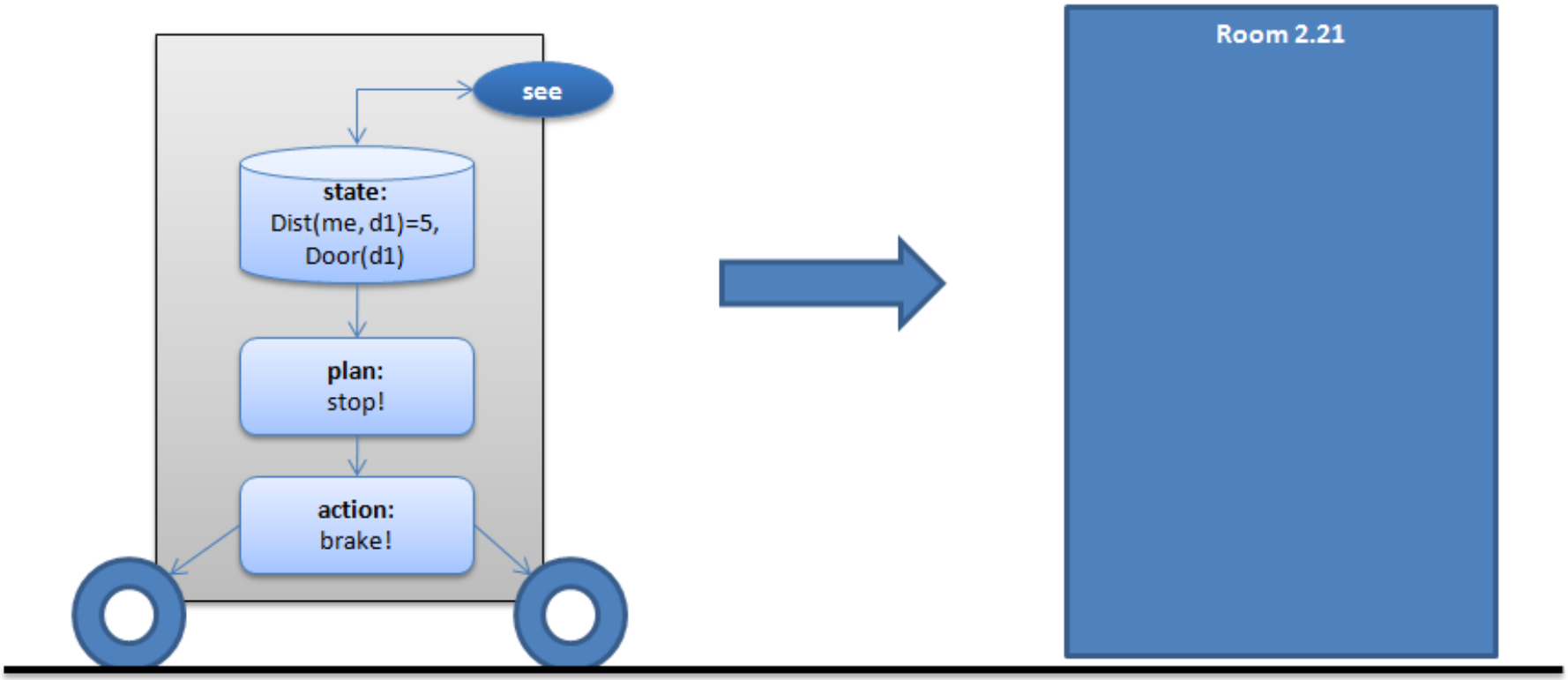
Agenty wnioskujące dedukcyjnie (*Deductive Reasoning Agents*)

Na podstawie "An Introduction to MultiAgent Systems" oraz slajdów Michaela Wooldridge'a

Agenty wnioskujące dedukcyjnie

- Tradycyjne podejście do budowy systemów inteligentnych (symboliczna AI)
- Symboliczna reprezentacja środowiska i akcji/zachowania agenta $\rightarrow$ *formuły logiczne*
- Manipulacja syntaktyczna na poziomie przyjętej reprezentacji $\rightarrow$ *dedukcja* (*dowodzenie twierdzeń*)

Kluczowe wyzwania (1)



Kluczowe wyzwania (2)

- Transdukcja
  - *Sprawne* tłumaczenie obserwacji stanu środowiska na *odpowiednią* reprezentację logiczną
  - ... rozpoznawanie mowy, obrazu, uczenie się
- Reprezentacja i wnioskowanie
  - Odpowiednia reprezentacja przetłumaczonych obserwacji i *sprawne* wnioskowanie na jej podstawie
  - ... reprezentacja wiedzy, automatyczne wnioskowanie i planowanie

