

POLITECHNIKA WARSZAWSKA
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH



REPREZENTACJA WIEDZY

Programy działań z efektami
domyślnymi

Raport końcowy

Autorzy:

Dragan Łukasz
Flis Mateusz
Izert Piotr
Pielat Mateusz
Rząd Przemysław
Siry Roman
Waszkiewicz Piotr
Zawadzka Anna

20 czerwca 2016

Spis treści

1	Opis zadania	2
2	Opis klas	3
2.1	Klasy fluentów, aktorów i akcji	3
2.2	Klasy zdań logicznych	3
2.3	Klasy zdań	3
2.4	Klasa dziedziny	4
2.5	Klasa stanu	4
2.6	Klasa Graph	4
2.7	Klasa Edge	4
2.8	Klasa World	5
2.9	Klasa kroku scenariusza	5
2.10	Klasa scenariusza	5
2.11	Klasy kwerend	5
3	Algorytmy	6
3.1	Wyznaczanie zbioru stanów	6
3.2	Wyznaczanie stanu początkowego	7
3.3	Obliczanie zbiorów ResN i ResAb	8
3.4	Konstrukcja grafu zależności	9
3.5	Obliczanie odpowiedzi na zadaną kwerendę	10
4	Moduły dodatkowe	19
4.1	Biblioteka Microsoft Automatic Graph Layout	19
5	Podział prac	20

1 Opis zadania

Zadaniem projektu jest opracowanie i zaimplementowanie języka akcji dla specyfikacji podanej klasy systemów dynamicznych oraz odpowiadający mu język kwerend.

System dynamiczny spełnia podane założenia:

1. Prawo inercji
2. Niedeterminizm i sekwencyjność działań
3. Pełna informacja o wszystkich akcjach i wszystkich ich skutkach bezpośrednich
4. Z każdą akcją związany jest:
 - (a) Warunek początkowy (ew. true)
 - (b) Efekt akcji
 - (c) Jej wykonawca
5. Skutki akcji:
 - (a) Pewne (zawsze występują po zakończeniu akcji)
 - (b) Domyślne (preferowane. Zachodzą po zakończeniu akcji, o ile nie jest wiadomym, że nie występują)
6. Efekty akcji zależą od jej stanu, w którym akcja się zaczyna i wykonawcy tej akcji
7. W pewnych stanach akcje mogą być niewykonalne przez pewnych (wszystkich) wykonawców

Programem działań nazywać będziemy ciąg $((A_1, w_1), (A_2, w_2), \dots, (A_n, w_n))$, gdzie A_i jest akcją, zaś w_i jej wykonawcą, $n = 0, 1, 2, \dots$ gdzie $w_i = \epsilon$ oznacza dowolnego wykonawcę.

Język kwerend zapewnia uzyskanie odpowiedzi na następujące pytania:

1. Czy podany program działań jest wykonywalny zawsze/kiedykolwiek?
2. Czy wykonanie podanego programu działań z dowolnego stanu spełniającego warunek π prowadzi zawsze/kiedykolwiek/na ogół do stanu spełniającego warunek celu γ ?
3. Czy z dowolnego stanu spełniającego warunek π cel γ jest osiągalny zawsze/kiedykolwiek/na ogół?
4. Czy wskazany wykonawca jest zaangażowany w realizację programu zawsze/kiedykolwiek?
5. Czy po wykonaniu podanego programu działań zawsze/kiedykolwiek/na ogół formuła γ jest spełniona w stanie wynikowym?

2 Opis klas

Przygotowany program, realizujący postawiony cel, został napisany w języku programowania C# i wykonany w technologii Windows Forms. Składa się on z szeregu klas będących odzwierciedleniem pojęć występujących w teorii reprezentacji wiedzy, mających na celu łatwiejsze napisanie i zrozumienie sposobu działania programu.

2.1 Klasy fluentów, aktorów i akcji

Klasy te służą do identyfikowania poszczególnych elementów występujących w dziedzinie wprowadzonej przez użytkownika. Rozróżnialne są za pomocą unikalnych nazw nadawanych im w trakcie działania programu.

