Reprezentacja wiedzy w języku logiki

Metody przeszukiwania w przestrzeni stanów sformułowane były w postaci dość ogólnej, jednak wymagały reprezentacji zagadnienia we właściwej formie, tzn. przestrzeni stanów, zbioru operatorów, a dodatkowo przydatna/potrzebna była informacja heurystyczna w formie funkcji oceny stanów.

Ogólnie, format i sposób reprezentacji wiedzy o zagadnieniu są niezwykle istotne i mają bezpośredni wpływ na efektywność — lub w ogóle zdolność — znalezienia rozwiązania.

Istnieje szereg opracowanych ogólnych podejść do problemu reprezentacji, i różne reprezentacje mają zwykle związane z nimi techniki **wnioskowania**, czyli formowania pewnych ustaleń pomocniczych (wniosków), mogących służyć do znalezienia ostatecznego rozwiązania problemu.

Jednym z najpopularniejszych schematów reprezentacji wiedzy jest język logiki matematycznej.

Dlaczego logika matematyczna jest dobrym językiem reprezentacji wiedzy dla sztucznej inteligencji?

Z jednej strony, język logiki jest zbliżony do sposobu w jaki ludzie myślą o świecie, i myśli swe wyrażają w zdaniach języka naturalnego. Czasami mówi się kolokwialnie, że człowiek myśli „logicznie". Kategorie, którymi myśli i mówi człowiek obejmują takie konstrukcje jak: obiekty, związki między obiektami (relacje), stwierdzenia faktów prostych i złożonych, zdania, spójniki zdaniowe, wyrażenia faktów warunkowych, a nawet kwantyfikatory.

Z drugiej strony, logika matematyczna dostarcza precyzyjnego aparatu wnioskowania opartego na dowodzeniu twierdzeń. Ludzie, myśląc, również stosują podobne wnioskowanie logiczne, zatem aparat logiki matematycznej wydaje się dobrą platformą reprezentacji wiedzy agenta inteligentnego, którego sposób wyrażania faktów i wnioskowania byłby zbliżony do ludzkiego.

Przykład: świat wumpusa

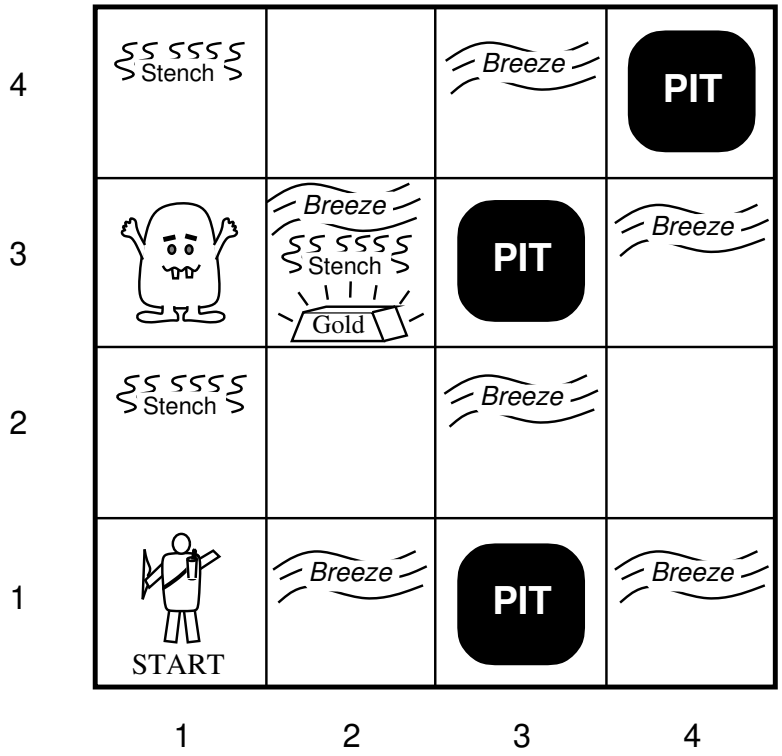
Do sprawdzenia działania wielu metod przydatne jest środowisko testowe dostatecznie proste, aby można było intuicyjnie określać właściwe reprezentacje i sprawdzać proste koncepcje, ale jednocześnie dostatecznie bogate, aby pozwoliło konfrontować te metody z coraz bardziej realnymi przeszkodami.

Jednym z takich testowych środowisk podręcznikowych jest **świat wumpusa**.[1] W tym środowisku porusza się agent dążący do znalezienia złota (i bezpiecznego wyniesienia go z jaskini). Na przeszkodzie stoją zapadliny (*pits*), w które agent może wpaść, i potwór (tytułowy *wumpus*), który może agenta zjeść.

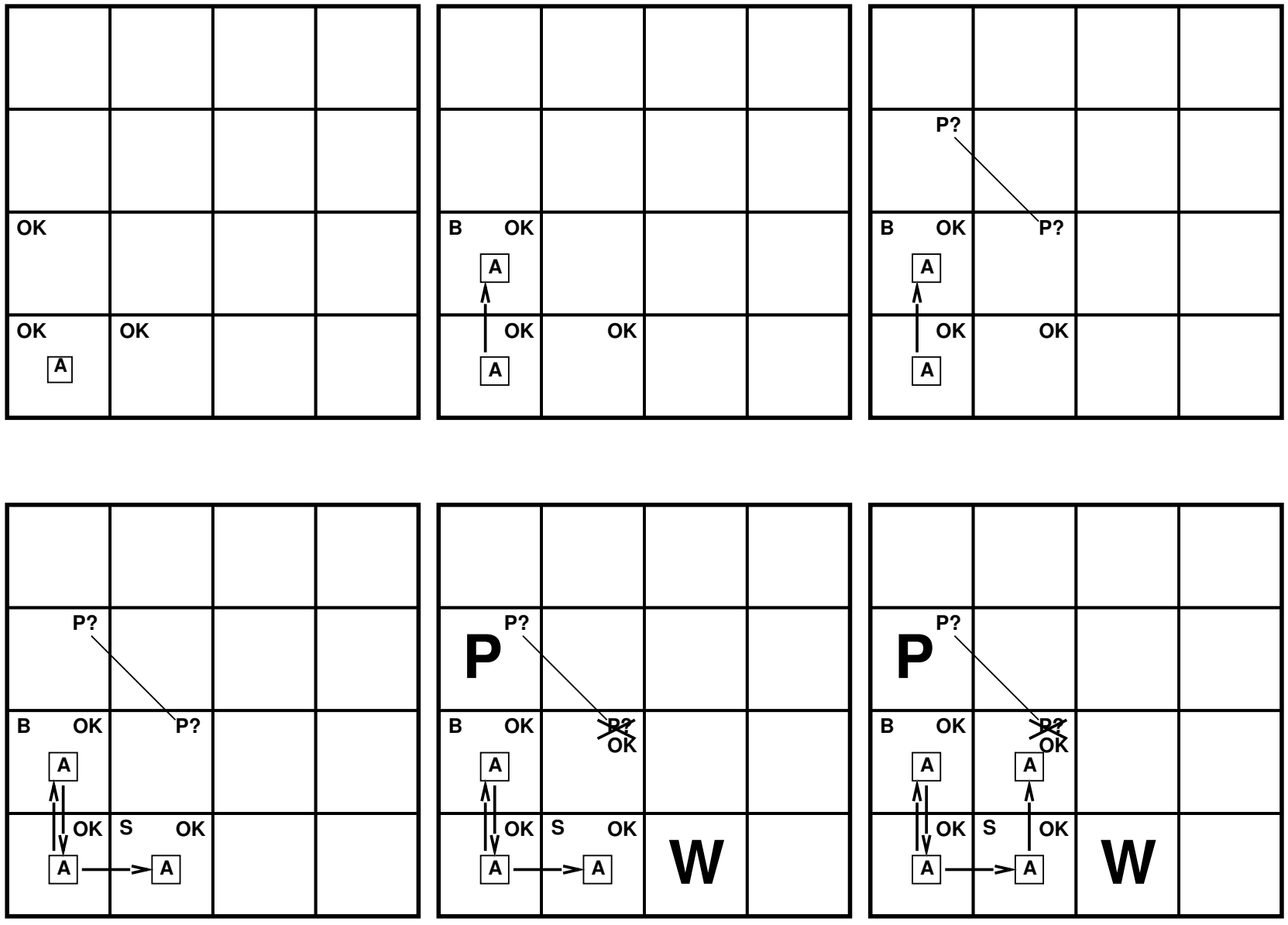
Agent może jedynie obracać się w prawo lub w lewo, poruszać się po jednym kroku do przodu, wystrzelić z łuku jedyną posiadaną strzałę (na wprost), podnieść złoto, gdy je znajdzie, i wyjść z jaskini, jeśli znajduje się w punkcie startowym.

[1] Przedstawione tu przykłady i diagramy świata wumpusa zaczerpnięte zostały z podręcznika Russella i Norviga „Artificial Intelligence A Modern Approach" i materiałów udostępnionych na stronie internetowej Stuarta Russella.

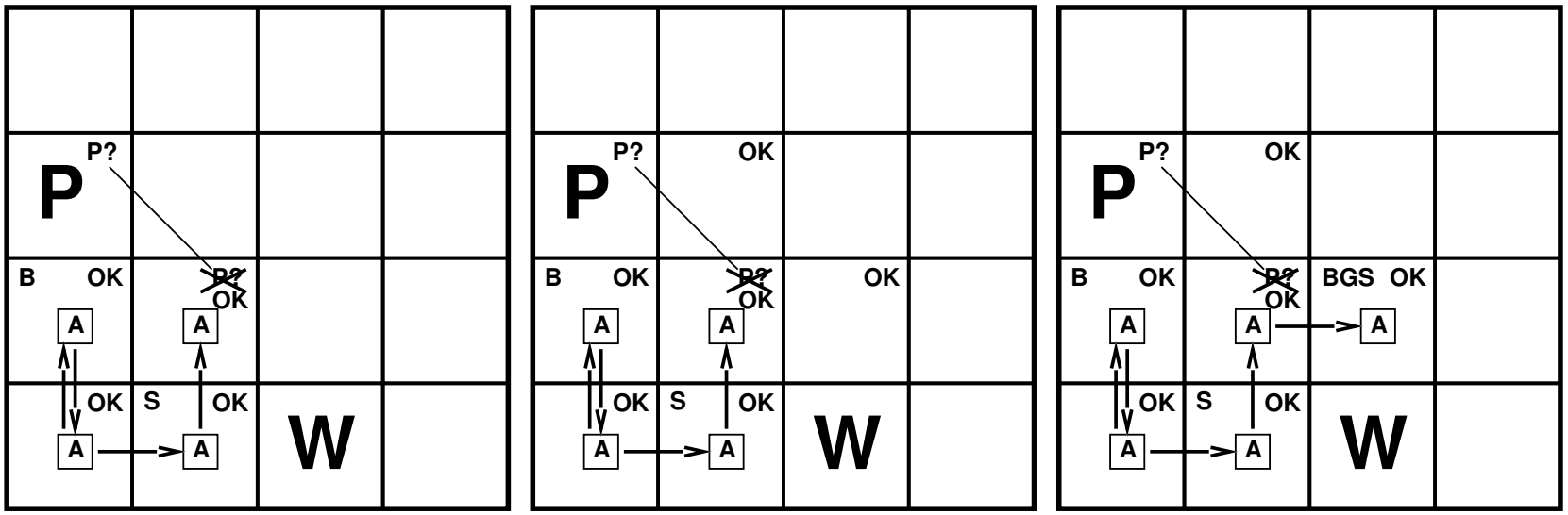
Agent otrzymuje pewne informacje o środowisku (dane otrzymywane z receptorów agenta nazywa się **perceptami**): wyczuwa smród wumpusa (*stench*) i powiew powietrza z zapadlin (*breeze*), jeśli znajduje się w polu sąsiadującym z nimi. Ponadto zauważa złoto (*gold*), ale tylko jeśli jest w tym samym polu co ono. Nie może jednak sprawdzić swojej bezwzględnej pozycji (*à la* GPS), może jedynie sam swoją pozycję rejestrować. Ściany jaskini wyczuwa jedynie przez próby wejścia w nie, które powodują odbicia.



Przykład: poruszanie się w świecie wumpusa



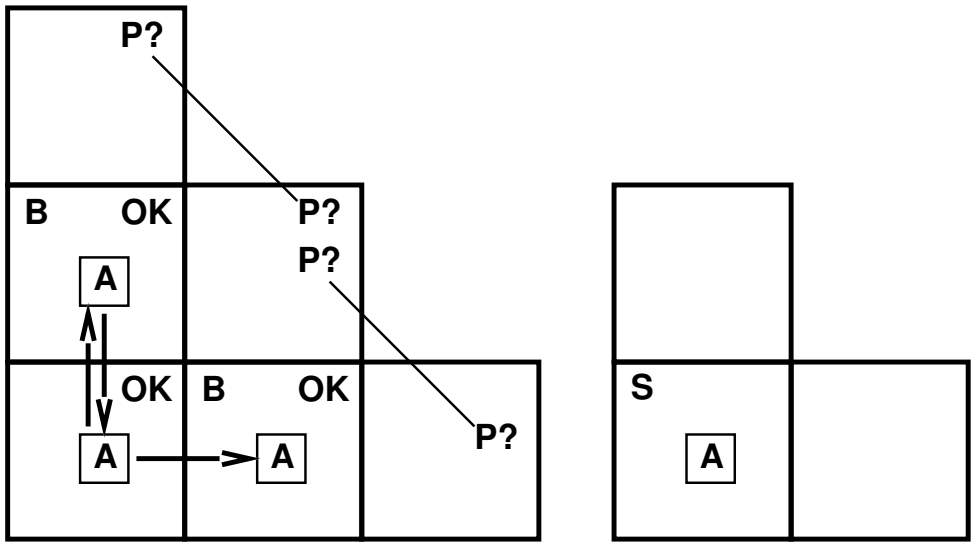
Przykład: poruszanie się w świecie wumpusa (cd.)



Jednak nie zawsze można skutecznie działać w świecie wumpusa posługując się tylko wnioskowaniem logicznym.

W niektórych przypadkach jedynym rozwiązaniem jest „strzelanie", czyli wybór ruchu na ślepo, i dopiero potem analizowanie sytuacji.

Oczywiście o ile przeżyjemy!!



Rachunek zdań: składnia i formuły poprawne formalnie (wff)

Logika zdań jest bardzo prostym językiem logicznym. Pozwala na pisanie **formuł atomowych** w oparciu o **symbole zdaniowe**. Pisząc formułę logiczną, stwierdzamy pewien fakt. Przykłady formuł atomowych: $P, Q, R, WumpusAt\_1\_5, HaveGold$.

Możemy również pisać **formuły złożone**, które są konstruowane z innych formuł przy użyciu **spójników logicznych**: $\neg$ (negacja), $\wedge$ (koniunkcja), $\vee$ (alternatywa), $\Rightarrow$ (implikacja) i $\Leftrightarrow$ (równoważność) (ang. *biconditional*).

Formuły złożone mogą składać się z innych formuł złożonych z użyciem nawiasów lub bez nich, jeśli nie są niejednoznaczne. Te zasady tworzenia wyrażeń formalnie poprawnych, zwanych *well-formed formulas* lub wffs, tworzą razem **składnię** języka.

Przykłady wyrażeń formalnie poprawnych (wff):

$(P \wedge Q) \vee (\neg P \wedge \neg Q)$
$\neg\neg P$
$(AgentAt\_1\_1 \wedge PitAt\_1\_2) \Rightarrow Breeze$
$HaveGold \vee \neg HaveGold$
$HaveGold \wedge \neg HaveGold$

Przykłady wyrażeń niepoprawnych formalnie (nie-wff):

$P \wedge \wedge Q$
$P \neg Q$
$P(WumpusAt\_1\_5)$

Spróbuj wyjaśnić, dlaczego te wyrażenia nie są wff.

Rachunek zdań: semantyka

Składnia definiuje język. W przypadku logiki zdań składa się ona z: zestawu symboli zdaniowych (mogą one być dowolne), zestaw spójników logicznych (to w zasadzie tylko te pięć, które wprowadziliśmy) oraz reguły ich użycia.

Składnia nie określa znaczenia formuł. To jest rola **semantyki** języka. Semantyka nadaje znaczenie każdemu z symboli zdaniowych. Po zdefiniowaniu znaczenia formuły możemy zacząć mówić o tym, czy jest ona prawdziwa czy fałszywa. I to jest właściwym celem reprezentacji logicznej.

