Warstwa deliberatywna

* Rozwiąż prosty problem deliberacji za pomocą Strips, mając na uwadze początkowy model świata, operatory, tabelę różnic i ewaluator różnic. Po każdym kroku pokaż stan modelu świata.

* Opisz organizację planowania misji, nawigatora i pilota zagnieżdżonego kontrolera hierarchicznego.

* Zdefiniuj fizyczny problem uziemienia symbolu i zakotwiczenia; daj przykład.

* Porównaj i porównaj zalety i wady lokalnych przestrzeni percepcyjnych z wielopoziomowymi modelami hierarchicznymi do tworzenia, utrzymywania i używania modeli światowych.

* Opisz rolę i działanie agentów Sequencer i Skill Manager w architekturze 3T oraz sposób, w jaki uwzględnia ona reaktywność i rozważania.

* Porównaj podejście do rozumowania w górę i w górę na podstawie modelu w wykrywaniu, identyfikacji i odzyskiwaniu błędów (FDIR) i podaj zalety i wady każdego z nich.

* Zdefiniuj umiejętności, wirtualny czujnik i świadomą awarię.

Przegląd

Części od 6 do 8 zawierają podstawowe informacje na temat robota opartego na zachowaniu i jak osiągnąć autonomię podobną do zwierząt. Jednak to, co cenimy jako „prawdziwą" inteligencję poznawczą, jest rozważane, a nie reaktywne. Chęć zrozumienia deliberacji prowadzi do pytania postawionego w tym rozdziale: Jak myślą roboty? Wracając do części 4, inteligencja deliberatywna wiąże się z generowaniem celów i zamiarów, wyborem lub planowaniem, jak najlepiej osiągnąć te cele i intencje oraz wdrażaniem konkretnych działań i zasobów w celu realizacji planu, a na koniec monitorowaniem wykonania planu i przeplanowaniem, jeśli to nie działa. Metody inteligencji deliberatywnej obejmują większość aspektów sztucznej inteligencji, zwłaszcza dyscypliny planowania i rozwiązywania problemów, wraz z ich poddziedzinami planowania, alokacji zasobów i rozumowania. Ponieważ niemożliwe jest omówienie całego zestawu postępów w zestawie dyscyplin, w niniejszym rozdziale podjęta zostanie próba wprowadzenia czytelnika w terminologię i ramy pojęciowe potrzebne do dalszej lektury. Na szczęście większość terminologii i ram koncepcyjnych wywodzi się z powstania pierwszego robota AI, Shakey w SRI w 1967. Shakey podobno otrzymał swoją nazwę, ponieważ był bardzo ciężki; gdy toczył się do przodu, antena telewizyjna chwiała się i drżała. Shakey był kwintesencją systemu hierarchicznego: SENSE, potem PLAN, potem ACT. Zbadano pomysł ogólnego rozwiązania problemów w planowaniu i rozwiązywaniu problemów, uchwycony w algorytmie Strips, który umożliwiłby Shakeyowi rozumowanie i planowanie ścieżki. Paski są rzadko używane w robotyce, ale znajomość ich jest ważna z trzech powodów: Po pierwsze, posłuży do zmotywowania czytelnika do zbadania wyzwań komputerowych związanych nawet z tak prostym zadaniem, jak przejście przez pokój. Po drugie, Strips jest interesujący z historycznego punktu widzenia, ponieważ sformalizował koncepcje warunków wstępnych, założenia zamkniętego świata, założenia otwartego świata i problemu ramowego. Po trzecie i najważniejsze, wybory projektowe programowania dotyczące reprezentacji wiedzy i algorytmów rozumowania wprowadzone w Strips nadal przenikają robotykę AI, a zatem zrozumienie Strips jest warunkiem wstępnym dla dalszych zaawansowanych badań. Shakey i Strips skoncentrowali się na stworzeniu idealnego planu planowania działań robota, ale metodologia przesłoniła krytyczną kwestię tego, jak to, co wyczuwany przez robot, można przekształcić w reprezentację świata. Ponieważ Shakey mógł „widzieć" tylko duże obiekty o jasnych kolorach, i tak nie było ich wiele do przedstawienia. Jednak w przypadku nowoczesnych robotów czujniki i detekcje są bardziej zaawansowane, zapewniając w ten sposób bogatszy zestaw danych wejściowych, a oczekiwania inteligencji wymuszają tworzenie bardziej wyrafinowanych modeli świata. W tej części