2.2 Klasy zdań logicznych

W tym zbiorze znajdują się klasy, które są realizacją podstawowych operatorów z logiki klasycznej:

- Negacja
- Konjunkcja
- Alternatywa
- Implikacja
- Równoważność

2.3 Klasy zdań

W ramach przygotowanego programu zostały zrealizowane klasy dla każdego z typów zdań możliwych do zdefiniowania:

- **initially α**
Zawiera formułę α zdefiniowaną za pomocą klas zdań logicznych.
- **(A, W) causes α if π**
Zawiera akcję, listę wykonawców, informację o wykluczeniu wykonawców typu logicznego oraz formuły α i π zdefiniowane za pomocą klas zdań logicznych. Warunek π może być pominięty, co oznacza, że jest zawsze prawdziwy, wtedy powyższe zdanie jest postaci **(A, W) causes α** .
- **(A, W) typically causes α if π**
Klasa ta zbudowana jest analogicznie do klasy powyżej, lecz reprezentuje ona zdanie, którego efekt jest typowy.
- **always α**
Zawiera formułę α zdefiniowaną za pomocą klas zdań logicznych.

- **impossible** (A, W) **if** π
Zawiera akcję, listę wykonawców, informację o wykluczeniu wykonawców typu logicznego oraz warunek π zdefiniowany za pomocą klas zdań logicznych.
- (A, W) **releases** f **if** π
Zawiera akcję, listę wykonawców, informację o wykluczeniu wykonawców typu logicznego, fluent oraz warunek π zdefiniowany za pomocą klas zdań logicznych.
- (A, W) **preserves** f **if** π
Zawiera akcję, listę wykonawców, informację o wykluczeniu wykonawców typu logicznego, fluent oraz warunek π zdefiniowany za pomocą klas zdań logicznych.

2.4 Klasa dziedziny

Klasa ta jest programową reprezentacją domeny wprowadzanej przez użytkownika. Zawiera ona listy zdań wszystkich typów, dopuszczalnych w ramach programu. Ponieważ wraz z uzupełnianiem danych liczba i rodzaj zdań może się zmieniać, oferuje ona niezbędne metody służące do ich zmiany, usuwania i modyfikowania.

2.5 Klasa stanu

Ponieważ w trakcie działania programu istnieje potrzeba rozróżniania możliwych stanów opisywanego systemu, powstała klasa odpowiadająca takiemu pojedynczemu stanowi. Każda taka klasa zawiera listę wartościowań dla każdego fluentu występującego w dziedzinie i jest ich konstruowanych w programie tyle, ile występuje unikalnych wartościowań ($2^{|fluentset|}$).

2.6 Klasa Graph

Reprezentuje graf zależności między poszczególnymi stanami.

2.7 Klasa Edge

Każda ścieżka w konstruowanym grafie łączy dwa wierzchołki reprezentujące poszczególne stany między którymi istnieje połączenie opisane w dziedzinie - akcja wykonywana przez pewny zbiór wykonawców. Ponieważ akcje mogą mieć skutki typowe i nietypowe, wyróżnia się także rodzaj krawędzi. Klasa zawiera dwa stany (wierzchołki), akcję i jej wykonawcę oraz informację o nietypowości efektu akcji jako zmienną logiczną.

2.8 Klasa World

Jej składowymi są zbiory fluentów, akcji, aktorów oraz dziedzina. Klasa ta odpowiedzialna jest za budowanie struktury $S = (\Sigma, \sigma_0, ResAb, ResN)$, gdzie:

- Σ - zbiór stanów
- $\sigma_0 \in \Sigma$ - stan początkowy
- $ResAb, ResN : A \times V \times \Sigma \rightarrow 2^\Sigma$ są funkcjami przejść. $ResAb$ jest funkcją przejść nietypowych, $ResN$ jest funkcją przejść typowych.

Klasa ta dostarcza także metodę budowania grafu zależności na podstawie skonstruowanej struktury S .

2.9 Klasa kroku scenariusza

Krok scenariusza zdefiniowany jest poprzez akcję i wykonawcę tejże akcji.

2.10 Klasa scenariusza

Składa się z listy kroków scenariusza.

