2021

Metody Sztucznej Inteligencji. 1. System agentowy w logice

1. SYSTEM AGENTOWY W LOGICE

WŁODZIMIERZ KASPRZAK

AGENT SZTUCZNEJ INTELIGENCJI, BAZA WIEDZY, WYNIKANIE I WNIOSKOWANIE, PRZESTRZEŃ STANÓW, UCZENIE, KATEGORIE AGENTÓW, CHARAKTERYSTYKA ŚRODOWISKA, RACHUNEK ZDAŃ, LOGIKA PREDYKATÓW

W pierwszym module bloku MSI przedstawione zostanie pojęcie i struktura systemu agentowego Sztucznej Inteligencji. Wyróżnione zostaną podstawowe zagadnienia procesu projektowania systemu agentowego: język reprezentacji wiedzy, reguły i procedury wnioskowania, realizacja funkcji agentowej (sterowania) w postaci algorytmów przeszukiwania przestrzeni stanów, poszukiwania celu i planowania działań, uczenie się własności i pojęć dziedziny zastosowania oraz uczenie się strategii działania. Przedstawione zostaną kategorie „systemów agentowych” i kryteria oceny złożoności środowiska działania agenta. Następnie skupimy się na projekcie agenta bazującego na logicznej bazie wiedzy. Przedstawiony zostanie najprostszy język logiczny (rachunek zdań) – jego składnia i semantyka, typowe reguły wnioskowania i korzystające z nich funkcje

Spis treści

1 System agentowy

1.1 Wstęp

Pionierskie prace Alana Turinga [1] zaowocowały zdefiniowaniem głównych zadań obszaru wiedzy zwanego „Sztuczną Inteligencją": reprezentacja wiedzy, wnioskowanie, przeszukiwanie, uczenie. Badania naukowe i rozwój metod w tym obszarze doprowadziły do rozwiązań technicznych znanych jako „systemy ekspertowe" i „systemy agentowe". Zasadniczą różnicę pomiędzy nimi można wyrazić poprzez sformułowanie celu ich pracy: „myśleć racjonalnie" (system ekspertowy) i „działać racjonalnie" (system agentowy). Ten drugi cel jest szerszy - racjonalne działać oznacza „wykonywać właściwą akcję". Właściwa akcja to taka, która maksymalizuje osiągnięcie zadanego celu przy dostępnej informacji, a „racjonalne myślenie" powinno być częścią „racjonalnego działania". Główne zastosowania metod sztucznej inteligencji to:

1. **Systemy ekspertowe,** realizujące podejście „myśleć racjonalnie", wspomagające człowieka w problemach logistycznych, diagnostyce medycznej, reagowaniu w nieoczekiwanych sytuacjach lub podejmowaniu decyzji biznesowych;
2. **Systemy agentowe**, realizujące podejście "działać racjonalnie", mające wyższy poziom autonomii dzięki zdolności percepcji otoczenia, uczenia się i wykonywania akcji.

Oprócz podstawowych zadań Sztucznej Inteligencji, które sformułował Alan Turing, system agentowy wymaga nowych metod i technik przeznaczonych do percepcji środowiska (sensory – czujniki i analiza pozyskiwanych przez nie danych) i wykonywania akcji (efektory i ich sterowanie). Współpraca takiego systemu z człowiekiem wymaga korzystania z nowoczesnych sposobów komunikacji człowiek-maszyna opartych o rozpoznawanie i syntezę mowy oraz rozpoznawanie gestów człowieka w obrazach.

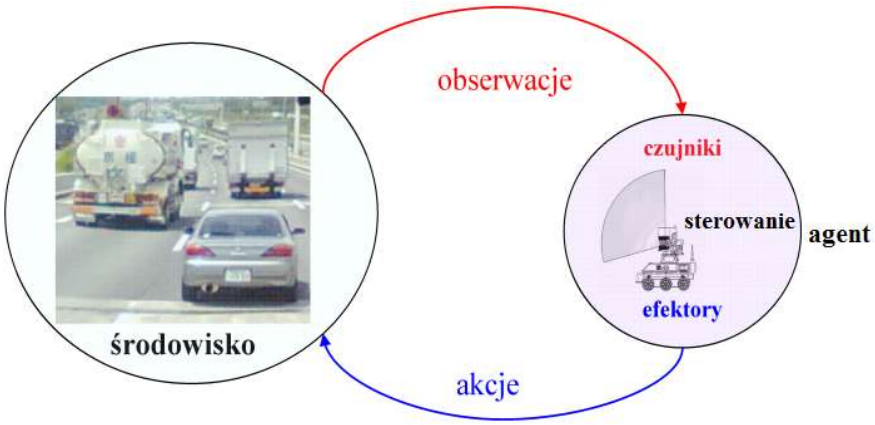
1.2 Agent Sztucznej Inteligencji

1.2.1 Funkcja agenta

Agentem Sztucznej Inteligencji (w skrócie: agentem) jest każdy obiekt (jednostka), który obserwuje (odbiera) swoje otoczenie (rozumiane jako obserwowany fragment środowiska) dzięki czujnikom (sensorom) i oddziałuje na to środowisko przy pomocy „efektorów" (wykonuje akcje). Formalnie biorąc, działanie agenta opisuje funkcja z dziedziny historii obserwacji (percepcji) w wykonywane akcje (Rysunek 1):

$$f: P^* \rightarrow A$$

Przez „system agentowy" będziemy rozumieli implementację agenta uwzgledniającą jego działanie w konkretnym środowisku.



Rysunek 1. Elementy systemu agentowego

Człowiek („*ludzki* agent") posiada oczy, uszy i inne organy pełniące role czujników oraz ręce, nogi, usta i inne części ciała będące efektorami. Agent upostaciowiony (robot, maszyna) posiada: kamery, czujniki podczerwieni, skanery laserowe, itp., będące czujnikami oraz różne silniki, napędy i manipulatory będące efektorami. Agent nieupostaciowiony (program komputerowy, *softbot*) to odpowiedni program wykonujący się na fizycznym komputerze (o określonej architekturze) i realizujący funkcję *f* agenta.

Przykład implementacji agenta upostaciowionego (robot „Shopbot") przedstawiono na Rysunek 2. Zadania agenta polegają na: rozpoznawaniu człowieka, prowadzeniu dialogu z człowiekiem za pomocą mowy, uczeniu się mapy otoczenia i strategii wyboru akcji oraz samodzielnej nawigacji do zadanego celu.

Rysunek 2. Przykład systemu agentowego - Shopbot (Niemcy) - robot społeczny

Głównym wyznacznikiem przy projektowaniu agenta będzie znalezienie najbardziej efektywnego działania w zależności od charakteru środowiska i rodzaju zadania, przy uwzględnieniu ograniczeń wynikających z dostępnych zasobów.

1.2.2 Racjonalny agent

Jedna klasa możliwych funkcji agenta ma charakter **racjonalny**. Idealnie racjonalny agent w oparciu o dany ciąg obserwacji i wiedzę o środowisku, w którym działa, powinien wykonywać akcje

prowadzące do maksymalizacji przyjętej przez niego **miary skuteczności**. Miara skuteczności (użyteczności) to pewne obiektywne kryterium oceny stopnia sukcesu w zachowaniu agenta. Np. dla „agenta odkurzającego" miarami skuteczności mogą być: usunięta masa zanieczyszczeń, czas pracy, ilość pobranej energii, ilość wygenerowanego szumu, itd. Racjonalność nie oznacza „nieomylności", gdyż nie mamy pełnej (nieskończonej) wiedzy o wszystkim. Stąd racjonalność powinna być wspierana przez eksplorację środowiska i uczenie się. Agenty mogą wykonywać akcje po to, aby zmodyfikować przyszłe obserwacje i w ten sposób pobrać pożyteczną informację (**eksploracja środowiska**). Agent jest autonomiczny wtedy, gdy na jego zachowanie wpływa własne doświadczenie (ma zdolności **uczenia się** i adaptowania do środowiska), np. poprzez samodzielne nabywanie wiedzy i uczenie się reguł wyboru akcji.

Podczas projektowania racjonalnego agenta powinniśmy uwzględnić następujące zagadnienia (Tabela 1):

1. charakter zadania,
2. miara skuteczności,
3. środowisko,
4. efektory,
5. czujniki.

Np. gdy naszym celem jest zaprojektowanie automatycznej taksówki to zapewne będziemy rozpatrywać następujące zagadnienia:

- agent upostaciowiony – robot mobilny na podłożu jezdnym;
- miara skuteczności: bezpieczny, szybki, zarejestrowany, komfortowa podróż, maksymalny zysk;
- środowisko: drogi, inne agenty, piesi, klienci;
- efektory: wyposażenie robota w silnik, sterowanie kierunkiem i przyspieszeniem, hamulce, sygnalizacja, sygnał dźwiękowy;
- czujniki: kamery, sonar, prędkościomierz, GPS, odometria, czujniki silnika, klawiatura.

Tabela 1. Charakterystyka wybranych racjonalnych agentów

Typ zadania	Percepcja	Akcje	Miara skuteczności	Środowisko
Diagnostyka medyczna	Objawy, badania, odpowiedzi pacjenta	Pytania, testy, leczenie	Zdrowie pacjenta, minimalizacja kosztów	Pacjent, szpital
Analiza obrazów satelitarnych	Piksele kolorowe obrazu	Klasyfikacja elementów sceny	Prawidłowa klasyfikacja	Obrazy z satelity
Manipulator sortujący	Obraz, sygnał czujnika	Uchwyć część, sortuj do pojemników	Części w prawidłowych pojemnikach	Taśma produkcyjna z częściami
Sterownik przemysłowy w rafinerii	Temperatura, ciśnienie	Otwórz/zamknij zawory, dopasuj temperaturę	Maksymalizacja stabilności i bezpieczeństwa	Rafineria
Zdalna nauka języka	Słowa pisane	Ćwiczenia, dyskusja, poprawianie	Maksymalizacja punktów z testu	Grupa studentów

1.2.3 Środowisko agenta

Własności środowiska w znacznym stopniu wpływają na projekt agenta. Dlatego spróbujemy podać kilka najważniejszych kryteriów dla charakteryzowania środowisk.

- W pełni **obserwowalne** lub częściowo obserwowalne

Środowisko jest w pełni obserwowalne, gdy czujniki agenta dostarczają pełnej informacji o stanie środowiska w każdej chwili czasu. Z zasady nie rozpatrujemy przypadku, gdy brak jest jakiejkolwiek możliwości obserwacji środowiska przez agenta.

- **Deterministyczne** lub niedeterministyczne (stochastyczne)

Determinizm środowiska oznacza, że następny stan środowiska jest w pełni (jednoznacznie) określony przez aktualny stan i akcję wykonywaną przez agenta. Jeśli środowisko jest deterministyczne za wyjątkiem akcji innych agentów to mówimy, że jest ono **strategiczne**.

- **Modularne** lub sekwencyjne

Wiedza agenta o środowisku składa się z atomowych "epizodów" - każdy epizod składa się z pojedynczej obserwacji agenta i następującej po niej sekwencji akcji agenta, czyli wybór akcji w każdym epizodzie zależy jedynie od samego (całościowego) epizodu.

- **Statyczne** lub dynamiczne

Statyczne środowisko nie ulega zmianie podczas zastanawiania się agenta – w czasie pomiędzy obserwacją a wykonaniem akcji. Pół-dynamiczne środowisko wprawdzie nie zmienia się w tym czasie ale zmienia się wynik funkcji skuteczności dla akcji agenta.

- **Dyskretne** lub ciągłe

W dyskretnym przypadku występuje ograniczenie liczby możliwych obserwacji i akcji (skończone wartości).

- **O pojedynczym agencie** lub wieloagentowe

Pojedynczy agent operuje w środowisku lub jest ich wiele.

Przykłady środowisk:

- Agent grający w szachy z zegarem działa w środowisku, które jest: w pełni obserwowalne, strategiczne, sekwencyjne, pół-dynamiczne, dyskretne i wieloagentowe.
- Agent grający w szachy bez zegara nie musi uwzględniać upływu czasu i dlatego jego środowisko różni się od poprzedniego tylko tym, że jest statyczne.
- Środowiskiem agenta - kierowcy taksówki jest realny świat. Jest on: częściowo obserwowalny, niedeterministyczny, niemodularny, dynamiczny, ciągły i wieloagentowy.

