



Akademia Tarnowska w Tarnowie
Instytut Politechniczny

Informatyka: Inżynieria Oprogramowania

Narzędzie wykorzystujące kwadraty logiczne w celu wygenerowania maszyny stanów

Autorzy:

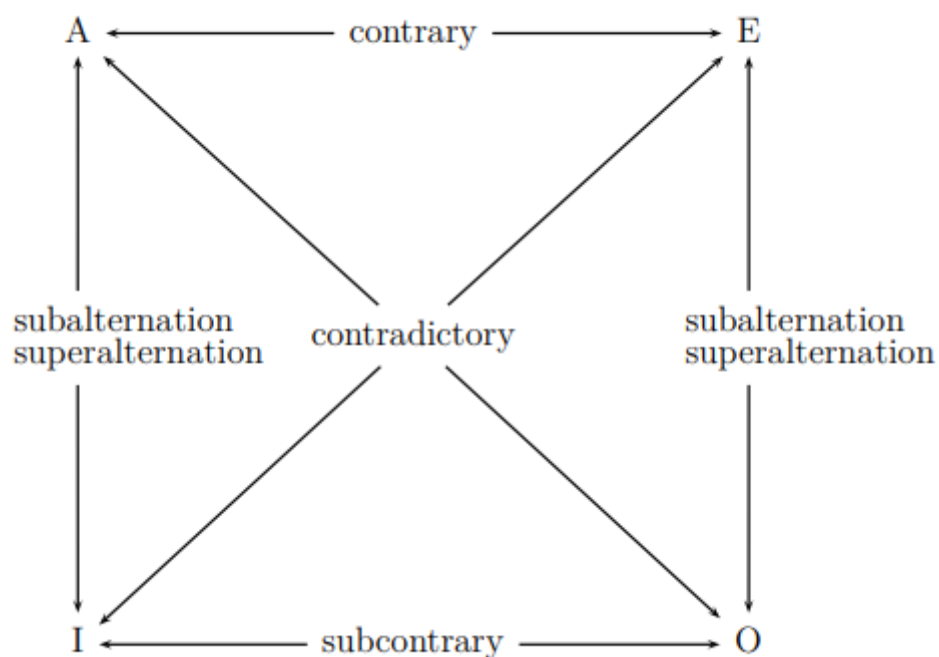
Grzegorz Urban
Dawid Kusion

1.Spis treści

1. Kwadrat logiczny...
2. Maszyna stanów...
3. Sformułowanie zadania projektowego...
4. Algorytm...
5. Opis działania programu...
6. Implementacja w języku programowania...
7. Przykład użycia...
8. Podsumowanie...

1.Kwadrat logiczny

Kwadrat logiczny jest to koncept przedstawienia relacji między zdaniami logicznymi. Można go przedstawić za pomocą grafu:



Zdania te dzieli się ze względu na “ilość” (**quantity**) lub “jakość” (**quality**) t.j. kolejno wszystkie lub niektóre oraz twierdzenie lub przeczenie. Zachodzą między nimi odpowiednie relacje:

Implikacja (podporządkowanie):

Ze zdania **ogólnie twierdzącego** (**A**) wynika zdanie **szczegółowo twierdzące** (**I**) gdzie rozważając dla S (prawda/fałsz):

Zdanie **ogólnie twierdzące** „Każde S jest P”

Zdanie **szczegółowo twierdzące** „Niektóre S są P”

Ze zdania **ogólnie przeczącego** (**E**) wynika zdanie **szczegółowo przeczące** (**O**) gdzie rozważając dla S (prawda/fałsz):

Zdanie **ogólnie przeczące** „Żadne S nie jest P”

Zdanie **szczegółowo przeczące** „Niektóre S nie są P”

Przeciwnieństwo:

Zachodzi dla zdań odpowiednio **A** i **E** i wynika z tego że jeżeli jedno zdanie jest prawdziwe to drugie jest fałszywe lub oba są fałszywe.

Sprzeczne:

Zdania te nie mogą być jednocześnie prawdziwe ani fałszywe. Sprzeczność zachodzi między zdaniem **ogólno-twierdzącym (A)** a **szczegółowo-przeczącym (O)** oraz między **ogólno-przeczącym (E)** a **szczegółowo-twierdzącym (I)**. Oznacza to że jeżeli “każde S jest P” to nie może zachodzić “niektóre S nie są P”

Podprzeciwieństwo:

Zdania podprzeciwne to zdania **I** oraz **O**. Nie mogą one być jednocześnie fałszywe natomiast mogą być przeciwne lub prawdziwe jednocześnie np. “Niektóre S są P ale również niektóre S nie są P”.