2.11 Klasy kwerend

Każda kwerenda, która może zostać zdefiniowana w programie, reprezentowana jest przez osobną klasę. Wyróżniamy następujące rodzaje kwerend:

- **always/ever executable Scenario**
Klasy reprezentujące te kwerendy (odpowiednio *always* i *ever*) przechowują obiekt klasy Scenariusza.
- **always/ever/typically accessible γ if π**
Klasy reprezentujące te kwerendy (odpowiednio *always*, *ever* i *typically*) przechowują formuły γ i π zdefiniowane za pomocą klas zdań logicznych.
- **always/ever/typically accessible γ if π when Scenario**
Klasy te zbudowane są analogicznie jak klasy zdefiniowane powyżej, z tym, że dodatkowo przechowują obiekt klasy Scenariusza.
- **always/ever partakes w when Scenario**
Klasy reprezentujące te kwerendy (odpowiednio *always* i *ever*) przechowują wykonawcę oraz obiekt klasy Scenariusza.
- **always/ever/typically γ after Scenario from π**
Klasy reprezentujące te kwerendy (odpowiednio *always*, *ever* i *typically*) przechowują formuły γ i π oraz obiekt klasy Scenariusza.

Każda klasa reprezentująca kwerendę dziedziczy po klasie Query i implementuje metodę Evaluate, która daje odpowiedź na zadane pytanie na podstawie zbudowanego wcześniej grafu zależności.

3 Algorytmy

3.1 Wyznaczanie zbioru stanów

Algorytm na podstawie zbioru fluentów wyznacza zbiór wszystkich stanów. Oparty jest na idei algorytmu z powrotami, który wyznacza wszystkie kombinacje n -elementowego ciągu składającego się z 0 i 1.

```
 $B \leftarrow$  pusty stos  
 $n \leftarrow$  liczba fluentów  
 $k \leftarrow 0$   
 $S \leftarrow$  lista stanów  
function BACKTRACK( $B, n, k$ )  
  if  $k = n$  then APPEND( $S, B$ )  
  end if  
  PUT( $B, 1$ )  
  BACKTRACK( $B, n, k + 1$ )  
  POP( $B$ )  
  PUT( $B, 0$ )  
  BACKTRACK( $B, n, k + 1$ )  
  POP( $B$ )  
  return  $S$   
end function
```

3.2 Wyznaczanie stanu początkowego

Algorytm wyznaczania stanu początkowego (stanów początkowych) polega na wygenerowaniu wszystkich możliwych kombinacji wszystkich fluentów danego świata oraz *przefiltrowaniu* ich przez zdania **always**. Otrzymane w ten sposób dozwolone stany świata są następnie kolejno sprawdzane pod kątem zgodności ze zdaniami **initially**. Dozwolone jest istnienie więcej niż jednego stanu początkowego.

```
 $H \leftarrow$  zbiór wszystkich kombinacji fluentów  $\mathcal{F}$ 
for all always  $\alpha \in \mathcal{D}$  do
  for all  $\sigma \in H$  do
    if  $\sigma \not\models \alpha$  then
       $H \leftarrow H \setminus \{\sigma\}$ 
    end if
  end for
end for
for all initially  $\alpha \in \mathcal{D}$  do
  for all  $\sigma \in H$  do
    if  $\sigma \not\models \alpha$  then
       $H \leftarrow H \setminus \{\sigma\}$ 
    end if
  end for
end for
```

3.3 Obliczanie zbiorów $ResN$ i $ResAb$

Algorytm obliczania zbiorów $ResN$ i $ResAb$ oraz zbiorów pośrednich Res_0 , Res^- , Res_0^+ działa na analogicznej zasadzie *filtrowania* zbioru stanów zdaniami w algorytmie wyznaczania stanu początkowego. Na wyższym poziomie abstrakcji pseudokod wygląda następująco:¹

```

 $\Sigma \leftarrow$  zbiór wszystkich kombinacji fluentów  $\mathcal{F}$ 
 $\Sigma \leftarrow$  stany  $\sigma \in \Sigma$  zgodne ze zdaniami always.
 $Res_0 \leftarrow \Sigma$ 
if dowolne zdanie impossible blokuje  $A$  dla  $Akt$  i  $\sigma$  then
     $Res_0 \leftarrow \emptyset$ 
end if
 $Res_0 \leftarrow$  stany  $\sigma \in Res_0$  zgodne ze zdaniami causes dla  $A$  i  $Akt$ 
 $Res_0 \leftarrow$  stany  $\sigma \in Res_0$  zgodne ze zdaniami preserves dla  $A$  i  $Akt$ 
 $Res^- \leftarrow$  stany  $\sigma \in Res_0$  o minimalnych zbiorach  $New$ 
 $Res_0^+ \leftarrow$  stany  $\sigma \in Res_0^+$  zgodne ze zdaniami typically causes dla  $A$  i  $Akt$ 
 $ResN \leftarrow$  stany  $\sigma \in Res_0^+$  o minimalnych zbiorach  $New$ 
 $ResAb \leftarrow Res^- \setminus ResN$ 