Zauważ, że zapisaliśmy niektóre symbole zdaniowe już w taki sposób, aby zasugerować, co mają oznaczać: $AgentAt\_1\_1, PitAt\_1\_2, HaveGold$. Inne symbole są po prostu ogólne; można im przypisać dowolne znaczenie, abstrakcyjne lub bardzo konkretne: P, Q, R.

Jednak język logiki musi być elastyczny i bardzo ogólny — z samego zapisu symbolu zdaniowego nigdy nie możemy wnosić co ona rzeczywiście znaczy.

Rachunek zdań: semantyka — światy możliwe, interpretacje

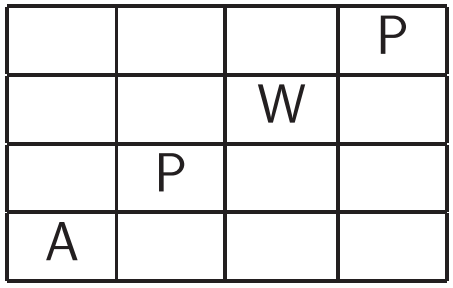
Jeśli skojarzymy formułę (zapisaną pojedynczym symbolem zdaniowym) $AgentAt\_1\_1$ ze takim znaczeniem, że agent świata wumpusa jest aktualnie na pozycji (1,1), to nadal nie daje nam to możliwości sprawdzenia, czy formuła stwierdza prawdę lub nie. Jest całkiem możliwe, że w jednym konkretnym przypadku gry jest to prawdą, podczas gdy w wielu innych przypadkach jest fałszem.

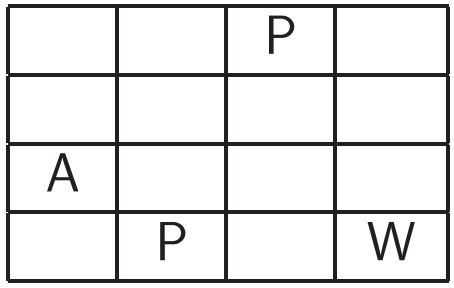
Semantyka rozwiązuje ten problem przez powiązanie każdej formuły atomowej ze **światem możliwym**, który jest konkretną konfiguracją opisywanej dziedziny problemowej, w której wszystkie opisywane obiekty znajdują się w ściśle określonych stanach. Odbywa się to za pomocą **funkcji interpretacji**, która wiąże każdą formułę atomową (lub symbol zdaniowy) z określonym znaczeniem w odniesieniu do takiego świata możliwego, a tym samym definiuje wartość logiczną takiej formuły.

Ważne jest, aby funkcja interpretacji była całkowicie zdefiniowana, tj. każdy symbol zdaniowy występujący w języku był powiązany z jakimś aspektem świata możliwego, a odpowiadająca mu formuła atomowa mogła być jednoznacznie zinterpretowany jako mająca wartość 1 lub 0.

Rachunek zdań: semantyka — interpretacje, modele

Interpretacja wykorzystująca świat możliwy po lewej stronie przypisuje formule $AgentAt\_1\_1$ wartość prawdy 1 (czyli: Prawda), podczas gdy inna interpretacja wykorzystująca świat możliwy po prawej przypisuje tej samej formule wartość prawdy 0 (czyli: Fałsz):





Światy możliwe są bardziej precyzyjnie określane jako **modele**.

Zauważ, że lokalizacje wszystkich obiektów (agenta, wumpusa, dziur), jeśli są opisane przez symbole zdaniowe języka, muszą być określone przez każdy model, niezależnie od tego, czy agent świata wumpusa zna te lokalizacje, czy nie.

W tych modelach nie jest możliwe np. posiadanie innego obiektu F, którego położenie mogłoby opisane formułami takimi jak FAt_2_2. Gdyby tak było, wtedy powyższe konfiguracje nie byłyby modelami dla takiej dziedziny problemowej, ponieważ nie odzwierciedlają położenia obiektu F.

Rachunek zdań: semantyka — spełnianie formuł

Mając konkretną formułę zdaniową, atomową lub złożoną, niektóre modele przypisują jej wartość logiczną 1, podczas gdy inne przypisują jej wartość logiczną 0. Zauważ, że nie ma innej opcji. Wszystkie obiekty opisane przez symbole zdaniowe są obecne w modelu, a wszystkie ich własności są tam odzwierciedlone.

Mówimy, że model m **spełnia** formułę f, jeśli przypisze jej wartość prawdy 1.
Powiemy również, że model m, który spełnia formułę f, jest **modelem tej formuły**.

Zwróć uwagę na inne znaczenie słowa: model. Dowolny model (dla określonej dziedziny problemowej) może spełniać daną formułę lub ją sfalsyfikować. Ale jeśli ją spełnia, to jest **modelem tej formuły**.

Z definicji, jeśli wzór spełnia wszystkie modele, to nazywamy ją **tautologią**.
Przykładem tautologii jest $P \vee \neg P$. Jej wartość logiczna nie zależy od modelu — musi mieć przypisaną wartość logiczną 1 przy dowolnym modelu.

I odwrotnie, jeśli formuła nie może być spełniony przez żaden model, to jest nazywana **niespełnialną**. Przykładem formuły niespełnialnej może być $P \wedge \neg P$.
Jej wartość prawdy również nie zależy od modelu — jest to stała 0.

Rachunek zdań: semantyka — spełnianie formuł złożonych

Zatrzymajmy się na chwilę, aby zwrócić uwagę na istotny szczegół. Przede wszystkim model definiuje prawdziwe wartości wszystkich formuł atomowych (symboli zdaniowych). Mając to, wszystkie inne formuły (złożone) mają swoje wartości prawdy określone przez semantykę konkretnych spójników logicznych.

Na przykład załóżmy, że model m przypisuje symbolowi $AtAgent\_1\_1$ wartość 1. Następnie rozważmy formułę $\neg AtAgent\_1\_1$. Nie możemy wybrać dla niej dowolnej wartości prawdy; musimy przyjąć, że jego wartość to 0. To samo dotyczy wszelkich formuł używających $\wedge, \vee$ itd. Wartości prawdy wszystkich formuł, które je zawierają, są definiowane przez ich tabele prawdy.

Jednocześnie chcielibyśmy upewnić się, że model ten przypisuje 0 wszystkim formułom (atomowym) typu: $AtAgent\_1\_2, AtAgent\_2\_1, AtAgent\_2\_2, \ldots$. Ale reguły logiki zdań nie wymuszają tego. Powyższe formuły nie są w żaden sposób powiązane, więc model może przypisać im dowolne wartości logiczne. Posiadanie którejkolwiek z nich równej 1 złamałoby zasady świata wumpusa (ponieważ powinien być tylko jeden agent i może być tylko w jednym miejscu na raz), ale z logicznego punktu widzenia nic złego by się nie stało.

Rachunek zdań: semantyka — funkcje interpretacji

W zasadzie funkcja interpretacji przypisuje formule atomowej jakiś model (który może być modelem tej formuły lub nie). Istnieje wiele modeli (możliwych światów), które można rozważać. Ale z punktu widzenia ustalania prawdziwości formuł ważne jest tylko to, które symbole zdaniowe dane modele spełniają, a które nie.

Ponieważ w określonej dziedzinie problemowej może występować tylko pewna liczba symboli zdaniowych (określona liczbą obiektów i ich reprezentowanych własności), to są tylko 2^N (N - liczba symboli zdaniowych) typów modeli które naprawdę się liczą: te, które spełniają określony symbol, i te, które go falsyfikują.

Z tego powodu, dla logiki zdań, często odrzucamy wielką różnorodność światów możliwych i redukujemy zbiór modeli do zbioru różnych zerojedynkowych N-krotek kojarzących wartości prawdy ze wszystkimi symbolami zdań:

	$AgentAt\_1\_1$	$WumpusAt\_1\_5$	...
m_1	0	0	...
m_2	0	1	...
m_3	1	0	...
m_4	1	1	...
...	...	...	...

Rachunek zdań: semantyka — zbiory modeli

Biorąc pod uwagę poprzednie uogólnienie, możemy alternatywnie zacząć patrzeć na formuły logiczne jako na zwarty sposób przedstawiania zbiorów modeli, a mianowicie tych, które są modelami określonej formuły.

Na przykład o formule $AgentAt\_1\_1$ można myśleć jako o reprezentacji wszystkich modeli, w których agent znajduje się w pozycji (1,1).

Rachunek zdań: kilka praw logicznych

Ponieważ dla danej (złożonej) formuły interesuje nas przede wszystkim wyznaczenie jej wartości prawdy, może być korzystne korzystanie z pewnych przekształceń, które można wykonać na formułach logicznych z zachowaniem ich wartości prawdy.
W poniższych wzorach używa się symbolu równoważności $\equiv$, który oznacza, że w procesie oceny prawdziwości formuły jedna strona może być zastąpiona przez drugą.

Łączność:

$$\begin{aligned} p \wedge (q \wedge r) &\equiv (p \wedge q) \wedge r \\ p \vee (q \vee r) &\equiv (p \vee q) \vee r \end{aligned}$$

Ze względu na łączność zarówno koniunkcji, jak i alternatywy, możemy zapisać wiele kolejnych wystąpień każdego z tych spójników bez nawiasów. Dzieje się tak dlatego, że wartość logiczna formuły z wielokrotnymi wystąpieniami któregoś z tych spójników nie zależy od kolejności, w jakiej spójniki są interpretowane:

$$\begin{aligned} p_1 \wedge (p_2 \wedge (p_3 \wedge (...))) &\equiv p_1 \wedge p_2 \wedge p_3 \wedge ... \\ p_1 \vee (p_2 \vee (p_3 \vee (...))) &\equiv p_1 \vee p_2 \vee p_3 \vee ... \end{aligned}$$

Rozdzielczość:

$$\begin{aligned} p \wedge (q \vee r) &\equiv (p \wedge q) \vee (p \wedge r) \\ p \vee (q \wedge r) &\equiv (p \vee q) \wedge (p \vee r) \end{aligned}$$

Prawa de Morgana:

$$\begin{aligned} \neg(p \wedge q) &\equiv (\neg p) \vee (\neg q) \\ \neg(p \vee q) &\equiv (\neg p) \wedge (\neg q) \end{aligned}$$

Inne przydatne tożsamości:

$$p \Rightarrow q \equiv \neg p \vee q$$

Wnioskowanie logiczne — wynikanie

Baza wiedzy (KB) jest zbiorem formuł reprezentujących koniunkcję wszystkich formuł składowych. Zbiór modeli takiego zbioru jest przecięciem zbiorów modeli formuł składowych.

W sztucznej inteligencji bazą wiedzy jest zwykle baza danych wszystkich faktów, które posiada agent AI. Wiedza agenta AI jest w sposób naturalny koniunkcją wszystkich faktów z tej bazy. Typowym działaniem agenta jest próba odpowiedzi na pytanie, czy z taką bazą wiedzy obowiązuje jakiś inny fakt, reprezentowany np. przez formułę f.

Może się zdarzyć, że w przypadku gdy wszystkie formuły ze zbioru KB są prawdziwe, inna formuła f będzie zawsze prawdziwa, przy wszystkich możliwych interpretacjach.

Mówimy wtedy, że f **wynika logicznie** z KB, co zapisujemy KB $\models f$.

Na przykład: $\{P, Q\} \models P \wedge Q$ $\quad$ $\{P \wedge Q\} \models P$ $\quad$ $\{P \vee Q, \neg P\} \models Q$ $\quad$ $\{P \Rightarrow Q, P\} \models Q$

W logice zdań jednym ze sposobów stwierdzania wynikania jest użycie tablic prawdy. W takiej tablicy wypisujemy wszystkie modele (przypisania prawdy dla symboli zdaniowych) i sprawdzamy, czy wszystkie te modele, które spełniają pełną bazę KB, również spełniają f.

Krótkie podsumowanie - pytania sprawdzające

Dla następujących przykładów odpowiedz czy zachodzi podane wynikanie logiczne.

1. $\{P \vee Q\} \models P \wedge Q$
2. $\{P \wedge Q\} \models P \vee Q$
3. $\{P, Q\} \models P \Rightarrow Q$
4. $\{P, Q\} \models \neg P \vee Q$
5. $\{P \Rightarrow Q, \neg Q\} \models \neg P$
6. $\{P \Rightarrow Q, \neg P\} \models \neg Q$
7. $\{P \Rightarrow Q, \neg P\} \models Q$
8. $\{P \Rightarrow Q, Q\} \models P$
9. $\{P \Rightarrow Q, Q \Rightarrow R\} \models R$

Wnioskowanie logiczne — modus ponens

W niektórych przypadkach na formułach logicznych możemy zastosować proces zwany **wnioskowaniem**. Przykład:

Pada deszcz.	(PadaDeszcz)
Kiedy deszcz pana, szosa jest mokra.	(PadaDeszcz⇒SzosaMokra)
Wniosek: szosa jest mokra.	(SzosaMokra)

Jest to przykład **reguły wnioskowania** zwanej **modus ponens**:

Dla dowolnych symboli zdaniowych p i q:

$$\frac{p, \quad p \Rightarrow q}{q}$$

albo, bardziej ogólnie:

Dla dowolnych $p_1, \ldots, p_k, q$:

$$\frac{p_1, \ \ldots \ , p_k, \quad (p_1 \wedge \cdots \wedge p_k) \Rightarrow q}{q}$$

Wnioskowanie logiczne — reguły wnioskowania

Możliwe są inne reguły wnioskowania, ogólnie zapisane zgodnie ze schematem:

$$\frac{f_1, ..., f_k \text{ (przesłanki)}}{g \quad \text{(wnioski)}}$$

Reguły wnioskowania możemy wykorzystać w procesie wnioskowania, stosując je sukcesywnie na formułach z bazy wiedzy, aż do uzyskania pożądanego wniosku. Jest to proces **wnioskowania** (ang. *inferencji*), a o każdej formule f uzyskanej w tym procesie mówi się, że jest **wyprowadzona** z KB, zapisywane $\text{KB} \vdash f$.

Algorytm **wnioskowania wprzód**:

```
repeat until no change to KB:
    foreach inference rule  f1,...,fk / g
        if  f1, ..., fk ∈ KB ∧ g ∉ KB
            add g to KB
```

Jeśli f zostanie ostatecznie dodana do KB, to $\text{KB} \vdash f$.

Wynikanie a wnioskowanie formuł

Zauważmy, że proces wnioskowania działa ściśle w domenie składni. Działa na formułach tak, jak są napisane i nie odnosi się do modeli ani sprawdzania prawdziwości.

Pojawia się zatem pytanie:

Jak ma się: KB $\models f$ do: KB $\vdash f$?

Czy są one równoważne?
Czy można wyprowadzić dowolną formułę, która wynika logicznie?
A także, czy dowolna wyprowadzalna formuła, wynika logicznie?

Krótkie podsumowanie - pytania sprawdzające

Dla następujących przykładów odpowiedz czy zachodzi podane wyprowadzanie.
Na początek przyjmij, że jedyną regułą wnioskowanie jest modus ponens: $\frac{\phi,\ \phi \Rightarrow \psi}{\psi}$
W dodatku do kroków wyprowadzania możesz wykorzystać prawa równoważności logicznej takie jak przedstawione na stronach 15 do 16 aby przekształcić formuły do pożądanej postaci równoważnej, zarówno w zbiorze KB jak i po prawej stronie.

1. $\{P, Q\} \vdash P \wedge Q$
2. $\{P \wedge Q\} \vdash P$
3. $\{P \wedge Q\} \vdash P \vee Q$
4. $\{P \Rightarrow Q, \neg Q\} \vdash \neg P$

Z kolei przyjmij, że oprócz modus ponens możesz dodatkowo użyć następujących reguł wnioskowania: $\frac{\phi \wedge \psi}{\phi}$ (eliminacja koniunkcji), $\frac{\phi,\ \psi}{\phi \wedge \psi}$ (wprowadzenie koniunkcji), i $\frac{\phi}{\phi \vee \psi}$ (wprowadzenie alternatywy).
Najpierw ponownie rozwiąż powyższe przykłady, a potem jeszcze następujące:

5. $\{P, Q\} \vdash P \Rightarrow Q$
6. $\{\neg P, Q\} \vdash P \Rightarrow Q$
7. $\{P \Rightarrow Q, P \vee Q\} \vdash Q$
8. $\{P \Rightarrow Q, P \wedge R\} \vdash Q \wedge R$

Reguły wnioskowania: poprawność i kompletność reguł

Reguła wnioskowania jest **poprawna** (ang. *sound*), jeśli pozwala na wyprowadzenie z dowolnej KB tylko takich formuł, które wynikają logicznie z tej KB. (Ale może nie wywodzić WSZYSTKICH takich formuł.)

