POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Informatyki

Praca dyplomowa licencjacka

ZASTOSOWANIE UCZENIA ZE WZMOCNIENIEM W ZADANIU WERYFIKACJI MODELOWEJ

Paweł Błoch, 145375

Promotor
r Iwo Błądek

POZNAŃ 2024

Tutaj będzie karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wstęp

Gałaź nauki zajmujące się "Model Checking"to obecnie bardzo szybko rozwijająca się gałąź nauki. Jest to spowodowane licznymi zastosowaniami w których można skorzystać z narzędzii które dostarcza werfikacji modeli. Obecnie zastosowania jakie możemy znaleźć między innymi to

- gry planszowe [1]
- planowanie misji dla autonomicznych agentów [2]
- analiza i dowodzenia poprawności kodów źródłowych [3].

Celem który przyświecał podczas tworzenia tej pracy była próba stworzenia alternatywnych sposobów werfikacji modeli, skuteczniejszych od tych obecnie istniejących i wyznaczających standardy. Laureatka Nagrody Nobla Françoise Barré-Sinoussi w dziedzinie medycny powiedziała

> We are not making science for science. We are making science for the benefit of humanity.

Ciekawym obszarem jest analiza i dowodzenie poprawności kodów programistycznych. Zwłaszcza w obecnych czasach gdy ilość kodu wytrzwarzana przez progmiastów z całego świata jest bardzo duża. Rok do roku liczba projektów programistycznych rozpoczętych przez programistów wzrasta w dynamicznym tempie. Jednym z powodów jest pojawienie się ostatio modeli sztucznej inteligencji potrafiących automatycznie generować oraz odpowiadać na pytania. Więcej o statystykach można poczytać w [4].

W tej pracy główny nacisk został położony na anlizę gier planszowych. Będziemy próbowali odpowiedzieć na pytanie czy i w jakim zakresie algorytmy uczenia ze wzmocnienieum i/lub algorytmy heurystyczne są w stanie polepszyć narzędzie typu "model checker". Narzędzie te umożliwiają automatyczną werfikację modeli. Narzędzie te przyjmują na wejściu parę składającą się z modelu oraz formuły która będzie badana dla danego modelu. Obecnie istnieje kilka tego typu narzędzi których opis znajdzie się w dalszej cześci pracy.

Rozdział 2

Podstawy teoretyczne

2.0.1 Logika

Aby mówić o werfikacji modeli musimy się wyposażyć w elementarz teoretyczny. Potrzebne nam są podstawowe definicje i pojęcia zanim zaczniemy mówić o skomplikowanych modelach. Ważne jest również wprowadzenie i usystematyzowanie informacji z różnych źródeł. Rodział ten został oparty na świetnej książce [5] i będzie stanowił fundament dalszej tej pracy. Podstawowe elementy logiki można znaleźć w świetnej książce [6, Podstawy Logiki] do której serdecznie odsyłam.

System wieloagentowy (ang. multi-agent system, MAS) to system składający się z autonomicznych podmiotów, działających w tym samym środowisku. Jest on wykorzystywany do modelowania i opisywania systemów z którymi spotykamy się w świecie informatyki i nie tylko. Agenci mogą podejmować akcje w jednym środowisku. Przykładem takiego środowiska mogą być wybory, gdzie grupa n wyborców (agentów) bierze udział w procesie wyborczym.

Logika modalna którą będziemy się posługiwali jest rozszerzeniem klasycznej logiki o nowe operatory $\Box$ i konieczności $\diamond$. Formuła $\Box\phi$ oznacz, że ϕ jest prawdziwa zawsze w każdym stanie. Formuła $\diamond\phi$ oznacza, że ϕ jest prawdziwe w pewnym stanie. Spełniona jest przy tym zasada

$$\Diamond\varphi \iff \neg\Box\neg\varphi$$

Wprowdzaimy teraz definicję modele Kripkiego.