```

¹Dla uproszczenia pseudokodu przez Res i New rozumie się kolejno $Res(A, Akt, \sigma)$ oraz $New(A, Akt, \sigma_0, \sigma_1)$

3.4 Konstrukcja grafu zależności

Graf przejść między stanami generowany jest na podstawie obliczonych zbiorów $ResN$ i $ResAb$. Pseudokod algorytmu wygląda następująco:

```
States  $\leftarrow$  zbiór wszystkich stanów
Actions  $\leftarrow$  zbiór wszystkich akcji
Actors  $\leftarrow$  zbiór wszystkich aktorów
E  $\leftarrow \emptyset$ 
for all state  $\in$  States do
  for all action  $\in$  Actions do
    for all actor  $\in$  Actors do
      for all target  $\in ResAb_{action,actor,state}$  do
        e  $\leftarrow$  nowa krawędź
        e.From  $\leftarrow$  state
        e.To  $\leftarrow$  target
        e.Action  $\leftarrow$  action
        e.Actor  $\leftarrow$  actor
        e.Abnormal  $\leftarrow$  True
        E  $\leftarrow E \cup \{e\}$ 
      end for
      for all target  $\in ResN_{action,actor,state}$  do
        e  $\leftarrow$  nowa krawędź
        e.From  $\leftarrow$  state
        e.To  $\leftarrow$  target
        e.Action  $\leftarrow$  action
        e.Actor  $\leftarrow$  actor
        e.Abnormal  $\leftarrow$  False
        E  $\leftarrow E \cup \{e\}$ 
      end for
    end for
  end for
end for
G  $\leftarrow (States, E)$ 
```

3.5 Obliczanie odpowiedzi na zadaną kwerendę

3.5.1 Kwerendy *always/ever executable Scenario*

Kwerendy *always/ever executable Scenario* sprawdzają, czy dany scenariusz jest wykonywalny zawsze/kiedykolwiek ze stanu początkowego. Pseudokod algorytmu wygląda następująco:

```
Scenario ← zadany scenariusz
States ← zbiór stanów początkowych
for all step ∈ Scenario do
  NewStates ← ∅
  for all state ∈ States do
    Edges ← wszystkie krawędzie dostępne w danym stanie dla kroku step
    if Wersja ALWAYS i Edges.count() == 0 then
      return FALSE - ścieżka niewykonywalna - brak krawędzi
    end if
    for all edge ∈ Edges do
      NewStates.add(edge.to)
    end for
  end for
  if Wersja EVER i NewStates.count() == 0 then
    return FALSE - nie było krawędzi z żadnego stanu - żadna ścieżka nie
    jest wykonywalna
  end if
  States ← NewStates
end for
return TRUE
```

3.5.2 Kwerendy *always/ever Actor partakes when Scenario*

Kwerendy *always/ever Actor partakes when Scenario* sprawdzają, czy dany aktor zawsze/kiedykolwiek bierze udział w wykonywaniu podanego scenariusza. Scenariusz jest wykonywany z dowolnego stanu początkowego. Pseudokod algorytmu wygląda następująco:

```
Actor ← wybrany aktor
Scenario ← zadany scenariusz
States ← zbiór stanów początkowych
if Scenario jest niewykonywalny then
    return FALSE
end if
for all step ∈ Scenario do
    NewStates ← ∅
    for all state ∈ States do
        Edges ← wszystkie krawędzie w danym stanie dla kroku step
        for all edge ∈ Edges do
            if edge.Actor == Actor then
                if Wersja EVER then
                    return TRUE - dany aktor bierze udział
                end if
            else- inny aktor
                NewStates.add(edge.to) - dalsze przeszukanie ścieżki
            end if
        end for
    end for
    if Wersja ALWAYS i NewStates.count() == 0 then
        return TRUE - nie dodano żadnego nowego stanu, czyli na wszystkich
        ścieżkach znalazł się już szukany aktor
    end if
    States ← NewStates
end for
return FALSE
```

