REPREZENTACJA WIEDZY

Projekt 1: Programy działań w środowisku wieloagentowym

Mateusz Tkaczyk, Kacper Król, Kamil Kamiński, Kamil Bańkowski,
Igor Staręga, Jakub Kępka, Stanisław Gwiazda, Piotr Baranowski

Semestr letni 2024/2025

Contents

1 Wstęp

Celem projektu jest opracowanie i zaimplementowanie języka akcji dla pewnej klasy systemów dynamicznych oraz odpowiadającego mu języka kwerend. Język kwerend powinien zapewniać uzyskanie odpowiedzi na stawiane mu pytania. System będzie działał wieloagentowo, gdzie wiele niezależnych agentów może równocześnie wykonywać różne działania, a z każdą akcją związany jest zbiór wykonawców (agentów). Program musi oferować język kwerend, który pozwoli ustalić, czy podany program działań jest możliwy do realizacji zawsze lub kiedykolwiek w stanie początkowym, czy wykonanie programu ze stanu początkowego prowadzi zawsze lub kiedykolwiek do stanu spełniającego warunek celu, oraz czy dany agent jest aktywny w realizacji tego programu. W poniższym dokumencie zostaną przedstawione takie rzeczy jak: szczegółowy opis problemu, przyjętej klasy systemów dynamicznych, zarys implementacji systemów i plan testów.

2 Opis systemu

Dana jest klasa systemów dynamicznych, oznaczana jako $\mathbb{DS}_1$, dla której implementowany będzie język akcji. W tym środowisku wiele działań może być wykonywanych przez różne grupy agentów, a zachowanie systemu wynika z warunków przedstawionych i opisanych poniżej. Najważniejszymi warunkami systemu są:

Prawo inercji Prawo inercji oznacza, że jeśli dla danej akcji nie modyfikujemy jakiejś zmiennej, to pozostaje ona bez zmian. Przyjmujemy, że wszystko pozostaje takie samo, o ile nie zostanie wprost zmienione przez akcję.

Sekwencyjność i niedeterminizm akcji W systemie akcje będą wykonywane sekwencyjnie, jedna po drugiej. Jednocześnie dana akcja może mieć różne skutki, zależne od warunków, w których została wykonana.

Pełna informacja Zakładamy, że system posiada wszystkie informacje o wszystkich zmiennych, agentach oraz wszystkich akcjach i ich skutkach bezpośrednich. Dla systemu wszystkie informacje są jawne oraz nie ma ukrytych czynników wpływających na funkcjonowanie systemu.

Jasne zdefiniowane akcje Z każdą akcją w systemie jest związany:

- warunek początkowy i końcowy (reprezentowane przez formuły zdaniowe);
- zbiór jej wykonawców (agentów).

Efekt końcowy Efekt końcowy zależy od stanu, w którym akcja się zaczyna, oraz zbioru jej wykonawców. W praktyce oznacza to, że różni agenci mogą uzyskać różne rezultaty tej samej akcji w różnych okolicznościach.

Niewykonalność akcji w pewnych stanach i przez pewnych agentów Nie każda akcja może być wykonana zawsze i przez każdego – istnieją stany, w których akcja jest niedostępna.

Dozwolony jest częściowy opis stanów Niektóre elementy i zmienne mogą pozostać nieokreślone w czasie wnioskowania. Nie wszystkie zmienne są zawsze istotne do przeprowadzenia niektórych zapytań. Dlatego uwaga może zostać poświęcona wyłącznie tym zmiennym, które są rzeczywiście istotne w kontekście danej akcji czy analizowanego problemu.

Warunki integralności wpływają tylko na skutki pośrednie akcji Warunek ten oznacza, że istniejące w systemie warunki integralności oddziałują tylko na efekty pośrednie akcji, a nie dodatkowo np. na przyczyny akcji.

Odpowiadający systemowi język kwerend będzie umożliwiał uzyskanie odpowiedzi na następujące pytania:

- **Q1**: Czy podany program działań jest możliwy do realizacji zawsze/kiedykolwiek w stanie początkowym?
- **Q2**: Czy wykonanie podanego programu działań ze stanu początkowego prowadzi zawsze/kiedykolwiek do stanu spełniającego warunek celu γ?
- **Q3**: Czy agent ag jest aktywny w realizacji programu Π działań?

3 Opis matematyczny

W poniższych sekcjach zostanie opisana syntaktyka i semantyka zdefiowanego języka

3.1 Syntaktyka języka akcji

Poniższy rozdział będzie zawierać opis zaproponowanego języka akcji. Sygnatura języka określona będzie jako trójka zbiorów: $\Upsilon = (\mathcal{F}, Ac, Ag)$, gdzie:

- $\mathcal{F}$ to zbiór zmiennych zwanych fluentami, przy czym zbiór fluentów inercyjnych oznaczamy jako $\mathcal{F}_I \subseteq \mathcal{F}$,
- Ac to niepusty zbiór dostępnych akcji,
- Ag to niepusty zbiór agentów wykonujących akcje.

W tym języku literałem f określamy fluent $f \in \mathcal{F}$ lub jego zaprzeczenie $\neg f$.
Język akcji będzie miał reprezentację składającą się z różnych zdań i stwierdzeń definiowanych za pomocą następujących elementów:

1. **Fluenty** (f): zmienne $f_1, f_2, \ldots, f_l \in \mathcal{F}$.
2. **Akcje** (A): zmienne $A_1, A_2, \ldots, A_m \in Ac$, wykonywane przez pojedynczych agentów lub ich grupy.
3. **Agenci** (g): zmienne $g_1, g_2, \ldots, g_k \in Ag$, przypisani do wykonywania akcji.
4. **Formuły logiczne** (α, β, π): wyrażenia logiczne budowane z fluentów i operatorów $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$.
5. **Formuły agentowe** (Fg): formuły logiczne powiązane z agentami $g \in Ag$, np. $g|\neg F_{g0}|F_{g0} \wedge F_{g1}| \ldots$.