Definicja 2.0.1 (Model Kripkiego) *Niech $PV = p, p', p'', \ldots$ będzie zbiorem zmiennych zdaniowych. Modele logiki modalnej nazywane są modelami Kripkego lub modelami możliwych światów i obejmują zbiór możliwych światów (lub stanów) St, modalną relację dostępności $R \subseteq St \times St$ oraz interpretację zmiennych $\mathcal{V} : PV \to 2^{St}$.*

Jeśli trójka $M = (St, R, V)$ będzie modelem kripkiego i q będzie możliwym słowem w M, to prawdziwość formuły M, q jest dana relacją $\models$ i jest zdefiniowana indukcyjnie przez zasady

$M, q \models p$	wtedy i tylko wtedy, gdy	$q \in V(p)$,	dla	$p \in \mathcal{PV}$;				
$M, q \models \neg\varphi$	wtedy i tylko wtedy, gdy	nie	$M, q \models \varphi$	(często pisane	$M, q \not\models \varphi$);			
$M, q \models \varphi \wedge \psi$	wtedy i tylko wtedy, gdy	$M, q \models \varphi$	i	$M, q \models \psi$;				
$M, q \models \varphi \vee \psi$	wtedy i tylko wtedy, gdy	$M, q \models \varphi$	lub	$M, q \models \psi$;				
$M, q \models \Box\varphi$	wtedy i tylko wtedy, gdy	dla każdego	$q' \in St$	takiego, że	qRq',	mamy	$M, q' \models \varphi$;	
$M, q \models \Diamond\varphi$	wtedy i tylko wtedy, gdy	dla pewnego	$q' \in St$	takiego, że	qRq',	mamy	$M, q' \models \varphi$.	

Prosty przykład modelu Kripkiego może wyglądać tak:

2.0.2 Uczenie ze wzmocnieniem

Uczenie ze wzmocnieniem to dynamicznie rozwijająca się część sztucznej inteligencji. Jest to proces, w którym agent ucz się jakie działania podejmować w środowisku, aby uzyskać jak największą nagrodę. Agent uczy się w pewien sposób metodą prób i błędów, nie jest dla niego jasne które akcje przyniosą mu największe nagrody. Problemy ten charakterysuje się opóźnioną nagrodą. Oznacza to, że za akcję podjęte w pewnym momencie wpływają nie tylko na bieżącę nagrody, ale również na przyszłe stany i nagrody z nimi związane.

Uczenie ze wzmocnieniem samo w sobie jest szerokim zagadnieniem, zawiera w sobie zarówno sam problem, klasę metod którą są w stanie rozwiązać problem jak i całą dziedzinę nauki. Kluczowym aspektem jest rozróżnienie pomiędzy problemem a metodami służącymi do jego rozwiązania.

Problemy uczenia ze wzmocnieniem formalizujemy za pomocą procesów decyzyjnych Markova. Agent musi w pewien sposób odczuwać stan środowiska, podejmować działania wpływające na środowisko i mieć cele związane ze stanem środowiska. Procesy decyzyjne Markova uwzględniają te trzy aspekty.

Uczenie ze wzmocnieniem rózni się od dobrze znanego uczenia nadzorowanego, w którym system uczy się bazując na podstawie poetykietownych przez nadzorcę przykładów ze zbioru uczącego. Różni się również od uczenia nienadzorowanego, które zazwyczaj polega na znajdowaniu struktur i wzorców w nieoznaczonych zbiorach danych. Uczenie ze wzmocnieniem dąży uzyskania jak największej nagrody, a nie do odrykwania struktur w zbiorach danych.

Jednym z wyzwań przed którym stajemy wykorzystując uczenie ze wzmocnieniem jest przetarg pomiędzy eksploracją i eksplatacją. Agent musi wybierać pomiędzy akcjami, które przyniosły mu już nagrodę, a wybieraniem nowych akcjii i stanów których jeszcze nie zna, a te mogą mu przynieść nagrody w przyszłości. Ani eksplatacją ani eksploracja nie mogą być wykonywane wyłącznie, gdyż uniemożliwi to agentowi uzsykanie celu.

2.0.3 Algorytmy uczenie ze wzmocnieniem

Podczas badań zostały wykorzystane różne algorytmy uczenia ze wzmocnieniem. Poniżej zostanie dokonana ich krótka charakterystyka.

AlphaZero

Algorytm AlphaZero jest zaawansowany algorytmem uczenia ze wzmocnieniem. Zmień on sposób myślenia w tej gałęzi nauki. Pokazał on, że sztuczna inteligencja jest w stanie osiągnąć poziom nie możliwy do osiągnięcia dla człowieka w grach planszowych dla dwóch graczy. W artykule [12] zaprezentowane zostałe osiągnięcia dla dobrze znanych gier jak szachy, shogi czy go. Algorytm ten nauczył się grać bez wiedzy dziedzinowej. Proces nauki polega na grze z samym sobą, co stopniowo prowadzi do polepszania swoich umiejętności. AlphaZero składa się z kilku kluczowych elementów.