Kwadrat logiczny może być wykorzystywany do identyfikowania stanów oraz projektowania maszyny stanów będącej podstawowym elementem modelu behawioralnego i do łatwiejszego ich rozumienia. Wyróżniamy trzy podstawowe operacje na stanach w odniesieniu do kwadratu logicznego:

ekstrakcję, ekspansję i dekompozycję. Ekstrakcja jest procesem wyodrębniania stanów dla maszyny stanowej z kwadratu logicznego jako źródła podstawowych relacji. Rozszerzanie to proces zwiększania rozmiaru rozważanej maszyny stanów poprzez zastąpienie wybranego stanu nowymi stanami (maksymalnie trzema). Dekompozycja to proces rozbijania pojedynczego stanu na oddzielny automat stanów.

Tylko trzy przypisania spełniają wszystkie powyższe warunki: $\{A, \neg E, I, \neg O\}$, $\{\neg A, E, \neg I, O\}$ oraz $\{\neg A, \neg E, I, O\}$. Rozważymy zdania, które w danym momencie są dopuszczalne, a powyższe zbiory zostaną uproszczone do następujących trzech przypadków: $\{A, I\}$, $\{E, O\}$ oraz $\{I, O\}$. Identyfikują one możliwe stany, jako połączone narożniki przyszłej maszyny stanów.

2. Maszyna stanów

Maszyna stanów, znana również jako automat skończony lub automaton skończony, to abstrakcyjny model matematyczny, który opisuje system, który może znajdować się w jednym z ograniczonej liczby stanów w danym momencie, oraz którego zachowanie jest determinowane przez przejścia pomiędzy tymi stanami. Maszyna stanów składa się z zestawu stanów, zbioru zdarzeń (sygnałów wejściowych) oraz funkcji przejścia, która określa, jakie działania są podejmowane w odpowiedzi na dane zdarzenie w danym stanie.

Podstawowe elementy maszyny stanów to:

- **Stany (States):** Ograniczony zestaw określonych warunków, w jakich może znajdować się system w danym momencie.
- **Wyzwalacze (Triggers):** Sygnały wejściowe, które mogą wpływać na stan systemu, wywołując przejście do innego stanu.
- **Funkcja Przejścia (Transition Function):** Określa, jakie akcje lub zmiany stanu zachodzą w odpowiedzi na dane zdarzenie w konkretnym stanie.
- **Stan Początkowy (Initial State):** Stan, w którym system znajduje się na początku działania.
- **Stan Końcowy (Final State):** Ostateczny stan, w którym system osiąga zamierzony cel.

Maszyny stanów są używane w różnych dziedzinach, takich jak informatyka, telekomunikacja, automatyka, a także w modelowaniu procesów i systemów. Ich zastosowanie obejmuje projektowanie interfejsów użytkownika, sterowanie systemami, analizę protokołów komunikacyjnych oraz modelowanie procesów biznesowych. W dokumentacji maszyny stanów istotne jest uwzględnienie szczegółów dotyczących poszczególnych stanów, zdarzeń, oraz przejść pomiędzy nimi, aby umożliwić zrozumienie i implementację tego modelu w praktyce.

3. Sformułowanie zadania projektowego

Zadaniem projektu jest utworzenie narzędzia które pozwoli użytkownikowi na przekształcenie wprowadzonych przez niego kwadratów logicznych będących reprezentacją stanów w procesie, którego opis i zautomatyzowanie interesuje użytkownika. Użytkownik korzystając z narzędzia powinien móc:

1. Zgodnie z konwencją kwadratu logicznego wprowadzić rogi kwadratów zawierających stany procesu który chce opisać
2. Rozszerzać pojedynczy kwadrat w celu dodania nowej puli stanów
3. Przedstawione powinno zostać drzewo rozpinające reprezentujące ów kwadraty logiczne w postaci stanów oraz ich ewentualne rozszerzenia.
4. Użytkownik powinien, po wykonaniu przez narzędzie drzewa rozpinającego, zdefiniować przejścia między stanami wraz z wyzwalaczami powodującymi to przejście
5. Użytkownik powinien posiadać również możliwość zdefiniowania zmiennych dla danego stanu w celu późniejszego ich użycia we właściwym kodzie maszyny stanów
6. Narzędzie tworzyć będzie z wprowadzonych danych maszynę stanów i przedstawia ją na grafie skierowanym
7. Program generuje również schemat pseudokodu dla danej maszyny stanów zawierający przejścia oraz zmienne

Po przejściu przez wszystkie etapy projektowania maszyny stanów za pomocą przedstawionego narzędzia użytkownik powinien dostać do swojej dyspozycji pseudokod pozwalający mu na zdefiniowanie wszelkich funkcji i działania dla danej maszyny stanów.