1.2.4 Symulator środowiska

Podczas procesu projektowania, wiele elementów sterowania agentów upostaciowionych (robotów) a także działania informatycznych soft-botów weryfikowanych jest wstępnie w warunkach symulowanego środowiska. Do generacji obserwacji agentów i przechowywania stanu środowiska w trybie symulacji posłużymy się **generatorem środowiska**. Jego ogólną postać przedstawia procedura w Tabela 2.

Tabela 2. Symulator środowiska dla wstępnych testów agenta

```
procedure GenerujŚrodowisko(stan, ModF, agenty, koniec)
/* Parametry: stan – początkowy stan środowiska, ModF – funkcja modyfikująca stan środowiska,
agenty – zbiór agentów, koniec – predykat, warunek zakończenia symulacji */
  do
     foreach (agent in agenty)
          Obserwacje[agent] ← pobierzObserwacje(agent,stan);
     foreach (agent in agenty)
          Akcje[agent] ← Program[agent](Obserwacje[agent];
     stan ←ModF(Akcje,agenty,stan);
  while koniec(stan);
}
```

1.3 Klasy złożoności agenta

Funkcja sterująca racjonalnie działającego agenta, o zdolności percepcji i uczenia się, jest bardzo złożona. Często proste zadania stawiane agentowi mogą być realizowane przez agenty o znacznie uproszczonej funkcji w porównaniu z agentem racjonalnym. Wyróżnimy tu następujące klasy złożoności funkcji sterującej (w porządku o najprostszej do najbardziej złożonej):

A. Prosty agent reaktywny;
B. Agent reaktywny ze stanem;
C. Agent realizujący cel;
D. Agent racjonalny (realizujący cel w sposób optymalny).

1.3.1 Prosty agent reaktywny

Taki agent reaguje wyłącznie na bieżącą obserwację, gdyż nie przechowuje on informacji o aktualnym stanie świata ani o przeszłych obserwacjach (Tabela 3). Agent stosuje reguły decyzyjne o postaci: *warunek* → *akcja*.

Tabela 3. Funkcja prostego agenta reaktywnego

```
function ProstyAgentReaktywny(obserwacja) returns akcja
{ static: reguły; // Zbiór reguł „warunek→akcja”
  stan ← ANALIZA_OBSERWACJI(obserwacja);
  reguła ← DOPASUJ_REGUŁĘ(stan,reguły);
  akcja ← AKCJA[reguła];
  return akcja;
}
```

1.3.2 Agent reaktywny ze stanem

Agent reaguje na bieżącą obserwację w oparciu o aktualny stan środowiska (tzn. opis środowiska dostępny mu na podstawie wszystkich dotychczasowych obserwacji) (Tabela 4).

Tabela 4. Funkcja agenta reaktywnego ze stanem

```
function AgentReaktywnyZeStanem(obserwacja) returns akcja
{ static:  stan, // opis aktualnego stanu środowiska
          reguły; // zbiór reguł „warunek→akcja”
  stan ← MODYFIKUJ_STAN(stan,obserwacja);
  reguła ← DOPASUJ_REGUŁĘ(stan,reguły);
  akcja ← AKCJA[reguła];
  stan ← MODYFIKUJ_STAN(stan,akcja);
  return akcja;
}
```

1.3.3 Agent realizujący cel

Agent realizujący cel podejmuje akcje w oparciu o stan środowiska i zadany z góry cel (Tabela 5). Problem agenta zwykle reprezentujemy łącznie w postaci:

- **celu** (inaczej: warunku zatrzymania – warunku stopu),
- **przestrzeni stanów** zdefiniowanej dla problemu i
- zbioru wykonywalnych **akcji** odpowiadających operacjom przejścia pomiędzy stanami.

Wyróżnimy dwie główne strategie realizacji celu:

1. akcje wybierane są w wyniku **przeszukiwania** przestrzeni stanów dla problemu;
2. poprzez konstruowanie **planu** (rozumianego jako sekwencja akcji prowadząca do celu); w szczególności plan powstaje w wyniku przeszukiwania właściwie zdefiniowanej przestrzeni planów.

Tabela 5. Funkcja agenta realizującego cel

```
function AgentRealizującyCel(obserwacja) returns akcja
{ static:  sekwAkcji, // sekwencja akcji, początkowo pusta
           stan, // opis aktualnego stanu środowiska
           cel, // cel agenta, początkowo zerowy
           problem; // sformułowanie (reprezentacja) problemu
  stan ← MODYFIKUJ_STAN(stan, obserwacja);
  if (sekwAkcji jest pusta)
  { cel ← WYZNACZ_CEL(stan);
    problem ← WYZNACZ_PROBLEM(stan, cel);
    sekwAkcji ← PRZESZUKIWANIE_LUB_PLANOWANIE(problem);
  }
  akcja ← PIERWSZA(sekwAkcji);
  sekwAkcji ← RESZTA(sekwAkcji);
  return akcja;
}
```

Przeszukiwanie przestrzeni stanów problemu oznacza, że system agentowy wybiera akcje (podejmuje decyzje) w oparciu o aktualny stan problemu i zadany cel. Sposób wyboru zależy od konkretnej strategii przeszukiwania realizowanej przez agenta.

Na potrzeby przeszukiwania reprezentacja stanów jest zwykle prostą strukturą danych, zawierającą dane potrzebne jedynie dla *generacji następnika, funkcji oceny* i *warunku stopu.* Poza tym z punktu widzenia strategii przeszukiwania stan ma zwykle charakter „czarnej skrzynki". Reprezentacja akcji ma postać procedury, która generuje następne stany problemu. Takie reprezentacje problemu nie dają agentowi wskazówek zależnych od analizy własności stanu środowiska.

Planowanie ma na celu „usprawnienie" przeszukiwania przez „otwarcie" reprezentacji stanów, celów i akcji. Reprezentacja planu jest to sekwencja operatorów w przestrzeni planów, która (a) rozszerza plan początkowy (prosty, niepełny plan) do końcowego; lub (b) modyfikuje pełny, ale błędny plan do końcowego. Algorytmy planujące korzystają z formalnych języków, zwykle logiki predykatów lub podzbiorów tego języka, do opisu stanów, celów i akcji. Stany i cel są reprezentowane poprzez zbiory formuł, które są spełnione w danym stanie (względnie stanie docelowym).

1.3.4 Agent racjonalny (optymalny)

Agent racjonalny realizuje zadany cel w sposób optymalny. Ogólna postać funkcji agenta racjonalnego jest taka sama jak podana w Tabela 5. W porównaniu do agenta realizującego cel podfunkcja PRZESZUKIWANIE_LUB_PLANOWANIE dodatkowo korzysta z funkcji użyteczności stanów i akcji w przestrzeni stanów problemu. Zazwyczaj funkcja ta określa stopień spełnienia celu przez dany stan lub akcję wykonywaną w danym stanie. Tym samym agent może uzależnić wybór akcji od wartości użyteczności stanu lub pary stan-akcja.

1.3.5 Przykład wpływu środowiska na projekt agenta

Projekt funkcji agenta silnie zależy od tego, czy środowisko jest w pełni obserwowalne lub nie, oraz czy środowisko jest deterministyczne lub nie. Stąd wynikają szczegółowe wymagania wobec agenta realizującego cel i agenta racjonalnego.

- Agent działa w pełni obserwowalnym i deterministycznym środowisku

Prowadzi to do prostego problemu „jednostanowej decyzji" – w każdej sytuacji decyzyjnej znany jest aktualny, pojedynczy stan problemu. Agent dokładnie zna stan przed i po wykonaniu operacji. Rozwiązaniem problemu jest z góry przewidywalna (warunkowana wykonaniem obserwacji) sekwencja akcji prowadząca do zadanego celu.

- Agent działa w deterministycznym, ale nie w pełni obserwowalnym środowisku

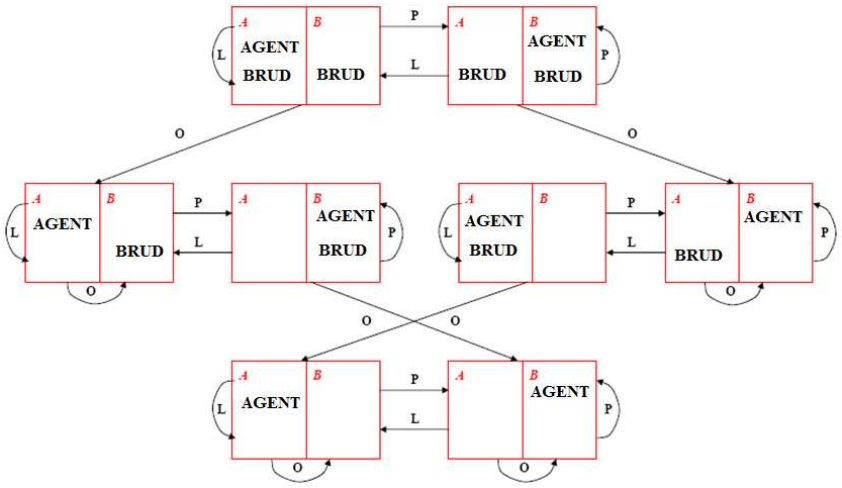
Prowadzi to do problemu „wielostanowej decyzji" - występuje niepewność agenta co do aktualnego stanu środowiska, co sprawia, że sytuacji decyzyjnej odpowiada wiele możliwych stanów środowiska. Agent nie ma pełnej wiedzy o stanie środowiska, ale zna wpływ wykonania swoich akcji na środowisko. Rozwiązaniem jest taka sekwencja akcji, która najpierw redukuje niepewność co do stanu środowiska po której następnie sekwencja akcji (wybierana w trybie „jednostanowej decyzji") prowadząca z rozpoznanego stanu środowiska do celu.

- Środowisko jest niedeterministyczne

Występuje tu problem „ewentualności decyzji" powodowany niepewnością następnego stanu środowiska. Potrzebne jest ciągłe korzystanie z wyników obserwacji środowiska w celu zawężenia niepewności co do aktualnego stanu środowiska. Rozwiązaniem w takiej sytuacji jest dodatkowe do przyjętego stanu i akcji warunkowanie wykonania następnej akcji od aktualnej obserwacji.

1.3.6 Przykład przestrzeni stanów

Zdefiniujemy przestrzeń stanów dla problemu „agenta odkurzającego" działającego w środowisku złożonym z dwóch pomieszczeń A i B (Rysunek 3). Agent może znajdować się w jednym z tych pomieszczeń a każde pomieszczenie może być albo „brudne" albo „czyste". Stany dla tego problemu wyznaczone są łącznie przez stopnie zabrudzenia obu kratek i pozycję odkurzacza. Wystarczą nam trzy akcje wykonywane przez agenta: [*w lewo* (*L*)*, w prawo* (*P*), odkurzanie *(O)*].. Warunek stopu to: „obie kratki są czyste (brak brudu)". Niech koszt akcji wynosi zawsze 1 za każdą akcję.