omówimy podstawowe pojęcia. Strips również w dużej mierze zignorował sposób realizacji działań; znowu, pragmatycznie, był to podstawowy robot, więc tak naprawdę nie było żadnych opcji. Zagnieżdżony kontroler hierarchiczny stał się kamieniem węgielnym planowania nawigacyjnego w latach 90. XX wieku. Później, w połowie lat 90., robotycy AI zaczęli rozwiązywać problemy z ogólnym powiązaniem planów i harmonogramów z ich realizacją. RAPS i architektura 3T były pierwszymi próbami połączenia planowania i zachowań. 3T jest doskonałym przykładem systemu, który może generować plany wysokiego poziomu i propagować te plany aż do sterowania określonymi siłownikami, co zostało podsumowane w tej części. Shakey i Strips założyli, że robot wykona cały cykl SENSE, PLAN, ACT przy każdej aktualizacji, ale w przeciwieństwie do paradygmatu Reactive, etap SENSE obejmował aktualizację kompleksowego modelu globalnego. Wyczuwanie zmiany na świecie z działania wytworzyło de facto monitorowanie realizacji planu. Jeśli robot miał przemieścić się do drzwi, a nie dotarł do nich, nowy plan powinien zapewniać najbardziej aktualną, optymalną ścieżkę lub rozwiązanie. Oczywiście, jeśli robot miał awarię i nie miał wewnętrznych czujników do zauważenia, nie mógł się zregenerować. Późniejsze prace nad sztuczną inteligencją zajęły się monitorowaniem, jak zauważać, kiedy coś jest nie tak, a następnie wnioskować o tym. Wysiłki te zazwyczaj obejmują planowanie i rozwiązywanie problemów, wnioskowanie i wyszukiwanie. Jedna z głównych szkół myślenia w monitorowaniu i rozumowaniu w celu rozwiązania problemów wykorzystuje modele robota, takie jak reprezentacje wiedzy o komponentach robota, ich powiązaniach i statystycznych modelach wydajności. Najlepszym przykładem jest sonda Deep Space One. Shakey, Strips i podejście do rozważań, które zrodziły, jest głównym konceptualnym odejściem od tradycyjnej teorii kontroli. Skupienie się na jawnych reprezentacjach wiedzy i wykorzystaniu logiki predykatów może być nużące i oszałamiające dla osób przeszkolonych w teorii sterowania. Kolejną barierą w zrozumieniu i realizacji rozważań jest wybór języków programowania. Badacze sztucznej inteligencji przechodzą od kodowania zachowań w proceduralnych językach programowania, takich jak C++, do kodowania funkcji deliberatywnych w językach funkcjonalnych, takich jak Lisp, które lepiej nadają się do odkodowywania i wykonywania klasycznych algorytmów sztucznej inteligencji.

STRIPS

Shakey, pierwszy robot mobilny AI, potrzebował uogólnionego algorytmu do planowania, jak osiągnąć cele. Na przykład przydałoby się, aby ten sam program pozwalał człowiekowi na wpisanie, że robot jest w Office 311 i powinien przejść do Office 313 lub że robot jest w 313 i powinien dostarczyć czerwone pudełko.

OGÓLNE ROZWIĄZYWANIE PROBLEMÓW (GPS)

STRIPS

ANALIZA ŚRODKÓW I KOŃCÓW

Ostatecznie wybrana metoda była wariantem metody rozwiązywania problemów ogólnych o nazwie Strips. Strips wykorzystuje podejście zwane analizą środków i celów, gdzie jeśli robot nie może wykonać zadania lub osiągnąć celu jednym „ruchem”, wybiera działanie, które zmniejszy różnicę między stanem, w jakim był teraz (np. gdzie był) w porównaniu ze stanem docelowym (np. gdzie chciał być). Ta metoda została zainspirowana zachowaniami poznawczymi u ludzi; jeśli nie możesz zobaczyć, jak rozwiązać problem, spróbuj rozwiązać część problemu, aby zobaczyć, czy przybliży cię do pełnego rozwiązania.