4. Algorytm

Algorytm przyjmujący na wejściu kwadrat logiczny a generujący maszynę stanów oraz trójskładnikowe drzewo rozpinające wygląda następująco:

Input: logic square

Output: stateMachine, spanTree

1. stateMachine = null;
2. spanTree = "0";
3. **while** stateMachine not finished **do**
4. take a leaf to expand:
5. A := establishSentence(A);
6. E := 0, O := 0; I := 0;
7. E := establishSentence2Rel(A,contrary);
8. O:=establishSentence2Rel(A,contradictory);
9. I;+establishSentence2Rel(A,subalternate);
10. newStates := {{A,I},{E,O},{I,O}}
11. spanTree := expandTree(newStates);
12. **end while**
13. spanTree2stateMachine(spanTree, stateMachine);
14. createTrasition(stateMachine);
15. generateEvents(stateMachine);

W powyższym algorytmie każde wykonanie ciała pętli jest pojedynczym procesem ekstrakcji, a każda iteracja pętli jest pojedynczym procesem ekspansji.

5. Opis działania programu

Pod uruchomieniem narzędzia użytkownikowi pokazuje się okno w którym dostaje instrukcje postępowania oraz możliwość wprowadzenia kwadratów logicznych i ich rozszerzania.

User Instructions:


1. Enter values for the A, E, I, O categories in the respective fields.
2. Click the (+) button next to the corresponding category to add a new logic square.
3. Repeat steps 1-2 for each category as needed.
4. Click the 'Confirm' button to confirm the user-entered data.
5. Add variables to the states by clicking on nodes on the span tree graph

Logic Square 1

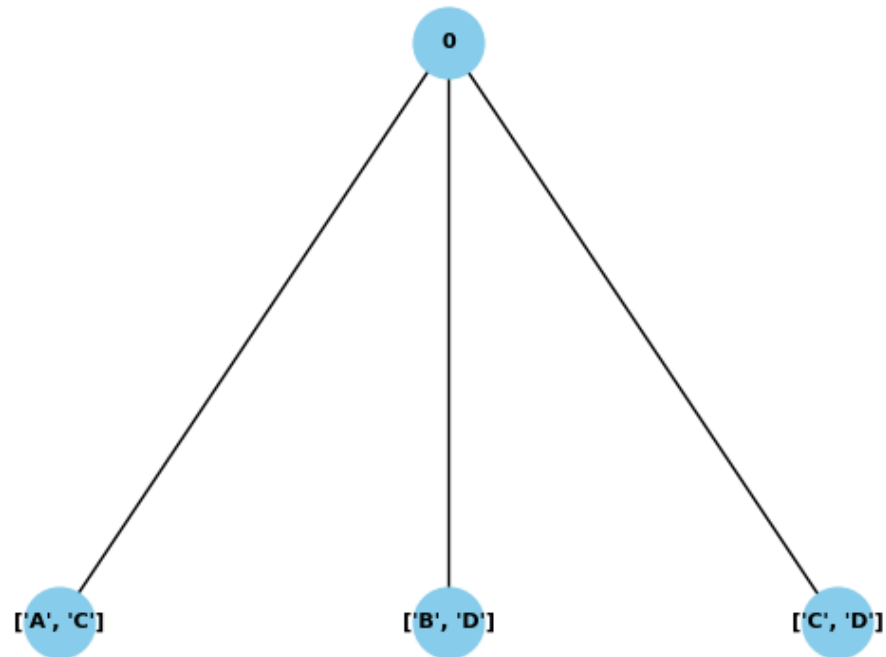
<input type="text"/>	<-->	<input type="text"/>
<input data-bbox="453 947 603 994" type="button" value="+"/>		<input data-bbox="951 947 1101 994" type="button" value="+"/>
<input type="text"/>	<input data-bbox="702 1084 852 1131" type="button" value="+"/>	<input type="text"/>