Sama reprezentacja będzię składała się z określonych rodzajów zdań i stwierdzeń:

1. **Zdania wartości i obserwacji**
   Wyrażenie logiczne α jest spełnione po wykonaniu ciągu akcji przez agentów.
   - α **after** A_1 **by** Fg_1**,** A_2 **by** Fg_2**, ...,** A_n **by** Fg_n: wyrażenie α jest zawsze spełnione po wykonaniu sekwencji akcji.
   - **observable** α **after** A_1 **by** Fg_1**,** A_2 **by** Fg_2**, ...,** A_n **by** Fg_n: wyrażenie α może być spełnione po wykonaniu sekwencji akcji.
2. **Zdania efektu**
   Opisują skutki wykonania akcji przez agentów, które prowadzą do spełnienia wyrażeń.
   - A **by** Fg **causes** α **if** π: wykonanie akcji A przez agentów spełniających warunek π powoduje, że α zachodzi.
   - Skróty: jeśli π jest zawsze prawdziwe używany A **by** Fg **causes** α oraz jeśli α jest zawsze fałszywe to **impossible** A **by** Fg **if** π

3. **Zdania release**
   Określają sytuację, w której wykonanie akcji przez agentów może zmienić wartość fluentu inercjalnego.

   - A **by** Fg **releases** f **if** π: akcja może zmienić wartość fluentu f w stanie spełniającym warunek π.
   - Skrót: A **by** Fg **releases** f, gdy warunek π jest zawsze spełniony.

4. **Zdania integralności**
   Wyrażenie jest zawsze spełnione niezależnie od stanu.

   - **always** α: wyrażenie α obowiązuje w każdym stanie.

Niepusty zbiór R składający się ze stwierdzeń i zdań typu 1–4 nazywamy **Reprezentacją akcji**.

Poniżej zostały przedstawione przykładowe składnie języka reprezentujące jego syntaktykę:

Przykład 1 Trzech studentów: Kacper, Kamil i Mateusz próbuje zbudować dom. Aby dom został zbudowany, wszyscy muszą poprawnie wykonać swoje zadania: Kacper dostarcza materiały, Kamil stawia ściany, a Mateusz montuje dach. Niestety, Kamil popełnia błąd w budowie ścian, co uniemożliwia ukończenie budowy.

Reprezentacja:
initially $\neg house\_built$;
initially $\neg walls\_correct$;
$SUPPLY\_MATERIALS$ **by** $Kacper$ **causes** $materials\_ready$;
$BUILD\_WALLS$ **by** $Kamil$ **causes** $walls\_correct$ **if** $materials\_ready$;
$INSTALL\_ROOF$ **by** $Mateusz$ **causes** $house\_built$ **if** $walls\_correct$;
$MISTAKE$ **by** $Kamil$ **releases** $\neg walls\_correct$;

Przykład 2 Kacper i Kamil chcą zbudować dom. Budowę domu może prowadzić Kacper lub Kamil, ale tylko Kacper potrafi przygotować potrzebne materiały. Budowanie domu sprawia, że dom staje się gotowy, ale zużywa materiały. Na początku dom nie jest zbudowany, a materiały nie są przygotowane.

Reprezentacja:
initially $\neg built\_house \wedge \neg materials\_ready$;
$PREPARE\_MATERIALS$ **by** Kacper **causes** $materials\_ready$;
$BUILD$ **by** Kacper $\vee$ Kamil **causes** $\neg materials\_ready$;
$BUILD$ **by** Kacper $\vee$ Kamil **causes** $built\_house$ **if** $materials\_ready$;

3.2 Semantyka języka akcji

Programem działań w tak zdefiniowanym systemie jest ciąg $\Pi = ((A_1, G_1), \ldots, (A_n, G_n))$, gdzie A_i jest akcją, zaś G_i jest grupą agentów. Mówimy, że agent ag jest aktywny w wykonaniu akcji Ac w stanie σ, jeśli z opisu dziedziny jednoznacznie wynika jego przynależność do grupy realizującej tę akcję w stanie σ.

Stan (oznaczany jako σ) reprezentuje aktualną konfigurację wartości fluentów w danym momencie, tj. po wykonaniu pewnej sekwencji akcji oraz odpowiadającej jej sekwencji formuł agentowych. Zbiór wszystkich możliwych stanów oznaczamy jako Σ.
Każdy stan $\sigma \in \Sigma$ definiowany jest jako odwzorowanie $\sigma : F \to \{0, 1\}$, gdzie dla dowolnego fluenta $f \in F$ zachodzi:

- $\sigma(f) = 1$, jeśli fluent f jest spełniony w stanie σ,
- $\sigma(f) = 0$, jeśli fluent f nie zachodzi w stanie σ.

Zapis ten można równoważnie przedstawić jako:

$$\sigma(f) = 1 \iff \sigma \models f \qquad \sigma(f) = 0 \iff \sigma \not\models f$$

czyli stan σ spełnia fluent f wtedy i tylko wtedy, gdy jego wartość w tym stanie wynosi 1.