STAN CELOWY

STAN POCZĄTKOWY

OPERATOR

RÓŻNICA

EWALUATOR RÓŻNIC

Rozważ próbę zaprogramowania robota, aby dowiedzieć się, jak dostać się do Stanford AI Lab (SAIL). O ile robot nie znajduje się w SAIL (reprezentowanym w Strips jako zmienny stan docelowy), trzeba będzie zorganizować jakiś rodzaj transportu. Załóżmy, że robot znajduje się w Tampa na Florydzie (stan początkowy). Robot może reprezentować proces decyzyjny, jak dotrzeć do lokalizacji, jako funkcję nazywaną operatorem, która uwzględniałaby odległość euklidesową (zmienna nazwana różnicą) między stanem docelowym a stanem początkowym. Różnicę między lokalizacjami można obliczyć w celach porównawczych, formalnie nazywanych „oceną". Funkcja matematyczna, która obliczyła odległość, nazywana jest ewaluatorem różnicy. Na przykład, używając arbitralnego układu odniesienia (X, Y), który umieszcza Tampę w centrum świata z wymyślonymi odległościami do Stanford:

initial state:	Tampa, Florida (0,0)
goal state:	Stanford, California (1000,2828)
difference:	3,000

TABELA RÓŻNIC

To podejście zorientowane na różnice może prowadzić do struktury danych zwanej tabelą różnic, określającą sposób podejmowania decyzji:

difference	operator
d≥200	fly
100<d < 200	ride_train
d ≤ 100	drive
d< 1	walk

Różne środki transportu są odpowiednie dla różnych odległości. Sposób transportu, latanie, jazda_pociągiem, jazda, spacer, w tabeli to tak naprawdę funkcja w programie robota. Nazywa się go również operatorem, ponieważ jeśli zostanie zastosowany, zmniejsza odległość między początkowym stanem Tampa a stanem docelowym. Robot podążający za tą tabelą różnic zaczynałby od planowania lotu jak najbliżej SAIL.

WSTĘPNE WARUNKI Załóżmy jednak, że robot wleciał na lotnisko w San Francisco. Znajdowałby się w promieniu 100 mil od SAIL, więc wydawałoby się, że robot podjął mądrą decyzję. Ale robot nie osiągnął swojego celu, a zatem ma nową różnicę, którą należy zredukować, to znaczy pokonać te 100 mil między lotniskiem a SAIL. Analizuje tabelę różnic z nową wartością różnicy. Stół mówi, że robot powinien jeździć. Co prowadzić? Samochód? Ups: gdyby robot miał samochód osobisty, wróciłby do Tampy. Robot musi być w stanie odróżnić prowadzenie swojego samochodu od prowadzenia wynajętego samochodu. To rozróżnienie jest dokonywane przez wypisanie warunków wstępnych, które muszą być spełnione, aby wykonać ten konkretny operator. Warunki wstępne są reprezentowane przez kolumnę w tabeli różnic; zwróć uwagę, że jeden operator może mieć wiele warunków wstępnych. W praktyce lista warunków wstępnych jest dość długa, ale na potrzeby tego przykładu tylko drive_rental, drive_personal będą wyświetlane z warunkami wstępnymi.

difference	operator	preconditions
d≤200	fly	
100<d<200	ride_train	
d≤100	drive_rental	at airport
	drive_personal	at home
d<1	walk	

DODAJ LISTĘ

USUŃ LISTĘ

Tabela różnic umożliwia robotowi prowadzenie wynajętego samochodu, jeśli znajduje się on na lotnisku. Ale ten dodatek do tabeli różnic wprowadza nowy problem: Skąd robot wie, gdzie się znajduje? Robot wie, gdzie się znajduje, monitorując swój stan i świat. Jeśli robot poleciał samolotem z Tampy na lotnisko w San Francisco, jego stan się zmienił. Jego stanem początkowym jest teraz lotnisko w San Francisco, a nie Tampa. Dlatego za każdym razem, gdy robot wykonuje operatora, prawie zawsze trzeba dodać coś do wiedzy robota o stanie świata (co jest wprowadzane do listy dodawania) i coś, co należy usunąć (usuń listę). Te dwie listy są przechowywane w tabeli różnic, więc gdy robot wybierze operatora na podstawie różnicy, a następnie ją wykona, może łatwo zastosować odpowiednie modyfikacje do świata. Poniższa tabela różnic została rozszerzona, aby pokazać listy dodawania i usuwania.

difference	operator	pre-conditions	add-list	delete-list
d≤200	fly		at Y at airport	at X
100<d<200	train		at Y at station	at X
d≤100	drive_rental	at airport		
	drive_personal	at home		
d<1	walk			