Kolejnym Etapem po wprowadzeniu kwadratów logicznych jest potwierdzenie zmian przyciskiem **“Confirm”** lub restartowanie programu do stanu początkowego przyciskiem **“Restart”**

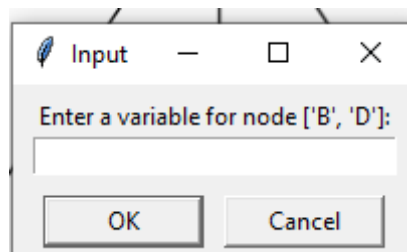
Następnie użytkownikowi zostaje przedstawione drzewo rozpinające wygenerowane zgodnie z konwencją przekształcania kwadratów na stany

 Graph

— □ ×



Po wciśnięciu odpowiedniego stanu na grafie drzewa użytkownik proszony jest o wprowadzenie ewentualnej zmiennej dla danego stanu i może to robić wielokrotnie (nie jest to konieczne).



Input — □ ×

Enter a variable for node ['B', 'D']:

OK Cancel

Kolejnym krokiem jest wprowadzenie przejść między stanami oraz odpowiednich wyzwalaczy które spowodują zmianę stanu

Enter transition from ['A', 'C'] to ['B', 'D']:

Trigger1

Enter transition from ['A', 'C'] to ['C', 'D']:

Enter transition from ['B', 'D'] to ['A', 'C']:

Enter transition from ['B', 'D'] to ['C', 'D']:

Trigger2

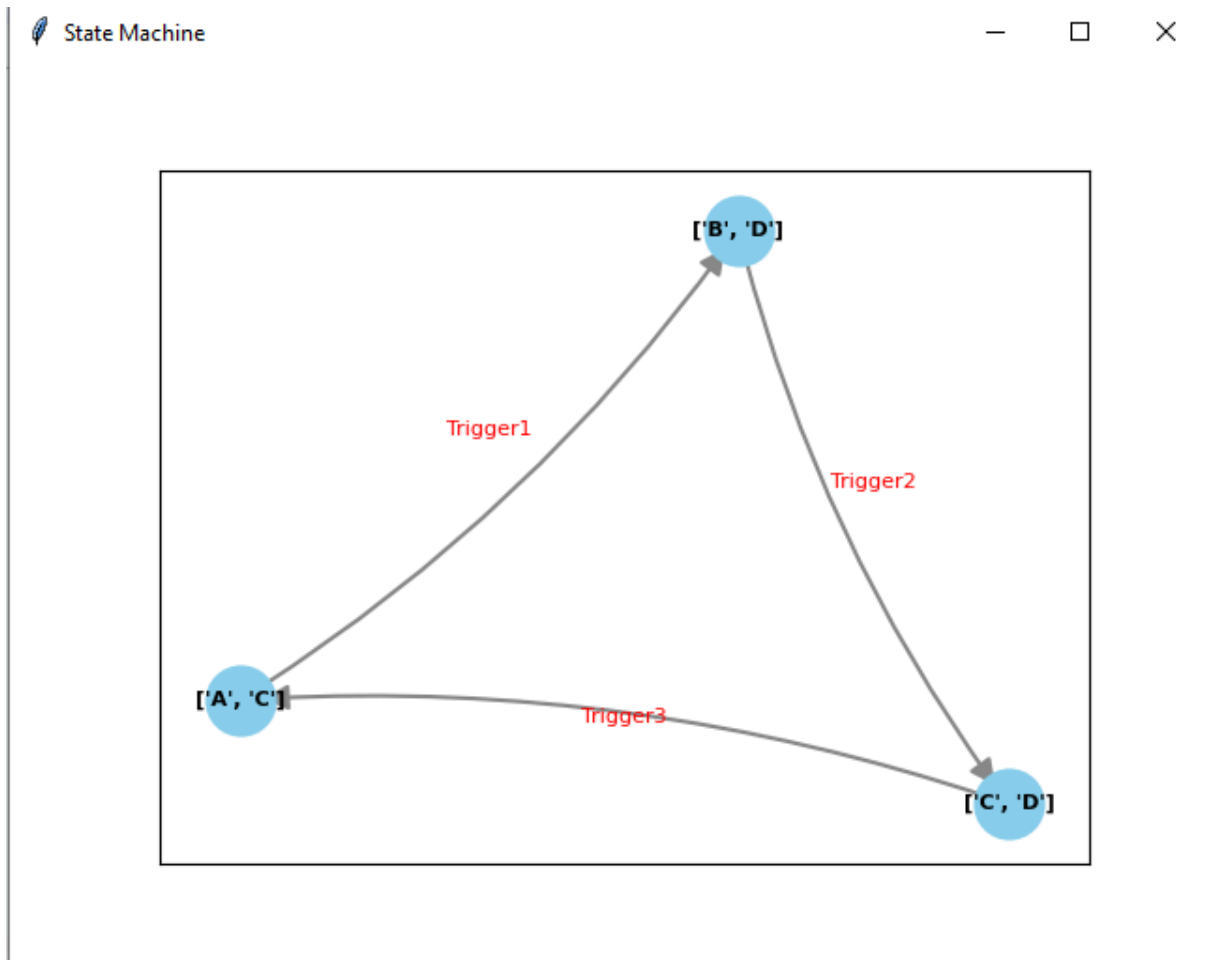
Enter transition from ['C', 'D'] to ['A', 'C']:

Trigger3

Enter transition from ['C', 'D'] to ['B', 'D']:

OK

Po naciśnięciu przycisku “**OK**” pokazany zostanie graf maszyny stanów ze zdefiniowanymi przejściami



oraz wygenerowany zostanie szablon kodu oparty na języku python, niebędący jednak w pełni funkcjonalnym i kompilowalnym kodem. Kod może zostać wykorzystany w celu wprowadzenia wszelkich działań które powinny się znaleźć w konkretnych stanach.

```
2 #Presented code is just a pseudocode based on python programming language, cannot be compiled without
3 def state_machine(start_state):
4     current_state = start_state
5     condition = true
6     while condition:
7         trigger = Null
8         if current_state == ['A', 'C']:
9             #code to handle trigger and variables
10            if trigger == Trigger1:
11                current_state = ['B', 'D']
12                continue
13            else:
14                print("Missing or invalid trigger")
15                break
16        if current_state == ['B', 'D']:
17            #Variables:
18            Variable1 = Null
19            Variable3 = Null
20            #code to handle trigger and variables
21            if trigger == Trigger2:
22                current_state = ['C', 'D']
23                continue
24            else:
25                print("Missing or invalid trigger")
26                break
27        if current_state == ['C', 'D']:
```

6. Implementacja w języku programowania

Narzędzia napisane zostało z użyciem języka programowania **python**, wykorzystuje ono biblioteki dla tego języka dostarczone przez strony trzecie. Są to:

- Biblioteka **Tkinter** - pozwalająca na utworzenie aplikacji okienkowej i wszelkich funkcjonalności ułatwiających korzystanie z programu
- Biblioteka **matplotlib** - służąca do tworzenia wykresów i wizualizacji
- Biblioteka **networkx** - służąca do analizy i wizualizacji grafów

Zdefiniowane zostały 3 klasy, odpowiednio klasa drzewa rozpinającego, kwadratu logicznego oraz główna klasa aplikacji

```
#Spanning tree class
class SpanTree:...

#Logic square class
class Square:...

#Main app class
class LogicSquareApp:...
```

Określenie zmiennych typu self dla klasy aplikacji

```
def __init__(self, root):
    self.root = root
    self.root.title("Logic Square")
    self.root.geometry("1600x1200")

    #Widgets of the app window
    self.LP = 1
    self.squares = []
    self.buttons = []
    self.entries = []
    self.expands = []
    self.create_widgets()
    self.add_extract_button(5)
    self.add_restart_button(7)

    #Dictionary of triggers assigned to certain transitions
    self.transitions_dict = {}

    #If there are any, variable for each state in the machine
    self.variables = {}
```

Funkcje odpowiedzialne za generowanie strony wizualnej aplikacji

```

def create_widgets(self, row_index=2):...

def add_extract_button(self, row_index):...

def add_restart_button(self, row_index):...

def restart_program(self):...

def add_logic_square(self, row_index, entry1, entry2, entry3, entry4, buttonID):...
def print_squares_info(self):...

def confirm(self):...