Agenty i dowodzenie twierdzeń (1)

- Teoria agencji (*theory of agency*) φ – teoria opisująca zachowanie agenta (optymalizująca pewną przyjętą miary oceny przebiegów)
- Teoria φ może być zastosowana (wykonana) w celu wyboru odpowiedniej akcji

Agenty i dowodzenie twierdzeń (2)

- Wewnętrzny stan agenta zrealizowany jako baza formuł logicznych

Przykład

Open(valve221)
Temperature(reactor4726, 321)
Pressure(tank776, 28)

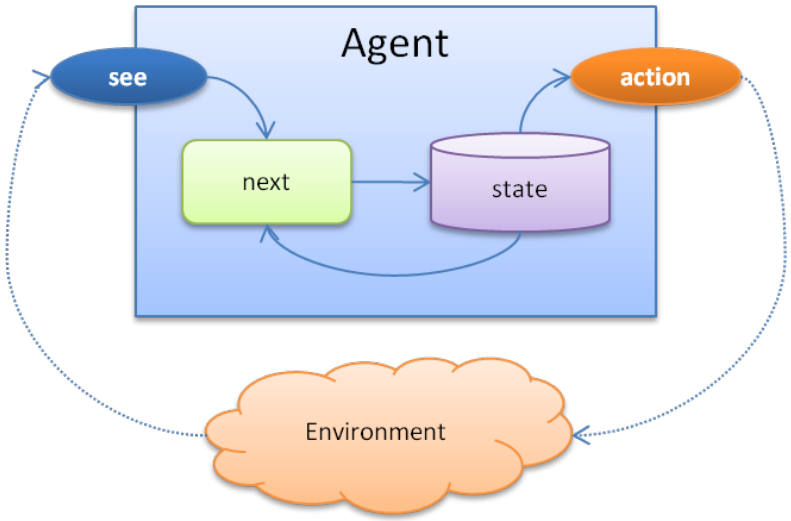
- Stan wewnętrzny odpowiada przekonaniom (*beliefs*) u ludzi – stan wewnętrzny może zawierać niewłaściwe (błędne lub „przeterminowane”) informacje

Agenty i dowodzenie twierdzeń (3)

Niech

- L będzie zbiorem formuł w logice pierwszego rzędu
- $D = \mathbf{2}^L$ będzie zbiorem baz zawierających formuły z L
  - DB, DB_1, ... są elementami zbioru D
  - DB reprezentuje wewnętrzny stan agenta
- ρ będzie zbiorem reguł dedukcyjnych modelujących proces decyzyjny agenta związany z wyborem akcji
- $DB \vdash_\rho \varphi$ oznacza, że formuła φ może być wyprowadzona (udowodniona) z bazy DB za pomocą reguł ρ

Agenty i dowodzenie twierdzeń (4)



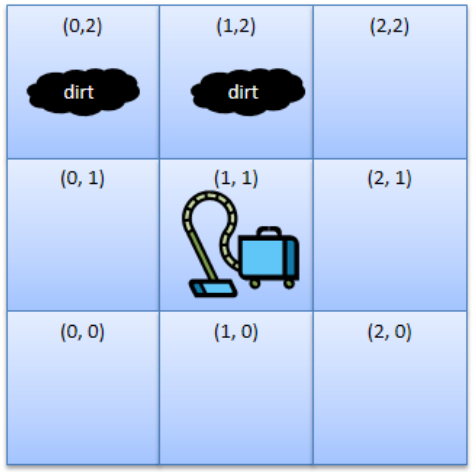
- Funkcja percepcji
  $see : E \rightarrow Per$
- Funkcja następnego stanu
  $next : D \times Per \longrightarrow D$
- Funkcja wyboru akcji
  $action : D \rightarrow Ac$

Wybór akcji jako dowodzenie twierdzenia

foreach $\alpha \in Ac$ **do**
  if $DB \vdash_\rho Do(\alpha)$ **then**
    return α;
  end
end
foreach $\alpha \in Ac$ **do**
  if $DB \nvdash_\rho \neg Do(\alpha)$ **then**
    return α;
  end
end

