지 않고 마무리되어야 합니다.

6. 어려운 점과 해결 방안

초기 완성형이 아닌 조합형으로 오토마타를 설계

처음에는 한글이 완성형이라고 생각하지 못하고, 하나씩 입력이 된다고 생각하여 CHO, CHO_CHK, JUNG, JUNG_CHK, JONG, JONG_CHK 상태로 세분화시켜 설계를 진행하였습니다.

CHK는 각 상태가 복모음 또는 복자음이 가능한지를 판단하는 하나의 상태라고 설정하였습니다. 각 상태를 다음과 같이 가정하였습니다.

- CHO: 초성 입력된 상태
- CHO_CHK: 자음이 연속 입력되었을 때 초성자리에 들어갈 수 있는지 파악하는 상태
- JUNG: 중성이 입력된 상태
- JUNG_CHK: 모음이 연속으로 입력되어서 복모음으로 결합이 가능한지를 파악하는 상태
- JONG: 종성이 입력된 상태
- JONG_CHK: 자음이 연속으로 입력되어서 겹받침이 가능한지를 파악하는 상태

따라서 다음과 같이 ㄱ,ㄱ,ㅏ,ㄹ,ㄱ이 순서대로 입력된다고 했을 때의 예시입니다.

순서	**문자**	**상태 변화**	**상태 내 저장 값**
1	ㄱ	START → CHO	cho = ㄱ
2	ㄱ	CHO → CHO_CHK → CHO	cho = ㄱㄱ(쌍기억)
3	ㅏ	CHO → JUNG	jung = ㅏ
4	ㄹ	JUNG → JONG	jong = ㄹ
5	ㄱ	JONG → JONG_CHK → JONG	jong = ㄹㄱ (겹받침)

하지만, 각 입력이 자모 단위가 아닌 완성형 한글로 들어온다는 것을 파악하였습니다.

따라서 이를 분해하기 위한 decompose_hangeul() 함수를 만들어서 각 글자를 초성, 중성, 종성으로 분리한 뒤 이를 process() 함수로 넘겨서 처리하였습니다. 이 decompose_hangeul() 함수는 유니코드 계산을 통해서 하나의 완성된 한글 음절을 분해한 뒤 자모 리스트로 반환해 주는 역할을 합니다.

또한, 입력을 분해하였더니, 기존에 정의했던 CHK 상태들이 정의하지 않아도 하나의 입력에서 조건문으로 처리가 가능하고, 그것이 더 효율적이라는 것을 파악하고 CHK 상태를 제거하여 복잡도를 낮추고 간결성을 높였습니다.

실시간 입력과 종료 처리

프로그램은 실시간으로 입력을 받고 이를 처리해야 하기 때문에, 종료(Ctrl+C)를 통해서 처리 중이던 글자가 조합되지 않고 유실되는 문제가 발생했습니다.

따라서 이를 해결하기 위해 종료 직전에 flush()를 수행하는 finalize() 함수를 통해서 마지막까지 완성되지 않은 자모들도 모두 유실되지 않고 처리할 수 있도록 구현하였습니다.

유니코드의 조합에 의한 문제

초기에 설정해 두었던 **CHOSUNG, JUNGSUNG, JONGSUNG** 리스트의 순서를 그냥 마음대로 ㄱㄴㄷㄹㅁㅂㅅ 순으로 정렬해 놓았습니다. 하지만, 다음의 combine()을 하는 부분에서 index를 가져오는 부분이 있는데, 순서가 뒤바뀌어서 입력한 값과 완전히 다른 값이 조합되는 현상이 발생했습니다.

다음 코드에서는 다음과 같이 구성이 되어 있습니다. 즉, 리스트에 어떤 자모가 어떤 순서로 있느냐가 조합 결과에 영향을 미칩니다.

- 초성_index는 CHOSUNG 리스트에서 자음의 순서
- 중성_index는 JUNGSUNG 리스트에서 모음의 순서
- 종성_index는 JONGSUNG 리스트에서 받침의 순서

```
def combine(self):
  if self.cho is None or self.cho not in self.CHOSUNG:
    return ""

    # 각각 초성, 중성, 종성의 인덱스를 구한다
  cho_idx = self.CHOSUNG.index(self.cho)
  jung_idx = self.JUNGSUNG.index(self.jung) if self.jung else 0
  jong_idx = self.JONGSUNG.index(self.jong) if self.jong else 0

  # 완성형 한글 유니코드 = 0xAC00 + (초성 * 21 * 28) + (중성 * 28) + 종성
  return chr(0xAC00 + (cho_idx * 21 * 28) + (jung_idx * 28) + jong_idx)
```

따라서 초기의 CHOSUNG, JUNGSUNG, JONGSUNG 리스트를 유니코드에 맞춰서 수정해주었습니다.