Reguła wnioskowania jest **kompletna**, jeśli pozwala na wyprowadzenie z dowolnej KB WSZYSTKICH formuł, które wynikają logicznie z tej KB. (Ale może również wywodzić inne formuły, które być może nie wynikają logicznie z tej KB.)

Z powyższego wynika, że gdybyśmy mieli regułę wnioskowania, która byłaby zarówno poprawna, jak i kompletna, moglibyśmy użyć procesu wnioskowania, zamiast sprawdzania wszystkich modeli (tablic prawdy).

Ale nie jest łatwo znaleźć taką regułę wnioskowania. Na przykład, modus ponens jest poprawna, ale nie jest kompletna. Aby to zobaczyć, zauważmy, że w poprzednim przykładzie:

$$\text{KB}=\{\text{PadaDeszcz}, \text{PadaDeszcz}\Rightarrow\text{SzosaMokra}\}$$

byliśmy w stanie wyprowadzić za pomocą modus ponens formułę SzosaMokra, ale z tej bazy wiedzy wynika logicznie również $(\text{PadaDeszcz}\wedge\text{PadaDeszcz}\Rightarrow\text{SzosaMokra})$, której nie można wyprowadzić za pomocą reguły modus ponens.

Reguły wnioskowania: poprawność i kompletność zbiorów reguł

Załóżmy, że mamy więcej niż jedną regułę wnioskowania. Jak w tym przypadku działa poprawność/kompletność?

Powinno być jasne, że niezależnie od tego, jak wiele reguł wnioskowania rozważymy, chcemy, aby każda z nich była (indywidualnie) poprawna. Pojedyncza niepoprawna reguła pozwoliłaby algorytmowi wnioskowania na wprowadzenie do bazy wiedzy sprzecznych wniosków, niezależnie od tego, co mogą zaoferować inne reguły.

Przykład: rozważ dwie hipotetyczne reguły wnioskowania: $\frac{p,\ q}{p \vee q}$, $\frac{p,\ q}{\neg(p \vee q)}$
Pierwsza jest poprawna, ale druga nie.

Jeśli obie są obecne w systemie, algorytm wnioskowania będzie musiał wyprowadzić zarówno $(p \vee q)$, jak i $\neg(p \vee q)$. Bez względu na to, ile istnieje poprawnych reguł wnioskowania, jedna nieprawidłowa reguła może zepsuć cały system, pozwalając na wywnioskowanie fałszywej formuły.

Reguły wnioskowania: poprawność i kompletność zbiorów reguł (cd.)

Kompletność to inna historia. Może się zdarzyć, że dla pewnego zbioru reguł wnioskowania $\{\mathsf{IR}_1, \mathsf{IR}_2, ..., \mathsf{IR}_n\}$ system wnioskowania będzie w stanie z dowolnej KB wyprowadzić wszystkie formuły, które wynikają logicznie z tej KB. Wtedy taki zbiór reguł wnioskowania byłby kompletny jako całość, nawet jeśli którakolwiek, lub nawet wszystkie z tych reguł, mogą z osobna nie być kompletne.

Tak więc jednym ze sposobów stworzenia systemu wnioskowania, w którym wyprowadzenie formuł było równoważne z wynikaniem logicznym, byłoby znalezienie takiego zestawu poprawnych reguł wnioskowania, które razem tworzyłyby zbiór kompletny.

Jest to możliwe, ale jest też inne rozwiązanie.

Postać Koniunkcyjna Normalna

Trochę terminologii:
Literałem nazwiemy dowolną formułę atomową lub jej negację.
Klauzulę nazwiemy formułę, która jest alternatywą literałów.
O formule, która jest koniunkcją klauzul będziemy mówić, że jest w **postaci koniunkcyjnej normalnej** (ang. *Conjunctive Normal Form CNF*).

Krótko możemy powiedzieć, że formuła CNF jest koniunkcją klauzul.

Przykłady: $[(P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge R], (P \wedge Q), (P \vee Q), \neg P, P$
Przykłady formuł nie-CNF: $(P \Rightarrow Q) \wedge (Q \Rightarrow P), \neg(P \wedge Q)$

Fakt: dowolną formułę rachunku zdań można przekonwertować na równoważną formułę w postaci CNF.

Formuły CNF są przydatne, ponieważ pozwalają na wnioskowanie przy użyciu rezolucji.

I, co równie ważne, proces ten można całkowicie zautomatyzować.

Reguły wnioskowania: rezolucja

Korzystając ze znanej i użytecznej tożsamości:

$$(p \Rightarrow q) \equiv (\neg p \vee q)$$

możemy przepisać modus ponens do innej postaci:

$$\frac{p, \quad \neg p \vee q}{q}$$

Zauważ, że wyrażenia podświetlone na czerwono w pewnym sensie kasują się. Ma to sens, ponieważ jeśli wiemy, że p jest prawdziwe, to $\neg p$ jest z pewnością fałszywe i można je usunąć z alternatywy, otrzymując q jako nowy wniosek.

Obserwację tę można uogólnić do następującej reguły wnioskowania zwanej **rezolucją**:

$$\frac{p_1 \vee \cdots \vee p_n \vee q \ , \quad \neg q \vee r_1 \vee \cdots \vee r_m}{p_1 \vee \cdots \vee p_n \vee r_1 \vee \cdots \vee r_m}$$

Rezolucja może być używana jako jedyna reguła wnioskowania w poprawnym i kompletnym systemie dowodzenia twierdzeń.

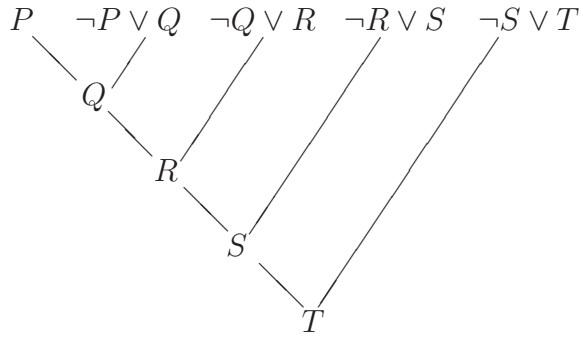
Wnioskowanie logiczne z wykorzystaniem rezolucji — przykłady

Rozważmy kilka typowych schematów wnioskowania logicznego. Załóżmy, że ilekroć wiemy, że P, to również Q jest prawdziwe, a jeśli Q to także R, jeśli R to S, oraz jeśli S to także T. Załóżmy, że wiemy również, że prawdą jest P. Wtedy powinniśmy być w stanie wywnioskować wszystkie te fakty w wyniku szeregu zastosowań reguły wnioskowania modus ponens. Zobaczmy, jak to działa z postacią CNF i rezolucją.

Oryginalne fakty: $P, P \Rightarrow Q, Q \Rightarrow R, R \Rightarrow S, S \Rightarrow T$

Te same fakty w postaci CNF i graficzna reprezentacja łańcucha kroków wnioskowania rezolucji:

(Zauważ, że w postaci graficznej łańcuch kroków wnioskowania tworzy rodzaj drzewa. Jest to typowe.)

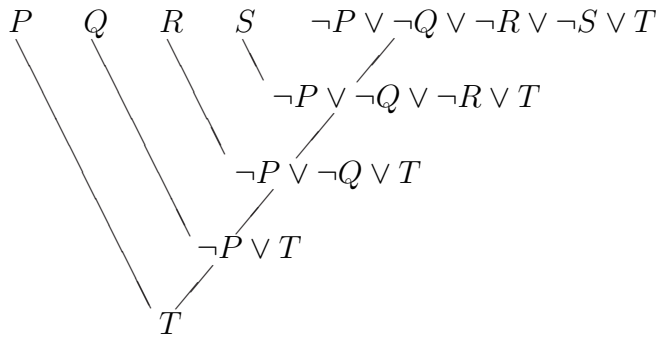


W powyższym schemacie wnioskowania wszystkie kroki wykonywane na klauzulach CNF mogą być równie dobrze wykonane z regułą modus ponens na oryginalnych formułach implikacyjnych. Jednak nie zawsze działa to w taki sposób.

Rozważmy inny schemat rozumowania. Załóżmy, że znanych jako prawdziwe jest kilka faktów: P, Q, R, S. Załóżmy dalej, że wszystkie te fakty razem wzięte implikują T.

Originalne fakty: $P, Q, R, S, (P \wedge Q \wedge R \wedge S) \Rightarrow T$.
Fakty w postaci CNF: $P, Q, R, S, (\neg P \vee \neg Q \vee \neg R \vee \neg S \vee T)$.

The resolution tree:



Tym razem wnioskowanie końcowej formuły T nie mogło być uzyskane za pomocą modus ponens. W tym celu musielibyśmy najpierw otrzymać formułę $P \wedge Q \wedge R \wedge S$, która wynika ze zbioru pierwotnych faktów, ale za pomocą modus ponens wyprowadzić jej się nie da. Tak samo zresztą nie można tego zrobić za pomocą rezolucji. Rezolucją udało się wyprowadzić T z postaci CNF oryginalnych faktów, ale nie uda się wyprowadzić prostej koniunkcji oryginalnych faktów, ponieważ rezolucja może tworzyć wnioski tylko przez połączenie dwóch klauzul ze skasowanymi literałami konfliktowymi.

Aby móc wyprowadzić formuły takie jak powyższa koniunkcja, musimy użyć rezolucji w specjalny sposób.

Puste klauzule

O pojedynczym literale możemy mówić jak o klauzuli unarnej, czyli o alternatywie tylko tego jednego literału. Ponadto dopuszczamy puste klauzule, które są traktowane jako alternatywy zero literałów. Można to wyjaśnić za pomocą notacji funkcyjnej dla alternatywy, dzięki asocjacyjności:

$$\begin{aligned} p_1 \vee p_2 \vee \ldots \vee p_n &\equiv \vee(p_1, p_2, \ldots, p_n) \\ p \vee q \vee r &\equiv \vee(p, q, r) \\ p \vee q &\equiv \vee(p, q) \\ p &\equiv \vee(p) \\ \sqcup &\equiv \vee() \end{aligned}$$

O ile prawdziwość dowolnej klauzuli niepustej zależy od prawdziwości jej składników, to klauzula pusta musi mieć stałą interpretację logiczną. Przez proste uogólnienie definicji wartości logicznych alternatywy możemy otrzymać, że klauzula pusta jest formułą fałszywą (niespełnialną). Ponieważ będziemy musieli posługiwać się pustą klauzulą w notacji logicznej, do jej oznaczenia używamy symbolu $\square$.

Klauzula pusta może być traktowana jako element neutralny dla spójnika alternatywy:

$$\square \vee p \equiv p \equiv p \vee \square$$

Wnioskowanie nie-wprost oparte na rezolucji

Poprawny i kompletny system dowodzenia twierdzeń może być stworzony przy użyciu rezolucji w procesie **wnioskowania nie wprost** (*refutation reasoning*). Aby uzyskać:

$$KB \vdash f$$

dodajemy negację formuły twierdzenia $\neg f$ do zbioru KB i — mając nadzieję, że jest to teraz niespójny zbiór formuł logicznych — spróbujmy wyprowadzić formułę niespełnialną (fałszywą). Zakładając, że oryginalny KB jest spełnialny, jedynym źródłem niespełnialności może być dodana formuła $(\neg f)$, a to dowodzi prawdziwości f.

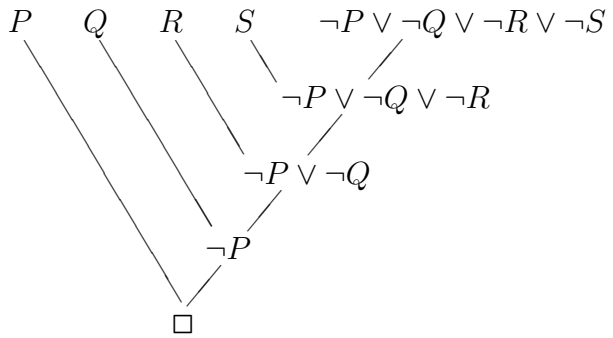
Ponieważ system wnioskowania oparty na rezolucji działa z klauzulami, wynikiem tego procesu nie wprost będzie pusta klauzula. Jeśli można ją uzyskać, dowód jest kompletny. Jeśli nie da się uzyskać klauzuli pustej, wtedy twierdzenie, które próbowaliśmy udowodnić, musi być fałszywe (w logice zdań).

Zauważmy jednak, że niepowodzenie wyprowadzenia klauzuli pustej samo w sobie nie jest dowodem fałszywości twierdzenia, podobnie jak nieumiejętność znalezienia dowodu nie oznacza, że takowy nie istnieje. Ale jeśli poszukiwanie pustej klauzuli jest prowadzone w systematyczny sposób i jest zupełne, na przykład przez zapewnienie, że wypróbowane zostały wszystkie możliwe kroki wnioskowania w rozwiązaniu, wtedy wniosek, że twierdzenie jest fałszywe, może być poprawnie sformułowany.

Wnioskowanie nie wprost z wykorzystaniem rezolucji — przykład

Dla prostej ilustracji dowodu nie wprost z wykorzystaniem rezolucji rozważmy poprzedni przypadek posiadania w bazie danych czterech faktów: P, Q, R, S i próby wyprowadzenia ich koniunkcji: $P \wedge Q \wedge R \wedge S$. Wyprowadzenie: $\{P, Q, R, S\} \vdash (P \wedge Q \wedge R \wedge S)$ nie jest możliwe ani przez modus ponens, ani przez rezolucję używaną w sposób bezpośredni.

Próbując rezolucji nie wprost, negacja twierdzenia okazuje się być pojedynczą klauzulą: $\neg P \vee \neg Q \vee \neg R \vee \neg S$. A kolejność kroków prowadzących do pustej klauzuli jest prosta:



Krótkie podsumowanie — pytania sprawdzające

Dla poniższego zestawu formuł napisz wszystkie możliwe do uzyskania rezolwenty.
Jeśli nie można wykonać żadnego kroku rezolucji, podaj krótkie wyjaśnienie.
Porównaj obliczone rezolwenty z logicznymi konsekwencjami, które możesz wyprowadzić intuicyjnie z podanych formuł.
Zwróć uwagę na przecinki, aby poprawnie zidentyfikować formuły w zbiorach.

1. $\{ P \vee Q \ , \ \neg P \vee \neg Q \}$
2. $\{ P \Rightarrow Q \ , \ Q \Rightarrow R \}$
3. $\{ \neg P \Rightarrow Q \ , \ Q \Rightarrow R \}$
4. $\{ P \vee Q \vee R \ , \ \neg P \vee Q \vee R \}$
5. $\{ P \vee Q \vee R \ , \ \neg P \vee \neg Q \vee \neg R \}$
6. $\{ P \vee Q \ , \ P \vee \neg Q \ , \ \neg P \vee Q \}$
7. $\{ P \Rightarrow (Q \vee R) \ , \ \neg Q \wedge \neg R \}$
8. $\{ P \Rightarrow Q \ , \ R \Rightarrow Q \ , \ P \vee R \}$

Rachunek predykatów pierwszego rzędu — termy

Termy reprezentują w języku logiki obiekty, i mogą być: stałymi (oznaczają konkretny obiekt), zmiennymi (mogą przybierać wartości różnych obiektów), lub funkcjami (wyznaczają jakiś obiekt na podstawie wartości swoich argumentów, inaczej, przypisują jednym obiektom inne).

Przykłady termów: A, 123, x, $f(A)$, $f(g(x))$, $+(x, 1)$

Umownie zapisujemy termy stałe wielkimi literami, a zmienne małymi.

Zauważmy, że w powyższych przykładach ostatni term jest niejawnym zapisem następnika x, a nie operacją odejmowania. W czystej logice nie ma odejmowania. Będziemy mieli do czynienia z tym problemem w wielu sytuacjach.

Rachunek predykatów pierwszego rzędu — predykaty

Predykaty reprezentują relacje na zbiorze termów. Możemy je traktować jako funkcje mające wartość prawdy lub fałszu (1 lub 0), które przypisują prawdę każdej n-ce termów spełniających relację, a fałsz każdej n-ce niespełniającej.

Zapis predykatu z zestawem termów nazywamy **formułą atomową**.

Przykłady formuł atomowych: P, $Q(A)$, $R(x, f(A))$, $>(x, 10)$

Zapis funkcyjny: $>(x, 10)$ jest odpowiednikiem relacji: $x > 10$. W arytmetyce potraktowalibyśmy taki zapis jako nierówność i moglibyśmy ją rozwiązywać. Natomiast formuły logiczne możemy jedynie **wartościować**, to znaczy wyznaczać ich wartość logiczną prawdy lub fałszu. Gdy formuła zawiera zmienną to często nie da się wyznaczyć jej wartości logicznej.