Przykład – Vacuum World

- Agent to robot, który czyści podłogę w pewnym pomieszczeniu podzieloną na kwadratowe pola
- Agent jest wyposażony w czujnik kurzu oraz odkurzacz
- Agent ma zawsze ustaloną orientację (N, S, E, W) oraz położenie
- Agent może przesunąć się o jedno pole do przodu lub obrócić się o 90 stopni
- Dla uproszczenia przyjmujemy, że podłoga ma rozmiar 3×3, agent znajduje się w polu (0,0), a jego orientacja to N



Reprezentacja w Vacuum World (1)

- Możliwe percepty
  $Per = \{dirt, null\}$
- Predykaty dziedzinowe opisujące stan wewnętrzny
  $In(x, y)$ – agent na polu (x, y)
  $Dirt(x, y)$ – kurz na polu (x, y)
  $Facing(d)$ – agent skierowany w kierunku d
- Możliwe akcje
  $Act = \{forward, clean, turn\}$

Reprezentacja w Vacuum World (2)

- Niech $old(DB)$ oznacza stare predykaty w bazie DB, które powinny zostać usunięte przez funkcję *next*
  $old(DB) =$
  $\{P(t_1, \ldots, t_n) | P \in \{In, Dirt, Facing\} \wedge P(t_1, \ldots, t_n) \in DB\}$
- Wprowadzamy funkcję *new* definiującą nowe predykaty, które powinny być dodane do bazy DB
  $new : D \times Per \rightarrow D$
- Funkcja *next* może zostać przedefiniowana za pomocą funkcji *old* i *new* jako
  $next(DB, p) = (DB \backslash old(DB)) \cup new(DB, p)$

Wnioskowanie w Vacuum World

Reguły dedukcji

1. $In(x,y) \wedge Dirt(x,y) \longrightarrow Do(clean)$
2. $In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \longrightarrow Do(forward)$
3. $In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \longrightarrow Do(forward)$
4. $In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \longrightarrow Do(turn)$
5. $In(0,2) \wedge Facing(east) \wedge \neg Dirt(0,2) \longrightarrow Do(forward)$
6. $\ldots$

Problemy z wnioskowaniem

- Podejmowanie decyzji poprzez dowodzenie twierdzeń jest złożone (teoretycznie brak gwarancji zakończenia procesu wnioskowania)
- Proces decyzyjny zakłada statyczne środowisko $\rightarrow$ *racjonalność obliczeniowa* (*calculative rationality*)
  - Sugerowana decyzja była optymalna w momencie rozpoczęcia procesu wnioskowania, ale możliwe są zmiany środowiska
  - Jeśli wnioskowanie przebiega szybko, można pominąć ten problem
- Trudności z uwzględnieniem (reprezentacja, wnioskowanie) aspektów temporalnych w „czystej” logice

Programowanie zorientowane agentowo

Agent-oriented programming, AOP

new programming paradigm, based on a societal view of computation

- Odpowiedź na problemy z „czystym” dowodzeniem twierdzeń (np. poprzez uproszczenie procesu wnioskowania)
- Głowna idea to programowanie agentów z wykorzystaniem pojęć intencjonalnych ($\rightarrow$ systemy intencyjne) – przekonań, zobowiązań, intencji
- Pierwsza eksperymentalna realizacja AOP to język AGENT0
- Idea AOP jest cały czas rozwijana – AgentSpeak/Jason

Komponenty AGENT0

Każdy agent w AGENT0 składa się z 4 komponentów

1. zbiór zdolności (rzeczy, które agent może zrobić)
2. zbiór początkowych przekonań
3. zbiór początkowych zobowiązań (rzeczy, które agent zrobi)
4. zbiór reguł zobowiązujących (*commitment rules*)

Reguły zobowiązujące w AGENT0

- Podstawowy element determinujący zachowanie się agenta
- Każda reguła zawiera
  - warunek narzucony na wiadomości ($\rightarrow$ warunek komunikacyjny)
  - warunek narzucony na stan wewnętrzny (przekonania) ($\rightarrow$ warunek mentalny)
  - akcję do wykonania (zobowiązanie do przyjęcia)

Cykl decyzyjny w AGENT0

1. Warunek komunikacyjny reguły jest dopasowywany do wiadomości, które agent poprzednio otrzymał
2. Warunek mentalny jest dopasowywany do stanu wewnętrznego agenta
3. Jeśli reguła zostaje dopasowana, wówczas agent zobowiązuje się wykonać wskazaną akcję (akcja zostaje dodana do zbioru zobowiązań agenta)