Definicja 3.1 *Stan $\sigma : F \to \{0, 1\}$ spełnia formułę logiczną α, gdy wartość $\sigma(\alpha) = 1$. W takim przypadku mówimy, że σ jest modelem formuły α, co zapisujemy jako $\sigma \models \alpha$.*

Definicja 3.2 *Stan $\sigma : F \to \{0, 1\}$ uznawany jest za poprawny względem opisu dziedziny D wtedy i tylko wtedy, gdy dla każdego ograniczenia postaci always α zawartego w D zachodzi relacja $\sigma \models \alpha$.*

Innymi słowy, stanem dopuszczalnym w danej dziedzinie nazywamy takie przypisanie wartości fluentom, które spełnia wszystkie formuły integralności zdefiniowane w opisie tej dziedziny.
Każdemu podzbiorowi agentów $G \subseteq Ag$ można przyporządkować funkcję charakterystyczną $\chi_G : Ag \to \{0, 1\}$, która dla każdego agenta $g \in Ag$ przyjmuje wartość 1 wtedy i tylko wtedy, gdy g należy do zbioru G (czyli $g \in G$), oraz wartość 0 w przeciwnym przypadku.
Wartościowanie formuł agentowych względem funkcji charakterystycznych definiujemy w sposób analogiczny do wartościowania formuł logicznych względem stanów.

Definicja 3.3 *Zbiór agentów $G \subseteq Ag$ spełnia formułę agentową F_g, gdy zachodzi $\chi_G(F_g) = 1$. Wówczas G jest modelem tej formuły, co zapisujemy jako $G \models F_g$.*

Ponieważ istnieje wiele różnych zbiorów agentów mogących spełniać daną formułę agentową, przez $G(F_g) := \{G \subseteq Ag : G \models F_g\}$ oznaczamy zbiór wszystkich takich zbiorów G, które są modelami formuły F_g. Innymi słowy, $G(F_g)$ to kolekcja wszystkich podzbiorów agentów spełniających daną formułę. Zbiór $\hat{G}$ jest zbiorem wszystkich zbiorów $G(F_g)$ (gdzie formuły F_g występują w danej dziedzinie) oraz ich dopełnień $G^c(F_g) = 2^{A_g} \setminus G(F_g)$. Elementy zbioru $\hat{G}$ będziemy oznaczać symbolem G.

Przykład

- $\neg$ negacja. Dla agentów $A_g = \{a, b, c\}$ formuła $F_g = \neg a \wedge b$ generuje zbiór $G = \{\{b\}, \{b, c\}\}$, ponieważ każdy zbiór agentów $G \in G$ spełnia formułę F_g, tzn. $\{b\} \models F_g$ oraz $\{b, c\} \models F_g$.
- $\wedge$ koniunkcja. Dla agentów $A_g = \{a, b, c\}$ formuła $F_g = a \wedge b$ generuje zbiór $G = \{\{a, b\}, \{a, b, c\}\}$.
- $\vee$ alternatywa. Dla agentów $A_g = \{a, b, c\}$ formuła $F_g = a \vee b$ generuje zbiór $G = \{\{a\}, \{b\}, \{a, c\}, \{b, c\}, \{a, b\}, \{a, b, c\}\}$.
- $\rightarrow$ implikacja. Dla agentów $A_g = \{a, b, c\}$ formuła $F_g = a \rightarrow b$ generuje zbiór $G = \{\{b\}, \{c\}, \{b, c\}, \{a, b\}, \{a, b, c\}\}$.
- $\leftrightarrow$ równoważność. Dla agentów $A_g = \{a, b, c\}$ formuła $F_g = a \leftrightarrow b$ generuje zbiór $G = \{\{c\}, \{a, b\}, \{a, b, c\}\}$.

Definicja 3.4 *Strukturą dla języka L z klasy języków AR z wykonawcami akcji nazywamy trójkę $S = (\Sigma, \sigma_0, Res)$, spełniającą następujące warunki:*

- $\Sigma \neq \emptyset$ *to zbiór stanów,*
- $\sigma_0 \in \Sigma$ *to stan początkowy,*
- $Res : Ac \times \hat{G} \times \Sigma \rightarrow 2^{\Sigma}$ *to funkcja przejścia, gdzie Ac oznacza zbiór wszystkich akcji, a $\hat{G}$ to zbiór wszystkich zbiorów G.*

Intuicyjnie, dla dowolnej akcji $A \in \mathrm{Ac}$, dowolnego zbioru zbiorów agentów $G = \{G_1, \ldots, G_n\} \in \hat{G}$, gdzie $G_i \subseteq A_g$, oraz dowolnego stanu $\sigma \in \Sigma$, funkcja przejścia Res przypisuje wszystkie stany osiągalne ze stanu σ poprzez wykonanie akcji A przez wszystkie grupy wykonawców G_i ze zbioru G. Dodatkowo, funkcja $\mathrm{Res}(A, G, \sigma)$ określa wszystkie stany będące „najbliżej jak się da" stanu σ, a Σ to zbiór stanów spełniających wszystkie ograniczenia.

Definicja 3.5 *Niech $S = (\Sigma, \sigma_0, Res)$ będzie strukturą języka AR z wykonawcami akcji. Odwzorowanie częściowe dla zadanego języka $\Psi_S : (Ac \times 2^{A_g})^* \times \Sigma \rightarrow \Sigma$ definiuje się w następujący sposób:*

- $\Psi_S(\epsilon, \sigma) = \sigma$
- *Jeżeli istnieje (jest określone) przejście $\Psi_S(((A_1, G_1), \ldots, (A_n, G_n)), \sigma)$ dla $n \geq 1$, gdzie grupa agentów G_i odpowiada akcji A_i, musi zachodzić następująca relacja:*

$$\Psi_S(((A_1, G_1), \ldots, (A_n, G_n)), \sigma) \in Res(A_n, G_n, \Psi_S(((A_1, G_1), \ldots, (A_{n-1}, G_{n-1})), \sigma))$$

Definicję 3.5 możemy rozumieć w następujący sposób:

- Brak akcji (i wykonawców akcji) nie wpływa na stan,
- Jeżeli istnieje częściowe przejście ze stanu σ dla ciągu n akcji i zbiorów agentów, to funkcja przejścia musi definiować taką transformację również dla ciągu $n - 1$ akcji i zbiorów agentów.