Reprezentacja faktów za pomocą formuł

Jaki sens ma język logiki predykatów?

Możemy przy jego użyciu opisać fakty, które chcemy wyrazić, np.:

$$At(Wumpus, 2, 2)$$
$$At(Agent, 1, 1)$$
$$At(Gold, 3, 2)$$

Wybór zestawu symboli, którymi zamierzamy opisać obiekty i relacje pewnego świata, nazywamy **konceptualizacją**. Na przykład, alternatywna do powyższej konceptualizacja świata wumpusa mogłaby zawierać formuły:

$$AtWumpus(loc(2, 2))$$
$$AtAgent(loc(1, 1))$$
$$AtGold(loc(3, 2))$$

Powyższe konceptualizacje są podobne, aczkolwiek mają nieco odmienne właściwości, np. w drugiej wumpus, agent, ani złoto nie wystąpią w jawnej postaci. Ogólnie od przyjętej konceptualizacji może zależeć łatwość, a nawet możliwość wyrażania różnych faktów o dziedzinie problemowej.

Reprezentacja faktów za pomocą formuł (cd.)

Przykładem problemu konceptualizacyjnego świata wumpusa jest opis istnienia i położenia dziur. Możemy nadać dziurom prawa obywatelskie i tożsamość:

$$At(Pit_4, 3, 3)$$

W ten sposób możemy łatwo opisać cały świat widziany z lotu ptaka, nadając poszczególnym dziurom dowolnie wybrane nazwy (termy stałe). Z punktu widzenia agenta poruszającego się w świecie wumpusa ta konceptualizacja jest jednak bardzo niewygodna. Trudno byłoby w ten sposób opisać świat stopniowo poznawany, gdy na początku agent nie zna nawet liczby dziur. Istnienie dziury trzeba wtedy opisać zmienną:

$$At(x, 3, 3)$$

Jednak z tego zapisu nie wynika, że x jest dziurą i wymaga to uzupełniających opisów. Wygodną alternatywą jest postrzeganie dziur jako anonimowych, i tylko zapisywanie faktów istnienia lub nieistnienia dziur w konkretnych miejscach:

$$PitAt(3, 3)$$
$$NoPitAt(1, 1)$$

Spójniki logiczne i formuły złożone

Formuły złożone języka predykatów pierwszego rzędu można konstruować za pomocą **spójników logicznych** takich jak: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Jako szczególny przypadek, formułę która jest formułą atomową lub negacją formuły atomowej nazywamy **literałem**.
Przykłady formuł złożonych (pierwsza jest pojedynczym literałem):

$$\neg At(Wumpus, 1, 1)$$
$$PitAt(2, 1) \vee PitAt(1, 2)$$
$$[At(Agent, 1, 1) \wedge PitAt(2, 1)] \Rightarrow Percept(Breeze)$$

Zauważmy, że ostatnia formuła jest zupełnie innej natury, niż dwie pierwsze. Dwie pierwsze formuły mogą stanowić fragment opisu świata otrzymanego i/lub budowanego przez agenta inteligentnego w trakcie jego pracy w świecie wumpusa. Natomiast ostatnia formuła wyraża jedno z praw świata wumpusa. Agent zna to prawo, i może posiadać taką formułę w swojej bazie wiedzy.

Fakty ogólnie słuszne w danej dziedzinie problemowej nazywamy **aksjomatami świata**. Natomiast fakty opisujące stan konkretnej instancji problemu, nazywamy **incydentalnymi**.

Kwantyfikatory

Formuły złożone można również budować za pomocą **kwantyfikatorów**: $\forall, \exists$, które **wiążą** zmienne w formułach. Ogólny schemat formuły z kwantyfikatorem:

$$\forall x P(x)$$

Zmienną niezwiązaną kwantyfikatorem w formule nazywamy **wolną**. Formuła:

$$\exists y Q(x, y)$$

zawiera dwie zmienne, jedną wolną, a drugą związaną kwantyfikatorem.

Zdaniem nazywamy formułę bez wolnych zmiennych.

Przykłady:

$$\exists x, y\ At(Gold, x, y)$$
$$\forall x, y\ [At(Wumpus, x, y) \wedge At(Agent, x, y)] \Rightarrow AgentDead$$
$$\forall x, y\ [At(Wumpus, x, y) \wedge At(Agent, -(x, 1), y)] \Rightarrow Percept(Stench)$$

Zwróćmy uwagę, że w ostatnim przykładzie $-(x, 1)$ jest niejawnym zapisem współrzędnej na lewo od x, a nie odejmowaniem. W logice nie ma odejmowania.

Krótkie podsumowanie — pytania sprawdzające

1. Opracuj kompletną konceptualizację świata wumpusa w rachunku predykatów pierwszego rzędu. To znaczy: wprowadź symbole termów (stałych i funkcji termowych), oraz symbole predykatów niezbędne do opisywania instancji problemów w tej dziedzinie.

   Uwaga: nie rozważamy procesu poszukiwania rozwiązania, analizy alternatywnych ruchów i ich skutków, opisywania sekwencji kroków, itp. Poszukujemy jedynie formatu pozwalającego na statyczny opis stanu zagadnienia, tzw. *snapshot*.

2. Wykorzystując reprezentację opracowaną w poprzednim punkcie, opisz instancję problemu przedstawioną na stronie 4.

3. Spróbuj zapisać aksjomatykę świata wumpusa, to znaczy: ogólne reguły rządzące tym światem.

Przekształcanie formuł logicznych do postaci klauzulowej

Formułę bez zmiennych wolnych możemy przekształcić do postaci **klauzulowej** (inaczej: **prenex**) gdzie wszystkie kwantyfikatory zapisane są przed formułą:

(i) przemianuj zmienne związane kwantyfikatorami na unikalne,
(ii) zastąp koniunkcjami i alternatywami pozostałe spójniki logiczne,
(iii) przesuń negacje do wewnątrz formuł (do symboli predykatów),
(iv) wyodrębnij kwantyfikatory poza formułę,
(v) przekształć formułę do postaci koniunkcyjnej (CNF),
(vi) zastąp wszystkie kwantyfikatory egzystencjalne funkcjami Skolema.

Pierwsze pięć kroków są równoważnościowymi przekształceniami logicznymi (przy zachowaniu właściwej kolejności wyodrębnianych kwantyfikatorów w kroku (iv)). Krok (vi), tzw. **skolemizacja**, polega na zastąpieniu formuł postaci:

$$\forall x_1 \forall x_2 ... \forall x_n \exists y \ \Phi(x_1, x_2, ..., x_n, y)$$

formułą

$$\forall x_1 \forall x_2 ... \forall x_n \ \Phi(x_1, x_2, ..., x_n, f_y(x_1, x_2, ..., x_n))$$

gdzie f_y jest nowo wprowadzonym symbolem funkcyjnym zwanym **funkcją Skolema**. (W przypadku braku kwantyfikatorów $\forall$ będzie to **stała Skolema**.)

Twierdzenie Skolema

Ostatni krok w algorytmie przekształcenia formuły do postaci klauzulowej nie jest równoważnościowym przekształceniem logicznym. To znaczy, dla oryginalnej formuły logicznej Φ, i otrzymanej w wyniku przekształcenia jej postaci klauzulowej Φ', w ogólności $\Phi \not\equiv \Phi'$.

Zachodzi jednak następująca własność, zwana **twierdzeniem Skolema**: dla zamkniętej formuły Φ, jeśli Φ' jest jej postacią klauzulową, to Φ jest spełnialna wtedy i tylko wtedy gdy Φ' jest spełnialna.

Zatem, o ile nie możemy w ogólności posługiwać się postacią klauzulową Φ' zamiast oryginalnej formuły Φ, to jednak możemy posługiwać się tą postacią dla celów dowodzenia spełnialności (lub niespełnialności).

Istnieje niezwykle przydatny schemat wnioskowania, wykorzystujący formuły w postaci klauzulowej, zapisywane często w postaci zbioru (lub listy) klauzul, gdzie poszczególne klauzule są zapisane w postaci zbiorów (lub list) literałów.

Krótkie podsumowanie — pytania sprawdzające

Przekształć do postaci prenex następujące formuły rachunku predykatów:

1. $\forall x\ [(P(x) \Rightarrow Q(x)) \wedge (P(x) \Rightarrow R(x))]$

2. $\forall x\ [(P(x) \wedge Q(x)) \vee (R(x) \wedge S(x))]$

3. $\forall x \exists y\ [P(x) \Rightarrow Q(x, y)]$

4. $\exists x \forall y\ [P(x, y) \Rightarrow Q(A, x)]$

5. $\forall x \exists y\ [P(x, y) \Rightarrow Q(y, f(y))]$

Podstawienia zmiennych w formułach

Będziemy rozważali przekształcenia formuł polegające na zastępowaniu zmiennych w formułach innymi wyrażeniami (termami). Ponieważ zmienne w formułach w postaci prenex domyślnie związane są kwantyfikatorami uniwersalnymi, zastępowanie zmiennych innymi termami oznacza branie szczególnych przypadków formuły.

Podstawieniem (*substitution*) nazwiemy zbiór odwzorowań określających termy podstawiane pod poszczególne zmienne. Podstawiane termy nie mogą zawierać zastępowanej zmiennej. Przykład podstawienia: $s = \{x \mapsto A, y \mapsto f(z)\}$.

Wykonanie podstawienia polega na syntaktycznym zastąpieniu wszystkich wystąpień danej zmiennej w formule związanym z nią termem. Wszystkie zastąpienia wykonywane są jednocześnie, czyli wynikiem wykonania podstawienia $s = \{x \mapsto y, y \mapsto A\}$ na termie $f(x, y)$ będzie term $f(y, A)$.

Zauważ, że w ten sposób nie ma znaczenia w jakiej kolejności zmienne są zastępowane, pomimo iż podstawienie jest zbiorem (nieuporządkowanym).

Złożeniem podstawień s_1 i s_2 (zapisywanym: $s_1 s_2$) nazwiemy podstawienie uzyskane przez zastosowanie podstawienia s_2 do termów z s_1, oraz dopisanie do otrzymanego zbioru wszystkich par z s_2 ze zmiennymi nie występującymi w s_1.

$$\begin{aligned} \Phi s_1 s_2 &= (\Phi s_1) s_2 \\ s_1(s_2 s_3) &= (s_1 s_2) s_3 \end{aligned}$$

Unifikacja

Unifikacją nazywamy takie podstawienie termów pod zmienne w zbiorze formuł, aby sprowadzić wszystkie formuły w zbiorze do identycznych (lub równoważnych logicznie, patrz wyjaśnienie niżej) formuł, czyli zbioru singletonowego.

Unifikator zbioru formuł to jest podstawienie redukujące zbiór do jednoelementowego. Zbiór jest **unifikowalny** jeśli istnieje jego unifikator.

Na przykład: zbiór $\{P(x), P(A)\}$ jest unifikowalny, i jego unifikatorem jest $s = \{x \mapsto A\}$.

Podobnie, zbiór $\{P(x), P(y), P(A)\}$ jest unifikowalny, a jego unifikatorem jest $s = \{x \mapsto A, y \mapsto A\}$.

Zbiór $\{P(A), P(B)\}$ nie jest unifikowalny, podobnie jak zbiór $\{P(A), Q(x)\}$.

Unifikacja (cd.)

Unifikacja jest ogólną procedurą, ale tutaj będziemy wykonywać unifikacje wyłącznie na zbiorach klauzul. Rozważmy poniższe, przykładowe zbiory klauzul:

$$\begin{aligned}\Phi &= \{P \vee Q(x), P \vee Q(A), P \vee Q(y)\} \\ \Psi &= \{P \vee Q(x), P \vee Q(A), P \vee Q(f(y))\} \\ \Omega &= \{P \vee Q(x), P \vee Q(A) \vee Q(y)\}\end{aligned}$$

Zbiór Φ jest unifikowalny, jego unifikatorem jest $s = \{x \mapsto A, y \mapsto A\}$, a zunifikowanym zbiorem jest singletonowy zbiór $\Phi s = \{P \vee Q(A)\}$.

Zbiór Ψ nie jest unifikowalny.

Zbiór Ω jest bardziej skomplikowanym przypadkiem. Stosując czysto **syntaktyczną unifikację**, nie jest on unifikowalny, bo po wykonaniu samego podstawienia formuły nie są identyczne. Jednak stosując **semantyczną unifikację**, jest on unifikowalny, ponieważ po wykonaniu podstawienia formuły są logicznie równoważne. Będziemy dopuszczali unifikację semantyczną z zastosowaniem łączności i przemienności alternatywy.

Najogólniejszy unifikator (mgu)

Najogólniejszym unifikatorem unifikowalnego zbioru formuł, albo **mgu** (*most general unifier*), nazywamy najprostszy (minimalny) unifikator dla tego zbioru.

Dla unifikowalnego zbioru formuł zawsze istnieje jego mgu, a dowolny unifikator dla tego zbioru można otrzymać przez złożenie mgu z jakimś innym podstawieniem.
Algorytm unifikacji pozwala wyznaczyć mgu zbioru formuł.

Krótkie podsumowanie — pytania sprawdzające

Dla poniższych zbiorów klauzul odpowiedz, czy dany zbiór jest unifikowalny. Jeśli tak, to podaj jego unifikator. Spróbuj podać zarówno mgu, jak i inny unifikator, który nie jest mgu. Jeśli zbiór nie jest unifikowalny, to krótko uzasadnij dlaczego.
Zwróć uwagę na przecinki, aby prawidłowo odczytać formuły w zbiorach.

1. $\{P(x)\ , P(f(x))\}$

2. $\{P(x, y)\ , P(y, x)\}$

3. $\{P(x, y)\ , P(y, f(x))\}$

4. $\{P(x, y)\ , P(y, f(y))\}$

5. $\{P(x, y)\ , P(y, z)\ , P(z, A)\}$

Rezolucja — przypadek ogólny

Rezolucja w ogólnym przypadku: gdy dla dwóch klauzul (zbiorów literałów) $\{L_i\}$ i $\{M_i\}$ istnieją ich podzbiory $\{l_i\}$ i $\{m_i\}$, zwane **literałami kolidującymi**, takie, że zbiór $\{l_i\} \cup \{\neg m_i\}$ daje się zunifikować i s jest jego mgu, wtedy ich rezolwentą jest zbiór $[\{L_i\} - \{l_i\}]s \cup [\{M_i\} - \{m_i\}]s$.

Mogą istnieć różne rezolwenty danych klauzul dla różnych wyborów podzbiorów ich literałów. Na przykład, rozważmy następujące klauzule:

$$P[x, f(A)] \vee P[x, f(y)] \vee Q(y) \quad \text{oraz} \quad \neg P[z, f(A)] \vee \neg Q(z)$$

Wybierając $\{l_i\} = \{P[x, f(A)]\}$ oraz $\{m_i\} = \{\neg P[z, f(A)]\}$ otrzymujemy rezolwentę:

$$P[z, f(y)] \vee \neg Q(z) \vee Q(y)$$

Natomiast wybierając $\{l_i\} = \{P[x, f(A)], P[x, f(y)]\}$ oraz $\{m_i\} = \{\neg P[z, f(A)]\}$ otrzymujemy:

$$Q(A) \vee \neg Q(z)$$

Krótkie podsumowanie — pytania sprawdzające

Dla poniższych zbiorów klauzul, zapisz wszystkie możliwe do otrzymania rezolwenty. Dla każdej rezolwenty zanotuj, z których klauzul została otrzymana, i jakie było zastosowane podstawienie. Jeśli nie jest możliwe wykonanie rezolucji, to wyjaśnij dlaczego nie.
Zwróć uwagę na przecinki, aby prawidłowo odczytać formuły w zbiorach.

1. $\{\neg P(x) \vee Q(x)\ ,\ P(A)\}$

2. $\{\neg P(x) \vee Q(x)\ ,\ \neg Q(x)\}$

3. $\{\neg P(x) \vee Q(x)\ ,\ P(f(x))\ ,\ \neg Q(x)\}$

Rezolucja jako reguła wnioskowania

Rezolucja jest poprawną regułą wnioskowania ponieważ klauzula otrzymana z pary klauzul w wyniku zastosowania rezolucji, wynika z nich logicznie. Jednak nie jest kompletna, to znaczy nie możemy z jej pomocą wygenerować z danej formuły Δ każdego wniosku φ, takiego że $\Delta \vdash \varphi$.

Na przykład, dla $\Delta = \{P, Q\}$ nie możemy rezolucją wywieść formuł $P \vee Q$ ani $P \wedge Q$, a dla $\Delta = \{\forall x R(x)\}$ nie możemy wywieść formuły $\exists x R(x)$.