Komunikacja w AGENT0

- Akcje wskazane przez regułę mogą być
  - *prywatne* – obliczenia wykonywane wewnętrznie przez agenta
  - *komunikacyjne* (*communicative*) – wysyłanie wiadomości
- Wiadomości są ograniczone do trzech typów
  - *request* – zobowiązanie do wykonania akcji
  - *unrequest* – anulowanie zobowiązania
  - *inform* – przekazanie informacji

Przykładowa reguła zobowiązania

```
COMMIT(
        (agent,
                REQUEST, DO(time, action)
        ), ;;; msg condition
        (B,
                [now, Friend agent] AND
                CAN(self, action) AND
                NOT [time, CMT(self, anyaction)]
        ), ;;; mental condition
        self,
        DO(time, action)
)
```

Jeśli otrzymam wiadomość od innego agenta, który chce, abym wykonał pewną akcję we wskazanym czasie, oraz jestem przekonany, że:

- ten agent jest aktualnie przyjacielem
- potrafię wykonać tę akcję
- we wskazanym czasie nie zobowiązałem się wykonać innej akcji

wówczas zobowiązuje się do wykonania żądnej akcji.

Concurrent MetateM

- Agentowy język programowania wykorzystujący formuły w logice temporalnej (klasyczna logika uwzględniająca zależności czasowe)
- System w Concurrent MetateM składa się z wielu działających jednocześnie agentów, którzy wymieniają asynchronicznie wiadomości
- Każdy agent ma własny zestaw reguł, które sterują jego zachowaniem

Logika temporalna

- $\bigcirc\phi$ – ϕ będzie prawdziwe w kolejnym stanie
- $\Diamond\phi$ – ϕ będzie prawdziwe w przyszłości
- $\Box\phi$ – ϕ będzie prawdziwe zawsze w przyszłości
- $\odot\phi$ – ϕ było prawdziwe w poprzednim stanie
- $\phi \mathscr{U} \psi$ – ϕ będzie prawdziwe, aż zostanie spełnione ψ

Program w Concurrent MetateM

- Program w Concurrent MetateM jest zestawem reguł $past \Longrightarrow future$
- Podczas działania systemu reguły są dopasowywane do zapamiętanej przez agenta historii
- Przesłanki aktywowanych reguł stają się zobowiązaniami, które agent musi zrealizować ($\rightarrow$ analogicznie do AGENT0)

System w Concurrent MetateM

- System w Concurrent MetateM składa się z z wielu agentów, z których każdy jest opisany za pomocą 3 atrybutów
  - nazwa
  - interfejs komunikacyjny – wiadomości, które agent może odebrać i wysłać
  - program (zbiór reguł w logice temporalnej)
- Przykład: stack agent
  $stack\,(pop, push)\,[popped, stackfull]$

Cykl działania w Concurrent MetateM

1. Odbierz wiadomości od innych agentów i zaktualizuj na ich podstawie stan agenta
2. Dopasuj reguły do historii i aktualnego stanu
3. Wykonaj wybrane zobowiązania wskazane przez reguły aktywowane w obecnym cyklu oraz pochodzące z wcześniejszych cykli
4. Idź do kroku 1

Przykład Concurrent MetateM (1)

Prosty system z 3 agentami:

- *rp* – producent zasobów (*resource producer*)
  - w danym momencie może dać tylko zasób jednemu agentowi
  - zobowiązuje się przekazać zasób temu agentowi, który o to poprosi
- *rc1*, *rc2* – konsumenci zasobów (*resource consumers*) – mniej i bardziej zachłanny

Przykład Concurrent MetateM (2)

- $rp\,(ask1, ask2)\,[give1, give2]$:

  $start \longrightarrow \quad \Box\neg\,(give1 \wedge give2)$

  $\odot ask1 \longrightarrow \qquad\qquad \Diamond give1$

  $\odot ask2 \longrightarrow \qquad\qquad \Diamond give2$

- $rc1\,(give1)\,[ask1]$:

  $start \longrightarrow \quad ask1$

  $\odot ask1 \longrightarrow \quad ask1$

- $rc2\,(ask1, give2)\,[ask2]$:

  $\odot\,(ask1 \wedge \neg ask2) \longrightarrow \quad ask2$

Przykład Concurrent MetateM (3)

	Agent		
Cykl	*rp*	*rc*1	*rc*2
0		*ask*1	
1	*ask*1	*ask*1	*ask*2
2	*ask*1, *ask*2, *give*1	*ask*1	
3	*ask*1, *give*2	*ask*1, *give*1	*ask*2
4	*ask*1, *ask*2, *give*1	*ask*1	*give*2
5	...		