Głęboka sieć nueronowa ocenia pozycję na planszy, prognozuje prawodopodobieństwa wykonania ruchu i ocenia wynik gry. Sieć możemy oznaczyć jako $(p, v) = f_\theta(s)$. Parametry θ to parametry sieci, s to stan gry przyjmowany na wejście, a wynikiem działania sieci jest wektor prawdopodobieństwa p oznaczający prawdopodobieństwo wykonania ruchów oraz wartość oczekiwaną pozycji v.

Klasyczny algorytm przeszukiwania drzewa *alfa-beta* został zastąpiony przez *Monte Carlo Tree Search (MCTS)* który jest bardziej ogólnym podejściem. Każda symulacja w MCTS wybiera

ruchy na podstawie małej liczby odwiedzin (ruchów dotychczas mało eksplorowanych), wysokiego prawdopodobieństwa ruchu i wysokiej wartości stanu.

Parametry θ sieci neuronowej są aktualizowane na podstawie wyników rozegranych gier oraz różnicy pomiędzy przewidywanym wynikiem v a rzeczywistym wynikiem gry. Minimalizowana jest minimalizacja funkcji straty, która składa się z błędu średniokwadratowego i straty entropii krzyżowej

$$l = (z - v)^2 - \sum_a p_a \log q_a + c||\theta||^2$$

gdzie c jest parametrem regularyzacji $L2$.

Alphazero rozpoczyna proces treningu od losowych ruchów bez posiadania wiedzy domenowej oprócz reguł gry. Wraz z procesem treningu AlphaZero zwiększa swoje umiejętności wskutek aktualizacji wag modelu. Algorytm ten jest algorytmem który bardzo szybko się uczy. Dowodzi tego fakt, że zaledwie po czterech godzinach treningu model był w stanie pokonać dotychczas najlepszy silnik szachowy *Stockfish*

Rozdział 3

Rozwinięcie

3.1 Wprowadzenie do problemu

Celem tej sekcji jest porównanie werfikacji modelowej i technik uczenia ze wzmocnieniem w konteście analizy modeli reprezentujących grę dla dwóch graczy. Przeanalizujemy znaną grę Kółko i Krzyżyk. Analiza będzie polegała na wykonaniu dwóch eksperymentów. Pierwszym w nich będzie zamodelowanie gry jako modelu logicznego. Zostanie to wykonane i opisane w formacie *Interpreted Systems Programming Language (ISPL)*. Po zamodelowaniu będzie konstruowali formuły do sprawdzenia przez model checker **MCMAS** które będą odpowiadały problemowi znajdowania strategii wygrywającej dla jednego z graczy. Będziemy zaczynali w pewnej pozycji, początkowej lub planszą częściowo zajętą. Pytanie o strategię wygrywającą, jest równoznaczne z pytaniem czy istnieje strategia dla danego gracza która niezależnie od strategii drugiego gracza doprowadzi go do zwycięstwa. Będziemy również wykorzystywali algorytmy uczenia ze wzmocnieniem do nauki modelu gry. Następnie będziemy rozgrywali gry pomiędzy nauczonymi agentami (lub losowymi) i na podstawie serii wyników będzie stawiali hipotezę co do prawdziwości danej formuły.

Wybór gry kółko krzyżyk jako pierwszego środowiska jest umotywowany kilkoma aspektami. Gra kółko krzyżyk jest znane praktycznie każdemu i to sprawia, że próg wejścia jest niski. Co więcej każdy czytelnik zrozumie zagadnienie które będziemy omawiali. Sama gra pozostaje cały czas interesującym tematem badań i opracowań. Możemy je znaleźć między innymi w [7], [8], [9]. Gra jest cały czas problemem otwartym. Dokładniej mówiąc, otwarty jest problem dowiedzenia czy istnieje strategię wygrywającą dla gier o różnym rozmiarze. Jak podaje [10] nadal otwartym problemem pozostaje gra $5 \times 5 \times 5$. Dowiedzione zostałe strategie dla gier $3 \times 3 \times 3$ oraz $4 \times 4 \times 4$. W obu tych grach istnieje strategia wygrywająca dla graczy rozpoczynających rozgrywkę. Dla gry $5 \times 5 \times 5$ przypuszcza się, że będzie to gra remisowa.