Oczywiście powyższa tabela różnic jest dość niekompletna. Prowadzenie wypożyczonego samochodu powinno mieć warunek, że jest dostępny wypożyczony samochód. (I że robot ma zgodę na prowadzenie pojazdu eksperymentalnego i ma satysfakcjonującą metodę płatności). z punktu widzenia programowania. Ignorując na razie szczegóły, najważniejsze jest to, że tabela różnic wydaje się być dobrą strukturą danych do reprezentowania tego, czego robot potrzebuje do zaplanowania podróży. Można napisać funkcję rekurencyjną, aby zbadać każdy wpis w tabeli w celu znalezienia pierwszego operatora, który zmniejsza różnicę, a następnie powtarzać i zmniejszać tę różnicę i tak dalej, aż do osiągnięcia celu. Powstała lista operatorów to właściwie plan: lista kroków (operatorów), które robot musi wykonać, aby osiągnąć cel. Zwróć uwagę, że robot konstruuje cały plan przed przekazaniem go do wykonania innemu programowi, ponieważ algorytm planowania może wypróbować operatora, a później odkryć, że to nie zadziała i musi się cofnąć, aby wypróbować innego operatora. W tej chwili jest mało prawdopodobne, aby robot wsiadł do samolotu, a następnie jechał. Być może więc krytyka Strips jest spowodowana tym, że użyty przykład jest zbyt skomplikowanym zadaniem, aby był realistyczny. Przyjrzyjmy się, czy Strips staje się bardziej uproszczony, gdy stosuje się go do prostego zadania, jakim jest przechodzenie z jednego pokoju do drugiego.

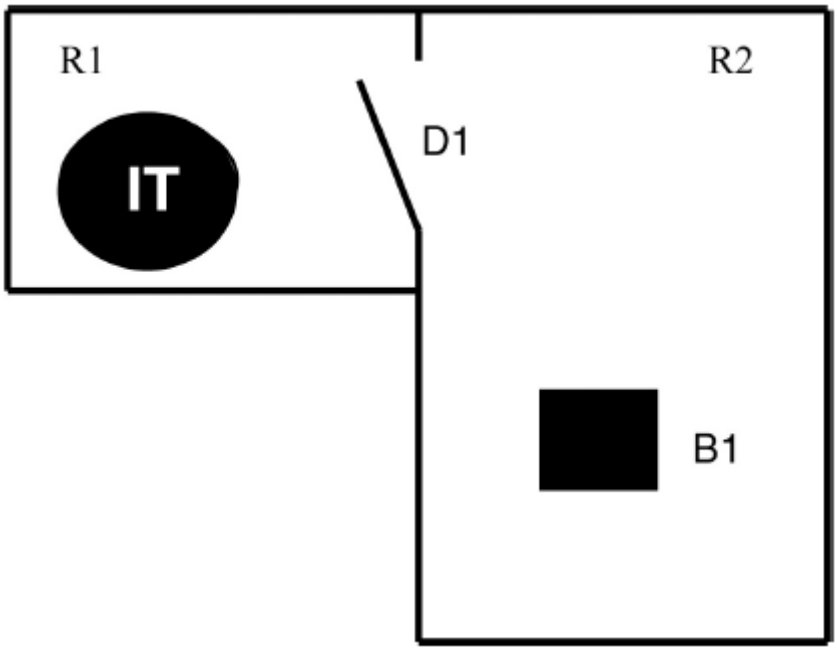
Bardziej realistyczny przykład Strips

TWORZENIE MODELU ŚWIATA

AKSJOMY

PREDYKATY

Pierwszym krokiem w tworzeniu planera Strips jest skonstruowanie reprezentacji świata opartej na Strips lub modelu świata. Wszystko na świecie, co jest istotne dla problemu, jest reprezentowane przez fakty lub aksjomaty w logice predykatów. Predykaty to funkcje, których wynikiem jest PRAWDA lub FAŁSZ. Zgodnie z konwencją programowania AI, predykaty są pisane wielkimi literami. Rozważmy problem robota o nazwie IT w pomieszczeniu R1, gdzie robot musi przejść do innego pomieszczenia R2, w domu pokazanym na rysunku