7. 결과 및 분석

입력: ㄱ,ㅗ,ㅏ,ㄹ,ㄱ,ㅜ,ㅑ

예상: 괄구ㅑ

결과: 괄구ㅑ

```
Korean Automata 시작! (종료: Ctrl+C)
[입력: ㄱ] 상태: START, 초성: None, 중성: None, 종성: None
[입력: ㅘ] 상태: CHO, 초성: ㄱ, 중성: None, 종성: None
[입력: ㄹ] 상태: JUNG, 초성: ㄱ, 중성: ㅘ, 종성: None
[입력: ㄱ] 상태: JONG, 초성: ㄱ, 중성: ㅘ, 종성: ㄹ
[입력: ㅜ] 상태: CHO, 초성: ㄱ, 중성: None, 종성: None
[입력: ㅑ] 상태: JUNG, 초성: ㄱ, 중성: ㅜ, 종성: None
[입력: ㅑ] 상태: START, 초성: None, 중성: None, 종성: None

프로그램을 종료합니다.
최종 결과: 괄구ㅑ
```

분석

START → ㄱ → CHO: START 상태에서 자음(ㄱ)이 들어왔기 때문에 CHO 상태로 전이

CHO → ㅘ → JUNG: CHO 상태에서 모음(ㅘ)이 들어왔기 때문에 JUNG 상태로 전이

JUNG → ㄹ → JONG: JUNG 상태에서 자음(ㄹ)이 들어왔기 때문에 JONG 상태로 전이

JONG → ㄱ → CHO: JONG 상태에서 자음(ㄱ)이 연속해서 들어왔기 때문에 겹받침이 가능한지를 판단해야함. ㄹㄱ은 겹받침이 가능하기 때문에 ㄱ을 cho 변수에 저장하고, CHO 상태로 전이함

CHO → ㅜ → JUNG: CHO 상태에서 모음(ㅜ)가 들어왔기 때문에 바로 JUNG 상태로 이전

JUNG → ㅑ → START (process()): JUNG 상태에서 모음(ㅑ)가 연속해서 들어왔기 때문에 flush()를 진행해주고, START 상태로 다시 처음부터 시작해야 함. 따라서 ㅑ를 process()를 통해서 재귀 호출을 시켜서 START 상태부터 재시작 하도록 함.

JUNG → ㅑ: 재귀 호출의 결과로 콘솔창에 한 번 찍혔음.

입력: ㅇ,ㅗ,ㅐ,ㅐ,ㅐ

예상: 왜ㅐㅐ

결과: 왜ㅐㅐ

```
Korean Automata 시작! (종료: Ctrl+C)
[입력: ㅇ] 상태: START, 초성: None, 중성: None, 종성: None
[입력: ㅐ] 상태: CHO, 초성: ㅇ, 중성: None, 종성: None
[입력: ㅐ] 상태: JUNG, 초성: ㅇ, 중성: ㅙ, 종성: None
[입력: ㅐ] 상태: START, 초성: None, 중성: None, 종성: None
[입력: ㅐ] 상태: START, 초성: None, 중성: None, 종성: None

프로그램을 종료합니다.
최종 결과: 왜ㅐㅐ
```

분석

START → ㅇ → CHO: START 상태에서 자음(ㅇ)이 들어왔기 때문에 CHO 상태로 전이

CHO → ㅐ → JUNG: CHO 상태에서 모음(ㅐ)이 들어왔기 때문에 JUNG 상태로 전이

JUNG → ㅐ → START(process()): JUNG 상태에서 이전에 이어서 모음(ㅐ)가 연속해서 들어왔기 때문에 초기 상태로 돌아가야 함. 따라서 flush()를 해준뒤, process() 재귀 호출을 진행하여 START 상태부터 시작하도록 함.

ㅐ → 재귀 호출로 인한 콘솔창에 출력

START → ㅐ → START: START 상태에서 모음이 들어왔기 때문에 START 상태를 유지

입력: ㄱ,ㅏ,ㅁ,ㅏ

예상: 가마

결과: 가마

```
Korean Automata 시작! (종료: Ctrl+C)
[입력: ㄱ] 상태: START, 초성: None, 중성: None, 종성: None
[입력: ㅏ] 상태: CHO, 초성: ㄱ, 중성: None, 종성: None
[입력: ㅁ] 상태: JUNG, 초성: ㄱ, 중성: ㅏ, 종성: None
[입력: ㅏ] 상태: JONG, 초성: ㄱ, 중성: ㅏ, 종성: ㅁ

프로그램을 종료합니다.
최종 결과: 가마
```