3.1.1 Model checker STV

Narzędzi *stv* [13] to jedno z najlepszych dostępnych narzędzi do werfikacji modelowej. Jego współautorem jest polski naukowiec prof. dr hab. Wojciech Jamroga. Zalicza się on do grona najlepszych ekspertów w tej tematyce. Jest on również autorem wielu publikacji z tego zakresu. Narzędzi *stv* posłuży nam jako pewien punkt odniesienie w prównaniu z innymi narzędziami i metodami.

Model gry - STV

Model gry zapisujemy w języku inspirowanym na języku *ISPL*, nie mniej jest to autorski pomysł i różnicą się od *ISPL*. Podobnie definiujemy agentów, formułę, inny jest mechanizm komunikacji

pomiędzy agentami. Przedstawię najważniejsze fragmenty modelu wraz z ich opisem.

```
Agent P1:
LOCAL: [last_move_1,P1_00,P1_01,P1_02,P1_10,P1_11,P1_12,P1_20,P1_21,P1_22]
...
INITIAL: [last_move_1:=0,P1_00:=0,P1_10:=0,P1_20:=0,P1_01:=0,P1_11:=0,P1_21:=0,P1_02:=0,P1_12:=0
```

Definicja agenta rozpoczyna się jego nazwy, następnie definiowane są zmienne lokalne naszego agenta. Po definicji zmiennych lokalnych w sekcji *INITIAL* ustawione są wartości początkowe zmienncyh lokalnych.

```
init idle
my_turn_1: idle -> start_turn
 play_local_1_00: start_turn[P1_00 == 0 ] -> end_turn[P1_00:=1, last_move_1:=0]
 play_local_1_01: start_turn[P1_01 == 0 ] -> end_turn[P1_01:=1, last_move_1:=1]
 ...
 play_local_1_22: start_turn[P1_22 == 0 ] -> end_turn[P1_22:=1, last_move_1:=8]
```

Przechodzimy do akcji i stanów agenta. Określenie stanu początkowego 'init idle' wskazuje, na kto który stan będzie stanem początkowym. Składania języka jaki obowiązuje dla akcji jest taka :

```
akcja : stan poczatkowy[warunki] -> stan koncowy[aktualizacja]
```

Akcja przeprowadza agenta ze stanu w jakim się znajduje jeśli spełnia warunki do wykonania akcji. Po wykonaniu akcji zostaje wprowadzony nowy stan i zostają zaktualizowane zmienne lokalne. Dla każdego stanu w którym jest agent zdefiniowane są akcje które może wykonać. W naszym modelu agent wykonuje ruch jako jedną z n^2 akcji jeśli pole jest wolne. Po wykonaniu ruchu oznacza pole jako zajętę w swojej zmiennej lokalnej.

```
shared[3] play_1_00[play]: end_turn[last_move_1==0] -> wait
shared[3] play_1_01[play]: end_turn[last_move_1==1] -> wait
...
shared[3] play_1_22[play]: end_turn[last_move_1==8] -> wait
...
shared[3] play_2_00[play]: wait[P1_00 == 0] -> start_turn[P1_00:=2]
shared[3] play_2_01[play]: wait[P1_01 == 0] -> start_turn[P1_01:=2]
...
shared[3] play_2_22[play]: wait[P1_22 == 0] -> start_turn[P1_22:=2]
```

Następnym krokiem po wykonaniu ruchu przez Agenta jest przekazanie tej informacji do pozostałych agentów. Składania narzędzia *STV* nie umożliwia wprowadzania zmienncyh globalnych, czy innego systemu współdzielenia pamięci pomiędzy kilku agentów. Do przekazywania informacji wykorzystany został mechanizm akcji współdzielonych, czyli takich które zostają wykonane przez pewną ilość agentów razem. W naszym przypadku liczbę agentów krzórzy wykonają akcje współdzieloną określamy przez *shared[3]*. W tym kroku również mamy n^2 możliwych akcji, nie mniej tylko jedna z nich jest możliwa do podjęcia , poprzez kontrolę wartości zmiennej *last_move_x*. Akcje z prefiksem *play_1* oznaczają, że ruch wykonał agent *P1*, a akcje z prefiksem *play_1* oznaczają, że akcje wykonał agent *P2*. Taki mechanizm pewnej synchronizacji umożliwia przekazywania pomiędzy agentami informacji o wykonanych ruchach i aktualizację ich pamięci lokalnej.