Przyjmijmy teraz, że D jest dziedziną akcji, a $S = (\Sigma, \sigma_0, \mathrm{Res})$ będzie strukturą języka.

Definicja 3.6 *Zdanie wartości* (α *after* A_1 *by* F_{g_1}, A_2 *by* $F_{g_2}, \ldots, A_n$ *by* F_{g_n}) *jest spełnione w* S *wtedy i tylko wtedy, gdy częściowe odwzorowanie* $\Psi_S(((A_1, G_1), \ldots, (A_n, G_n)), \sigma_0) \models \alpha$ *dla każdego odwzorowania* Ψ_S.

Definicja 3.6 oznacza, że gdy dla każdego odwzorowania Ψ_S wynikiem wykonania programu P jest stan spełniający α (każdy stan wynikowy musi spełniać warunek α), wówczas zdanie wartości jest prawdziwe w strukturze języka S.

Definicja 3.7 *Zdanie obserwacji* (*observable* α *after* A_1 *by* F_{g_1}, A_2 *by* $F_{g_2}, \ldots, A_n$ *by* F_{g_n}) *jest spełnione w* S *wtedy i tylko wtedy, gdy istnieje takie odwzorowanie* Ψ_S, *że:*

$$\Psi_S(((A_1, G_1), \ldots, (A_n, G_n)), \sigma_0) \models \alpha.$$

Definicja 3.7 oznacza, że gdy istnieje pewne odwzorowanie Ψ spełniające α, czyli istnieje taka ścieżka programu P, która kończy się stanem spełniającym α, to zdanie obserwacji jest prawdziwe w strukturze języka S.

Definicja 3.8 *Funkcję pomocniczą* $Res_0 : Ac \times \hat{G} \times \Sigma \to 2^{\Sigma}$ *definiuje się w następujący sposób:*

- *dla każdego* $A \in Ac$ *i każdego zbioru grup agentów* $G \in \hat{G}$ *oraz dla każdego* $\sigma \in \Sigma$*:*
  - *jeżeli istnieje zdanie* (A *by* F_g *causes* α *if* π) $\in D \wedge (G = G(F_g) \vee G = G^c(F_g))$, *to funkcja pomocnicza* Res_0 *jest określona w następujący sposób:*

    $$Res_0(A, G, \sigma) = \{\sigma' \in \Sigma : (A \text{ by } F_g \text{ causes } \alpha \text{ if } \pi) \in D \wedge (\sigma \models \pi) \wedge (G \models F_g) \Rightarrow (\sigma' \models \alpha)\},$$

  - *w przeciwnym przypadku funkcja* Res_0 *jest nieokreślona.*

Funkcja Res_0, opisana w definicji 3.8, jest to zbiór wszystkich stanów σ', takich że jeśli zadanie (A by F_g causes α if π) należy do dziedziny akcji D oraz stan σ modeluje warunek początkowy akcji π, a zbiór G zawiera wszystkie modele F_g (spełnia formułę F_g), to stan wynikowy spełnia α. Czyli jest to zbiór wszystkich stanów σ', w których spełnione są prawa akcji zaczynającej się w jakimś stanie σ. W przypadku, gdy pierwszy warunek implikacji jest fałszywy (np. zbiór $G \not\models F_g$), to implikacja również jest spełniona (gdyż z fałszu wynika prawda). Wówczas funkcja Res_0 zwraca wszystkie możliwe stany. Z tego względu, dla danego zdania *causes* rozważamy jedynie zbiory $G = G(F_g)$ oraz ich dopełnienia $G = G^c(F_g)$ dla formuły agentowej F_g z danego zdania *causes*, aby uniknąć przecięcia między rozważanymi zbiorami G dla danej akcji i stanu początkowego.
Funkcja Res_0 jest zbyt szerokim zbiorem stanów, aby modelować język, który opisujemy w niniejszej pracy, zgodnie z prawem inercji, dlatego definiuje się funkcję New .

Definicja 3.9 *Dla każdego* $\sigma \in \Sigma$, *dla każdego* $A \in Ac$ *i każdego zbioru grup agentów* $G \in \hat{G}$ *oraz dla każdego* $\sigma' \in Res_0(A, G, \sigma)$, *funkcja* $New(A, G, \sigma, \sigma')$ *jest zbiorem literałów* f, *takim że* $\sigma' \models f$ *oraz:*

- $f \in F_I \wedge \sigma(f) \neq \sigma'(f)$, *lub*
- *istnieje zdanie* (A *by* F_g *releases* f *if* π) $\in D \wedge \sigma \models \pi \wedge G \models F_g$.

Biorąc akcję, jej wykonawców, stan początkowy i stan wynikowy, za pomocą funkcji New otrzymuje się taki zbiór literałów ze zbioru wynikowego, który obejmuje zmiany wszystkich fluentów bądź uwolnione fluenty.

Definicja 3.10 *Niech D będzie dziedziną akcji dla języka L z klasy języków AR z wykonawcami akcji oraz niech $S = (\Sigma, \sigma_0, Res)$ będzie strukturą dla L. Mówimy, że S jest modelem D wtedy i tylko wtedy, gdy:*

- *Σ jest zbiorem wszystkich stanów dla D,*
- *każde zdanie wartości oraz każde zdanie obserwacji w D jest prawdziwe w S,*
- *dla każdego $A \in Ac$ i każdego zbioru grup agentów $G \in \hat{G}$ oraz każdego $\sigma \in \Sigma$, $Res(A, G, \sigma)$ jest zbiorem wszystkich stanów $\sigma' \in \Sigma$, dla których zbiory $New(A, G, \sigma, \sigma')$ są minimalne względem inkluzji.*

4 Problemy ramowe

4.1 Misja załogi czołgu

Załoga czołgu składa się z trzech osób: kierowcy, obserwatora i strzelca. Może ona przechwycić wiadomość wroga, pod warunkiem że w chwili jej nadawania przynajmniej jeden z członków załogi ją nasłuchuje. Załoga ma także możliwość zaatakowania stanowiska artylerii przeciwnika. Atak powiedzie się tylko wtedy, gdy wszyscy członkowie załogi w nim uczestniczą. Oczywiście przechwycenie wiadomości oraz atak na wroga są możliwe tylko wtedy, gdy czołg nie został wcześniej zniszczony. Niezniszczona artyleria może wycelować w czołg, przeprowadzając ostrzał, którego wynik może być zarówno pozytywny, jak i negatywny.