Rysunek 3. Przestrzeń stanów dla agenta odkurzającego

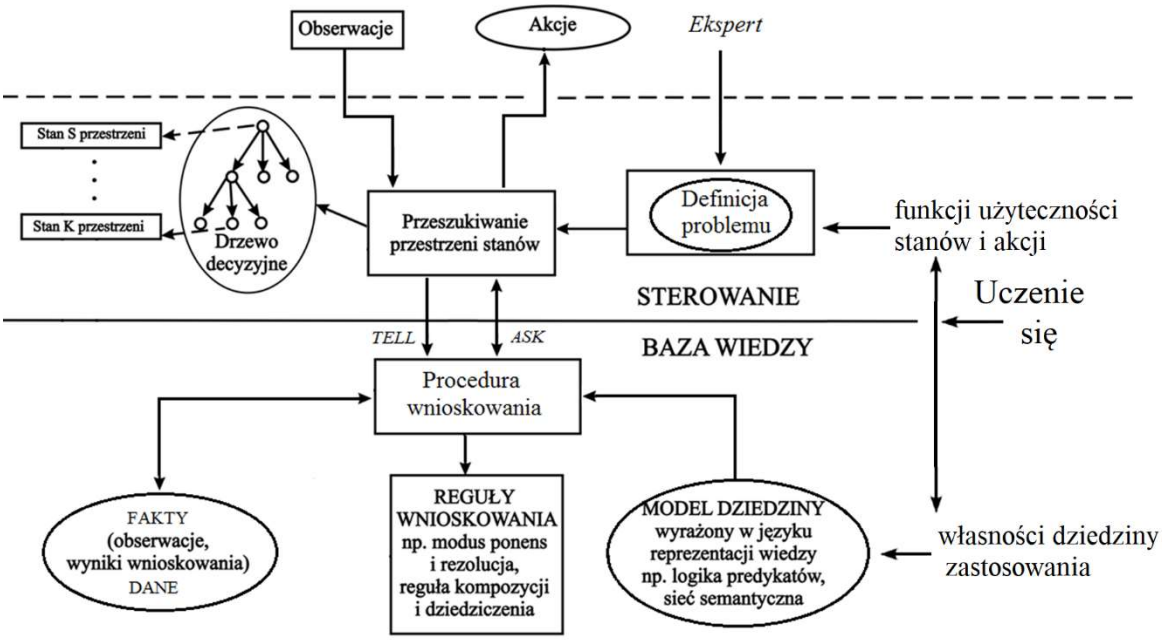
2 System z bazą wiedzy

System z bazą wiedzy posiada następujące możliwości:

- reprezentowanie stanu środowiska, percepcji i akcji agenta;
- dodawanie nowych obserwacji (lub nowo wprowadzonej informacji);
- wnioskowanie o ukrytych własnościach świata zgodnych z obserwacją;
- wybieranie odpowiednich akcji agenta (prowadzących do rozwiązania zadanego problem);
- uczenie się agenta (najczęściej oznacza to uczenie się funkcji użyteczności stanów i akcji oraz modelu dziedziny zastosowania).

2.1 Struktura systemu

Informatyczna implementacja funkcji agenta ma zwykle postać **systemu z bazą wiedzy** (Rysunek 4). Składa się on z dwóch zasadniczych podsystemów: sterowania i bazy wiedzy.

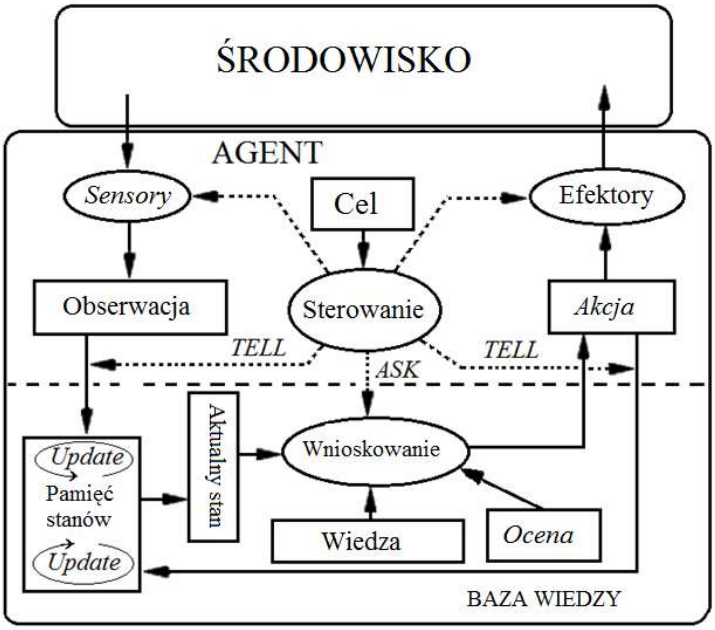


Rysunek 4. System z bazą wiedzy

Baza wiedzy (ang. „*knowledge base*”, KB) zawiera zbiór **aksjomatów i faktów** o modelowanym świecie wyrażony w języku reprezentacji wiedzy oraz odpowiednie dla tego języka **reguły wnioskowania** i **procedury wnioskowania**. Z punktu widzenia logiki matematycznej aksjomaty i fakty mają postać formuł zapisanych w języku reprezentacji wiedzy a reguły wnioskowania są **tautologiami** tego języka. Tautologia jest formułą zawsze prawdziwą w danym języku. Prawdziwość aksjomatów ograniczona jest do konkretnej dziedziny zastosowania systemu, natomiast prawdziwość formuł wyrażających fakty zależy od obserwacji aktualnego środowiska, od tego czy dane fakty zostały zaobserwowane (lub wywnioskowane z innych faktów)
Istnieją też odpowiednie dla języka reprezentacji wiedzy (alternatywne) **procedury wnioskowania**. Każda z nich sprawdza prawdziwość zadawanych pytań w oparciu o aktualny stan bazy wiedzy i reguły wnioskowania.
Sterowanie systemu odpowiada za realizację zadanego celu (lub inaczej mówiąc, za rozwiązanie zadanego problemu).

2.2 Agent racjonalny jako system z bazą wiedzy

Strukturę agenta racjonalnego wykonanego w technologii systemu z baza wiedzy przedstawiono na Rysunek 5.



Rysunek 5. Struktura agenta racjonalnego w technologii systemu z bazą wiedzy

Przyjmiemy, że podsystem sterowania komunikuje się z bazą wiedzy (KB) za pomocą dwóch operacji, TELL i ASK:

TELL: Agent → KB (powiedz bazie o nowych obserwacjach / faktach),

ASK : KB → Agent (zapytaj się bazy, np. co robić? czy zachodzi pewna własność?).

2.2.1 Wybór akcji w bazie wiedzy

W najprostszej postaci podsystem sterowania agenta zadaje jedynie kolejne pytania do bazy wiedzy o następną akcję do wykonania, następnie wykonuje taką akcję i przekazuje bazie wiedzy zwrotnie aktualne obserwacje i wykonane akcje (patrz Tabela 6, funkcja AgentZBaząWiedzy). Sterowanie odwołuje się do bazy wiedzy za pomocą podfunkcji TELL i ASK, przekazując im odpowiednie zdania wyrażone w języku reprezentacji wiedzy tej bazy. Generalnie są to zdania trzech rodzajów, tworzone przez ich podfunkcje:

- UtwórzZdanieObserwacji() – pobiera obserwację i indeks czasu a następnie generuje zdanie w języku bazy wiedzy reprezentujące obserwację w danej chwili czasu;
- UtwórzZapytanieAkcji() – pobiera indeks czasu i generuje zdanie w języku bazy wiedzy będące zapytaniem o to, jaką akcję należy wykonać;
- UtwórzZdanieAkcji() – generuje zdanie w języku bazy wiedzy reprezentujące wykonaną akcję w danej chwili czasu.

Konkretna realizacja tych funkcji zależy od typu reprezentacji wiedzy i własności agenta, takich jak wektor obserwacji, dostępne akcje i format pytań służących rozwiązaniu problemu.

Tabela 6. Funkcja komunikacji podsystemu sterowania agenta z bazą wiedzy

```
function AgentZBaząWiedzy(obserwacja) zwraca akcję
{ static:       KB, // baza wiedzy
                t;  // licznik (indeks czasu) – początkowo wynosi 1
        TELL(KB, UtwórzZdanieObserwacji(obserwacja,t));
        akcja ← ASK(KB, UtwórzZapytanieAkcji(t));
        TELL(KB, UtwórzZdanieAkcji(akcja, t));
        t ← t+1;
        return akcja;
}
```

2.2.2 Wybór akcji poprzez przeszukiwanie przestrzeni stanów

Uniwersalną postacią sterowania może być algorytm **przeszukiwania** przestrzeni stanów zdefiniowanej dla konkretnego problemu. Istnieje wiele możliwych strategii przeszukiwania. Możemy je podzielić na strategie **ślepego** przeszukiwania lub **poinformowanego** przeszukiwania (w sposób racjonalny - optymalny z punktu widzenia stosowanej funkcji użyteczności stanów).

2.3 Uczenie się

Ważnym zagadnieniem systemu agentowego jest posiadanie zdolności do **uczenia się**. Projekt funkcji uczenia się zależy od postaci głównych elementów agenta systemu, jaki element jest modyfikowany, a także jaki rodzaj sprzężenia zwrotnego stosujemy podczas uczenia. O wybranych

procedurach uczenia powiemy w innych modułach tego bloku. Natomiast sprzężenie zwrotne przyjmuje zwykle jedną z trzech poniższych postaci:

1. Uczenie z nadzorem (*supervised learning*): istnieją prawidłowe (wzorcowe) odpowiedzi dla każdego przykładu uczącego;
2. Uczenie ze wzmacnianiem (*reinforcement learning*): brak wzorcowej odpowiedzi ale istnieje krytyk nagradzający ”dobre” akcje (zbliżające agenta do prawidłowego stanu);
3. Uczenie bez nadzoru (*unsupervised learning*): brak wzorcowych odpowiedzi, brak krytyka.

W ramach bloku MSI przedstawimy zagadnienia nabywania wiedzy o własnościach środowiska (np. uczenie przez obserwację tworzenia pojęć i klasyfikacji pojęć) i uczenia się strategii decyzyjnych (uczenie się funkcji użyteczności stanów i akcji).

2.4 Eksploracja otoczenia

W potocznym znaczeniu pojęcie „uczenie się” jest czasem utożsamiane z eksploracją nieznanego środowiska, czyli z nabywaniem informacji o faktach w aktualnej instancji świata. Jest to nieodzowny element działania agenta wtedy, gdy projektant systemu nie dysponuje pełną informacją o środowisku. Poznawanie środowiska jest możliwe dzięki dwóch podsystemom agenta – percepcji otoczenia i wnioskowaniu o ukrytych własnościach otoczenia.

2.5 Język reprezentacji wiedzy

Wyróżnimy teraz podstawowe **języki reprezentacji wiedzy** i określimy ich charakterystykę ze względu na ontologię i epistemologię języka. **Ontologia** języka wskazuje na to czemu (jakim zjawiskom) odpowiadają symbole języka w rzeczywistym świecie. **Epistemologia** języka wyraża to, jak odbierane (oceniane) są te elementy świata - jakie wartości przyjmują jednostki wiedzy. W Tabela 7 zebranych jest 5 istotnych języków reprezentacji wiedzy. W tym module omówimy reprezentację wiedzy deterministycznej w języku logiki (rachunek zdań i logika predykatów).

Tabela 7. Ontologia i epistemologia wybranych języków reprezentacji wiedzy

Język	**Ontologia**	**Epistemologia**
Rachunek zdań	Fakty	true/false/unknown
Logika predykatów (1 rzędu)	Fakty, obiekty, relacje	true/false/unknown
Logika temporalna	Fakty, obiekty, relacje (wiedza niepełna)	true/false/unknown
Teoria probabilistyczna	Fakty (wiedza niepewna)	Rozkład prawdopodobieństwa
Logika rozmyta	Fakty (wiedza niedokładna)	Stopień przynależności <0, 1>

Składnia języka L podaje reguły tworzenia poprawnych zdań języka (w logice predykatów nazywanych *formułami*). **Semantyka** języka L definiuje „znaczenie" zdań (formuł):

1. podaje ona znaczenie wszystkich symboli **X** języka L (czyli zawiera pewne odwzorowanie: **X**→ {elementy modelowanego świata}) i

2. sposób, w jaki zdaniom (formułom) można przypisać znaczenie, co z kolei pozwala określić ich wartość (np. logiczną).