분석

START → ㄱ → CHO: START 상태에서 자음(ㄱ)이 들어왔기 때문에 CHO 상태로 전이

CHO → ㅏ → JUNG: CHO 상태에서 모음(ㅏ)이 들어왔기 때문에 JUNG 상태로 전이

JUNG → ㅁ → JONG: JUNG 상태에서 자음(ㅁ)이 들어왔기 때문에 JONG 상태로 전이

JONG → ㅏ → JUNG: JONG 상태에서 모음(ㅏ)가 들어왔기 때문에 현재 종성을 cho 변수에 넣어서 초성으로 바꾼 뒤 입력값이 ㅏ를 jung에 설정해준 뒤 JUNG 상태로 전이

입력: ㄱ,ㅏ,ㅁ,1,ㅏ

예상: 감1ㅏ

결과: 감1ㅏ

```
Korean Automata 시작! (종료: Ctrl+C)
[입력: ㄱ] 상태: START, 초성: None, 중성: None, 종성: None
[입력: ㅏ] 상태: CHO, 초성: ㄱ, 중성: None, 종성: None
[입력: ㅁ] 상태: JUNG, 초성: ㄱ, 중성: ㅏ, 종성: None
[입력: 1] 상태: JONG, 초성: ㄱ, 중성: ㅏ, 종성: ㅁ
[입력: ㅏ] 상태: START, 초성: None, 중성: None, 종성: None

프로그램을 종료합니다.
최종 결과: 감1ㅏ
```

분석

START → ㄱ → CHO: START 상태에서 자음(ㄱ)이 들어왔기 때문에 CHO 상태로 전이

CHO ->ㅏ → JUNG: CHO 상태에서 모음(ㅏ)이 들어왔기 때문에 JUNG 상태로 전이

JUNG → ㅁ → JONG: JUNG 상태에서 자음(ㅁ)이 들어왔기 때문에 JONG 상태로 전이

JONG → 1 → START: 숫자가 들어왔기 때문에 START 상태로 전이

START → ㅏ → START: START 상태에서 모음(ㅏ)이 들어왔기 때문에 다시 START 상태를 유지

입력: ㅂ,ㅜ,ㅔ,ㄹ,ㅂ

예상: 붸

결과: 뷃

```
Korean Automata 시작! (종료: Ctrl+C)
[입력: ㅂ] 상태: START, 초성: None, 중성: None, 종성: None
[입력: ㅞ] 상태: CHO, 초성: ㅂ, 중성: None, 종성: None
[입력: ㅃ] 상태: JUNG, 초성: ㅂ, 중성: ㅞ, 종성: None

프로그램을 종료합니다.
최종 결과: 뷃
```

분석

START → ㅂ → CHO: START에서 자음(ㅂ)이 들어왔기 때문에 CHO 상태로 전이

CHO → ㅞ → JUNG: CHO 상태에서 모음(ㅞ)이 들어왔기 때문에 JUNG 상태로 전이

JUNG → ㄹㅂ → JONG: JUNG 상태에서 자음(ㄹㅂ)이 들어왔기 때문에 JONG 상태로 전이

8. 결론

이번 과제를 통해서 한글의 구조적 특성 및 조합원리 그리고 컴퓨터에서 어떻게 한국어가 입력처리가 되는지를 Finite State Machine을 활용하여 이해할 수 있었습니다. 초성, 중성, 종성의 간단한 조합뿐만 아니라 복모음과 복자음에 관해서 고민해 보면서 다양한 상황들을 고려하며 상태들을 설계하였습니다.

처음에는 컴퓨터에서 처리할 때 완성형으로 처리한다는 것을 인지하지 못해서 상태가 지나치게 복잡했지만, 수많은 디버깅 작업을 통해 상태를 단순화하고 한글을 음절로 분해하여 효과적으로 해결할 수 있었습니다.

이번 구현을 통해서 지금껏 한 번도 해보지 못했던 한국어를 어떻게 입력하는지에 관해서 고민해 볼 수 있었고, 한국어 입력기를 만들 수 있을 것이라고는 상상하지 않았지만, 이를 통해서 한글에 대해서 더욱 깊이 있는 이해를 할 수 있었습니다. 또한, 유니코드와 한국어가 컴퓨터 내에서 어떻게 표현되는지에 대해서 알 수 있어 값진 시간이었습니다.

추후에는 한글 외의 다른 언어에 대한 오토마타도 만들어보고 싶고, 백스페이스 기능 등 다양한 기능을 추가하고 GUI를 추가하여 시각적인 입-출력이 가능하도록 발전시키고 싶습니다.