```

Główna funkcja implementująca algorytm

```

def extract_and_expand(self):
    #Initialize span tree with root '0'
    span_tree = SpanTree("0")
    #Working copy of node to access span tree
    leaf = span_tree

    #Loop for adding node to span tre
    for s in self.squares:
        leaf = self.select_leaf_to_expand(leaf)
        print(str(leaf.value))
        #New states using next logic square
        A = s.A
        E = s.E
        O = s.O
        I = s.I

        #Updating tree
        newStates = [SpanTree([A, I]), SpanTree([E, O]), SpanTree([I, O])]
        leaf.expand(newStates)

    #Drawing tree graph
    self.draw_tree(span_tree)
    #Generating and drawing state machine using spanning tree
    self.span_tree_to_directed_graph(span_tree)

```

Funkcje odpowiedzialne za wyrysowanie grafu drzewa i obsługi dodawania zmiennych dla stanów

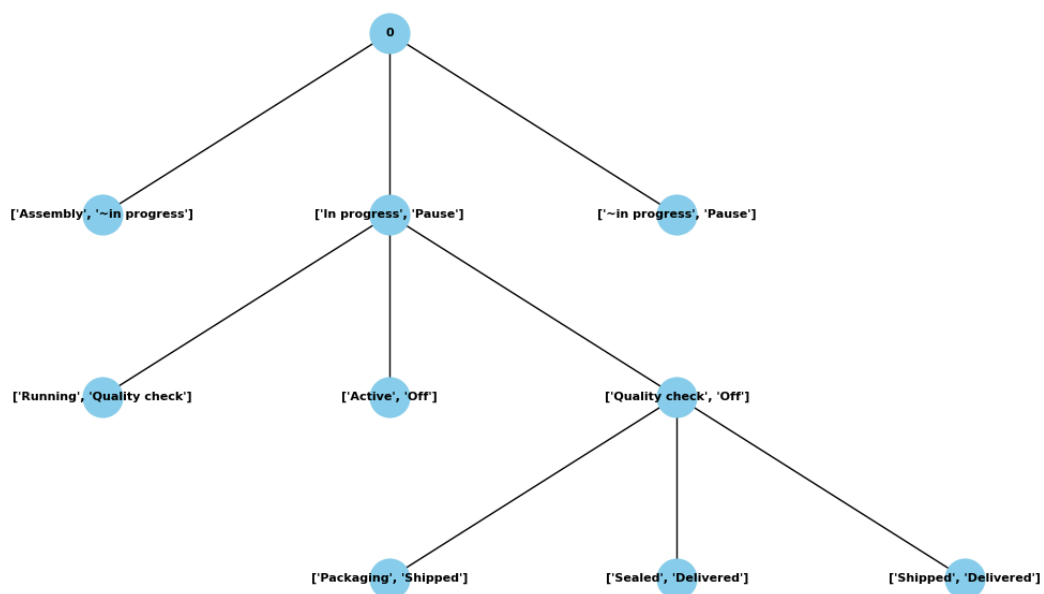
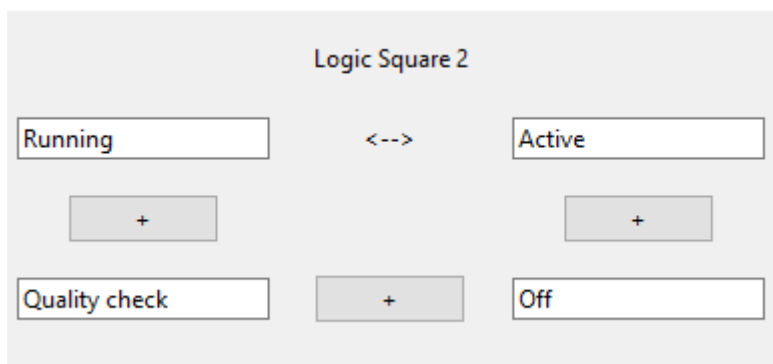
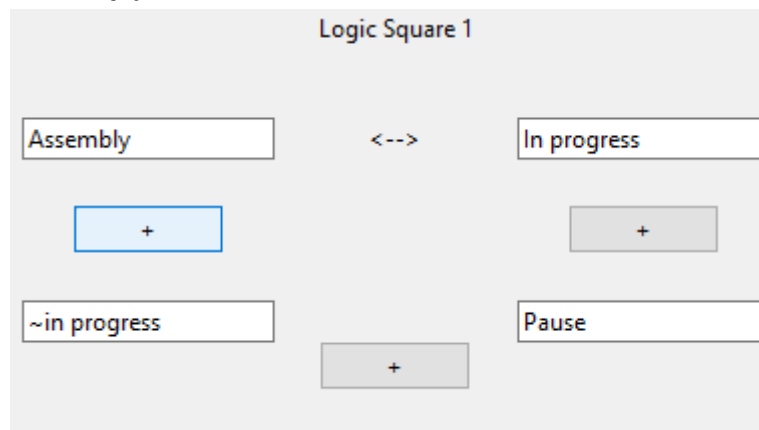
```
#Tree  
def draw_tree(self, span_tree):...  
  
def on_node_click(self, event):...  
  
def draw_graph_in_tkinter(self, graph, pos):...
```