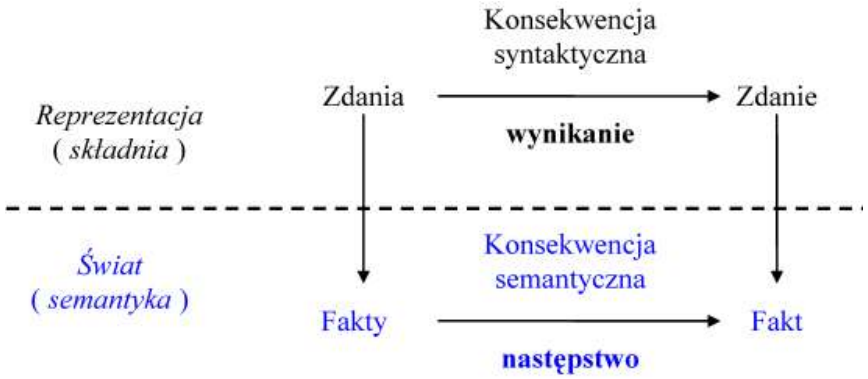
Dla przykładu język działań arytmetycznych jest językiem logiki. Zasady składni mówią np., że (x+2 ≥ y) jest zdaniem, a (x2+y > {}) nie jest zdaniem tego języka. Zasady semantyki mówią np., że:

- x+2 ≥ y jest prawdziwe wtw. gdy liczba x+2 jest nie mniejsza niż liczba y;
- x+2 ≥ y jest prawdziwe w świecie, w którym x = 7, y = 1;
- x+2 ≥ y jest fałszywe w świecie, w którym x = 0, y = 6.

3 System logicznego wnioskowania

3.1 Wynikanie zdań

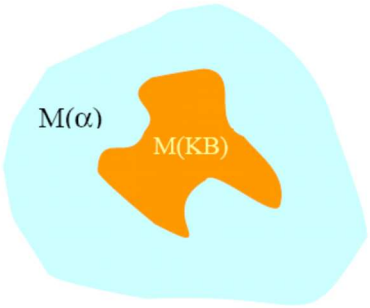
Wynikanie (ang. *entailment*) jest związkiem pomiędzy zdaniami. Mówimy, że „ze zbioru zdań X wynika zdanie A" i oznaczamy to jako, X ⊨ A , wtedy gdy odzwierciedla to następstwo (*konsekwencję semantyczną*) odpowiadających tym symbolom faktów w modelowanym świecie". Pojęcie wynikania zdań języka przedstawiono schematycznie na Rysunek 6. Wynikanie zdań w języku logiki.



Rysunek 6. Wynikanie zdań w języku logiki

Definicję relacji wynikania zdań możemy podać na gruncie teorii zbiorów. Semantyczną relację **następstwa** wyrazimy poprzez relację zawierania się zbiorów reprezentujących realizacje świata, zwane **modelami**. Modele w logice to formalnie zdefiniowane **światy**, względem których można określać to co jest **prawdziwe** a co **nie**. **Model** dla zbioru zdań ***X*** to każdy **świat**, w którym prawdziwe są wszystkie zdania ze zboru ***X***. Możemy powiedzieć, że zdanie *A* **wynika** ze zbioru zdań ***X***, co zaznaczamy $\boldsymbol{X} \models A$, jeśli *A* jest **prawdziwe** w każdym modelu dla ***X***. Odnosząc to stwierdzenie do bazy wiedzy, która zawiera jedynie zdania uznane za prawdziwe w obserwowanym środowisku (realizacji świata) powiemy, że z bazy wiedzy KB **wynika** zdanie α wtw. gdy α jest **prawdziwe** dla wszystkich **modeli zdań** zawartych w KB (oznaczamy, $\boldsymbol{KB} \models \alpha$). Niech M(α) będzie zbiorem wszystkich modeli zdania α. Wtedy (patrz Rysunek 7):

$$\boldsymbol{KB} \models \alpha \text{ wtw. } \mathrm{M}(\boldsymbol{KB}) \subseteq \mathrm{M}(\alpha).$$



Rysunek 7. Relacja wynikania zdań zdefiniowana jako zawieranie się zbiorów modeli

Np. niech baza wiedzy KB zawiera zdania "Legia *wygrała mecz*" i "Wisła *wygrała mecz*". Z nich wynika zdanie: "Legia w*ygrała mecz lub Wisła wygrała mecz*". Jest ono spełnione w szerszym zbiorze światów niż koniunkcja obu pierwszych zdań. Zauważmy też, że zdania uznane za **równoważne** w pewnej dziedzinie wiedzy są specyficznym przypadkiem wynikania. Np. ze zdania (x+y= 4) wynika, że (4 = x+y), a w *matematyce* są to zdania równoważne.

3.2 Wnioskowanie

Celem **procesu wnioskowania** jest sprawdzenie tego, czy zachodzi relacja wynikania (w aktualnym stanie środowiska) pomiędzy wybranymi zdaniami języka. Ogólnie rzecz biorąc, każdy proces **wnioskowania** (ang. *inference*) w systemie logicznym występuje w jednej z dwóch postaci:

1) Proces **wyprowadzania** (generowania) nowych zdań ze zdań przyjętych za prawdziwe (tzn. reprezentujących prawdziwe fakty).
2) Proces **sprawdzenia** (dowód), czy zadanie zdanie *A* (pytanie) wynika ze zbioru zdań ***X***, tzn. czy zachodzi $\boldsymbol{X} \vDash A$.

Dla danego języka logicznego możemy zdefiniować pojęcie **wyprowadzalności zdań** (konsekwencja syntaktyczna, *derivability*): „zdanie *A* jest wyprowadzalne ze zbioru zdań ***X*** przy użyciu procedury wnioskowania (dowodzenia) Π, co oznaczymy jako $\boldsymbol{X} \vdash_{\Pi} A$, wtw. gdy Π wyprowadza (znajduje dowód zdania) *A* ze zdań zbioru ***X***." Podobnie, oznaczenie, $\boldsymbol{KB} \vdash_{\Pi} \alpha$, powie nam, że zdanie α jest wyprowadzalne z bazy wiedzy ***KB*** przy użyciu procedury Π.

Interesują nas tylko takie procedury wnioskowania, które są **poprawne** i **zupełne**. Procedura wnioskowania Π jest **poprawna** wtedy i tylko wtedy, gdy dla każdego zbioru zdań ***X*** i każdego zdania *A*: zachodzenie $\boldsymbol{X} \vdash_{\Pi} A$, pociąga za sobą $\boldsymbol{X} \vDash A$. Innymi słowy, jeśli poprawna procedura wnioskowania wskazuje na istnienie relacji wynikania pomiędzy pewnymi zdaniami, to ona zawsze rzeczywiście zachodzi.

Procedura wnioskowania Π jest **zupełna** wtw., gdy dla każdego zbioru zdań ***X*** i każdego zdania *A*:

$\boldsymbol{X} \vDash A$ pociąga za sobą $\boldsymbol{X} \vdash_{\Pi} A$.

Tym samym wskazujemy, że zupełna procedura wnioskowania to taka, która jest w stanie wykazać każde rzeczywiście istniejące wynikanie.

3.3 System logiczny

Celem **systemu logicznego wnioskowania** (systemu logicznego lub po prostu **logiki**) jest to, że udostępnia on język formalny do takiej reprezentacji informacji, z której można wyciągać wnioski. Podsumowując, logika obejmuje język reprezentacji wiedzy (charakteryzowany poprzez składnię i semantykę) oraz odpowiedni mechanizm wnioskowania (dedukcji). W następnych rozdziałach zdefiniujemy dwa podstawowe systemy logiczne (rachunek zdań, logika pierwszego rzędu), które są wystarczająco „mocne", aby wyrazić większość interesujących nas rzeczy i dla których istnieją poprawne i zupełne procedury wnioskowania. Dla każdego języka logiki można zaproponować różne procedury wnioskowania – sam język nie wyznacza unikalnego mechanizmu wnioskowania, którym można się posługiwać. Jednak typowy system logicznego wnioskowania polega na stosowaniu **reguł wnioskowania**, czyli formuł zawsze prawdziwych (stanowiących **tautologie** języka).

4 Rachunek zdań

Prezentację klasycznych języków logiki rozpoczniemy od najprostszego z nich - **rachunku zdań**.

4.1 Składnia rachunku zdań

W skład alfabetu rachunku zdań wchodzą:

- stałe logiczne *True* i *False*,
- symbole zdaniowe (*P*, *Q*, *R*,...),
- spójniki logiczne (koniunkcja, alternatywa, implikacja, równoważność, negacja) $\wedge, \vee, \Rightarrow, \Leftrightarrow, \neg$
- nawiasy okrągłe (,).

Z liter alfabetu tworzone są **zdania** za pomocą następujących zasad:

- *True* , *False* i symbole zdaniowe są zdaniami atomowymi.
- Jeśli *A* i *B* są zdaniami to $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ i $(A \Leftrightarrow B)$ też są zdaniami języka.
- Jeśli *A* jest zdaniem to $\neg A$ też jest zdaniem.

Mianem **literału** określamy zdanie atomowe lub negację zdania atomowego.

4.2 Semantyka rachunku zdań

Semantyka rachunku zdań może być formalnie ujęta jako struktura algebraiczna: <D, φ >. **Dziedzina** D to zbiór faktów, do których odnoszą się symbole języka. **Interpretacja** φ to funkcja przyporządkowująca symbolom zdaniowym fakty należące do dziedziny i wartościująca zdania jako *Prawdę* (*True*) lub *Fałsz* (*False*). Wartościowanie zdań atomowych zależy od prawdziwości reprezentowanego faktu w aktualnym świecie a przy wartościowaniu zdań złożonych korzystamy z tabelki prawdy dla spójników (0 oznacza *Fałsz* a 1 – *Prawdę*) (Rysunek 8).

Model zdania A (oznaczmy go jako M(A) = <D, φ >) jest wyznaczony przez każdą interpretację φ dla dziedziny D, w której zdanie A jest prawdziwe. **Tautologia** w dziedzinie D jest to zdanie prawdziwe w każdej interpretacji (modelu) dla D.

Rachunek zdań jest **rozstrzygalny**, tzn. istnieje algorytm, który dla dowolnego zdania A stwierdza, czy A jest tautologią. Takim algorytmem jest chociażby sprawdzanie tabelki prawdy dla zdania. Ponieważ każde zdanie zawiera skończoną liczbę symboli i spójników, więc tabelka będzie miała skończony rozmiar co umożliwi sprawdzenie wszystkich wierszy.

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Rysunek 8. Tabela prawdy dla spójników logicznych

Dla przykładu, dzięki tabeli prawdy pokazujemy, że zdanie złożone, $((P \vee H) \wedge \neg H) \Rightarrow P$, jest tautologią, tzn. zawsze prawdziwą niezależnie od modelu dla P i H . Zauważmy, że ostatnia kolumna jest zawsze prawdziwa (Rysunek 9).

P	H	P∨H	(P∨H)∧¬H	((P∨H)∧¬H)⇒P
0	0	0	0	1
0	1	1	0	1
1	0	1	1	1
1	1	1	0	1

Rysunek 9. Sprawdzanie czy zdanie jest tautologią

4.3 Wnioskowanie

4.3.1 Wnioskowanie z tabelą prawdy

Procedura sprawdzająca wszystkie modele jest procedurą wnioskowania **poprawną** i **zupełną**. Może ona zostać efektywnie zaimplementowana wtedy, gdy formuły dają się uporządkować (np. odpowiednio do odległości miejsca od pozycji startowej agenta, do którego te formuły się odnoszą). Np. funkcja WnioskujTabPrawdy() (Tabela 8) jest efektywną implementacją procedury wnioskowania. Korzysta ona z rekursywnej funkcji TabPrawdy(), wywoływanej dla częściowego i stopniowo rozszerzanego modelu. Podfunkcja CzyModel() sprawdza poprawność zdania (lub bazy wiedzy) w częściowym modelu a podfunkcja Rozszerz() rozszerza model o wartość dla kolejnego symbolu (zdania).

Tabela 8. Procedura wnioskowania polegająca na sprawdzaniu tabelki prawdy

```
funkcja WnioskujZTabPrawdy(KB, α)
zwraca wynik: true lub false
{ symbole ← symbole zdaniowe w KB i α;
  return TabPrawdy(KB, α, symbole, []);
}
```

```
funkcja TabPrawdy(KB, α, symbole, model)
zwraca wynik: true lub false
{ if (symbole == ∅ ) {
        if CzyModel(KB, model) return CzyModel(α, model);
        else return true;
  } else {
    P ← Pierwszy(symbole); reszta ← Reszta(symbole);
    return (TabPrawdy(KB, α, reszta, Rozszerz(P, true, model)
      && TabPrawdy(KB, α, reszta, Rozszerz(P, false,model)); }
}
```

4.3.2 Twierdzenia o dedukcji

Wiemy już, że zdanie logiczne jest tautologią wtw. gdy jest prawdziwe we wszystkich modelach języka. Np. tautologiami są następujące zdania:

$$True, \quad A \vee \neg A, \quad A \Rightarrow A, \quad (A \wedge (A \Rightarrow B)) \Rightarrow B$$

Tautologia jest powiązana z wnioskowaniem poprzez pierwsze **twierdzenie o dedukcji**, które mówi, że: $KB \models \alpha$ wtw. gdy formuła $(KB \Rightarrow \alpha)$ jest tautologią.