Jednak jeśli zastosujemy rezolucję w procedurze dowodzenia **nie wprost**, czyli przez zaprzeczenie tezy i wyprowadzenie sprzeczności, reprezentowanej przez klauzulę pustą (oznaczaną $\square$), to możemy udowodnić nią każde twierdzenie. Zatem w tym sensie rezolucja jest kompletna (*refutation complete*).

Rozważmy powyższe przykłady. Dla $\Delta = \{P, Q\}$ zaprzeczeniem formuły $P \vee Q$ są klauzule $\neg P$ i $\neg Q$ i każda z nich pozwala natychmiast wygenerować klauzulę pustą z odpowiednią klauzulą z Δ. Zaprzeczeniem formuły $P \wedge Q$ jest klauzula $\neg P \vee \neg Q$ i możemy uzyskać klauzulę pustą w dwóch krokach rezolucji. Dla $\Delta = \{\forall x R(x)\}$ zaprzeczeniem formuły $\exists x R(x)$ jest $\neg R(y)$, która unifikuje się z klauzulą $R(x)$ otrzymaną z Δ i daje klauzulę pustą w jednym kroku rezolucji.

Dowodzenie twierdzeń oparte na rezolucji

Podstawowy schemat wnioskowania opartego na rezolucji, gdy posiadamy zbiór aksjomatów Δ i chcemy z niego wywieść formułę φ, jest następujący. Łączymy zbiory klauzul otrzymanych z Δ i $\neg\varphi$, i próbujemy wywieść sprzeczność (klauzulę pustą) z otrzymanego łącznego zbioru klauzul, przez zastosowanie rezolucji na kolejnych parach wybranych klauzul. W tym procesie uzyskana w bieżącym kroku rezolucji nowa klauzula zostaje każdorazowo dołączona do głównego zbioru klauzul, i procedura jest powtarzana.

Podstawowy wynik logiki matematycznej tu wykorzystywany jest następujący. Jeśli uruchomimy rezolucję na zbiorze klauzul otrzymanym z niespełnialnej formuły, z jakimś systematycznym algorytmem generowania rezolwent, to w pewnym momencie otrzymamy klauzulę pustą. I na odwrót, jeśli ze zbioru klauzul otrzymanego z jakiejś formuły można procedurą rezolucji wygenerować klauzulę pustą, to ten zbiór klauzul, ale także oryginalna formuła, są niespełnialne. Obejmuje to również klauzule otrzymane w wyniku skolemizacji, a więc jest zarazem potwierdzeniem poprawności tej procedury.

Dowodzenie twierdzeń: przykład

Wiadomo, że:

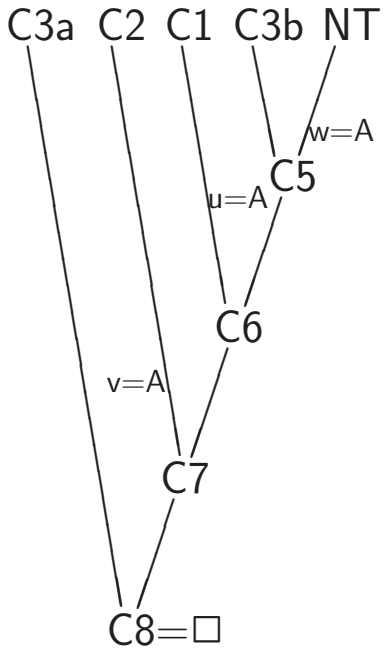
1. Kto potrafi czytać ten jest oświecony. $(\forall x)[R(x) \Rightarrow L(x)]$
2. Delfiny nie są oświecone. $(\forall x)[D(x) \Rightarrow \neg L(x)]$
3. Niektóre delfiny są inteligentne. $(\exists x)[D(x) \wedge I(x)]$

Należy udowodnić twierdzenie:

4. Są tacy inteligentni co nie potrafią czytać. $(\exists x)[I(x) \wedge \neg R(x)]$

Po konwersji do postaci prenex CNF otrzymujemy klauzule:

C1:	$\neg R(u) \vee L(u)$	z pierwszego aksjomatu
C2:	$\neg D(v) \vee \neg L(v)$	z drugiego aksjomatu
C3a:	$D(A)$	z trzeciego aksjomatu, cz.1
C3b:	$I(A)$	z trzeciego aksjomatu, cz.2
NT:	$\neg I(w) \vee R(w)$	z negacji twierdzenia



Z kolejnych kroków rezolucji otrzymujemy:

C5:	$R(A)$	rezolwenta klauzul C3b i NT
C6:	$L(A)$	rezolwenta klauzul C5 i C1
C7:	$\neg D(A)$	rezolwenta klauzul C6 i C2
C8:	$\square$	rezolwenta klauzul C7 i C3a

Dowodzenie twierdzeń: poważniejszy przykład

Rozważmy przykład z matematyki.[2] Chcemy udowodnić, że przekrój dwóch zbiorów zawiera się w dowolnym z nich. Zaczynamy od wypisania aksjomatów, z których rozumowanie będzie musiało korzystać. W tym przypadku są to definicje pojęć przekroju i zawierania się zbiorów.

$$\forall x \forall s \forall t \ \ (x \in s \ \wedge \ x \in t) \Leftrightarrow x \in s \cap t$$
$$\forall s \forall t \ \ (\forall x \ x \in s \Rightarrow x \in t) \Leftrightarrow s \subseteq t$$

Formuła do udowodnienia:

$$\forall s \forall t \ \ s \cap t \subseteq s$$

[2] Przykład zapożyczony z książki Geneseretha i Nilssona „Logical Foundations of Artificial Intelligence".

Po przetworzeniu do postaci klauzul otrzymujemy:

1.	$\{x \notin s, x \notin t, x \in s \cap t\}$	z definicji przekroju
2.	$\{x \notin s \cap t, x \in s\}$	z definicji przekroju
3.	$\{x \notin s \cap t, x \in t\}$	z definicji przekroju
4.	$\{F(s,t) \in s, s \subseteq t\}$	z definicji zawierania się
5.	$\{F(s,t) \notin t, s \subseteq t\}$	z definicji zawierania się
6.	$\{A \cap B \not\subseteq A\}$	z zaprzeczenia tezy

Zauważmy funkcje Skolema w klauzulach 4 i 5, oraz stałe Skolema w klauzuli 6. Dalej następuje wywód prowadzący dosyć prosto do klauzuli pustej.

7.	$\{F(A \cap B, A) \in A \cap B\}$	z klauzul 4. i 6.
8.	$\{F(A \cap B, A) \notin A\}$	z klauzul 5. i 6.
9.	$\{F(A \cap B, A) \in A\}$	z klauzul 2. i 7.
10.	$\{\}$	z klauzul 8. i 9.

To koniec dowodu. Cel osiągnięty. Trochę trudno poczuć satysfakcję jaka zwykle towarzyszy osiągnięciu tradycyjnego dowodu matematycznego. Również aby prześledzić rozumowanie i np. je sprawdzić, trzeba się trochę napracować, aczkolwiek w przypadku tego dowodu jest to jeszcze względnie proste.

Krótkie podsumowanie — pytania sprawdzające

Dla podanych poniżej zbiorów aksjomatów Δ i formuły φ, spróbuj udowodnić $\Delta \vdash \varphi$ metodą rezolucji nie wprost. Zacznij od zaprzeczenia formuły celowej, następnie z otrzymanego zaprzeczenia i zbioru aksjomatów utwórz podstawowy zbiór klauzul.

1. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Wino}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \mathsf{Lubi}(\mathsf{Zdzich}, \mathsf{Wino})\}$
   $\varphi = \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{Zdzich})$

2. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Rychu}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \neg\mathsf{Lubi}(\mathsf{zona}(\mathsf{Zdzich}), \mathsf{Rychu})\}$
   $\varphi = \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{zona}(\mathsf{Zdzich}))$

3. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Wino}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \mathsf{Lubi}(\mathsf{Zdzich}, \mathsf{Wino})\}$
   $\varphi = (\mathsf{Lubi}(\mathsf{Rychu}, \mathsf{Zdzich}) \vee \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{zona}(\mathsf{Zdzich}))$

4. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Wino}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \mathsf{Lubi}(\mathsf{Zdzich}, \mathsf{Wino})\}$
   $\varphi = (\mathsf{Lubi}(\mathsf{Rychu}, \mathsf{Zdzich}) \wedge \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{zona}(\mathsf{Zdzich}))$

Inżynieria wiedzy

Przedstawiony formalizm logiki predykatów pierwszego rzędu, wraz z rezolucją jako metodą dowodzenia twierdzeń nie wprost, pozwala na budowę inteligentnych agentów efektywnie rozwiązujących stawiane im problemy. Budowa takiego agenta wymaga konstrukcji reprezentacji, którą można sformułować w postaci następującego procesu, zwanego **inżynierią wiedzy**:

identyfikacja problemu
: określenie zakresu pytań, na które agent będzie musiał znajdować odpowiedzi, rodzaju faktów, którymi będzie mógł się posługiwać, itp.; na przykład, w odniesieniu do świata wumpusa, musimy określić, czy agent ma umieć planować działania, czy np. tylko tworzyć reprezentację stanu świata rozpoznanego w dotychczasowych działaniach?

pozyskanie wiedzy
: twórca oprogramowania agenta (inżynier wiedzy) może nie rozumieć wszystkich niuansów opisywanego świata, i musi współpracować z ekspertami aby pozyskać całą niezbędną wiedzę

definicja słownika reprezentacji

pojęcia i obiekty z dziedziny problemowej muszą zostać opisane formułami logicznymi; konieczne jest zdefiniowanie słownika predykatów i termów, tzn. funkcji termowych oraz stałych; ten etap może się okazać kluczowy dla zdolności do efektywnego rozwiązywania problemów, np. w świecie wumpusa, czy zapadliny lepiej przedstawić jako obiekty, czy własności miejsc

kodowanie wiedzy ogólnej

kodowanie aksjomatów zawierających ogólną wiedzę o dziedzinie problemowej, regułach rządzących światem, istniejących heurystykach, itp.

kodowanie wiedzy szczególnej

zapis konkretnego problemu do rozwiązania przez agenta, w tym zadanych mu faktów o konkretnych obiektach, oraz postawionego zadania jako pytania do odpowiedzenia lub, ogólniej, twierdzenia do udowodnienia

przedstawienie zapytań do urządzenia wnioskującego

uruchomienie procedury dowodzenia na skonstruowanej bazie wiedzy + faktach

debugowanie bazy wiedzy

niestety, podobnie jak w przypadku zwykłych programów, rzadko kiedy skonstruowany system będzie od razu poprawnie działał; mogą zdarzyć się takie problemy, jak brak kluczowych aksjomatów, albo aksjomaty nieprecyzyjnie sformułowane, które pozwalają na udowodnienie zbyt mocnych twierdzeń

Algorytmy pomocnicze: relacja równości

Jedną ze szczególnych relacji występujących w formułach logicznych jest relacja równości (identyczności) termów.

Przykład:
$\Delta = \{=(\text{żona}(\text{Zdzich}), \text{Gabrysia}), \text{Posiada}(\text{żona}(\text{Zdzich}), \text{alfa-8c})\}$.
Czy to znaczy, że Gabrysia posiada Alfę 8c Competizione?
Czy możemy to udowodnić metodą rezolucji?
$\text{Posiada}(\text{Gabrysia}, \text{alfa-8c})$?

Niestety, nie. Procedura dowodowa rezolucji nie traktuje predykatu równości w żaden szczególny sposób i nie wykorzysta posiadanej informacji o identyczności termów. Aby dowód w powyższym przykładzie był możliwy musielibyśmy sformułować dodatkowy aksjomat równości:

$$\forall x, y, z\ [\text{Posiada}(x, y) \wedge =(x, z) \Rightarrow \text{Posiada}(z, y)]$$

Za pomocą sformułowanego powyżej aksjomatu równości można udowodnić, że Gabrysia posiada Alfę, jak również podobne fakty o posiadaniu dla innych posiadaczy określanych jawnie lub niejawnie. Jednak aby móc podobne wnioskowanie rozciągnąć na równoważność przedmiotu posiadania, niezbędny jest jeszcze inny aksjomat:

$$\forall x, y, z\ [\mathsf{Posiada}(x, y) \wedge =(y, z) \Rightarrow \mathsf{Posiada}(x, z)]$$

Co gorsza, aby system mógł sprawnie posługiwać się znanymi faktami tożsamości termów w odniesieniu do wszystkich relacji, analogiczne aksjomaty równości należałoby napisać dla wszystkich symboli predykatów. Niestety, w języku logiki pierwszego rzędu nie można tego wyrazić jedną wspólną formułą typu:

$$\forall P, y, z[P(y) \wedge =(y, z) \Rightarrow P(z)]$$

Alternatywnym rozwiązaniem jest wbudowanie obsługi równości termów w procedurę dowodzenia. Istnieje kilka rozwiązań tego problemu: reguła redukcji formuł ze względu na relację równości zwana **demodulacją**, uogólniona reguła rezolucji uwzględniająca relację równości zwana **paramodulacją**, oraz wbudowanie relacji równości w procedurę unifikacji.

Algorytmy pomocnicze: odzyskiwanie odpowiedzi z drzewa rezolucji

Rozważmy następujący przykład, wiemy że:

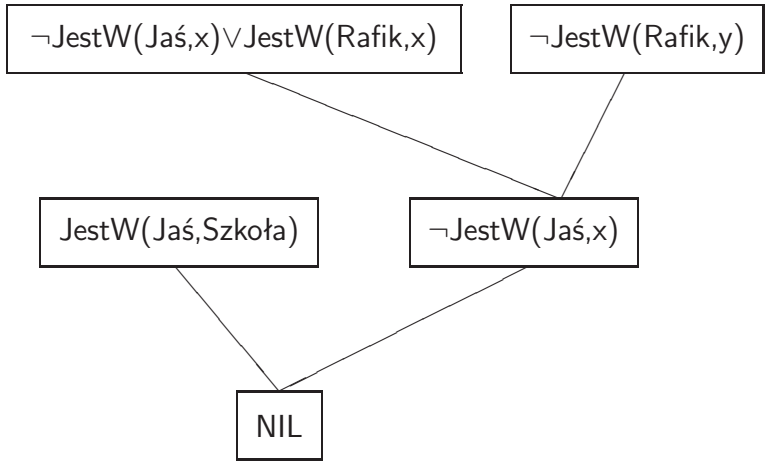
Gdzie jest Jaś, tam jest Rafik. $(\forall x)[\text{JestW}(\text{Jaś}, x) \Rightarrow \text{JestW}(\text{Rafik}, x)]$

Jaś jest w szkole. $\text{JestW}(\text{Jaś}, \text{Szkoła})$

Chcemy znaleźć odpowiedź na pytanie:

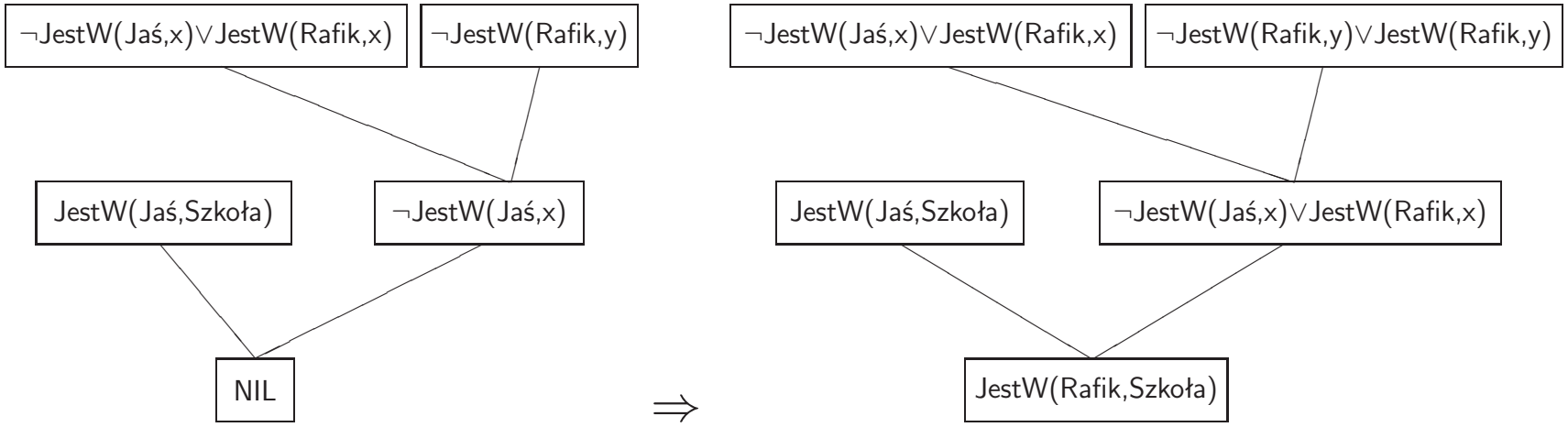
Gdzie jest Rafik? $(\exists x)[\text{JestW}(\text{Rafik}, x)]$

Formuła logiczna odpowiadająca oryginalnemu pytaniu różni się nieco od niego, ale dowód znajduje się łatwo:



Niestety, nie daje on odpowiedzi na pierwotnie postawione pytanie.