Aby rozwiązać ten problem za pomocą Strips, robotowi należy dać sposób reprezentowania świata, co z kolei wpłynie na tabelę różnic, narzędzie do oceny różnic oraz sposób napisania list dodawania i usuwania. Model świata w poprzednim przykładzie nigdy nie został formalnie zdefiniowany. Model świata jest generalnie budowany ze statycznych faktów (reprezentowanych jako predykaty) ze zbioru kandydatów i rzeczy ze świata, takich jak robot. Nazwa robota jest pisana wielkimi literami, ponieważ istnieje (formalnie jest PRAWDA). Identyfikatory pisane małymi literami wskazują, że rzecz jest zmienną, że rzeczywista rzecz nie została jeszcze przypisana do tego symbolu zastępczego. Załóżmy, że robot ograniczył się tylko do wiedzy, czy ruchomy przedmiot, taki jak pudełko, znajduje się w pokoju, obok drzwi lub innego ruchomego przedmiotu oraz czy drzwi są otwarte, czy zamknięte i z jakimi pokojami są połączone. W sensie programistycznym na świecie byłyby tylko trzy rodzaje rzeczy: movable_object (takie jak robot lub pudełko), pokój i drzwi. Wiedza robota może być reprezentowana przez następujące predykaty:

`INROOM(x, r)`	where `x` is an object of type `movable_object`, `r` is type `room`
`NEXTTO(x, t)`	where `x` is a `movable_object`, `t` is type `door` or `movable_object`
`STATUS(d, s)`	where `d` is type `door`, `s` is an enumerated type: `OPEN` or `CLOSED`
`CONNECTS(d, rx, ry)`	where `d` is type `door`, `rx, ry` are the `room`

Mając powyższe predykaty, model świata dla początkowego stanu świata na powyższym rysunku byłby reprezentowany przez:

```
initial state:
INROOM(IT, R1)
INROOM(B1,R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1,OPEN)
```

Ten model świata pokazuje, że konkretny ruchomy_obiekt o nazwie IT znajduje się w pomieszczeniu o nazwie R1, a B1 znajduje się w innym pomieszczeniu o nazwie R2. Drzwi D1 łączą R1 z R2 i łączą R2 z R1. (Dwa różne predykaty CONNECTS są używane do wskazania, że robot może przejść przez drzwi z dowolnego pokoju.) Drzwi o nazwie D1 mają wyliczoną wartość OTWARTE. Predykat NEXTTO nie został użyty, ponieważ nie był prawdziwy i nie byłoby z czym powiązać zmiennych. W tym stylu reprezentacji światowy model stanu docelowego byłby następujący:

```
goal state:
INROOM(IT,R2)
INROOM(B1,R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1,OPEN)
```

KONSTRUKCJA TABELI RÓŻNIC

Po ustaleniu modelu świata możliwe jest skonstruowanie tabeli różnic. Tabela różnic cząstkowych to:

operator	preconditions	add-list	delete-list
OP1: GOTODOOR(IT,dx)	INROOM(IT,rk) CONNECT(dx,rk,rm)	NEXTTO(IT,dx)	
OP2: GOTHRUDOOR(IT,dx)	CONNECT(dx,rk,rm) NEXTTO(IT,dx) STATUS(dx, OPEN) INROOM(IT,rk)	INROOM(IT,rm)	INROOM(IT,rk)

Ta tabela różnic mówi, że robot jest zaprogramowany tylko na dwie operacje: przejście do drzwi i przejście przez drzwi. Operator GOTODOOR można zastosować tylko wtedy, gdy spełnione są dwa następujące warunki wstępne:

* INROOM(IT, rk) Robot znajduje się w pomieszczeniu, któremu zostanie przypisany identyfikator rk.

* CONNECT(dx, rk, rm) Są drzwi, które zostaną przypisane do identyfikatora dx, które łączą rk z innym pokojem zwanym rm.

Etykieta IT służy do ograniczania predykatów. Zauważ, że tylko zmienne dx i rk są wiązane, gdy wywoływana jest funkcja GOTODOOR. rm może być wszystkim. Jeśli GOTODOOR jest wykonywany, robot znajduje się teraz obok drzwi o nazwie dx. Nic nie jest usuwane ze stanu świata, ponieważ robot nadal znajduje się w pokoju rk; drzwi dx nadal łączą dwa pokoje rk i rm. Jedyne, co się zmieniło, to fakt, że robot znajduje się teraz na godnej uwagi pozycji w pomieszczeniu: przy drzwiach. Tabela różnic określa, że operator GOTHRUDOOR będzie działał tylko wtedy, gdy robot znajduje się w pomieszczeniu, obok drzwi, drzwi są otwarte, a drzwi łączą pomieszczenie, w którym znajduje się robot, z innym pomieszczeniem. W takim przypadku predykaty muszą być dodawane i usuwane z modelu świata podczas wykonywania operatora. Kiedy robot znajduje się w pokoju rk i przechodzi przez drzwi,