To twierdzenie prowadzi do metod wnioskujących wprost.

Wiemy też, że formuła A jest spełnialna wtw. gdy posiada przynajmniej jeden model. Wtedy niespełnialną formułą jest taka, która nie posiada żadnego modelu. Np., $A \wedge \neg A$, jest niespełnialne. Spełnialność formuły jest powiązana z wnioskowaniem poprzez drugie **twierdzenie o dedukcji**:

$KB \models \alpha$ wtw. gdy formuła $(KB \wedge \neg \alpha)$ jest niespełnialna.

To twierdzenie prowadzi do wnioskowania nie wprost – do dowodu przez zaprzeczenie.

4.3.3 Przekształcenia równoważne

Na potrzeby procedur wnioskowania zdania w bazie wiedzy powinny przyjmować wymagane postacie *normalne*. Dla uzyskania takich postaci można stosować przekształcenia zdania do równoważnych zdań.

Istnieje szereg równoważnościowych przekształceń zdań logicznych. Mówimy, że dwa zdania są **logicznie równoważne** (oznaczamy ≡) wtw. gdy są poprawne w tych samych modelach. Innymi słowy logiczną równoważność możemy powiązać z wynikaniem:

$\alpha \equiv \beta$ wtw. $(\alpha \models \beta)$ i $(\beta \models \alpha)$

Oto podstawowe pary logicznie równoważnych zdań:

1. $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ – przemienność $\wedge$
2. $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ – przemienność $\vee$
3. $((\alpha \wedge \beta) \wedge \lambda) \equiv (\alpha \wedge (\beta \wedge \lambda))$– łączność $\wedge$
4. $((\alpha \vee \beta) \vee \lambda) \equiv (\alpha \vee (\beta \vee \lambda))$– łączność $\wedge$
5. $\neg(\neg \alpha) \equiv \alpha$ – eliminacja podwójnej negacji
6. $(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$ – kontrapozycja
7. $(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$ – eliminacja implikacji
8. $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ – eliminacja równoważności
9. $\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$ – prawo de Morgana
10. $\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$ – prawo de Morgana
11. $(\alpha \wedge (\beta \vee \lambda)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \lambda))$ – rozdzielczość $\wedge$ względem $\vee$
12. $(\alpha \vee (\beta \wedge \lambda)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \lambda))$ – rozdzielczość $\vee$ względem $\wedge$

4.3.4 Reguły wnioskowania

Dla rachunku zdań istnieją zupełne i poprawne **procedury wnioskowania** (dowodzenia formuł). Stosują one **reguły wnioskowania** o ogólnej postaci:

$$\frac{poprzednik}{nastepnik}$$

Reguła wnioskowania mówi, że jeśli spełniony jest warunek zadany *poprzednikiem* reguły to wnioskowana jest poprawność *następnika*. Oczywiście, aby móc stosować jakąś regułę wnioskowania musimy upewnić się, że jest ona poprawna we wszystkich modelach języka, tzn. że jest tautologią języka. Powiemy, że reguła wnioskowania o postaci

$$\frac{\alpha_1,\alpha_2,\cdots,\alpha_n}{\beta}$$

jest **poprawna**, jeśli zdanie β jest prawdziwe w każdej interpretacji (modelu), w której prawdziwe są zdania: $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Wybór poprawnych i najczęściej stosowanych reguł wnioskowania:

1. Reguła **odrywania** (*modus ponens*): $\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta}$
2. Reguła **eliminacji koniunkcji**: $\frac{\alpha_1 \wedge \cdots \wedge \alpha_n}{\alpha_i}$
3. Reguła **wprowadzania koniunkcji**: $\frac{\alpha_1,\cdots,\alpha_n}{\alpha_1 \wedge \cdots \wedge \alpha_n}$
4. Reguła **rezolucji**: $\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$

Sprawdźmy za pomocą tablicy prawdy poprawność powyższej reguły rezolucji (Rysunek 10).

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	1	1

Rysunek 10. Tabela prawdy dla reguły rezolucji

W powyższej tabeli wyróżniono kolorem tła 4 wiersze, dla których spełniony jest poprzednik tej reguły. Widzimy, że w tych wierszach następnik również jest zawsze spełniony. Tym samym reguła rezolucji jest poprawna.

4.4 Postacie normalne zdań

Procedury wnioskowania zakładają, że zdania występują we właściwej postaci **normalnej**. Pozwala to zdefiniować obliczeniowo efektywną procedurę wnioskowania.

4.4.1 Klauzula Horna i reguła odrywania

Dla procedur stosujących regułę *Modus Ponens* (zwaną także *regułą odrywania*) zdania (lub formuły) przyjmują postać tzw. **klauzul Horna**. Klauzula Horna to pojedynczy prosty literał lub implikacja o postaci: (*koniunkcja prostych literałów*) $\Rightarrow$ prosty literał .

„Prosty" oznacza „pozytywny", nie zanegowany. Np. w poniższym zdaniu występują trzy klauzule Horna: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Reguła odrywania (*Modus Ponens*) stosowana jest w procedurach wnioskowania dla rachunku zdań wtedy, gdy zdania w bazie wiedzy występują w postaci klauzul Horna. Tworzą one poprzednik reguły:

$$\frac{\alpha_1,\cdots,\alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Z reguły odrywania korzystają procedury wnioskowania wprost: **progresywna** (wnioskowanie w przód) i **regresywna** (wnioskowanie wstecz).

4.4.2 Postać normalna CNF i reguła rezolucji

Drugą postacią normalną dla zdań (lub formuł) jest **koniunkcyjna postać normalna** (ang. *Conjunctive Normal Form*, **CNF**) będąca koniunkcją alternatyw literałów.

Np.: w rachunku zdań poniższe zdanie jest w postaci CNF: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Postać CNF jest odpowiednia dla procedur wnioskowania korzystających z reguły rezolucji, czyli dla wnioskowania nie wprost, inaczej mówiąc, dowodzenia zdania zapytania przez zaprzeczenie.

Reguła rezolucji w rachunku zdań jest postaci:

$$\frac{l_1 \vee \ldots \vee l_k \ , \ m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

gdzie l_i i m_j są komplementarnymi literałami ($l_i = \neg m_j$).

Np., jeśli zachodzą zdania: $P_{13} \vee P_{22}$ i $\neg P_{22}$ to wnioskujemy stąd, że zachodzi P_{13} .

Pokażemy poprawność reguły rezolucji stosując jedynie przekształcenia zdań.

1) Zakładamy, że zachodzi poprzednik. Stosując równoważność, $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$, przejdziemy z postaci po prawej stronie do lewej strony dla obu zdań w poprzedniku. Wyłączamy przy tym komplementarne literały od reszty literałów. Otrzymujemy:

$$\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i$$

$$\neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

2) Z założenia, $l_i = \neg m_j$, i z przechodniości implikacji, $\alpha \Rightarrow \beta \Rightarrow \gamma$, wynika:

$$\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

3) Ponownie stosujemy równoważność z pkt. 1) ale teraz przekształcamy implikację z pkt. 2) na postać alternatyw literałów:

$l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n$.

Tym samym uzyskaliśmy postać następnika reguły rezolucji co kończy dowód.

4.4.3 Przykład

Konwersja zdania do postaci normalnej CNF. Przekształcimy zdanie, $B_{11} \Leftrightarrow (P_{12} \vee P_{21})$.

1. Usuwamy równoważność $\Leftrightarrow$, zamieniając $\alpha \Leftrightarrow \beta$ na $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Usuwamy obie implikacje $\Rightarrow$, zamieniając $\alpha \Rightarrow \beta$ na $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Wprowadzamy $\neg$ do środka nawiasów stosując reguły de Morgana i ewentualnie eliminujemy podwójną negację:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Stosujemy prawo rozdzielczości ($\wedge$ nad $\vee$) i rozpisujemy:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Uzyskaliśmy zdanie w postaci CNF.

4.5 Wnioskowanie poprzez rezolucję (przez zaprzeczenie)

Procedury wnioskowania (dowodzenia) zdań stosujące regułę rezolucji prowadzą **dowód przez zaprzeczenie**, tzn. aby dowieść, że zachodzi wynikanie ($KB \models \alpha$) pokazują one, że zdanie $(KB \wedge \neg\alpha)$ jest niespełnialne.

4.5.1 Algorytm wnioskowania

Schemat algorytmu wnioskowania przez rezolucję pokazuje Tabela 9. W funkcji Rezolucja() najpierw zamieniamy zdania w bazie wiedzy rozszerzonej o negację zapytania, $(KB \wedge \neg\alpha)$, na postać CNF. Następnie w pętli, dla każdej odpowiedniej pary zdań, funkcja KrokRezolucji() generuje *rezolwentę* swoich 2 argumentów, tzn. zdanie w następniku reguły rezolucji, które jest pozbawione pary komplementarnych symboli. Są możliwe dwa warunki zakończenia procedury:

1) nie można dodać nowych zdań, wtedy α jest fałszywe w modelu M(KB), lub

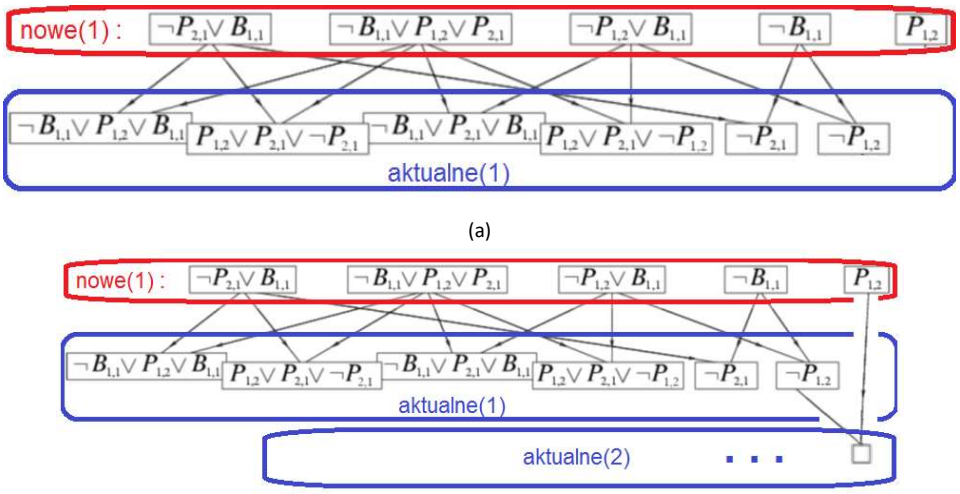
2) w wyniku rezolucji powstaje zdanie puste, co wynika ze sprzeczności w bazie; a to oznacza, że α jest prawdziwe w modelu M(KB).