Reprezentacja:
initially $tank\_alive$;
initially $enemy\_alive$;
initially $\neg intercepted$;
$INTERCEPT$ **by** $Driver \vee Spotter \vee Gunner$ **causes** $intercepted$;
$ATTACK$ **by** $Driver \wedge Spotter \wedge Gunner$ **causes** $\neg enemy\_alive$ **if** $tank\_alive$;
$STRIKE$ **releases** $\neg tank\_alive$ **if** $enemy\_alive$;

Ciąg akcji:
$INTERCEPT$ **by** $\{Spotter\}$
$STRIKE$
$ATTACK$ **by** $\{Driver, Spotter, Gunner\}$

Kwerendy:
necessary $intercepted$ **after** [...]; $\longrightarrow$ $true$
necessary $tank\_alive$ **after** [...]; $\longrightarrow$ $false$
possible $intercepted \wedge tank\_alive$ **after** [...]; $\longrightarrow$ $true$
possible $tank\_alive \wedge enemy\_alive$ **after** [...]; $\longrightarrow$ $false$
active $Driver$ **in** $ATTACK$ **by** $\{Driver, Spotter, Gunner\}$; $\longrightarrow$ $true$
active $Spotter$ **in** $INTERCEPT$ **by** $\{Driver, Spotter, Gunner\}$; $\longrightarrow$ $false$

4.2 Akcja drużyny piłkarskiej

W akcji ofensywnej zespołu piłkarskiego bierze udział trzech zawodników: pomocnik, skrzydłowy i napastnik. Celem jest zdobycie bramki. W początkowej sytuacji drużyna znajduje się w posiadaniu piłki, jednak zawodnicy nie są w odpowiedniej pozycji do strzału. Pomocnik może rozegrać piłkę, wymieniając się podaniami z jednym z pozostałych członków akcji. Wówczas piłka znajdzie się w odpowiednim położeniu do oddania strzału. Ponadto, skrzydłowy lub napastnik może zdecydować się na drybling, jednak ze względu na ich średnie umiejętności może to spowodować, że pozycja stanie się znowu nieodpowiednia do strzału. Dowolny z zawodników może również oddać strzał. W przypadku, gdy strzał jest oddawany przez napastnika w odpowiedniej pozycji, próba kończy się strzeleniem bramki. Jeśli strzał z dobrej pozycji oddaje skrzydłowy lub pomocnik, bramka może (ale nie musi) paść. Strzał z nieodpowiedniej pozycji, niezależnie od wykonawcy, kończy się niepowodzeniem, tj. brakiem zdobycia bramki. Ponadto, niezależnie od pozycji i wykonawcy, oddanie strzału jest równoważne stracie piłki (drużyna przeciwna przejmuje piłkę po nieudanym strzale lub wznawia grę, gdy padła bramka).

Reprezentacja:
initially $ball\_possesion$;
initially $\neg good\_position$;
initially $\neg goal$;
$PLAY$ **by** $(Midfielder \wedge Winger) \vee (Midfielder \wedge Striker)$ **causes** $good\_position$;
$DRIBBLE$ **by** $Winger \vee Striker$ **releases** $\neg good\_position$
$SHOOT$ **by** $Striker$ **causes** $goal$ **if** $good\_position$
$SHOOT$ **by** $Midfielder \vee Winger$ **releases** $goal$ **if** $good\_position$
$SHOOT$ **causes** $\neg ball\_possesion$

Ciąg akcji:
$PLAY$ **by** $\{Midfielder \wedge Winger\}$
$DRIBBLE$ **by** $\{Winger\}$
$SHOOT$ **by** $\{Striker\}$

Kwerendy:
necessary $\neg ball\_possesion$ **after** [...]; $\longrightarrow$ $true$
necessary $good\_position$ **after** [...]; $\longrightarrow$ $false$
possible $goal$ **after** [...]; $\longrightarrow$ $true$
possible $goal \wedge \neg good\_position$ **after** [...]; $\longrightarrow$ $false$
active $Midfielder$ **in** $PLAY$ **by** $\{Midfielder, Winger, Striker\}$; $\longrightarrow$ $true$
active $Winger$ **in** $SHOOT$ **by** $\{Midfielder, Winger, Striker\}$; $\longrightarrow$ $false$

4.3 Misja drużyny ratunkowej

initially $zone\_stable \wedge civilian\_injured \wedge \neg civilian\_rescued \wedge \neg team\_injured$;
$ASSIST$ **by** $Medic$ **causes** $\neg civilian\_injured$;
$SECURE$ **by** $Scout$ **causes** $zone\_stable$;
$EVACUATE$ **by** $Medic \wedge Technician \wedge Scout$ **causes** $civilian\_rescued$ **if** $zone\_stable$;
$EVACUATE$ **by** $Medic \wedge Technician \wedge Scout$ **causes** $team\_injured$ **if** $\neg zone\_stable$;
$COLLAPSE$ **releases** $\neg zone\_stable$;
$ALERT$ **causes** $\neg zone\_stable$;