znajduje się teraz w pokoju rm (który musi zostać dodany do modelu świata) i nie znajduje się już w pokoju rk (który należy usunąć). Jak dotąd konstruowanie modelu świata i tabeli różnic powinno wydawać się rozsądne, choć żmudne. Ale konstruowanie tabeli różnic jest bezcelowe bez funkcji oceny różnic. (Zauważ, że w powyższej tabeli nie było kolumny dla różnicy.) Elementem oceny różnicy w przykładzie podróży była odległość euklidesowa. W tym przykładzie oceniającym jest rachunek predykatów, w którym stan początkowy jest logicznie odejmowany od stanu docelowego. Logiczna różnica między początkowym stanem docelowym jest po prostu:

¬INROOM(IT, R2) or INROOM(IT, R2)=FALSE

REDUKCJA RÓŻNIC

Zmniejszanie różnic jest trochę jak rozwiązywanie układanki, w której Strips próbuje różnych podstawień, aby sprawdzić, czy konkretny operator zmniejszy różnicę. Aby zmniejszyć różnicę, Strips szuka dopasowania w tabeli różnic pod kolumną listy dodawania, zaczynając od góry. Wygląda w kolumnie listy dodawania, a nie w oddzielnej kolumnie różnic, ponieważ wpis listy dodawania dla operatora wyraża wynik tego operatora. Jeśli Strips znajdzie operator, który generuje stan docelowy, wówczas operator ten eliminuje istniejącą różnicę między stanem początkowym a stanem docelowym. Wpis na liście dodatków dla OP2: GOTHRUDOOR ma dopasowanie w formularzu. Jeśli rm=R2, to wynikiem OP2 byłoby INROOM(IT, R2), co wyeliminowałoby różnicę; więc OP2 jest kandydatem na operatora. Przed nałożeniem OP2 paski muszą sprawdzić warunki wstępne. Aby to zrobić, rm musi zostać zastąpione przez R2 w każdym predykacie w warunkach wstępnych. OP2 ma dwa warunki wstępne, dotyczy tylko CONNECTS(dx, rk, rm). Staje się CONNECTS(dx, rk, R2). Dopóki dx i rk nie zostaną powiązane, orzeczenie nie ma wartości prawdziwej ani fałszywej. Zasadniczo dx, rk to symbole wieloznaczne, CONNECTS(*, *, R2). Aby uzupełnić wartości tych zmiennych, Strips sprawdza aktualny stan modelu świata, aby znaleźć dopasowanie. Predykat w aktualnym stanie świata CONNECTS(D1, R1, R2) pasuje do CONNECTS(*, *, R2). D1 jest teraz związane z dx, a R1 jest związane z rk.

NIEUDANE WARUNKI WSTĘPNE

Teraz Strips propaguje powiązania do następnego warunku wstępnego na liście: NEXTTO(IT, dx). NEXTTO(IT, D1) ma wartość FALSE, ponieważ orzeczenie nie jest w bieżącym stanie świata. NEXTTO(IT, D1) jest określany jako warunek wstępny zakończony niepowodzeniem. Nieformalna interpretacja jest taka, że GOTHRUDOOR(IT, D1) doprowadzi robota do stanu docelowego, ale zanim to zrobi, IT musi być obok D1.

REKURENCJA W CELU ROZSTRZYGNIĘCIA RÓŻNIC

Zamiast się poddawać, STRIPS rekursywnie (wykorzystuje technikę programowania rekurencji), aby powtórzyć całą procedurę. Oznacza pierwotny stan celu jako G0, odkłada go na stos, a następnie tworzy nowy stan celu podrzędnego, G1, NEXTTO(IT, D1). Różnica między NEXTTO(IT, D1) a obecnym stanem świata to:

¬NEXTTO(IT, D1)