Tabela 9. Procedura wnioskowania przez rezolucję w rachunku zdań

```
funkcja Rezolucja(KB, α) zwraca wynik: True lub False
{
  zdania ← zbiór zdań w postaci CNF dla (KB ∧ ¬α);
  nowe ← {};
   while ( True ) {
      for each (Ci, Cj ∈ zdania ) {
          rezolwenty ← KrokRezolucji(Ci, Cj);
          if (rezolwenty zawierają zdanie puste) return True;
          nowe ← nowe ∪ rezolwenty;
      }
      if (nowe ⊆ zdania ) return False;
      zdania ← zdania ∪ nowe;
   }
}
```

4.5.2 Przykład wnioskowania przez rezolucję

Niech baza wiedzy agenta zawiera zdania: { $B_{11} \Leftrightarrow (P_{1,2} \vee P_{2,1})$, $\neg B_{11}$ }. Chcemy dowieść, że zachodzi zdanie, $\alpha = \neg P_{12}$. Procedura wnioskowania Rezolucja() wykona dwie iteracje bloku „while". Na Rysunek 11(a) górny rząd „nowe(1)" zawiera klauzule powstałe po normalizacji zdań w zbiorze (KB $\wedge \neg\alpha$) do postaci CNF. Dolny rząd (aktualne(1)) zawiera *rezolwenty* par zdań zawierających komplementarne literały. Na Rysunek 11(b) dodano wynik drugiej iteracji. Zbiór „aktualne(1)" staje się „nowe(1)" a wśród wyników generowanych w drugiej iteracji „aktualne(2)" pojawia się zdanie puste powstałe z rezolucji pary P_{12} i $\neg P_{12}$. Sprzeczność bazy wiedzy powstała po dodaniu negacji zapytania dowodzi tego, że zdanie zapytania jest spełnione w modelu bazy wiedzy, tzn. KB $\models \alpha$.



Rysunek 11. Przykład iteracji procesu wnioskowania przez rezolucję: (a) pierwsza iteracja, (b) druga iteracja. Wygenerowana zostanie formuła pusta.

4.6 Wnioskowanie z regułą odrywania (wnioskowanie wprost)

Procedury wnioskowania korzystające z reguły odrywania stosują dowodzenie wprost. Wyróżnimy tu zasadniczo dwa sposoby takiego wnioskowania:

- procedura progresywna (wnioskowanie w przód) – generowanie zdań – i
- procedura regresywna (wnioskowanie wstecz) – dowód wprost.

4.6.1 Wnioskowanie w przód

Idea procedury wnioskowania **progresywnego (w przód)**:

1. wykonaj każdą regułę, której warunek (poprzednik) jest spełniony w KB,
2. dodaj wynik wyprowadzenia (następnik reguły) do KB,
3. kontynuuj kroki 1-2 aż do znalezienia zdania zapytania lub niemożliwości wygenerowania nowych zdań.

Progresywna procedura wnioskowania jest poprawna i zupełna dla bazy wiedzy o postaci klauzul Horna.

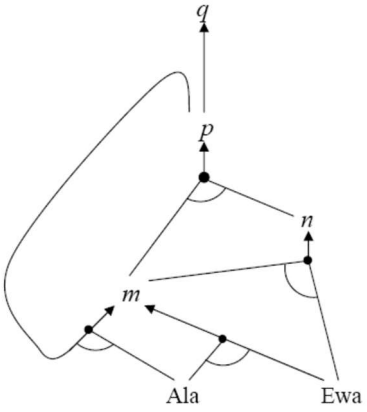
4.6.2 Przykład wnioskowania w przód.

Przyjmijmy, że dane są zdania w KB w postaci klauzul Horna (Rysunek 12).

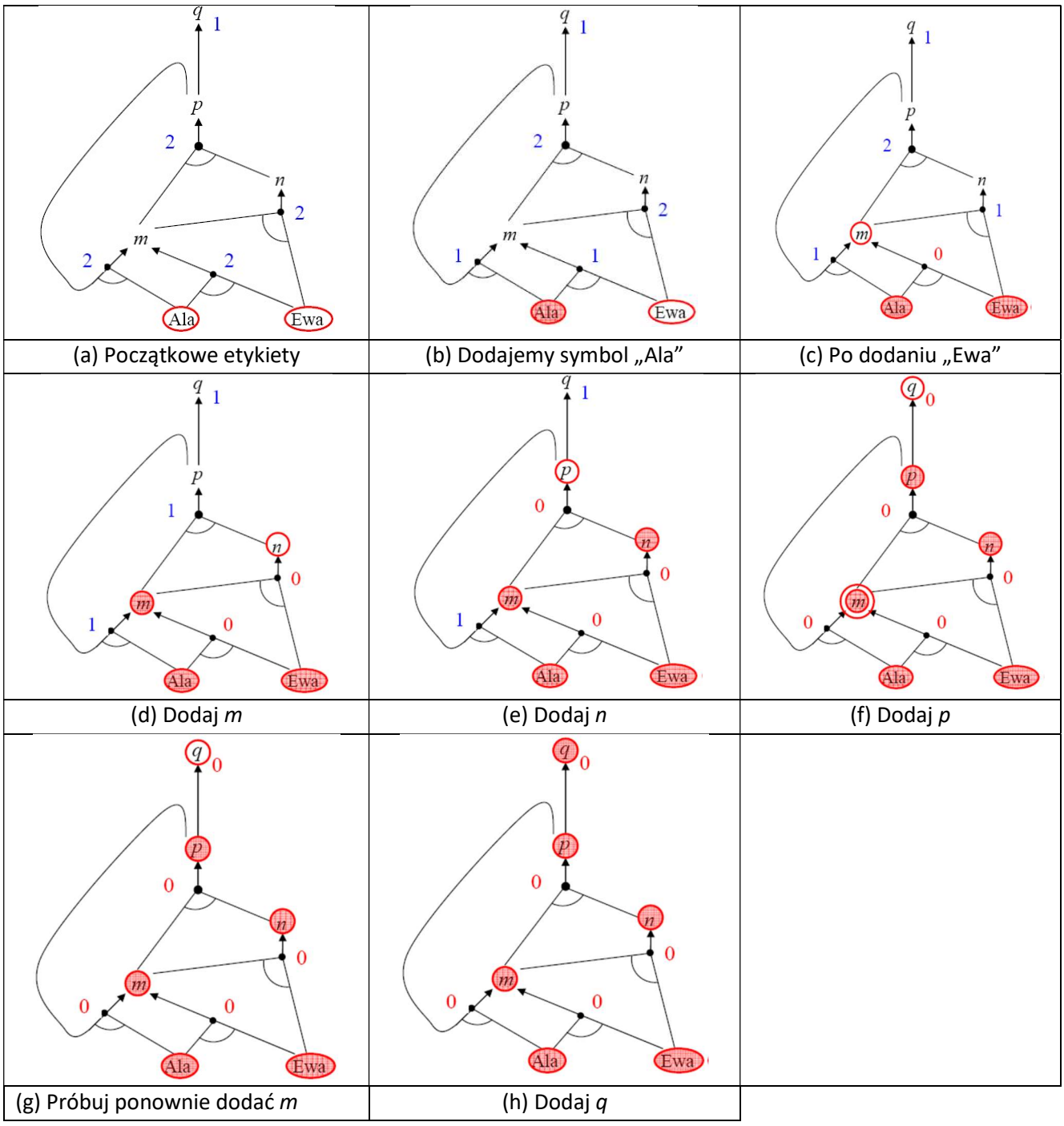
$$p \Rightarrow q$$
$$m \wedge n \Rightarrow p$$
$$Ewa \wedge m \Rightarrow n$$
$$Ala \wedge p \Rightarrow m$$
$$Ala \wedge Ewa \Rightarrow m$$
$$Ala$$
$$Ewa$$

Rysunek 12. Zdania zawarte w przykładowej bazie wiedzy.

Odpowiada im reprezentacja graficzna w postaci grafu I-LUB (Rysunek 13). Każdy węzeł typu „I" posiada etykietę – odpowiada ona liczbie warunków w poprzedniku reguły pozostałych jeszcze do spełnienia. Kolejno wykonywane kroki wnioskowania w przód ilustruje Rysunek 14.



Rysunek 13. Graf I-LUB dla reprezentacji zbioru klauzul Horna.



(a) Początkowe etykiety | (b) Dodajemy symbol „Ala” | (c) Po dodaniu „Ewa”

(d) Dodaj *m* | (e) Dodaj *n* | (f) Dodaj *p*

(g) Próbuj ponownie dodać *m* | (h) Dodaj *q*

Rysunek 14. Przykład kroków wnioskowania „w przód”

4.6.3 Wnioskowanie wstecz

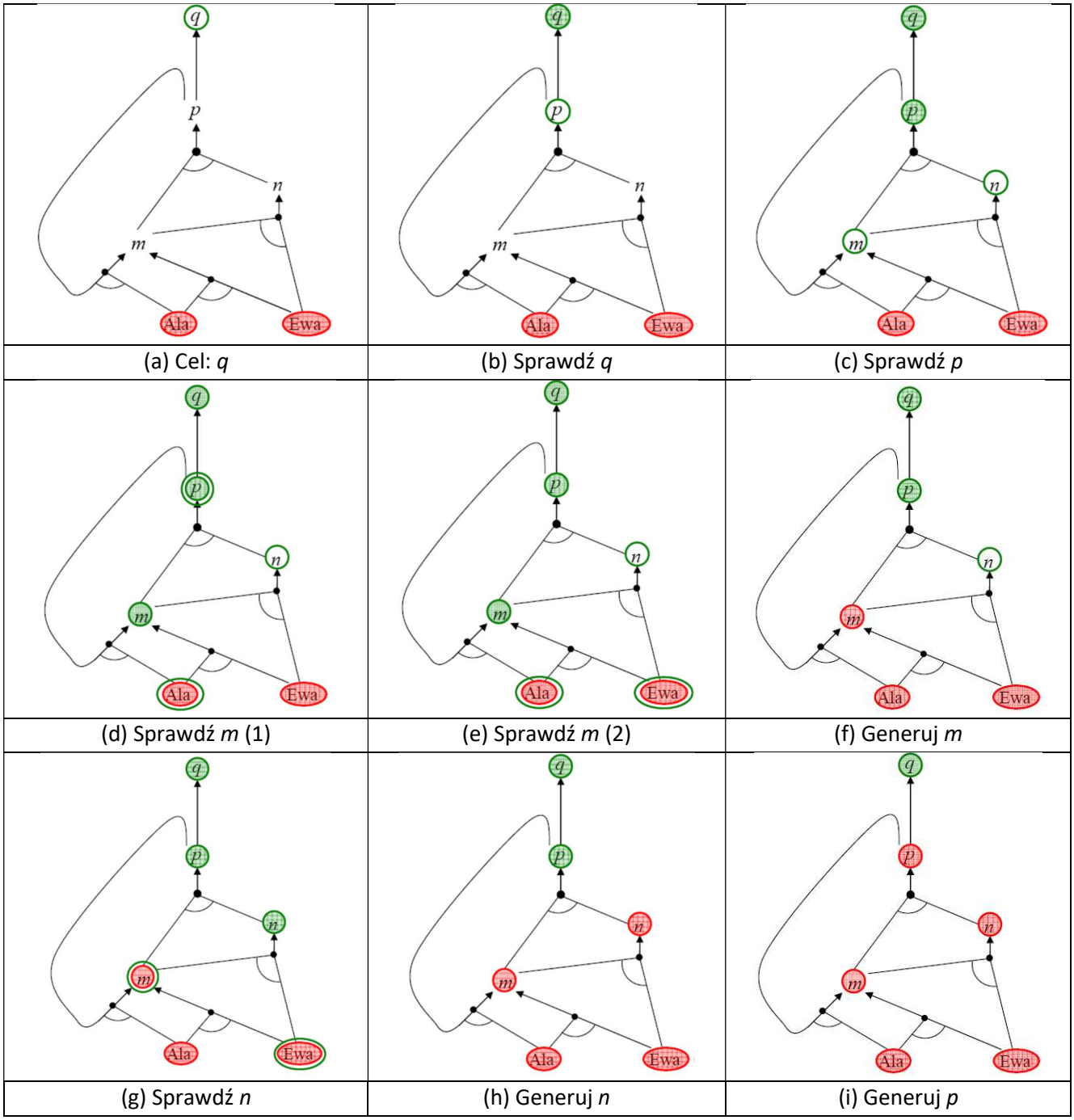
Idea procedury wnioskowania „wstecz” w rachunku zdań wygląda następująco:

1. Funkcja rozpoczyna od zdania zapytania (celu) *q*.
2. Aby sprawdzić prawdziwość *q* procedura sprawdza, czy *q* już występuje w KB a jeśli nie, to sprawdza czy istnieje przynajmniej jedna implikacja wyprowadzająca zdanie *q*. Jeśli tak, to literały stanowiące warunek tej implikacji stają się „pod-celami” i rekurencyjne badana będzie ich prawdziwość z punktu widzenia aktualnego modelu KB, podobnie jak poprzednio główny cel.