Przestrzeń stanów

$$\sigma_0 = \{zone\_stable, civilian\_injured, \neg civilian\_rescued, \neg team\_injured\}$$
$$\sigma_1 = \{zone\_stable, \neg civilian\_injured, \neg civilian\_rescued, \neg team\_injured\}$$
$$\sigma_2 = \{\neg zone\_stable, \neg civilian\_injured, \neg civilian\_rescued, \neg team\_injured\}$$
$$\sigma_3 = \{\neg zone\_stable, \neg civilian\_injured, civilian\_rescued, team\_injured\}$$
$$\sigma_4 = \{zone\_stable, \neg civilian\_injured, civilian\_rescued, \neg team\_injured\}$$

Agenci i grupy

$$Ag = \{Medic, Technician, Scout\}$$
$$G_1 = \{\{Medic\}\}$$
$$G_2 = \{\{Scout\}\}$$
$$G_3 = \{\{Medic, Technician, Scout\}\}$$
$$G_4 = \{\emptyset, \{Technician\}, \{Medic, Technician\}\}$$

Zbiory akcji i stanów

$$Ac = \{ASSIST, SECURE, EVACUATE, COLLAPSE, ALERT\}$$
$$\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$$

Funkcje Res i New

ASSIST:

$$Res_0(ASSIST, G_1, \sigma_0) = \{\sigma_1\}$$
$$New(ASSIST, G_1, \sigma_0, \sigma_1) = \{\neg civilian\_injured\}$$
$$Res(ASSIST, G_1, \sigma_0) = \{\sigma_1\}$$

COLLAPSE:

$$Res_0(COLLAPSE, \_, \sigma_1) = \{\sigma_2\}$$
$$New(COLLAPSE, \_, \sigma_1, \sigma_2) = \{\neg zone\_stable\}$$
$$Res(COLLAPSE, \_, \sigma_1) = \{\sigma_2\}$$

EVACUATE (strefa bezpieczna):

$$Res_0(EVACUATE, G_3, \sigma_1) = \{\sigma_4\}$$
$$New(EVACUATE, G_3, \sigma_1, \sigma_4) = \{civilian\_rescued\}$$
$$Res(EVACUATE, G_3, \sigma_1) = \{\sigma_4\}$$

EVACUATE (strefa niezabezpieczona):

$$Res_0(EVACUATE, G_3, \sigma_2) = \{\sigma_3\}$$
$$New(EVACUATE, G_3, \sigma_2, \sigma_3) = \{civilian\_rescued, team\_injured\}$$
$$Res(EVACUATE, G_3, \sigma_2) = \{\sigma_3\}$$

SECURE:

$$Res_0(SECURE, G_2, \sigma_2) = \{\sigma_1\}$$
$$New(SECURE, G_2, \sigma_2, \sigma_1) = \{zone\_stable\}$$
$$Res(SECURE, G_2, \sigma_2) = \{\sigma_1\}$$

ALERT:

$$Res_0(ALERT, \_, \sigma_0) = \{\sigma_2\}$$
$$New(ALERT, \_, \sigma_0, \sigma_2) = \{\neg zone\_stable\}$$
$$Res(ALERT, \_, \sigma_0) = \{\sigma_2\}$$

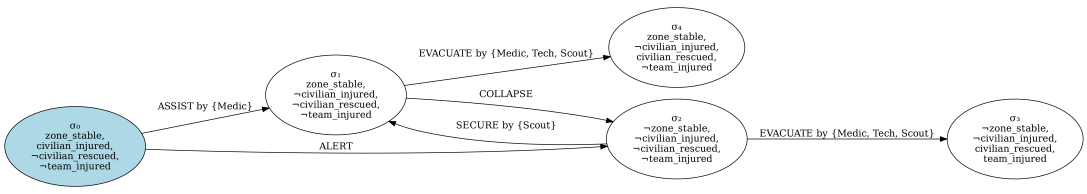

Figure 1: Diagram przejść stanów dla drużyny ratunkowej

Przykładowe kwerendy

Przykład 1

```
always executable ((ASSIST, {Medic}), (SECURE, {Scout}), (EVACUATE, {Medic,
Technician, Scout}))
```

Program zaczyna w stanie
$\sigma_0 = \{zone\_stable, civilian\_injured, \neg civilian\_rescued, \neg team\_injured\}$. Po wykonaniu akcji `ASSIST` przez `Medic` przechodzimy do σ_1. Następnie `SECURE` wykonane przez `Scout` nie zmienia stanu, ponieważ strefa była już stabilna. W końcu `EVACUATE` (przez trzech agentów) w bezpiecznej strefie powoduje przejście do σ_4. Wszystkie akcje są wykonywalne, więc kwerenda zwraca `TRUE`.

Przykład 2
`sometimes executable ((COLLAPSE, {}), (EVACUATE, {Medic, Technician, Scout}))`
Po rozpoczęciu w stanie σ_1 (np. po wcześniejszym ASSIST), COLLAPSE powoduje przejście do σ_2 – strefa przestaje być bezpieczna. EVACUATE nadal jest wykonywalne, lecz prowadzi do σ_3, gdzie zespół zostaje ranny. Program możliwy do wykonania, więc kwerenda zwraca `TRUE`.

Przykład 3
`always accessible (civilian_rescued) from (civilian_injured) in ((EVACUATE, {Medic, Technician, Scout}))`
Zaczynając od stanu z `civilian_injured`, pojedyncze EVACUATE nie prowadzi do `civilian_rescued`, ponieważ akcja wymaga, by cywil wcześniej nie był ranny. Zatem `civilian_rescued` nie jest zawsze osiągalny bez uprzedniego ASSIST, więc kwerenda zwraca `FALSE`.