Generowanie maszyny stanów oraz generowanie kodu

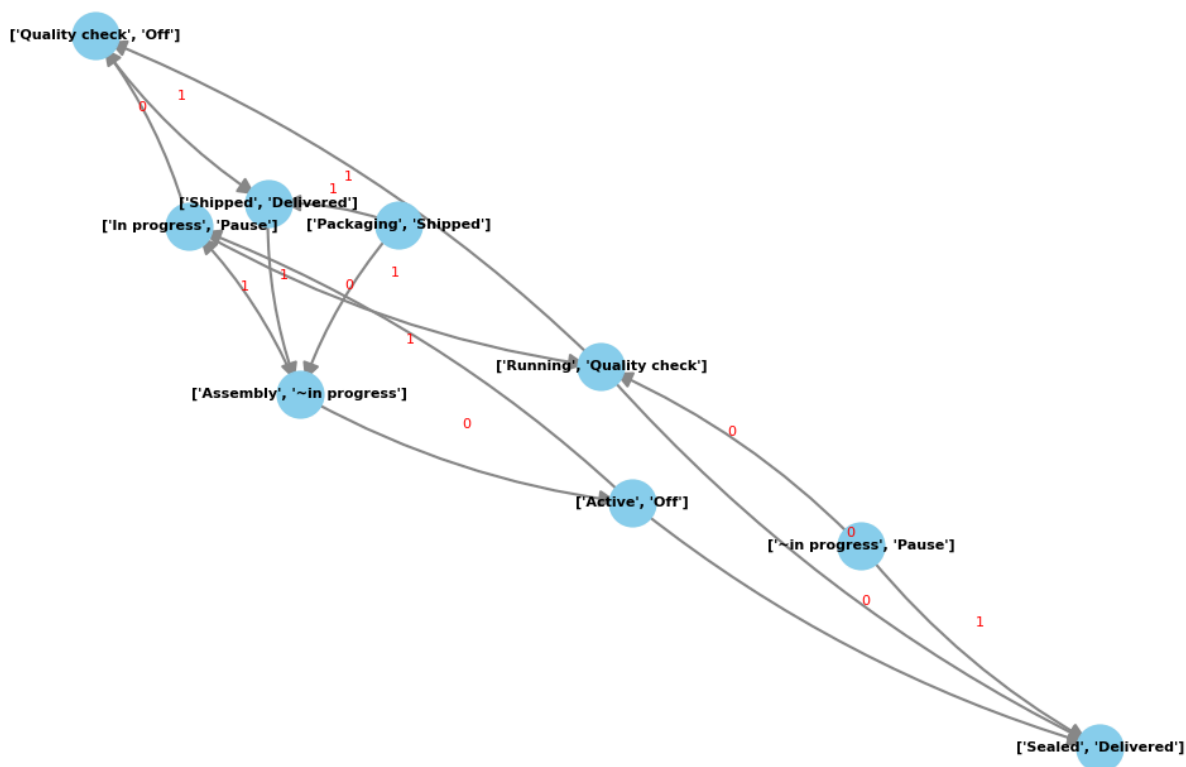
```
#Graph  
def span_tree_to_directed_graph(self, span_tree):...  
  
def collect_leaf_nodes(self, node, graph):...  
  
def add_user_transitions(self, directed_graph):...  
  
def draw_directed_graph(self, directed_graph):...  
  
def print_variables(self):...  
  
def generate_code(self, directed_graph):...
```

7. Przykład

Zarządzanie produkcją samochodów



Dodanie przejść między stanami



8. Podsumowanie

Narzędzie zostało zaprojektowane zgodnie z założeniami projektowymi. Pozwala więc w łatwy i przystępny dla użytkownika sposób konstruowanie maszyny stanów posiadając wiedzę o swoich potrzebach i umiejętność stworzenia z danych właściwego kwadratu logicznego zachowującego założenia z pkt 1.