Model agenta *P2* jest symetryczny do agenta *P1*. Trzecim agentem będącym w środowisku jest agent *Env*. Odpowiada on za kontrolę planszy.

```
shared[3] play_1_00[play]: turn_1[P3_00 == 0 && end ==0] ->
check_score_1[round := round+1, P3_00:=1]
 ...
shared[3] play_1_22[play]: turn_1[P3_22 == 0 && end ==0] ->
check_score_1[round := round+1, P3_22:=1]
shared[3] play_2_00[play]: turn_2[P3_00 == 0 && end ==0] ->
```

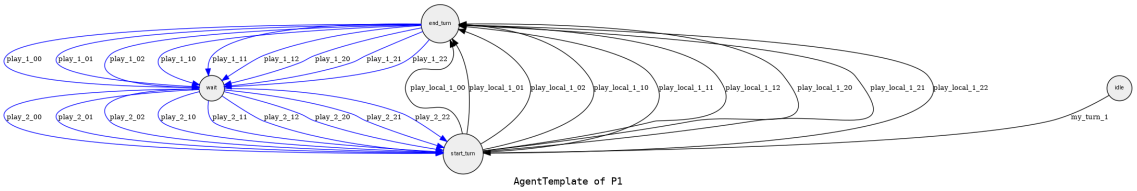
```
check_score_2[round := round+1, P3_00:=2]
...
shared[3] play_2_22[play]: turn_2[P3_22 == 0 && end ==0] ->
check_score_2[round := round+1, P3_22:=2]
```

Agent *Env* wykonuje akcje współdzielone z pozostałymi agentami które służą do synchronizacji informacji. Agent ten kontroluje to co się dzieje na planszy, liczy wykonane ruchy i śledzi stan rozgrywki Po każdy wykonanym ruchu dokonywana jest przez tego agenta kontrola stanu planszy w celu stwierdzenia przez niego czy wystąpił stan końcowy.

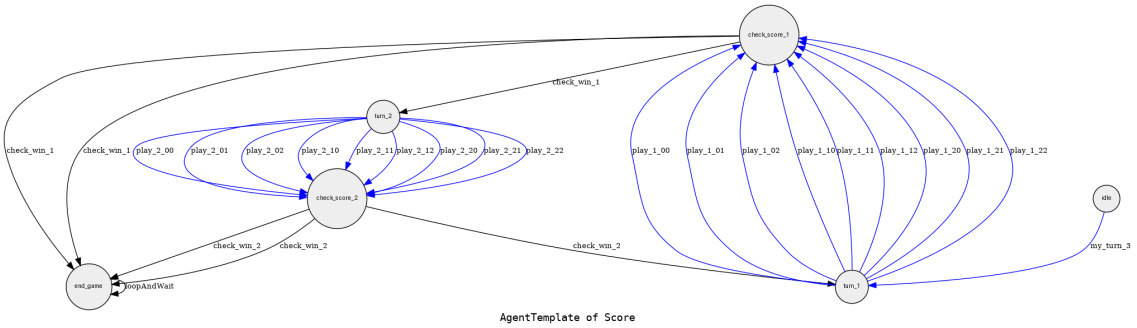
```
check_win_1 : check_score_1[(
(P3_00==1 && P3_01==1 && P3_02==1) ||
(P3_10==1 && P3_11==1 && P3_12==1) ||
(P3_20==1 && P3_21==1 && P3_22==1) ||
(P3_00==1 && P3_10==1 && P3_20==1) ||
(P3_01==1 && P3_11==1 && P3_21==1) ||
(P3_02==1 && P3_12==1 && P3_22==1) ||
(P3_00==1 && P3_11==1 && P3_22==1) ||
(P3_02==1 && P3_11==1 && P3_20==1))] ->
end_game[P1_WIN:=1, end:=1]
```

Klasyczne sprawdzenia planszy odbywa się przez sprawdenie n kolumn, n wierszy i dwóch głównych przekątnych. Jeśli któryś z warunków zostanie spełniony to gra zostaje zakończona, a zwycięzca zostaje wskazany. Jeśli żaden z warunków niezostałby spełniony, a na planszy znajdowałyby się wolne pola, to do następnej tury zostałby dopuszczony przeciwny gracz. Jeśli na planszy nie znajdowałyby się wolne pola, to gra zostałby zakończona remisem.