3. Unikanie zapętleń: procedura sprawdza, czy aktualny „pod-cel” nie znajduje się już na stosie wygenerowanych „pod-celów”.
4. Unikanie powielania przejść: sprawdza, czy nowy „pod-cel” został już sprawdzony i pokazano to, czy jest prawdziwy lub fałszywy.

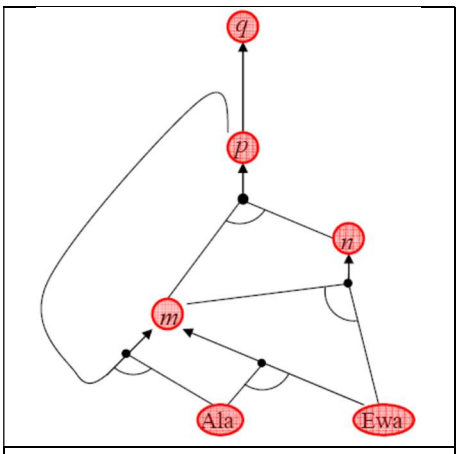
4.6.4 Przykład wnioskowania „wstecz”.

Załóżmy, że baza danych zawiera zdania z poprzedniego przykładu (Rysunek 12). Wśród nich występują też 2 fakty: „Ala” i „Ewa”. Niech zdanie zapytania to „*q*”. Kolejne kroki w procesie wnioskowania wstecz dla tej sytuacji ilustruje Rysunek 15 i Rysunek 16.



(a) Cel: *q*	(b) Sprawdź *q*	(c) Sprawdź *p*
(d) Sprawdź *m* (1)	(e) Sprawdź *m* (2)	(f) Generuj *m*
(g) Sprawdź *n*	(h) Generuj *n*	(i) Generuj *p*

Rysunek 15. Przykład wnioskowania „wstecz” (kroki a-i)



(j) Generuj *q*

Rysunek 16. Przykład wnioskowania „wstecz" – krok końcowy

Procedura wnioskowania w przód jest sterowana danymi – odpowiada to automatycznemu, „nieświadomemu" przetwarzaniu danych. Np. rozpoznawanie obiektów, rutynowe decyzje. Może wykonywać „nadmiarową" pracę, która nie zmierza bezpośrednio do celu.
Procedura wnioskowania wstecz jest sterowana celem – jest to odpowiednie dla rozwiązywania zadanego problemu. Np. dostarczenie odpowiedzi na pytanie: „Gdzie są moje klucze?" Złożoność procedury regresywnej może w praktyce być poniżej liniowej względem rozmiaru KB.

4.7 Własności zmienne w czasie

Prostota rachunku zdań sprawia, że posiada on małą siłę wyrazu – każdy kolejny fakt musi być reprezentowany oddzielnym symbolem języka. Dla przykładu zastanówmy się jak reprezentować w rachunku zdań akcje agenta *„świata Wumpusa"* wykonywane w określonym czasie. Wprowadzamy symbole $L_{i,j}$ dla oznaczenia, że agent działający w 2-wymiarowym środowisku znajduje się w kratce o współrzędnych [i,j]. Wtedy możliwym akcjom odpowiadałyby zdania w rodzaju:

$$L_{1,1} \wedge ZwróconyWPrawo \wedge RuchWPrzód \Rightarrow L_{2,1}$$

Jednak to nie prowadzi do prawidłowego wnioskowania. Po wykonaniu akcji oba zdania $L_{1,1}$ i $L_{2,1}$ będą w bazie danych uważane za prawidłowe, tymczasem już tak nie jest, gdyż świat zmienia się wraz z upływem czasu. Jak reprezentować te zmiany w rachunku zdań? Jedynym sposobem jest odpowiednie indeksowanie symboli. Np.:

$$L^1_{1,1} \wedge ZwróconyWPrawo^1 \wedge RuchWPrzód^1 \Rightarrow L^2_{2,1}$$

$$ZwróconyWPrawo^1 \wedge ObrótWLewo^1 \Rightarrow ZwróconyWGórę^2$$

Powyższy przykład ilustruje ograniczenia w sile wyrazu właściwe dla rachunku zdań. Baza wiedzy musi zawierać liczne formuły o podobnej formie różniące się „indeksami", gdzie każdy zbiór „indeksów" identyfikuje inne „fizyczne" miejsce świata lub czas (np. kratkę w „świecie Wumpusa" w pewnej chwili czasu). Nie ma możliwości wyrażenia w zwarty sposób wspólnej własności wszystkich „fizycznych" miejsc.
Podobnie jest z reprezentacją możliwych akcji agenta - musimy wprowadzić osobne symbole i zdania dla każdej chwili czasu *t* i każdego miejsca [x,y]. W „świecie Wumpusa" dla każdego kierunku, każdej kratki i czasu musiałaby istnieć formuły o postaci

$$L^t_{x,y} \wedge ZwróconyWPrawo^t \wedge RuchWPrzód^t \Rightarrow L^{t+1}_{x+1,y}$$

Efektem jest „eksplozja” liczby zdań w bazie wiedzy przy rosnącym rozmiarze świata i liczbie wykonanych akcji.

5 Pytania testowe

5.1 System agentowy

1. Wyjaśnić cztery definicje pojęcia „inteligencji”.
2. Jakie są zasadnicze **zastosowania** Sztucznej Inteligencji w informatyce?
3. Omówić zasadę pracy **systemu z bazą wiedzy** i główne funkcje komunikowania się sterowania z bazą wiedzy.
4. Wyjaśnić pojęcia: „**wynikanie”** i „**wnioskowanie”** zdań. Zilustrować odpowiedź na przykładach ze świata matematyki
5. Omówić pojęcie **przeszukiwania przestrzeni stanów** jako uniwersalnego sposobu rozwiązywania problemów.
6. Jakie są główne **paradygmaty uczenia** w systemie Sztucznej Inteligencji ?

5.2 Rachunek zdań

1. Omówić **elementy składni** rachunku zdań.
2. Wyjaśnić **semantykę** rachunku zdań.
3. Przedstawić dwa twierdzenia o **dedukcji** (poprawnym wnioskowaniu)
4. Przedstawić typowe **reguły** wnioskowania.
5. Jakie są postacie **normalne** zdań ?
6. Omówić wnioskowanie przez **rezolucję**.
7. Omówić wnioskowanie „**w przód**” i „**wstecz**”.
8. Przedstawić problem własności **zmiennych w czasie** i **wspólnych własności miejsc**.

6 Bibliografia

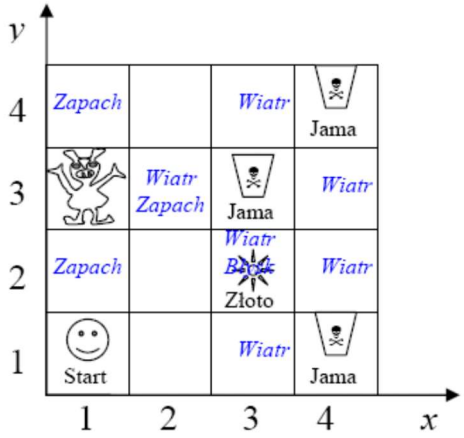
1. S. Russel, P. Norvig: *Artificial Intelligence. A modern approach.* Prentice Hall, 2002 (2nd ed.), 2013 (3d ed.). [Rozdziały 1, 2]
2. M. Flasiński: *Wstęp do sztucznej inteligencji*. Wydawnictwo Naukowe, PWN, 2011. [Rozdziały 2, 14]
3. D. L. Poole, A. K. Mackworth: *Artificial Intelligence - foundations of computational agents*. Cambridge University Press, 2009. [Rozdział 2]

7 Zadania

7.1 Wynikanie zdań

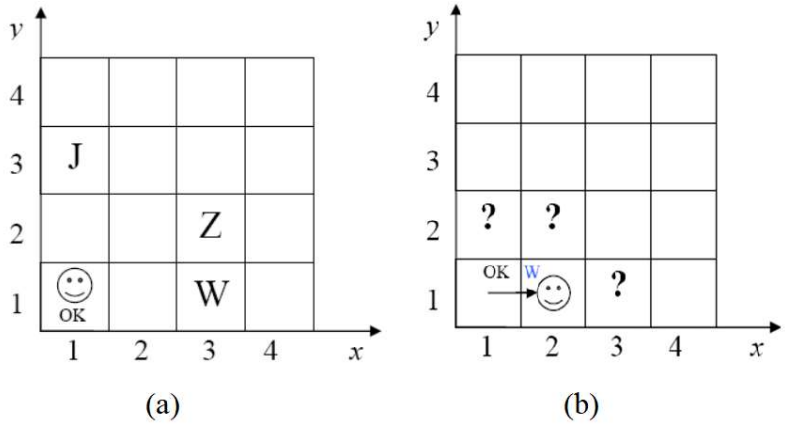
„*Świat Wumpusa*” to wczesna gra komputerowa. Środowisko ma postać sąsiadujących ze sobą 2-wymiarowych prostokątnych komórek w prostokątnym układzie współrzędnych XY (Rysunek 17).

Agent rozpoczyna grę w komórce startowej [1, 1]. Celem agenta jest znalezienie złota i wydostanie się z jaskini (powrót do kwadratu startowego [1,1]). Przeszkodami są jamy (może być ich wiele) lub Wumpus (zawsze jest tylko jeden). Wejście agenta do komórki z jamą lub Wumpusem jest dla niego śmiertelne. Nie zawsze agent może uzyskać dodatni wynik - świat może być „źle" zdefiniowany - złoto może być w jamie lub być niedostępne jeśli jest otoczone *jamami* lub w komórce z *Wumpusem*.



Rysunek 17. Przykład agenta w "świecie Wumpusa"

Obserwacje tworzą 5-elementowy wektor: [*zapach, wiatr, błysk, uderzenie, krzyk*]. W komórkach sąsiadujących z Wumpusem agent czuje jego zapach. W komórkach przylegających do jamy agent czuje wiatr. W kwadracie, gdzie znajduje się złoto agent obserwuje błysk. Jeśli agent wejdzie na ścianę to czuje uderzenie. Agent posiada jedną strzałę i może nią trafić Wumpusa. Kiedy Wumpus zostaje zabity, w jaskini słychać krzyk. Wektor możliwych akcji: { *ObrótWLewo,* ObrótWPrawo*, RuchWPrzód, Podnieść, Upuścić, Strzelić, Wyjść, Zginąć* }. *„Strzelić"* – agent zużywa jedyną strzałę w kierunku patrzenia agenta. *„Podnieść"* złoto, *„Upuścić"* złoto. „Wyjść" - pozwala agentowi opuścić jaskinię, o ile znajduje się on w kwadracie startowym. *„Zginąć*" - agent ginie jeśli wejdzie do komórki, w której znajduje się *Wumpus* lub *jama*.



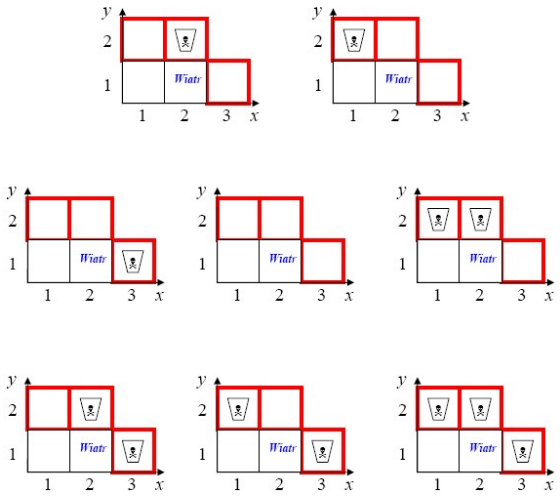
Rysunek 18. Sytuacja agenta po (a) pierwszej i (b) drugiej obserwacji

W *świecie Wumpusa* załóżmy sytuację agenta powstałą po: pustej obserwacji w [1,1] (Rysunek 18(a)), ruchu w prawo do [2,1] i obserwacji wiatru w [2,1] (Rysunek 18(b)) (gdzie ☺ oznacza położenie agenta, OK – wiedza o tym, że kratka jest „bezpieczna", **J** – jama, **W** – Wumpus, **Z** – złoto).