3.5.3 Kwerendy *always/ever/typically accessible γ if π when Scenario*

Kwerendy *always/ever/typically accessible γ if π when Scenario* sprawdzają, czy wykonanie podanego scenariusza z każdego stanu spełniającego warunek π zawsze/kiedykolwiek/na ogół doprowadzi do stanu spełniającego warunek γ . Pseudokod algorytmu wygląda następująco:

- Algorytm kwerendy *always*

```
StatesPi  $\leftarrow$  zbiór wszystkich stanów spełniających warunek  $\pi$ 
scenario  $\leftarrow$  scenariusz kwerendy
for all startState  $\in$  StatesPi do
  states  $\leftarrow$  {startState}
  for all step  $\in$  scenario do
    nextS  $\leftarrow$  zbiór stanów
    for all state  $\in$  states do
      nextS  $\leftarrow$  nextS  $\cup$  (stany z krawędzią z state z krokiem step)
    end for
    states  $\leftarrow$  nextS
  end for
  for all state  $\in$  states do
    if  $\neg$ DFSearch( $\emptyset$ , state) then
      return FALSE
    end if
  end for
end for
return TRUE
```

- Algorytm kwerendy *ever*

```

StatesPi ← zbiór wszystkich stanów spełniających warunek  $\pi$ 
scenario ← scenariusz kwerendy
for all startState ∈ StatesPi do
    states ← zbiór stanów
    dodaj startState do states
    for all step ∈ scenario do
        for all state ∈ states do
            nextS ← nextS ∪ (stany z krawędzią z state z krokiem step)
        end for
        states ← nextS
    end for
end for
for all startState ∈ states do
    open ← stany do których można przejść z startState
    close ← pusty zbiór stanów
    while open niepusty do
        state ← {dowolny stan z open}
        if state spełnia warunek  $\gamma$  then
            continue forall
        end if
        for all state2 ∈ stany, do których można przejść z state do
            if state2 ∉ close then
                dodaj state2 do open
            end if
        end for
    end while
    return FALSE
end for
return TRUE

```

- Algorytm kwerendy *typically*

```

StatesPi ← zbiór wszystkich stanów spełniających warunek  $\pi$ 
scenario ← scenariusz kwerendy
for all startState  $\in$  StatesPi do
    states ← {startState}
    for all step  $\in$  scenario do
        nextS ← zbiór stanów
        for all state  $\in$  states do
            nextS ← nextS  $\cup$  (stany z typową krawędzią z state z krokiem
step)
        end for
        states ← nextS
    end for
    for all state  $\in$  states do
        if  $\neg \text{DFSearch}(\emptyset, \text{state})$  then
            return FALSE
        end if
    end for
end for
return TRUE

```

- Algorytm procedury $\text{DFSearch}(\text{close}, \text{state})$

```

if state spełnia warunek  $\gamma$  then
    hasGammaBeenAchieved = TRUE
else
    hasGammaBeenAchieved = FALSE
end if
dodaj state do close
if  $\neg$ hasGammaBeenAchieved then
    if wersja typically then
        actionGroups  $\leftarrow$  typowe krawędzie z state zgrupowane wg par (Akcja,
Aktor)
    end if
    if wersja always then
        actionGroups  $\leftarrow$  krawędzie z state zgrupowane wg par (Akcja, Aktor)
    end if
    for all group  $\in$  actionGroups do
        isAtLeastOneUnSuccessful  $\leftarrow$  FALSE
        for all e  $\in$  group do
            s  $\leftarrow$  e.To
            if s  $\in$  close then
                isAtLeastOneUnSuccessful  $\leftarrow$  TRUE
                CONTINUE
            end if
            if  $\neg$ DFSearch(close, s) then
                isAtLeastOneUnSuccessful  $\leftarrow$  TRUE
            end if
        end for
        if  $\neg$ isAtLeastOneUnSuccessful then
            hasGammaBeenAchieved  $\leftarrow$  TRUE
        end if
    end for
end if
usuń state z close
return hasGammaBeenAchieved

```

3.5.4 Kwerendy *always/ever/typically accessible γ if π*

- Algorytm kwerendy *always accessible γ if π*

Kwerenda odpowiada twierdząco, jeżeli z każdego stanu spełniającego warunek π wykonanie dowolnego scenariusza zawsze doprowadzi do stanu spełniającego warunek γ . Pseudokod algorytmu wygląda następująco:

```
states  $\leftarrow$  zbiór wszystkich stanów spełniających warunek  $\pi$ 
open  $\leftarrow$  zbiór stanów do których można przejść z stanów w states
close - pusty zbiór stanów
while open niepusty do
    state  $\leftarrow$  {dowolny stan z open}
    usuń state z open
    dodaj state do close
    for all state2  $\in$  stany do których można przejść z state do
        if state2  $\notin$  close then
            dodaj state2 do open
        end if
    end for
end while
if close zawiera stan niespełniający  $\gamma$  then
    return FALSE
end if
return TRUE
```