Narzędzie *STV* umożliwia wygenerowanie różnego rodzaju grafów. Poniżej pokazane zostaną gfrafy lokalne stanów/akcji dla gracza i środowiska.



Rysunek 3.1: Model agenta P1



Rysunek 3.2: Model agenta Env

Na grfach możemy sprawdzić czy nasz zaprojektowany model działa tak jak zaplanowaliśmy. Jest to też ważna funkcjonalnośc w kontekście debugowania i analizy działania modelu. W naszym kontekście możemy przeanalizować, czy nasz model zadziała dobrze w kontekście naszej gry w kółko i krzyżyk.

Eksperymenty - STV

Analizę rozpoczniemy od analizy czasu jakiego potrzebuje *stv* do werfikacji formuły. Pomiary prowadzimy dla gry kółko i krzyżyk na planszy o rozmiarach 3×3. Dla liczby wolnych pól powyżej

Wolne pola	Czas wykonania (s)
1	0.007
2	0.01
3	0.01
4	0.02
5	0.07
6	0.45
7	4.35
8	32
9	522

Tablica 3.1: Wyniki checkera stv dla gry na planszy 3x3

7 czas werfikacji przekrasza $1s$, dla liczby wolnych 9 czas wynosi około 10 minut. Oznacza, to że do werfikacji czy istnieje strategia wygrywająca dla danego gracza w grze kółko i krzyżyk na planszy 3×3 potrzebny jest czas 10 minut.

3.2 Model gry

Gra została zamodelowana w języku *ISPL*. Jest to język który jest wykorzystywany przez narzędzie *MCMAS*. Przedstawimy kluczowe fragmenty modelu dla lepszego zrozumienia problemu i tego, w jakis posób działa werfikacka modelu.

```
Agent Environment
  Obsvars:
    turn : {nought, cross};
    b11 : {x, o, b}; b12 : {x, o, b}; b13 : {x, o, b}; b14 : {x, o, b}; b15 : {x, o, b};
    b21 : {x, o, b}; ...
   ...
  end Obsvars
```

Agent *Environment* przechowuje dwa typy zmiennych. Agent posiada zmienną typu *enumerate* czyli typu wyliczanego która przechowuje informację o tym, który z graczy jest aktualnie na ruchu. Kolejne zmienne przechowują informację, o stanie danego pola na planszy. Zmienna bij przechowuje informację na temat pola w i-tym wierszu i -jtej kolumnie. Standard $ISPL$ nie dopuszcza tablic, dlatego planszę musimy implementować za pomocą n^2 zmiennych.

```
  Evolution:
    turn=nought if turn=cross; turn=cross if turn=nought;
    b11 = o if turn = nought and Nought.Action = a11;
    b11 = x if turn = cross  and Cross.Action  = a11;
    ...
    b55 = o if turn = nought and Nought.Action = a44;
    b55 = x if turn = cross  and Cross.Action  = a44;
  end Evolution
end Agent
```

Tury graczy zmieniją się na przemian. Agent *Environment* pobiera od agentów graczy akcji jakie podjeli i aktualizuje planszę gry jak i aktualną turę.

```
Agent Nought
  Vars:
```

```
      null : boolean; -- for syntax reasons only
    end Vars
    Actions = {a11,a12,a13,a14,a15,a21,a22,a23,a24,
               a25,a31,a32,a33,a34,a35,a41,a42,a43,a44,a45,a51,a52,a53,a54,a55, };
    Protocol:
      Environment.b11=b:{a11}; Environment.b12=b:{a12}; ... Environment.b15=b:{a15};
      ...
      Environment.b51=b:{a51};
    end Protocol
  end Agent
```

Powyżej znajduje się szkic implementacji agenta *Nought*. Implementacja agenta *Cross* jest symetryczna. Agent gracza ma do wyboru n^2 akcji typu aij. Każda taka akcja oznacza wykonanie ruchu i postawienie znaku w i-tym wierszu i j-tej kolumnie. Akcja taka może zostać wykonana jeśli pole na planszy jest puste.

```
  Evaluation
    noughtwins if
      Environment.b11 = o and Environment.b12 = o and Environment.b13 = o or
      Environment.b14 = o and Environment.b15 = o or
      ...
      Environment.b11 = o and Environment.b21 = o and Environment.b31 = o or
      Environment.b41 = o and Environment.b51
      ...
      Environment.b11 = o and Environment.b22 = o and Environment.b33 and
      Environment.b44 = o and Environment.b55 = o
  end Evaluation
```