Teraz agent chciałby wiedzieć, czy w sąsiednich kratkach, [1,2], [2,2] i [3,1], występują jamy. Może to sprawdzić, zadając bazie wiedzy pytania o zachodzenie relacji wynikania poniższych zdań z aktualnej bazy wiedzy:

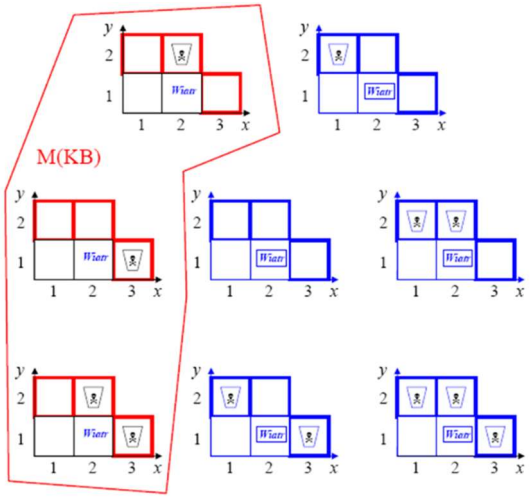
A) α1= „w kratce [1,2] nie ma jamy”
B) α2= „w kratce [2,2] nie ma jamy”

Przeprowadzimy wnioskowanie metodą „przeliczania modeli”, czyli sprawdzenia relacji zawierania się modelu KB w modelu zapytania. Możliwych jest 2^3 (= 8) stanów zajętości trzech kratek otoczenia agenta przez jamy (Rysunek 19).



Rysunek 19. Możliwe stany zajętości trzech kratek otoczenia agenta

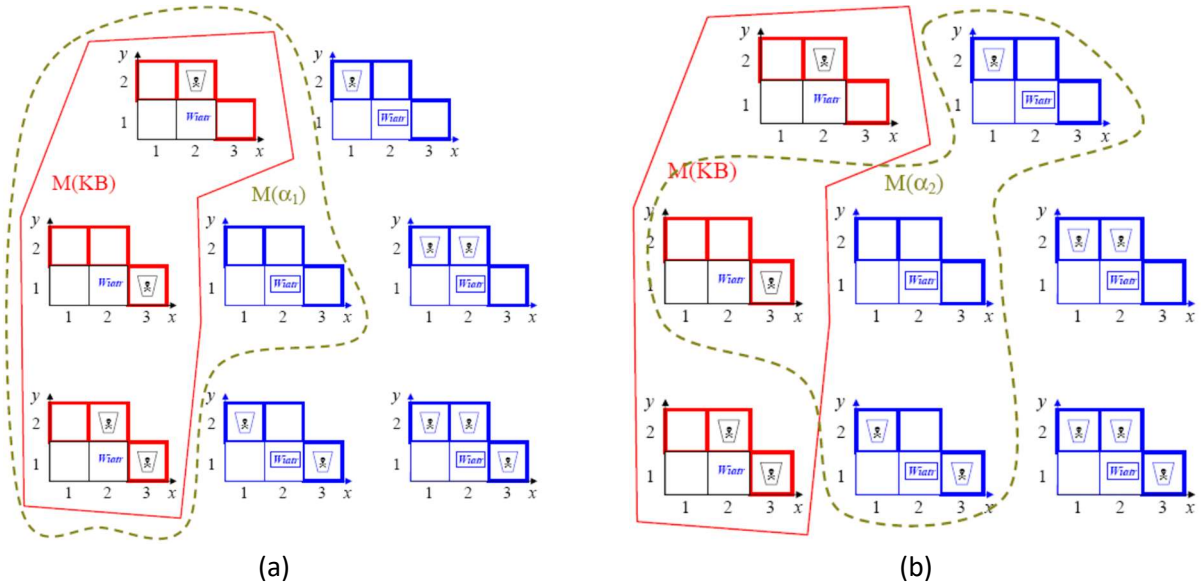
Aktualny model bazy wiedzy uwzgledniającej obie obserwacje, M(KB), zawiera jedynie trzy stany (Rysunek 20).



Rysunek 20. Aktualny model bazy wiedzy po drugiej obserwacji

Zdanie α_1 = „w kratce [1,2] nie ma jamy - jest ona bezpieczna” posiada model złożony z czterech stanów i obejmuje on model bazy wiedzy: $M(KB) \subseteq M(\alpha_1)$ (Rysunek 21(a)). Stąd wnioskujemy, że $(KB \models \alpha_1)$, czyli powyższe zdanie wynika z KB (jest prawdziwe w świetle posiadanej wiedzy).

Wprawdzie zdanie α_2 = „w kratce [2,2] nie ma jamy - jest ona bezpieczna” również posiada model złożony z czterech innych stanów, ale nie obejmuje on modelu bazy wiedzy: $M(KB) \not\subset M(\alpha_2)$ (Rysunek 21(b)). Czyli zdanie α_2 nie wynika z KB: $\neg(KB \models \alpha_2)$



Rysunek 21. Relacje modeli obu zapytań z modelem bazy wiedzy: (a) model zdania α_1 zawiera model bazy wiedzy, (b) model zdania α_2 nie zawiera modelu bazy wiedzy

7.2 Wnioskowanie z tablicą prawdy w rachunku zdań

Przyjmiemy symbole zdaniowe dla agenta w „świecie Wumpusa”:

- niech formuła $J_{i,j}$ będzie prawdziwa, gdy jest jama w kratce [i, j];
- niech formuła $W_{i,j}$ będzie prawdziwa, gdy jest wiatr w kratce [i, j].

Jak wiemy, *model* KB wyznacza wartość „*prawdy*” dla zdań zawartych w KB. Przyjmijmy założenie dla *sytuacji początkowej*, prawdziwe jest zdanie:

R_1: $\neg J_{11}$ („nie ma jamy w kratce [1,1]”)

Związek stanu świata z wynikiem obserwacji agenta: dla każdej kratki zachodzi „agent odczuwa wiatr gdy w sąsiedniej kratce jest jama”. Możemy to wyrazić dla dwóch pierwszych kratek jako:

zdanie R_2: $W_{11} \Leftrightarrow (J_{12} \vee J_{21})$

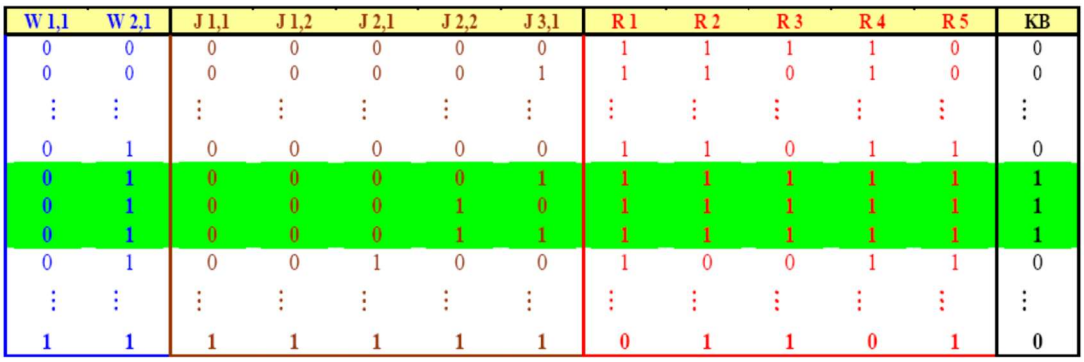
zdanie R_3: $W_{21} \Leftrightarrow (J_{11} \vee J_{22} \vee J_{31})$

Niech po dwóch obserwacjach (wykonanych w kratkach [1,1] i [2,1]) baza wiedzy zawiera też zdania:

zdanie R_4: $\neg W_{11}$ („agent nie odczuwa wiatru w [1,1]”)

zdanie R_5: $W_{2,1}$ („agent odczuwa wiatr w [2,1]”).

Metodą tablicy prawdy (przeliczeniem możliwych stanów) wyznaczymy model aktualnej bazy wiedzy ze względu na zdania od R_1 do R_5 (Rysunek 22).



W 1,1	W 2,1	J 1,1	J 1,2	J 2,1	J 2,2	J 3,1	R 1	R 2	R 3	R 4	R 5	KB
0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	1	1	1	0	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	0	0	0	0	0	1	1	0	1	1	0
0	1	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	1	0	0	1	0	0	1	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	1	0	1	1	0	1	0

Rysunek 22. Ilustracja metody przeliczania stanów (w tabeli prawdy) dla wyznaczania modelu bazy wiedzy

W zdaniach $R_1,...,R_5$ występuje 7 symboli zdaniowych (W_{11}, W_{21}, J_{11}, J_{12}, J_{21}, J_{22}, J_{31}). Stąd liczba potencjalnych stanów dla tej sytuacji wynosi 2^7 (= 128). Tylko dla trzech stanów wszystkie zdania od R_1 do R_5 są jednocześnie prawdziwe i te trzy stany tworzą model aktualnie rozpatrywanej bazy wiedzy KB.

Jeżeli zadamy pytanie, w którym występują wszystkie lub jedynie część powyższych symboli zdaniowych, to w podobny sposób co dla KB wyznaczymy model tego zapytania. Jeśli dla każdego wiersza tabeli prawdy spełnionego dla KB również zapytanie będzie spełnione (może być szerzej spełnione) to oznacza, że wynika ono z bazy wiedzy.

7.3 Poprawność reguł wnioskowania

Sprawdzić, które z podanych niżej reguł wnioskowania są poprawne?

$$\frac{\alpha \to \beta,\ \beta \to \gamma}{\alpha \to \gamma} \qquad \frac{\alpha \to \beta,\ \beta \to \gamma,\ \alpha}{\gamma} \qquad \frac{\alpha \vee \beta,\ \alpha \vee \neg\beta}{\alpha} \qquad \frac{\alpha \to \beta}{\neg\beta \to \neg\alpha}$$

$$\frac{\alpha \to \beta}{\neg\alpha \to \neg\beta} \qquad \frac{\alpha \to (\beta \to \gamma)}{\beta \to (\alpha \to \gamma)}$$

Można zastosować tabelę prawdy dla pokazania, że każdy model zdań (czyli takie stany wyznaczone przez wartościowania symboli zdaniowych α, β, γ, dla których zdania te są spełnione) występujących w poprzedniku danej reguły zawiera się w modelu zdania występującego w następniku reguły.

Można też wyprowadzić następnik z poprzednika w sposób analityczny stosując przekształcenia równoważnościowe lub wcześniej udowodnione reguły.

7.4 Postać normalna CNF

Przekształcić poniższe zdanie do postaci normalnej dla wnioskowania przez **rezolucję**:

$$A1 \Leftrightarrow (L1 \vee L2).$$

Wyjaśnić realizowane przekształcenia.

Stosujemy następujące kroki przekształceń:

1) Usuwamy $\Leftrightarrow$, zamieniając $\alpha \Leftrightarrow \beta$ na $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

$$(A1 \Rightarrow (L1 \vee L2)) \wedge ((L1 \vee L2) \Rightarrow A1)$$

2) Usuwamy ⇒, zamieniając α ⇒ β na ¬α∨ β.

$$(\neg A1 \vee L1 \vee L2) \wedge (\neg(L1 \vee L2) \vee A1)$$

3) Wprowadzamy negację ¬ do środka nawiasów stosując reguły de Morgana i ewentualnie eliminujemy podwójną negację:

$$(\neg A1 \vee L1 \vee L2) \wedge ((\neg L1 \wedge \neg L2) \vee A1)$$

4) Stosujemy prawo rozdzielczości (∨ nad ∧):

$$(\neg A1 \vee L1 \vee L2) \wedge (\neg L1 \vee A1) \wedge (\neg L2 \vee A1)$$

Uzyskaliśmy postać CNF: koniunkcję klauzul o postaci alternatyw literałów.
Do bazy wiedzy dołączamy zbiór trzech klauzul, stosując regułę eliminacji koniunkcji:

$$(\neg A1 \vee L1 \vee L2); (\neg L1 \vee A1); (\neg L2 \vee A1)$$