Odzyskiwanie odpowiedzi z drzewa rezolucji (c.d.)



- Podstawowa procedura zamienia dowód nie wprost na kompletny dowód tezy wprost.
- Jeśli twierdzenie zawiera alternatywy (po zaprzeczeniu stają się koniunkcjami) to w wyniku tej procedury otrzymujemy złożoną formułę, która może być trudna do interpretacji.
- Jeśli twierdzenie zawiera kwantyfikator ogólny to po zaprzeczeniu pojawiają się w niej stałe lub funkcje Skolema, które lądują w odpowiedzi, ale mogą być zamienione na zmienne kwantyfikowane uniwersalnie.

Algorytmy pomocnicze: strategie przyspieszające rezolucję

W dowodzeniu twierdzeń z wykorzystaniem rezolucji w procedurze dowodowej nie wprost, dążymy do wygenerowania klauzuli pustej, wskazującej na sprzeczność. Aby mieć pewność, że taką klauzulę wygenerujemy, zakładając, że jest to w ogóle możliwe, musimy generować rezolwenty w jakiś systematyczny sposób, na przykład realizując przeszukiwanie wszerz. Jednak przy większych bazach danych, ten sposób może prowadzić do generowania bardzo wielu wniosków, z których większość może nie mieć nic wspólnego z dowodzonym twierdzeniem.

Poszukuje się zatem **strategii przyspieszających**, które pozwoliłyby odciąć i nie generować niektórych rezolwent. Strategie takie mogą być **pełne**, tzn. dające gwarancję znalezienia rozwiązania (fałszu) jeśli to tylko możliwe, albo **niepełne**, czyli nie dające takiej gwarancji (ale typowo znacznie skuteczniejsze).

Strategie przyspieszające:

- preferencja pojedynczych literałów (normalnie niepełna, ale pełna jeśli jest traktowana tylko jako preferencja)
- strategia zbioru podtrzymania (*set of support*): tylko rezolucje z klauzulami z pewnego zbioru, początkowo uzyskanego z zaprzeczonej tezy (pełna)
- rezolucja wejściowa (*input resolution*) zezwala tylko na rezolucje z użyciem klauzuli wejściowej (pełna tylko w niektórych przypadkach, np. dla hornowskich baz danych)
- rezolucja liniowa (niepełna)
- eliminacja powtórzeń i specjalizacji (pełna)

Nierozstrzygalność rachunku predykatów

Rachunek predykatów wydaje się językiem reprezentacji dobrze nadającym się do wyrażania faktów i wnioskowania w systemach sztucznej inteligencji. Należy jednak zdawać sobie sprawę z pewnych jego ograniczeń, które zawężają jego użyteczność praktyczną.

Twierdzenie Churcha (1936, o nierozstrzygalności rachunku predykatów): nie istnieje procedura pozwalająca w ogólnym przypadku sprawdzać prawdziwości formuł rachunku predykatów. Mówimy, że rachunek predykatów jest **nierozstrzygalny** (*undecidable*).

Ta własność w istotny sposób ogranicza możliwości wnioskowania o faktach wyrażanych w języku predykatów. Co prawda istnieje szereg klas formuł, dla których procedura decyzyjna istnieje. Poza tym rachunek predykatów ma własność **półrozstrzygalności** (*semidecidability*), co oznacza, że istnieje procedura pozwalająca stwierdzić niespełnialność dowolnej formuły niespełnialnej w skończonej liczbie kroków. Niestety, dla formuł, które nie są niespełnialne, ta procedura może nigdy się nie zatrzymać.

Niezupełność w rachunku predykatów

Ktoś mógłby myśleć, że nierozstrzygalność rachunku predykatów można pokonać, korzystając z półrozstrzygalności. Chcąc udowodnić formułę φ ze zbioru aksjomatów Δ, uruchamiamy równolegle dwa dowody: $\Delta \vdash \varphi$ i $\Delta \vdash \neg\varphi$. Na mocy półrozstrzygalności, przynajmniej jeden z tych dowodów powinien zakończyć się powodzeniem. Niestety, to również może się nie udać.

Twierdzenie Gödla (1931, o niezupełności): w rachunku predykatów można sformułować teorie **niezupełne**, czyli takie, w których istnieją formuły zamknięte, których nie można wywieść, ani nie można wywieść ich zaprzeczenia. Co więcej, takie teorie są dość proste, na przykład taką teorią jest teoria liczb naturalnych, generowana opisującym je zbiorem aksjomatów.

Teorię $\mathcal{T}$ nazywamy **rozstrzygalną** jeśli istnieje algorytm pozwalający dla dowolnej formuły zamkniętej φ stwierdzić, czy $\varphi \in \mathcal{T}$, czy $\varphi \notin \mathcal{T}$. Teorie niezupełne są więc w oczywisty sposób nierozstrzygalne.

Twierdzenie Gödla ma taki skutek, że jeśli po pewnej liczbie kroków dowodu $\Delta \vdash \varphi$ (i, być może, równoległego dowodu $\Delta \vdash \neg\varphi$), nadal nie ma rozwiązania, to nie możemy mieć pewności, czy dowód jednak się zakończy (przynajmniej jeden z nich), czy mamy do czynienia z teorią niezupełną.

Zmiany zachodzące w czasie

Rachunek predykatów dobrze spisuje się jako język reprezentacji dla dziedzin statycznych, gdzie nic się nie dzieje, i wszystko co jest prawdziwe, pozostaje takie na zawsze. Świat realny niestety taki nie jest.

Na przykład, jeśli formuła: JestW(Jaś, Szkoła) poprawnie opisuje stan bieżący jakiegoś powszedniego dnia przed południem, to niestety, musimy się liczyć z tym, że Jaś pójdzie w pewnym momencie do domu. Jeśli aksjomatyka dobrze opisuje skutki działań agentów, to system mógłby nawet wywnioskować nowy fakt: JestW(Jaś, Dom). Niestety, wtedy w bazie danych powstanie sprzeczność, która dla systemu logicznego jest katastrofą. System dowodzenia zawierający fałsz w swoich aksjomatach może udowodnić dowolne twierdzenie!

Ćwiczenie: załóżmy, że w zbiorze aksjomatów Δ istnieją, między innymi, dwie klauzule: P i $\neg P$. Przedstaw dowód dowolnej formuły Q. Wskazówka: najpierw udowodnij, że $P \vee \neg P \vee Q$ jest tautologią (zdaniem zawsze prawdziwym) dla dowolnych P i Q. Można to sprawdzić wykonując dowód $\models (P \vee \neg P \vee Q)$, czyli dowodząc tezy z pustym zbiorem aksjomatów. Następnie dodaj tak udowodnioną tautologię do bazy Δ i przeprowadź dowód Q.

Logiki czasowe

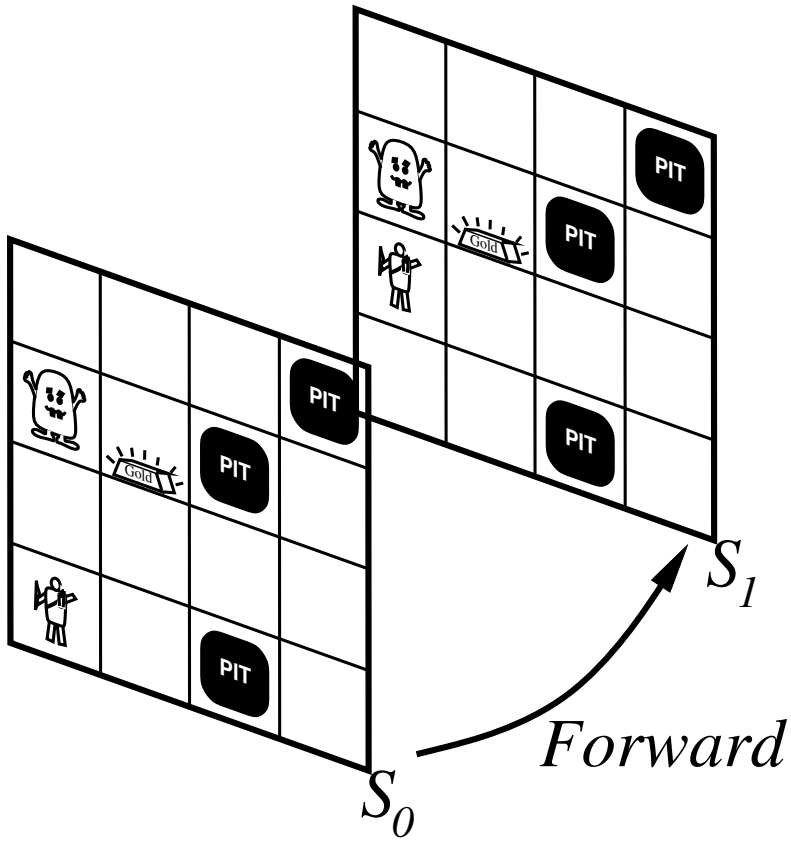
Dla rozwiązania problemu reprezentacji zmian stworzono wiele specjalistycznych teorii logicznych, zwanych **logikami czasowymi** (*temporal logics*). Zwykłe fakty wyrażane w tych logikach zachodzą w określonych momentach czasowych. Natomiast czas, jego własności, i specjalne reguły wnioskowania dotyczące jego upływu, są w logikach czasowych wbudowane w teorię (zamiast być reprezentowane jawnie, na równi z innymi własnościami świata).

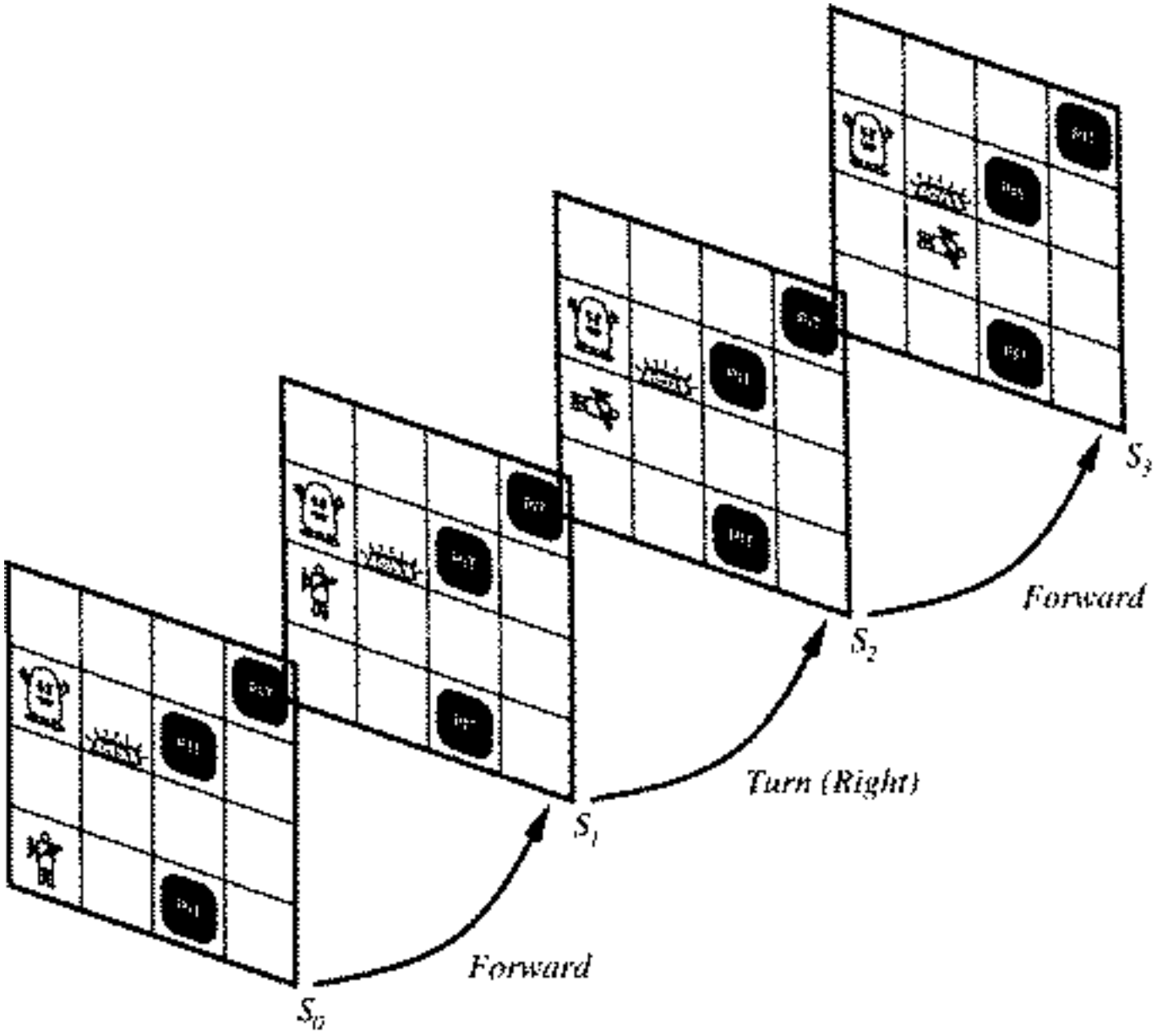
Jedną z głównych kwestii, różniących te teorie, jest sama reprezentacja czasu. Może on być dyskretny lub ciągły, może być określany w postaci punktów lub przedziałów, może być ograniczony lub nieograniczony, itp. Co więcej, czas może być pojęciem liniowym, lub rozgałęzionym. Zwykle powinien jednak być uporządkowany, choć istnieją kołowe reprezentacje czasu.

Dla każdej z takich logik czasowych, aby dało się efektywnie wnioskować o tworzonych formułach, reprezentujących zjawiska, z którymi ma do czynienia inteligentny agent, musi istnieć procedura dowodowa. Konstrukcja takiej procedury może opierać się na rzutowaniu danej teorii do logiki predykatów pierwszego rzędu.

Rachunek sytuacji

Alternatywą do logik czasowych jest bezpośredni zapis momentów czasowych w języku reprezentacji. Przykładem takiego podejścia jest **rachunek sytuacji**:
$At(Agent, [1,1], S_0) \wedge At(Agent, [1,2], S_1) \wedge S_1 = Result(Forward, S_0)$




S_0
S_1
S_2
S_3
Forward
Turn (Right)
Forward

Rachunek sytuacji (cd.)