Sekcja ewaluacji zawiera w sobie warunki na zakończenie gry. Gracz wygrywa jeśli zapełni jedną kolumnę, wiersz lub główną przekątną swoimi znakami. Warunków na zakończenie gry jest $2n+2$. Warunki zakończenia dla obu graczy są takie same, różnią się tylko znakami.

```
      InitStates
  Environment.b11=b and Environment.b12=b and Environment.b13=b
  and Environment.b14=b and Environment.b15  = b
  ...
  and Environment.turn = cross
end InitStates
```

Sekcja *InitStates* zawiera w sobie stan początkowy modelu. Wszystka pola planszy są puste, a jeden z graczy jest graczem startowym.

```
Groups
  nought = {Nought}; cross = {Cross};
end Groups

Formulae
  <cross> F (crosswins and ! noughtwins); -- TRUE
  <nought> F (noughtwins and ! crosswins); -- FALSE
end Formulae
```

Ostatnim elementem modelu jest zadeklarowanie koalicjii agentów. W naszym przypadku koalcjia agentów składa się z jednego gracza. Formuły jakie badamy, to formuły które sprawdzają czy dany gracz wygra. Zawsze sprawdzamy dwie formuły, aby uzyskać odpowiedź dla każdego z dwóch graczy.

3.2.1 Badania

Analizę wyników zaczniemy od analizy wyników uzyskanych za pomocą narzędzia *MCMAS*. Badając rozgrywkę dla pustej planszy $n \times n$ już dla $n = 5$ dostajemy timeout. Jest to związane

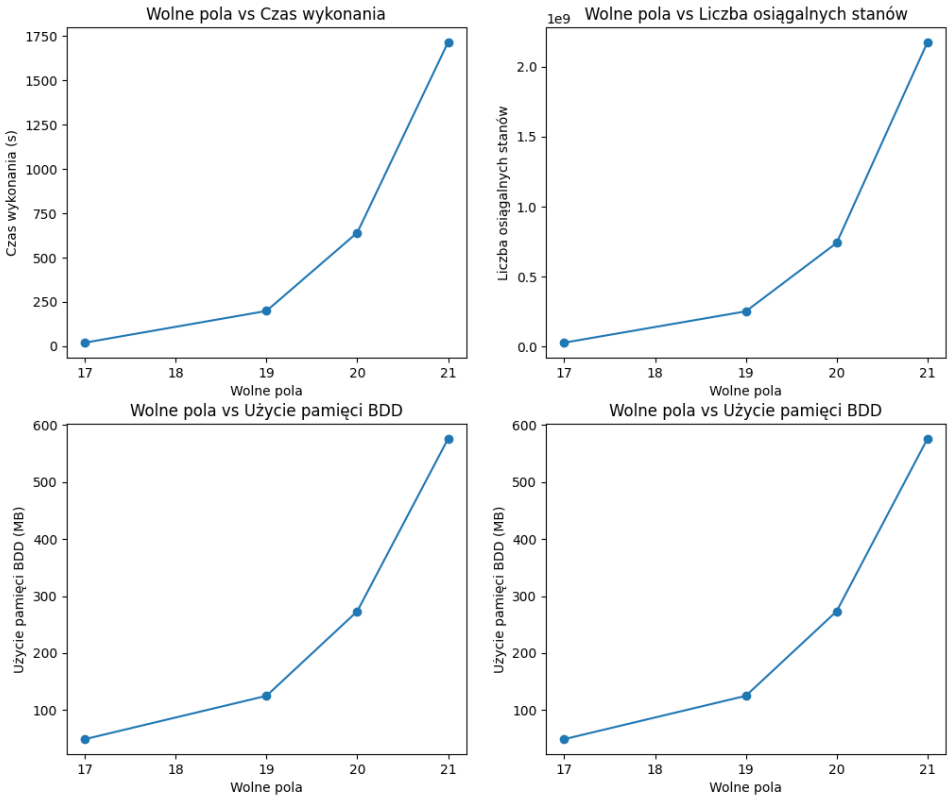
Rozmiar planszy	Czas wykonania (s)	Liczba osiągalnych stanów	Użycie pamięci BDD
2x2	0.017	41	8.8 MB
3x3	0.325	6172	13.2 MB
4x4	193.383	1.01786e+07	168 MB
5x5	timeout	timeout	timeout