Przykład 4
`sometimes accessible (civilian_rescued) from (civilian_injured) in ((ASSIST, {Medic}), (SECURE, {Scout}), (EVACUATE, {Medic, Technician, Scout}))`
Zaczynając w stanie σ_0, wykonywane kolejno: ASSIST $\rightarrow$ SECURE $\rightarrow$ EVACUATE. W efekcie dochodzimy do σ_4, gdzie `civilian_rescued` jest spełniony. Zatem osiągalność istnieje – kwerenda zwraca `TRUE`.

Przykład 5
`realisable ((ASSIST, {Medic}), (COLLAPSE, {}), (EVACUATE, {Medic, Technician, Scout})) by {Medic, Technician, Scout}`
Wszystkie akcje wykonywane są przez podany zbiór agentów lub jego podzbiory. Każda z akcji jest wykonywalna (ASSIST i EVACUATE odpowiednio przez odpowiednie grupy, COLLAPSE nie wymaga agenta), więc ciąg jest realizowalny. Kwerenda zwraca `TRUE`.

4.4 Misja drużyny strażackiej

Zespół strażaków składa się z trzech członków: Firefighter (strażak), Engineer (inżynier) i Medic (medyk), oraz istnieje agent środowiska Fire (pożar). Celem misji jest przygotowanie zapasu wody, wejście do budynku, ugaszenie pożaru i ewakuacja uwięzionej ofiary. Jeśli pożar jest aktywny, istnieje ryzyko zawalenia się budynku, co może uniemożliwić przebywanie wewnątrz.

Reprezentacja:
initially $fire\_active$;
initially $\neg water\_ready$;
initially $\neg in\_building$;
initially $victim\_trapped$;
initially $\neg fire\_extinguished$;

$PREPARE\_WATER$ **by** $Engineer$ **causes** $water\_ready$;
$ENTER\_BUILDING$ **by** $Firefighter$ **causes** $in\_building$ **if** $fire\_active$;
$EXTINGUISH$ **by** $Firefighter \wedge Engineer$ **causes** $fire\_extinguished$ **if** $water\_ready \wedge in\_building$;
$RESCUE$ **by** $Medic$ **causes** $\neg victim\_trapped$ **if** $fire\_extinguished \wedge in\_building$;
$COLLAPSE$ **by** $Fire$ **releases** $\neg in\_building$ **if** $fire\_active$;

Ciąg akcji:
$PREPARE\_WATER$ **by** $\{Engineer\}$
$ENTER\_BUILDING$ **by** $\{Firefighter\}$
$COLLAPSE$ **by** $\{Fire\}$
$EXTINGUISH$ **by** $\{Firefighter, Engineer\}$
$RESCUE$ **by** $\{Medic\}$

Kwerendy:
necessary $water\_ready$ **after** [...]; $\longrightarrow$ $true$
possible $\neg in\_building$ **after** [...]; $\longrightarrow$ $true$
necessary $victim\_trapped$ **after** [...]; $\longrightarrow$ $false$
possible $fire\_extinguished \wedge victim\_trapped$ **after** [...]; $\longrightarrow$ $false$
active $Firefighter$ **in** $EXTINGUISH$ **by** $\{Firefighter, Engineer\}$; $\longrightarrow$ $true$
active $Medic$ **in** $ENTER\_BUILDING$ **by** $\{Firefighter, Engineer, Medic\}$; $\longrightarrow$ $false$

4.5 Misja zespołu lekarzy diagnozujących pacjenta

Zespół medyczny składa się z trzech specjalistów: **Clinician** (lekarz przyjmujący pacjenta i stawiający diagnozę), **Radiologist** (radiolog) oraz **Pathologist** (patolog). Celem misji jest pobranie próbek, przeprowadzenie badań, analiza wyników oraz sformułowanie diagnozy.

Reprezentacja:
initially $patient\_admitted$;
initially $\neg sample\_taken$;
initially $\neg tests\_completed$;
initially $\neg diagnosis\_ready$;
initially $patient\_stable$;

$TAKE\_SAMPLE$ **by** $Clinician$ **causes** $sample\_taken$ **if** $patient\_stable$;
$PERFORM\_IMAGING$ **by** $Radiologist$ **causes** $tests\_completed$ **if** $sample\_taken$;
$PERFORM\_BIOPSY$ **by** $Pathologist$ **causes** $tests\_completed$ **if** $sample\_taken$;
$COMPLICATION$ **releases** $\neg patient\_stable$ **if** $sample\_taken$;
$ANALYZE\_RESULTS$ **by** $Radiologist \wedge Pathologist$
causes $diagnosis\_ready$ **if** $tests\_completed \wedge patient\_stable$;
$MAKE\_DIAGNOSIS$ **by** $Clinician$ **causes** $\neg patient\_admitted$ **if** $diagnosis\_ready$;

Ciąg akcji:
$TAKE\_SAMPLE$ **by** $\{Clinician\}$
$PERFORM\_IMAGING$ **by** $\{Radiologist\}$
$PERFORM\_BIOPSY$ **by** $\{Pathologist\}$
$COMPLICATION$
$ANALYZE\_RESULTS$ **by** $\{Radiologist, Pathologist\}$
$MAKE\_DIAGNOSIS$ **by** $\{Clinician\}$

Kwerendy:
necessary $sample\_taken$ **after** [...]; $\longrightarrow$ $true$
possible $\neg patient\_stable$ **after** [...]; $\longrightarrow$ $true$
necessary $tests\_completed$ **after** [...]; $\longrightarrow$ $false$
possible $diagnosis\_ready \wedge patient\_stable$ **after** [...]; $\longrightarrow$ $true$
possible $\neg diagnosis\_ready$ **after** [...]; $\longrightarrow$ $false$
active $Radiologist$ **in** $ANALYZE\_RESULTS$ **by** $\{Radiologist, Pathologist\}$; $\longrightarrow$ $true$
active $Clinician$ **in** $PERFORM\_IMAGING$ **by** $\{Clinician, Radiologist\}$; $\longrightarrow$ $false$

5 Składnia języka zapytań

Zdefiniowany wcześniej język akcji można analizować przy użyciu języka zapytań, który umożliwia udzielenie odpowiedzi typu TRUE/FALSE na pytania dotyczące możliwego przebiegu programów. Przykładowe zapytania obejmują:

1. Czy wykonując dany program działań, zaczynając od stanu początkowego spełniającego warunek π, można (lub zawsze można) osiągnąć stan spełniający warunek celu γ?