- Algorytm kwerendy *ever/typically accessible* γ if π

Kwerenda odpowiada twierdząco, jeżeli z każdego stanu spełniającego warunek π wykonanie dowolnego scenariusza kiedykolwiek/na ogół doprowadzi do stanu spełniającego warunek γ . Pseudokod algorytmu wygląda następująco:

```

StatesPi  $\leftarrow$  zbiór wszystkich stanów spełniających warunek  $\pi$ 
for all startState  $\in$  StatesPi do
  open  $\leftarrow$  stany do których można (typowo) przejść z startState
  close  $\leftarrow$  pusty zbiór stanów
  while open niepusty do
    state  $\leftarrow$  {dowolny stan z open}
    if state spełnia warunek  $\gamma$  then
      continue forall
    end if
    for all state2  $\in$  stany, do których można (typowo) przejść z state do
      if state2  $\notin$  close then
        dodaj state2 do open
      end if
    end for
  end while
  return FALSE
end for
return TRUE

```

3.5.5 Kwerendy *always/ever/typically* γ after *Scenario* from π

Kwerenda ta sprawdza, czy po wykonaniu podanego scenariusza ze stanu spełniającego warunek π , bądź ze stanu początkowego (w przypadku $\pi \Leftrightarrow \top$) zawsze/kiedykolwiek/na ogół w stanie wynikowym spełniona jest formuła γ . Pseudokod algorytmu wygląda następująco:

```
Scenario  $\leftarrow$  zadany scenariusz
States  $\leftarrow$  zbiór wszystkich dostępnych stanów
for all step  $\in$  Scenario do
  NewStates  $\leftarrow \emptyset$ 
  for all state  $\in$  States do
    Edges  $\leftarrow$  wszystkie krawędzie dostępne w danym stanie dla kroku step
    if Wersja ALWAYS/TYPICALLY i Edges.count() == 0 then
      return FALSE - ścieżka niewykonywalna - brak krawędzi
    end if
    for all edge  $\in$  Edges do
      if Wersja TYPICALLY i edge.Abnormal then continue;
      end if
      NewStates.add(edge.to)
    end for
  end for
  if Wersja EVER i NewStates.count() == 0 then
    return FALSE - nie było krawędzi z żadnego stanu - żadna ścieżka nie
    jest wykonywalna
  end if
  States  $\leftarrow$  NewStates
end for
for all state  $\in$  States do
  if state does not satisfy  $\gamma$  then
    return FALSE
  end if
end for
return TRUE
```

4 Moduły dodatkowe

4.1 Biblioteka Microsoft Automatic Graph Layout

Biblioteka ta została użyta do wizualizacji grafu zależności. Dostępna jest jako oprogramowanie open source pod adresem:

<https://github.com/Microsoft/automatic-graph-layout.git>.

Pakiet złożony jest z trzech bibliotek:

- Layout engine (Microsoft.MSAGL.dll)
- Drawing module (Microsoft.MSAGL.Drawing.dll)
- Viewer control (Microsoft.MSAGL.GraphViewerGDIGraph.dll)

Podstawowe funkcjonalności biblioteki:

- Przybliżanie, oddalanie i przesuwanie grafu
- Dowlona konfiguracja krawędzi i wierzchołków

5 Podział prac

- Dragan Łukasz - wyznaczanie zbioru stanów, ewaluacja formuł, testy
- Flis Mateusz - wizualizacja grafu zależności, testy
- Izert Piotr - implementacja kwerend `executable` i `partakes`, testy
- Pielat Mateusz - wyznaczanie zbiorów *Res*, testy
- Rząd Przemysław - konstrukcja grafu zależności, testy
- Siry Roman - implementacja kwerend `accessible` oraz `after`, testy
- Waszkiewicz Piotr - podstawowe klasy, graficzny interfejs użytkownika, implementacja kwerend `accessible` oraz `after`, testy
- Zawadzka Anna - podstawowe klasy, graficzny interfejs użytkownika, testy