Tablica 3.2: Wyniki checkera MCMAS

wykładniczą eksplozją stanów jakie *MCMAS* ma do przeanalizowania. Eksplozja stanów jest bezpośrednio związana z liczbą pól na jakich gracze mogą dokonywać ruchów. Do dalszej analizy wykorzystamy częściowo zapełnioną planszę o rozmiarze 5×5 z pozostałymi 17 polami lub więcej.

Wolne pola	Czas wykonania (s)	Liczba osiągalnych stanów	Użycie pamięci BDD (MB)
17	20.644	2.96682e+07	49.41
19	199.467	2.5328e+08	125.25
20	641.051	7.4155e+08	273.37
21	1714	2.1736e+09	575.39

Tablica 3.3: Wyniki checkera MCMAS



Rysunek 3.3: Wykres wyników dla MCMAS

Rozdział 4

Zakończenie

Zakończenie pracy zwane również Uwagami końcowymi lub Podsumowaniem powinno zawierać ustosunkowanie się autora do zadań wskazanych we wstępie do pracy, a w szczególności do celu i zakresu pracy oraz porównanie ich z faktycznymi wynikami pracy. Podejście takie umożliwia jasne określenie stopnia realizacji założonych celów oraz zwrócenie uwagi na wyniki osiągnięte przez autora w ramach jego samodzielnej pracy.

Integralną częścią pracy są również dodatki, aneksy i załączniki zawierające stworzone w ramach pracy programy, aplikacje i projekty.

Literatura

[1] Schlingloff, B.-H. (2021). Teaching Model Checking via Games and Puzzles. In A. Cerone & M. Roggenbach (Eds.), Formal Methods - Fun for Everybody (pp. 143-158). Springer International Publishing.

[2] Gu, R., Enoiu, E., Seceleanu, C., & Lundqvist, K. (2020). Verifiable and Scalable Mission-Plan Synthesis for Autonomous Agents. In M. H. ter Beek & D. Ničković (Eds.), Formal Methods for Industrial Critical Systems (pp. 73-92). Springer International Publishing.

[3] Chong, N., Cook, B., Kallas, K., Khazem, K., Monteiro, F. R., Schwartz-Narbonne, D., Tasiran, S., Tautschnig, M., & Tuttle, M. R. (2020). Code-level model checking in the software development workflow. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, 11-20. https://doi.org/10.1145/3377813.3381347

[4] GitHub. (2023). The State of Open Source and AI. https://github.blog/2023-11-08-the-state-of-open-source-and-ai/

[5] Jamroga, W. (2015). Logical Methods for Specification and Verification of Multi-Agent Systems. ICS PAS Publishing House.

[6] Batóg, T. Podstawy logiki. (Wydaw. Naukowe UAM,1994)

[7] Deepa, R., Velnath, R., Manojkumar, P. & Mohanraj, K. Artificial Neural Network-Based Tic-Tac-Toe Game. *Proceedings Of International Conference On Communication And Computational Technologies* . pp. 55-65 (2023)

[8] Dalffa, M., Abu-Nasser, B. & Abu-Naser, S. Tic-Tac-Toe Learning Using Artificial Neural Networks. *International Journal Of Engineering And Information Systems (IJEAIS)*. **3**, 9-19 (2019)

[9] Fogel, D. Using evolutionary programing to create neural networks that are capable of playing tic-tac-toe. *IEEE International Conference On Neural Networks*. pp. 875-880 vol.2 (1993)

[10] Beck, J. Combinatorial Games: Tic-Tac-Toe Theory. (Cambridge University Press,2008)

[11] Sutton, R. & Barto, A. Reinforcement Learning: An Introduction. (MIT Press,1998)

[12] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. & Others Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv Preprint ArXiv:1712.01815*. (2017)

[13] Kamiński, M., Kurpiewski, D. & Jamroga, W. STV+KH: Towards Practical Verification of Strategic Ability for Knowledge and Information Flow. *Proceedings Of The 23rd International Conference On Autonomous Agents And Multiagent Systems*. pp. 2812-2814 (2024)

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu LaTeX na platformie Overleaf.