   - *always/sometimes accessible* γ *from* π *in* P

   Oznacza to, że program P wykonany z każdego (lub przynajmniej jednego) stanu spełniającego π prowadzi do stanu spełniającego γ.

2. Czy grupa agentów G jest wystarczająca do wykonania programu P?

   - *realisable* P *by* G

   Innymi słowy, czy agenci w G mogą wspólnie zrealizować program P.

3. Czy program P może być wykonany zawsze lub w niektórych przypadkach?

   - *always/sometimes executable* P

   To zapytanie sprawdza, czy istnieje możliwość (lub gwarancja) wykonania programu P ze stanu początkowego σ_0.

6 Semantyka języka zapytań

Niech D oznacza dziedzinę działań, a Q konkretne zapytanie. Mówimy, że zapytanie Q jest logiczną konsekwencją programu P względem dziedziny D, co zapisujemy jako $P, D \models Q$, jeśli zachodzą poniższe warunki:

1. **Zapytanie *sometimes accessible* γ from π in P:**

   - Zwraca *TRUE*, jeśli dla każdego modelu $S = (\Sigma, \sigma_0, \text{Res})$ zgodnego z D, dla każdego stanu $\sigma \in \Sigma$ oraz pewnych odwzorowań częściowych Ψ_S, jeśli $\sigma \models \pi$, to $\Psi_S(P, \sigma)$ jest określone i spełnia warunek γ.

2. **Zapytanie *always executable* P:**

   - Zwraca *TRUE*, jeśli dla każdego modelu $S = (\Sigma, \sigma_0, \text{Res})$ zgodnego z D, odwzorowanie $\Psi_S(P, \sigma_0)$ jest zawsze określone.

3. **Zapytanie *realisable* P by G:**

   - Zwraca *TRUE*, jeśli dla każdego modelu $S = (\Sigma, \sigma_0, \text{Res})$ zgodnego z D spełnione są:
     - dla każdej grupy G_i pojawiającej się w P, istnieje $G \in \hat{G}$, takie że $G_i \subseteq G$ i $G \subseteq G_i$ (czyli są to grupy równoważne),

– dla każdej akcji A_i w programie P istnieje stan $\sigma \in \Sigma$, dla którego $\Psi_S((A_i, G), \sigma)$ jest określone.

4. **Zapytanie *always accessible* γ from π in P:**

   - Zwraca *TRUE*, jeśli dla każdego modelu $S = (\Sigma, \sigma_0, \text{Res})$, każdego stanu $\sigma \in \Sigma$ oraz każdego odwzorowania częściowego Ψ_S, spełnienie π w σ oznacza, że $\Psi_S(P, \sigma)$ jest określone i prowadzi do stanu spełniającego γ.

5. **Zapytanie *sometimes executable* P:**

   - Zwraca *TRUE*, jeśli dla każdego modelu $S = (\Sigma, \sigma_0, \text{Res})$ zgodnego z D istnieje pewne odwzorowanie częściowe Ψ_S, dla którego $\Psi_S(P, \sigma_0)$ jest określone.

Zaznaczmy, że przez $A \in P$ lub $G \in P$ rozumiemy, że dana akcja A lub grupa G pojawia się w programie P — istnieje para w P, której jednym z elementów jest odpowiednio A lub G.
Jeśli warunki określone przez zapytanie Q nie zostaną spełnione, to wynik zapytania jest *FALSE*.

7 Implementacja

System zostanie zaimplementowany w języku C#, który zapewnia automatyczne zarządzanie pamięcią (*garbage collector*) oraz szablonowe struktury danych, takie jak słownik czy stos. Ponadto środowisko .NET udostępnia wiele specjalizowanych *framework'ów*. Jednym z nich jest Windows Forms, z wykorzystaniem którego zostanie zaimplementowany graficzny interfejs użytkownika wraz z krótką instrukcją obsługi.

8 Testy

W ramach sprawdzania poprawności działania aplikacji do systemu zostanie wprowadzonych parę wcześniej przygotowanych scenariuszy. Każdy członek projektu przygotuje po kilka testów wraz z opisem i oczekiwanymi wynikami. Zagwarantuje to gruntowną niezawodność systemu oraz jego poprawność. Tak otrzymane wyniki testów zostaną zebrane i opisane, co pozwoli na wyciągnięcie wniosków i test funkcjonalności zaimplementowanego systemu.

9 Podział pracy

Poniżej zaprezentowano podział pracy w ramach części teoretycznej dokumentacji:

Osoba odpowiedzialna	**Zakres pracy**
Mateusz Tkaczyk	Wstęp, Opis systemu, Struktura projektu, Problemy ramowe
Kacper Król	Wstęp, Opis systemu, Syntaktyka języka akcji
Kamil Kamiński	Problemy ramowe
Igor Staręga	Semantyka języka akcji, Składnia i Semantyka języka zapytań, Problemy ramowe
Jakub Kępka	Problemy ramowe
Stanisław Gwiazda	Problemy ramowe
Piotr Baranowski	Problemy ramowe

Table 1: Podział pracy między członków zespołu zajmujących się częścią teoretyczną projektu