Rachunek sytuacji wykorzystuje pojęcia: **sytuacji**, **akcji**, i **fluentów**:

sytuacje: sytuacją jest stan początkowy s_0, i dla każdej sytuacji s i akcji a sytuacją jest również $Result(a, s)$; sytuacje odpowiadają sekwencjom akcji i w ten sposób są różne od stanów, tzn. agent może znaleźć się w danym stanie poprzez różne sytuacje,

fluenty: funkcje i relacje, które podlegają zmianom w czasie nazywane są fluentami i posiadają argument sytuacji, typowo jest to ostatni argument,

aksjomaty dopuszczalności akcji: opisują warunki stosowalności akcji, np. dla akcji $Shoot$: $Have(Agent, Arrow, s) \Rightarrow Poss(Shoot, s)$

aksjomaty następstwa akcji: opisują następstwa akcji dla wszystkich fluentów, np. dla akcji $Grab$ aksjomat powinien stwierdzać, że po prawidłowym wykonaniu tej akcji agent będzie trzymał to co podniósł; należy jednak pamiętać również o sytuacjach, kiedy fluent nie uległ zmianie w wyniku wykonania jakiejś akcji:

$$Poss(a, s) \Rightarrow$$
$$(Holding(Agent, g, Result(a, s)) \Leftrightarrow$$
$$a = Grab(g) \vee (Holding(Agent, g, s) \wedge a \neq Release(g))).$$

aksjomaty unikalności akcji: ze względu na obecność klauzul różności akcji w aksjomatach następstwa, musimy zapewnić mechanizm pozwalający agentowi stwierdzać takie fakty, na przykład przez aksjomaty unikalności akcji; dla każdej pary symboli akcji A_i i A_j musimy zapisać aksjomat (na pozór oczywisty) $A_i \neq A_j$; ponadto dla akcji z parametrami musimy zapisać też:

$$A_i(x_1, ..., x_n) = A_j(y_1, ..., y_n) \Leftrightarrow x_1 = y_1 \wedge ... \wedge x_n = y_n$$

Przykład z małpą i bananami — aksjomatyzacja

• wiedza ogólna o świecie i operatorach (częściowa i uproszczona):

A1: $\forall p \forall p_1 \forall s\ [\mathsf{At}(\textsc{box}, p, s) \Rightarrow \mathsf{At}(\textsc{box}, p, \mathsf{goto}(p_1, s))]$

A2: $\forall p \forall p_1 \forall s\ [\mathsf{At}(\textsc{bananas}, p, s) \Rightarrow \mathsf{At}(\textsc{bananas}, p, \mathsf{goto}(p_1, s))]$

A3: $\forall p \forall s\ [\mathsf{At}(\textsc{monkey}, p, \mathsf{goto}(p, s))]$

A4: $\forall p \forall p_1 \forall s\ [\mathsf{At}(\textsc{box}, p, s) \wedge \mathsf{At}(\textsc{monkey}, p, s) \Rightarrow \mathsf{At}(\textsc{box}, p_1, \mathsf{move}(\textsc{box}, p, p_1, s))]$

A5: $\forall p \forall p_1 \forall p_2 \forall s\ [\mathsf{At}(\textsc{bananas}, p, s) \Rightarrow \mathsf{At}(\textsc{bananas}, p, \mathsf{move}(\textsc{box}, p_1, p_2, s))]$

A6: $\forall p \forall p_1 \forall s\ [\mathsf{At}(\textsc{monkey}, p, s) \Rightarrow \mathsf{At}(\textsc{monkey}, p_1, \mathsf{move}(\textsc{box}, p, p_1, s))]$

A7: $\forall s\ [\mathsf{Under}(\textsc{box}, \textsc{bananas}, s) \Rightarrow \mathsf{Under}(\textsc{box}, \textsc{bananas}, \mathsf{climb}(\textsc{box}, s))]$

A8: $\forall p \forall s\ [\mathsf{At}(\textsc{box}, p, s) \wedge \mathsf{At}(\textsc{monkey}, p, s) \Rightarrow \mathsf{On}(\textsc{monkey}, \textsc{box}, \mathsf{climb}(\textsc{box}, s))]$

A9: $\forall s\ [\mathsf{Under}(\textsc{box}, \textsc{bananas}, s) \wedge \mathsf{On}(\textsc{monkey}, \textsc{box}, s) \Rightarrow \mathsf{Havebananas}(\mathsf{grab}(\textsc{bananas}, s))]$

A10: $\forall p \forall s\ [\mathsf{At}(\textsc{box}, p, s) \wedge \mathsf{At}(\textsc{bananas}, p, s) \Rightarrow \mathsf{Under}(\textsc{box}, \textsc{bananas}, s)]$

• dane zadania:

A11: $[\mathsf{At}(\textsc{monkey}, P_1, S_0) \wedge \mathsf{At}(\textsc{box}, P_2, S_0) \wedge \mathsf{At}(\textsc{bananas}, P_3, S_0)]$

• teza do udowodnienia:

$\exists s(\mathsf{Havebananas}(s))$

[2] Przedstawione tutaj rozwiązanie problemu małpy i bananów wzorowane jest na przykładzie z książki Philipa C. Jacksona Jr.'a: „Artificial Intelligence".

Przykład z małpą i bananami — zbiór klauzul

A1: $\{\neg\mathsf{At}(\textsc{box}, p, s_1), \mathsf{At}(\textsc{box}, p, \mathsf{goto}(p_1, s_1))\}$
A2: $\{\neg\mathsf{At}(\textsc{bananas}, q, s_2), \mathsf{At}(\textsc{bananas}, q, \mathsf{goto}(q_1, s_2))\}$
A3: $\{\mathsf{At}(\textsc{monkey}, r, \mathsf{goto}(r, s_3))\}$
A4: $\{\neg\mathsf{At}(\textsc{box}, u, s_4), \neg\mathsf{At}(\textsc{monkey}, u, s_4), \mathsf{At}(\textsc{box}, u_1, \mathsf{move}(\textsc{box}, u, u_1, s_4))\}$
A5: $\{\neg\mathsf{At}(\textsc{bananas}, t, s_5), \mathsf{At}(\textsc{bananas}, t, \mathsf{move}(\textsc{box}, t_2, t_3, s_5))\}$
A6: $\{\neg\mathsf{At}(\textsc{monkey}, v_1, s_6), \mathsf{At}(\textsc{monkey}, v_2, \mathsf{move}(\textsc{box}, v_1, v_2, s_6))\}$
A7: $\{\neg\mathsf{Under}(\textsc{box}, \textsc{bananas}, s_7), \mathsf{Under}(\textsc{box}, \textsc{bananas}, \mathsf{climb}(\textsc{box}, s_7))\}$
A8: $\{\neg\mathsf{At}(\textsc{monkey}, w, s_8), \neg\mathsf{At}(\textsc{box}, w, s_8), \mathsf{On}(\textsc{monkey}, \textsc{box}, \mathsf{climb}(\textsc{box}, s_8))\}$
A9: $\{\neg\mathsf{Under}(\textsc{box}, \textsc{bananas}, s_9), \neg\mathsf{On}(\textsc{monkey}, \textsc{bananas}, s_9),$
$\mathsf{Havebananas}(\mathsf{grab}(\textsc{bananas}, s_9))\}$
A10: $\{\neg\mathsf{At}(\textsc{box}, p, s_{10}), \neg\mathsf{At}(\textsc{bananas}, p, s_{10}), \mathsf{Under}(\textsc{box}, \textsc{bananas}, s_{10})\}$
A11a: $\{\mathsf{At}(\textsc{monkey}, P_1, S_0)\}$
A11b: $\{\mathsf{At}(\textsc{box}, P_2, S_0)\}$
A11c: $\{\mathsf{At}(\textsc{bananas}, P_3, S_0)\}$
NT: $\{\neg\mathsf{Havebananas}(z)\}$

Przykład z małpą i bananami — dowód

C1(A1,A11b) $\{\mathsf{At}(\textsc{box}, P_2, \mathsf{goto}(p_1, S_0))\}$

C2(C1,A4) $\{\neg\mathsf{At}(\textsc{bananas}, P_2, \mathsf{goto}(p_1, S_0)),$
$\mathsf{At}(\textsc{box}, u_1, \mathsf{move}(\textsc{box}, P_2, u_1, \mathsf{goto}(p_1, S_0)))\}$

C3(C2,A3) $\{\mathsf{At}(\textsc{box}, u_1, \mathsf{move}(\textsc{box}, P_2, u_1, \mathsf{goto}(P_2, S_0)))\}$

C4(C3,A10) $\{\neg\mathsf{At}(\textsc{bananas}, u_1, \mathsf{move}(\textsc{box}, P_2, u_1, \mathsf{goto}(P_2, S_0))),$
$\mathsf{Under}(\textsc{box}, \textsc{bananas}, \mathsf{move}(\textsc{box}, P_2, u_1, \mathsf{goto}(P_2, S_0)))\}$

C5(A2,A11c) $\{\mathsf{At}(\textsc{bananas}, P_3, \mathsf{goto}(q_1, S_0))\}$

C6(C5,A5) $\{\mathsf{At}(\textsc{bananas}, P_3, \mathsf{move}(\textsc{box}, t_2, t_3, \mathsf{goto}(q_1, S_0)))\}$

C7(C6,C4) $\{\mathsf{Under}(\textsc{box}, \textsc{bananas}, \mathsf{move}(\textsc{box}, P_2, P_3, \mathsf{goto}(P_2, S_0)))\}$

C8(C7,A7) $\{\mathsf{Under}(\textsc{box}, \textsc{bananas}, \mathsf{climb}(\textsc{box}, \mathsf{move}(\textsc{box}, P_2, P_3, \mathsf{goto}(P_2, S_0))))\}$

C9(A3,A6) $\{\mathsf{At}(\textsc{monkey}, v_2, \mathsf{move}(\textsc{box}, r, v_2, \mathsf{goto}(r, r_1)))\}$

C10(C9,A8) $\{\mathsf{At}(\textsc{box}, v_2, \mathsf{move}(\textsc{box}, r, v_2, \mathsf{goto}(r, r_1))),$
$\mathsf{On}(\textsc{monkey}, \textsc{box}, \mathsf{climb}(\textsc{box}, \mathsf{move}(\textsc{box}, r, r_2, \mathsf{goto}(r, r_1))))\}$

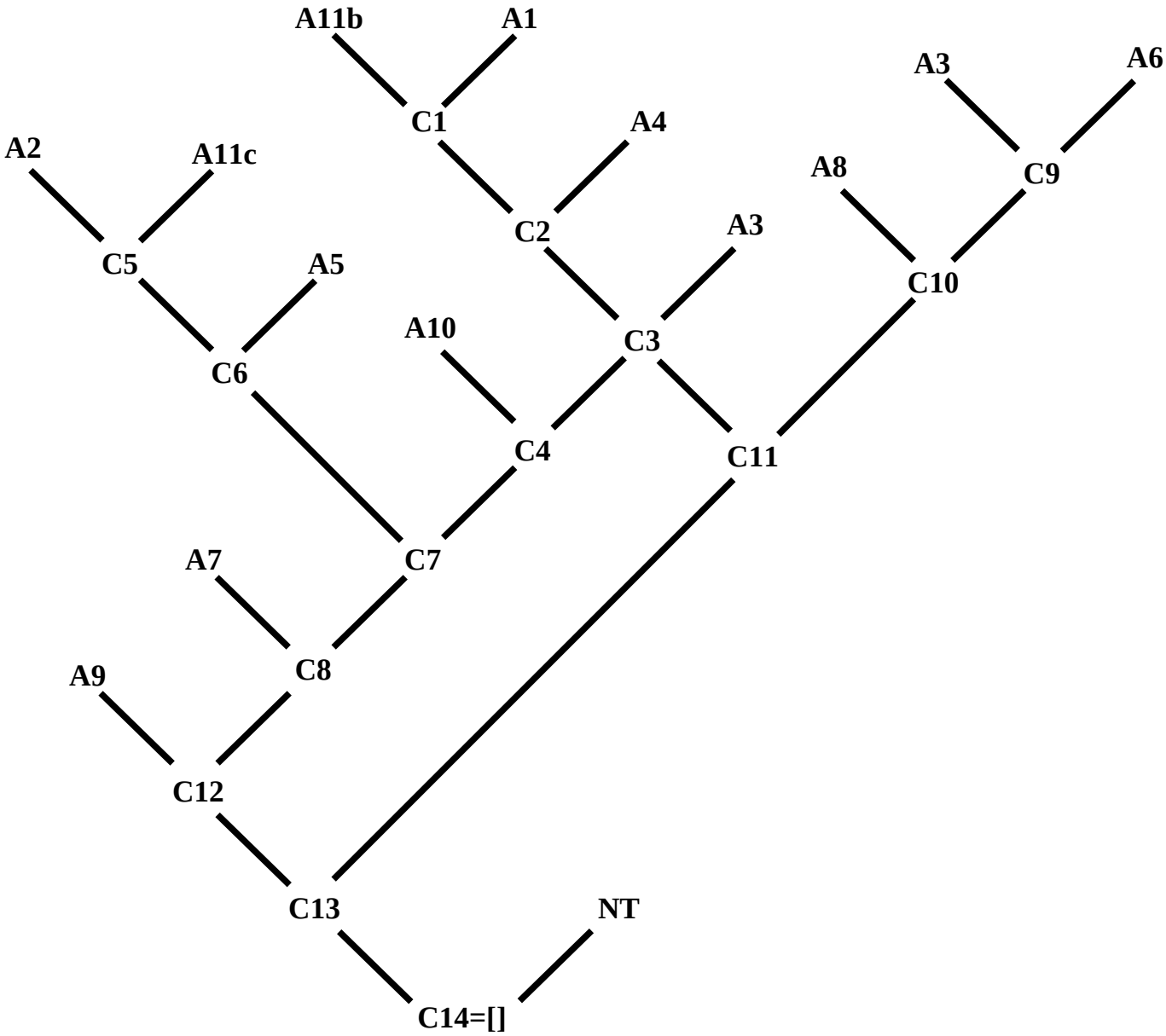
C11(C10,C3) $\{\mathsf{On}(\textsc{monkey}, \textsc{box}, \mathsf{climb}(\textsc{box}, \mathsf{move}(\textsc{box}, P_2, u_1, \mathsf{goto}(P_2, S_0))))\}$

C12(C8,A9) $\{\neg\mathsf{On}(\textsc{monkey}, \textsc{box}, \mathsf{climb}(\textsc{box}, \mathsf{move}(\textsc{box}, P_2, P_3, \mathsf{goto}(P_2, S_0)))),$
$\mathsf{Havebananas}(\mathsf{grab}(\textsc{bananas},$
$\mathsf{climb}(\textsc{box}, \mathsf{move}(\textsc{box}, P_2, P_3, \mathsf{goto}(P_2, S_0)))))\}$

C13(C11,C12) $\{\mathsf{Havebananas}(\mathsf{grab}(\textsc{bananas},$
$\mathsf{climb}(\textsc{box}, \mathsf{move}(\textsc{box}, P_2, P_3, \mathsf{goto}(P_2, S_0)))))\}$

C14(C13,NT) $\{\}$

Przykład z małpą i bananami — drzewo dowodu



Frame problem

Poprawna reprezentacja zagadnienia wymaga jawnego zapisania efektów działań agenta w środowisku, jak również zmian wywołanych przez inne czynniki (np. deszcz). Jednak, jak widzieliśmy w przykładach z wumpusem oraz małpą i bananami, konieczne jest również sformułowanie aksjomatów niezmienniczości, pozwalające na wnioskowanie o braku zmiany:

$$\forall a, x, s\ Holding(x, s) \wedge (a \neq Release) \Rightarrow Holding(x, Result(a, s))$$
$$\forall a, x, s\ \neg Holding(x, s) \wedge (a \neq Grab) \Rightarrow \neg Holding(x, Result(a, s))$$

Niestety, w świecie bardziej złożonym niż świat wumpusa, fluentów będzie bardzo wiele, i aksjomatyka musi opisywać ich zmiany oraz niezmienność, zarówno jako bezpośrednie, jak i uboczne, efekty wykonywanych akcji.

Te aksjomaty, zwane **aksjomatami tła** (*frame axioms*) trudno jest wyrazić w sposób ogólny, i w znacznym stopniu komplikują pierwotny opis świata.

Ponieważ w czasie pracy agent musi odpowiadać sobie na wiele pytań, prowadząc dowody logiczne, mnogość aksjomatów powoduje gwałtowne powiększanie się jego bazy danych, i w efekcie może doprowadzić do kompletnego paraliżu.

Krótkie podsumowanie — pytania sprawdzające

1. Opracuj opartą na rachunku sytuacji reprezentację dla świata wumpusa, przedstawionego na początku tego wykładu.

Problemy z brakiem informacji

Przedstawione dotychczas metody oparte na logice zakładały, że wszystkie informacje niezbędne do przeprowadzenia wywodów logicznych są agentowi dostępne, i są pewne. Niestety, nie jest to realistyczne założenie.

Jednym z problemów jest **problem niepełnej informacji**. Agent może mieć częściową, ale nie pełną informację o problemie, co często uniemożliwia wyciąganie potrzebnych wniosków logicznych. Innym problemem jest **niepewność informacji**. Agent może mieć dane pochodzące z różnych nie w pełni wiarygodnych źródeł, np.:

- fakty „typowe”,
- fakty „możliwe”,
- fakty „prawdopodobne”,
- wyjątki od faktów ogólnie słusznych.

Posiadanie takich informacji często jest kluczowe dla podejmowania właściwych decyzji. Ludzie tak postępują, czynią założenia, domniemania, i często potrafią skutecznie je wykorzystywać. Chcielibyśmy, by agent sztucznej inteligencji podobnie potrafił działać w braku pewnej informacji, wykorzystać informacje niepełne i niepewne, prowadzić wywody opierające się na możliwie najlepszych źródłach danych, a także oszacowując wiarygodność otrzymanych wniosków. Niestety, klasyczna logika predykatów nie dostarcza takich narzędzi, i nie potrafi robić żadnego użytku z tego rodzaju wiedzy.

Logika zdrowego rozsądku

Zastanówmy się, jakie informacje człowiek wie na pewno, podejmując decyzje w codziennym życiu. Wstając rano, ma zamiar pojechać do pracy. Ale jeśli jest jakaś awaria komunikacji miejskiej, to powinien wstać dużo wcześniej, i najpierw sprawdzić, czy kursują autobusy. Dzień wcześniej, zakupił produkty, aby przyrządzić z nich śniadanie. Ale czy wie na pewno, że jego sałatka śniadaniowa jest nadal w lodówce, czy nie zepsuła się, czy ktoś jej nie wykradł, itp.

Wniosek: logicznie funkcjonujący agent potrzebuje pewnych informacji do swego działania, i prędzej lub później zostanie sparaliżowany stuprocentową poprawnością swego mechanizmu wnioskowania. W świecie rzeczywistym nigdy nie będzie w stanie odważyć się na jakiekolwiek działanie, dopóki nie będzie miał pełnej informacji o otaczającym go świecie.

Jednak ludzie potrafią sprawnie poruszać się w świecie pełnym informacji niepewnej i niepełnej, faktów domyślnych i wyjątków. Jak to robią? Musimy uznać, że w swoim wnioskowaniu ludzie posługują się nieco inną logiką, niż rygorystyczna logika matematyczna. Możnaby ogólnie nazwać ten hipotetyczny mechanizm wnioskowania **logiką zdrowego rozsądku** (*common sense reasoning*).

Logiki niemonotoniczne

Część winy za problemy z wnioskowaniem przy użyciu logiki klasycznej ponosi jej własność określana jako **monotoniczność**. W logice klasycznej, im więcej wiemy, tym więcej możemy wywieść stosując wnioskowanie.

Człowiek stosuje inny model wnioskowania, o wiele bardziej elastyczny, wykorzystujący informację typową, domyślną, możliwą, a nawet brak informacji. Ten rodzaj wnioskowania wydaje się nie mieć własności monotoniczności.

Na przykład, o ile w braku dobrej informacji o sytuacji człowiek byłby gotów wyciągnąć pewne wnioski i podejmować decyzje (pochopne), to po zdobyciu pełniejszej informacji może już nie być w stanie wymyślić dobrego rozwiązania problemu.[3]

Stąd różne modele wnioskowania, zmierzające do pokonania tych problemów, i bardziej zbliżone do elastycznego modelu wnioskowania człowieka, określa się wspólnym mianem **logik niemonotonicznych**.

[3] Rozwiązanie, które wypracował wcześniej, w braku informacji, było błędne, ale może było lepsze niż brak jakiegokolwiek działania. Chociaż niekoniecznie.

Logiki niemonotoniczne — przykład

Wyzwanie Minsky-ego: opracowanie systemu, który pozwoliłby prawidłowo opisać ogólnie znany fakt, że ptaki potrafią fruwać.

$$\forall x[\textsc{bird}(x) \rightarrow \textsc{canfly}(x)]$$

Aby uwzględnić wyjątki, np. strusie, trzeba każdorazowo modyfikować poprzednią formułę.

$$\forall x[\textsc{bird}(x) \wedge \neg\textsc{ostrich}(x) \rightarrow \textsc{canfly}(x)]$$

Wyjątków jest więcej: ptaki skąpane w rozlanej ropie naftowej, ptaki bez skrzydeł, chore ptaki, martwe ptaki, namalowane ptaki, abstrakcyjne ptaki, ...

Pomysł: wprowadzamy operator modalny **M**:

$$\forall x[\textsc{bird}(x) \wedge \mathbf{M}\ \textsc{canfly}(x) \rightarrow \textsc{canfly}(x)]$$

Teraz wyjątki możemy wprowadzać modularnie:

$$\forall x[\textsc{ostrich}(x) \rightarrow \neg\textsc{canfly}(x)]$$

Dla następującego zbioru faktów:

$$\Delta = \{\text{BIRD}(\textit{Tweety}), \text{BIRD}(\textit{Sam}), \text{OSTRICH}(\textit{Sam})\}$$

możemy wywieść: $\neg\text{CANFLY}(\textit{Sam})$
zatem nie powinno nam się udać wyprowadzić:

$$\mathbf{M}\ \text{CANFLY}(\textit{Sam}) \qquad \text{ani} \qquad \text{CANFLY}(\textit{Sam})$$

Jednak przy użyciu normalnych procedur nie możemy udowodnić zdolności do latania *Tweety*:

$$\mathbf{M}\ \text{CANFLY}(\textit{Tweety}), \text{CANFLY}(\textit{Tweety})$$

W tym celu niezbędna jest procedura dowodowa zdolna do efektywnego (i automatycznego) dowodzenia twierdzeń w języku predykatów rozszerzonym o operator modalny **M**, zgodna z następującą regułą wnioskowania:

$$\frac{\text{Not}(\vdash \neg p)}{\mathbf{M}\ p}$$

Logiki niemonotoniczne — jaka procedura dowodowa?

Pomijając ograniczenia wynikające z odwołania do procedury dowodowej w powyższej definicji, taka procedura nie będzie jednak ani efektywna obliczeniowo, ani rozstrzygalna, ani nawet półrozstrzygalna, jak procedury dowodowe dla rachunku predykatów.

W przesłance powyższej reguły wnioskowania mamy bowiem stwierdzenie, że pewnej formuły nie da się udowodnić. To po pierwsze może być w ogóle niemożliwe do stwierdzenia. Zaś aby znaleźć pozytywne potwierdzenie tego faktu będzie na pewno konieczne przeprowadzenie globalnego wnioskowania na całej bazie danych, bo inaczej trudno byłoby stwierdzić, że czegoś nie da się udowodnić.

Dla odróżnienia, dowody w rachunku predykatów pierwszego rzędu mają charakter lokalny. Jeśli np. szczęśliwie wybierzemy od razu właściwe przesłanki to możemy uzyskać dowód w kilku krokach, nawet jeśli baza danych liczy tysiące faktów.

Systemy zachowania spójności logicznej (TMS)

Co ma zrobić system wnioskowania logicznego gdy chciałby wycofać jakiś fakt P posiadany w swojej bazie danych? Na przykład:

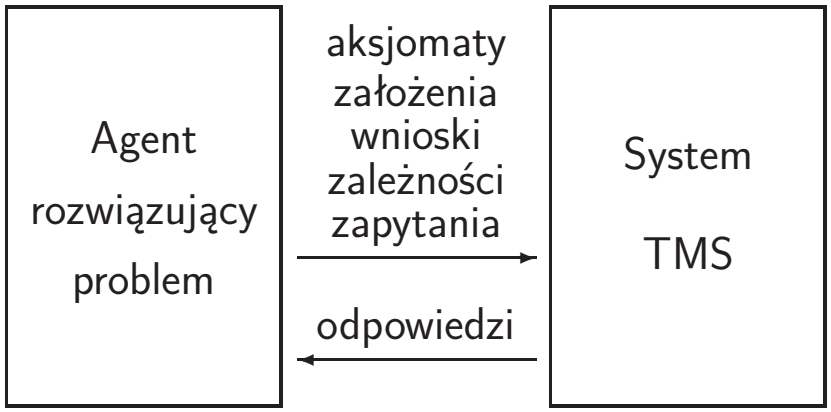
- system rejestruje bieżący stan rzeczy (nie uwzględniając upływu czasu ani historii) i ten stan uległ zmianie
- fakt był wynikiem założenia dokonanego *ad hoc* (być może w wyniku rozumowania niemonotonicznego) i obecnie są powody przypuszczać, że w istocie jest nieprawdziwy

Usunięcie błędnie dokonanego założenia lub nieaktualnego faktu nie może polegać na dodaniu faktu $\neg P$, ponieważ to nie wycofałoby błędnego faktu P, a tylko wprowadziło niespójność bazy danych, zawierającej obecnie zarówno P jak i $\neg P$. Zamiast tego, należy usunąć z bazy danych fakt P, oraz wszystkie inne fakty być może z niego wywiedzione.

> Na przykład, jeśli istniała implikacja $P \rightarrow Q$, to system mógł, wierząc w pewnym okresie w prawdziwość P, poprawnie wywnioskować Q.

Jednak fakty takie jak Q niekoniecznie muszą być nieprawdziwe tylko dlatego, że nieprawdziwe okazało się P. Po jego pierwotnym wywiedzeniu z P system mógł znaleźć inne, niezależne, potwierdzenia Q.

Systemy zachowania spójności logicznej (*truth maintenance systems*), wspomagają proces wnioskowania przez rejestrację takich zależności logicznych.



Funkcje systemu TMS:

1. przechowywanie faktów i wniosków
2. usuwanie faktów z obsługą konsekwencji
3. ponowne przywracanie faktów i konsekwencji
4. dostarczanie agentowi uzasadnień faktów
5. wykrywanie niespójności w założeniach
6. prowadzenie wnioskowania niemonotonicznego

W najprostszym przypadku system TMS może realizować swe funkcje przez usuwanie z bazy danych — w momencie wycofywania dokonanego wcześniej założenia — wszystkich wniosków otrzymanych przez system. Jest to metoda prosta i skuteczna, wymaga jednak każdorazowego powtórzenia dowodów wszystkich twierdzeń.

Nieco lepszą metodą jest usuwanie tylko wniosków wywiedzionych po wprowadzeniu do bazy danych wycofywanego aktualnie założenia. Wymaga to utrzymywania informacji o chronologii wprowadzania i wyprowadzania faktów, i nadal powtórzenia wielu dowodów, z których większość mogła nie mieć nic wspólnego z wycofywanym założeniem.

Jeszcze lepszą metodą jest rejestrowanie, dla wszystkich wniosków, założeń, na których były oparte ich wywody, a następnie, przy wycofywaniu jakiegoś założenia, usuwanie uzasadnień, których niezbędną częścią było wycofywane założenie, a także eliminacja faktów, które straciły wszystkie swoje uzasadnienia.

Przykład

Agent posiada następujące fakty pewne, oraz, uzyskane w wyniku swojej pracy, założenia niemonotoniczne i wnioski na nich oparte:

$P \rightarrow R$	{ fakt }
$Q \rightarrow S$	{ fakt }
$R \rightarrow S$	{ fakt }
P	{ założenie }
Q	{ założenie }
R	{ wniosek(P) }
S	{ wniosek(P), wniosek(Q) }

Gdyby następnie, w trakcie pracy, agent postanowił wycofać się z założenia P, i poinformował o tym system TMS, to ten skreśliłby fakt R z listy faktów uznawanych za słuszne, oraz skreśliłby jedno z uzasadnień faktu S.

Gdyby w dalszym ciągu agent wycofałby również założenie Q, to system TMS musiałby już ostatecznie usunąć fakt S.

Zauważmy, że gdyby agent następnie postanowił jednak przywrócić założenie P, to system TMS nie miałby sposobu odzyskania usuniętych wniosków R i S.

System JTMS

Historycznie najwcześniejszy system TMS Doyle'a jest właśnie oparty na uzasadnieniach (pierwotnie nazwany po prostu TMS, ale później dla odróżnienia od innych systemów nazywany również JTMS). System ten pamięta dla każdego faktu jego uzasadnienie, lub uzasadnienia.

System JMS nie usuwa raz utworzonych struktur danych, tylko określa status faktu jako „in" (fakt, w który wierzymy, bo ma uzasadnienia) albo „out" (fakt, w który nie wierzymy, bo nie ma uzasadnień).

Gdy status jakiegoś faktu zmienia się na „out", wtedy uzasadnienia niektórych innych faktów mogą również zmienić status, powodując propagację konsekwencji takiej zmiany w strukturach systemu JTMS.

Podobnie, gdy pojawia się jakiś fakt, który miał już poprzednio status „in", i jest częścią uzasadnień innych faktów, wtedy wystarczy zmienić status wszystkich takich uzasadnień i faktów na „in", i udowodnione wcześniej wnioski automatycznie pojawiają się znowu (tzw. „unouting").

Zatem system JTMS utrzymuje stan swojej bazy danych w postaci jednego spójnego zestawu „wierzeń", i o każdym stwierdzeniu możemy powiedzieć, czy aktualnie w nie wierzymy czy nie.

System ATMS

Innego rodzaju system TMS, oparty na założeniach i nazwany ATMS, zaproponował Johan de Kleer. ATMS nie etykietuje żadnych faktów jako „in" albo „out", a tylko rejestruje wszystkie fakty i założenia w postaci węzłów na jednym grafie zależności. System zaznacza, na podstawie informacji od użytkownika, które fakty są prawdziwe przy których założeniach, i nie ma pojęcia w co wierzy w danej chwili.

Użytkownik zgłasza systemowi wszystkie założenia, które go interesują, niezależnie czy są one wzajemnie spójne. ATMS aktualizuje graf zależności, gdzie etykiety węzłów zawierają założenia je uzasadniające, i w każdej chwili jest w stanie odpowiedzieć, przy jakim zestawie założeń dany fakt jest prawdziwy.

Metoda de Kleer'a pokonuje większość problemów związanych z systemami TMS (np. unouting), i nadaje się szczególnie do implementacji systemów rozważających alternatywne warianty, gdzie konieczne jest częste przełączanie się między wzajemnie wykluczającymi się punktami widzenia.

Jest to jednak metodą typowo dyskretną, działającą na niezbyt dużej liczbie założeń, ponieważ musi uwzględniać wszystkie możliwe ich kombinacje.

Generacja wyjaśnień

Systemy TMS mogą być również postrzegane jako generatory wyjaśnień. Jeśli dla danego faktu, który jest obserwowany, istnieje szereg możliwych uzasadnień, z których żadne nie jest obserwowane ani znane, to agent może analizować te uzasadnienia, i wybierać te, które np. są minimalne, a przez to najbardziej prawdopodobne, i skoncentrować się na znalezieniu przyczyny obserwowanej sytuacji.

Na przykład, jako uzasadnienie faktu, że nie da się uruchomić silnika samochodu, możemy mieć zapisaną niesprawność akumulatora, oraz jeszcze szereg innych możliwych przyczyn. Gdyby agent miał problemy z samochodem, i chciał określić jego przyczyny, mógłby zacząć analizować zbiory uzasadnienień takiej awarii, i na tej podstawie próbować wybrnąć z sytuacji. Na przykład, mógłby uporządkować te zbiory uzasadnień według liczności, i zacząć od najmniejszych, wychodząc z założenia, że najmniejszy zbiór uzasadnień związany jest z najprostszą okolicznością i przyczyną awarii. Zatem próby wybrnięcia z opresji mógłby nasz agent zacząć od sprawdzenia akumulatora, gdyby to on właśnie stanowił najprostsze w tym sensie wyjaśnienie awarii.

Problemy z metodami opartymi na logice

Podejście logiczne do reprezentacji wiedzy i rozwiązywania problemów budziło swojego czasu wiele emocji i nadziei na budowę wszechstronnych systemów sztucznej inteligencji. Istnieją jednak poważne przeszkody ograniczające zastosowanie tej metody do rozwiązywania problemów praktycznych:

- **eksplozja kombinatoryczna** procedury dowodowej — istnieją strategie usprawniające, jednak niewiele pomagają; jednocześnie trudno jest połączyć metody formalne z dostępną informacją heurystyczną
- **nierozstrzygalność** i gödlowska **niezupełność** rachunku predykatów
- wnioskowanie z uwzględnieniem **zmian** — rachunek sytuacji, logiki czasowe
  - pojawia się tu **problem tła** (*frame problem*) — poza określeniem co się zmieniło, konieczne jest śledzenie tego co się nie zmieniło
- wnioskowanie z użyciem informacji **niepełnej** i/lub **niepewnej** — inne wyzwanie dla metod formalnych, jednak nieodzowne w działaniu człowieka
  - uwzględnienie informacji niepełnej prowadzi do wnioskowania **niemonotonicznego**, którym ludzie posługują się sprawnie, podczas gdy tradycyjna logika matematyczna jest ściśle monotoniczna

Zastosowanie metod opartych na logice

Wymienione problemy z metodami opartymi na logice istotnie utrudniają ich wykorzystanie jako platformy budowy inteligentnych agentów. Powszechnie wykorzystywany w sztucznej inteligencji jest jedynie sam język logiki pierwszego rzędu jako język zapisu faktów.

Jednak w pewnych konkretnych zastosowaniach, w ograniczonych dziedzinach, powyższe problemy mają mniejsze znaczenie, i można skutecznie korzystać z tej metodologii.
Do tych zastosowań należą:

- synteza i weryfikacja programów, inżynieria oprogramowania,
- projektowanie i weryfikacja cyfrowej elektroniki obliczeniowej, w tym projektowanie układów VLSI,
- dowodzenie twierdzeń w matematyce; pozwala poszukiwać dowodów postulowanych twierdzeń, dla których nie udaje się znaleźć dowodu metodą tradycyjną.