Strips ponownie przeszukuje kolumnę listy dodawania w tabeli różnic, aby znaleźć operator, który zanegował to. Rzeczywiście, OP1: GOTODOOR(IT, dx) ma dopasowanie na liście dodatków NEXTTO(IT, dx). Strips musi zacząć od ponownego przypisania wartości do identyfikatorów, ponieważ program wszedł w nowe programowanie zakresu, więc dx=D1. Strips ponownie sprawdza warunki wstępne. Tym razem rk=R1 i rm=R2 mogą być dopasowane za pomocą CONNECTS(dx, rk, rm), a wszystkie warunki wstępne są spełnione (czyli są oceniane jako prawdziwe). Strips umieszcza operator OP1 na stosie planów i stosuje go do modelu świata, zmieniając stan. (Zauważ, że jest to odpowiednik „operacji

umysłowej". Robot tak naprawdę nie podchodzi do drzwi; po prostu zmienia stan, aby wyobrazić sobie, co by się stało, gdyby to zrobił). Przypomnijmy, że początkowy stan modelu świata był następujący:

```
initial state:
INROOM(IT, R1)
INROOM(B1,R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1,OPEN)
```

Zastosowanie operatora OP1 oznacza dokonanie zmian na liście dodawania i usuwanej liście. Istnieje tylko predykat do zmiany na liście add-list, a żaden na liście usuniętych. Po dodaniu NEXTTO(IT, D1) stan świata to:

```
state after OP1:
INROOM(IT, R1)
INROOM(B1,R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1,OPEN)
NEXTTO(IT, D1)
```

Strips następnie przywraca kontrolę do poprzedniego wywołania. Wznawia się tam, gdzie zakończył ocenę warunków wstępnych dla OP2 z dx=D1, rm=R2 i rk=R1. Dopiero teraz zmienił się model świata. Zarówno STATUS(D1, OPEN) jak i INROOM(IT, R1) są prawdziwe, więc wszystkie warunki wstępne dla OP2 są spełnione. Strips umieszcza OP2 na swoim stosie planów i zmienia model świata, stosując predykaty listy dodawania i listy usuwania. Oto, jaki będzie stan świata, gdy plan zostanie zrealizowany:

```
state after OP2:
INROOM(IT, R2)
INROOM(B1,R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1,OPEN)
NEXTTO(IT, D1)
```

Stripsy, a plan robota do wykonania (w odwrotnej kolejności na stosie) to: GOTODOOR(IT, D1), GOTHRUDOOR(IT, D1).

Podsumowanie Strips

Strips działa rekurencyjnie; jeśli nie może osiągnąć celu bezpośrednio, identyfikuje problem (niespełniony warunek wstępny), a następnie sprawia, że niespełniony warunek wstępny staje się podcelem. Po osiągnięciu celu podrzędnego Strips umieszcza operatora do osiągnięcia celu podrzędnego na liście, a następnie wykonuje kopię zapasową (zrzuca stos) i wznawia próbę osiągnięcia poprzedniego celu. Odcina plany, a nie wykonuje: tworzy listę operatorów do zastosowania, ale nie stosuje operatora w trakcie. Implementacje Strips wymagają od projektanta ustawienia:

* reprezentacja modelu świata,

* tabela różnic z operatorami, warunkami wstępnymi, dodawaniem i usuwaniem list oraz

* oceniający różnice.

Etapy wykonywania Pasków to:

1. Oblicz różnicę między stanem docelowym a stanem początkowym za pomocą funkcji oceny różnicy. Jeśli nie ma różnicy, zakończ.

2. Jeśli istnieje różnica, zmniejsz różnicę, wybierając pierwszy operator z tabeli różnic, którego lista dodatków ma predykat negujący różnicę.

3. Następnie sprawdź warunki wstępne, aby zobaczyć, czy można uzyskać zestaw powiązań dla zmiennych, z których wszystkie są prawdziwe. Jeśli nie, weź pierwszy warunek wstępny FALSE, ustaw go jako nowy cel i zapisz pierwotny cel, odkładając go na stos. Rekurencyjnie zmniejsz tę różnicę, powtarzając kroki 2 i 3.

4. Gdy wszystkie warunki wstępne dla operatora są zgodne, wepchnij operatora na stos planu i zaktualizuj kopię modelu świata. Następnie wróć do operatora z niespełnionym warunkiem wstępnym, aby mógł zastosować swój operator lub wykonać rekursję na innym nieudanym warunku wstępnym.

Powrót do założenia zamkniętego świata i problemu ramowego

ZAŁOŻENIE ŚWIATA ZAMKNIĘTEGO

PROBLEM Z RAMĄ

Strips uwrażliwił społeczność robotyczną na dwa wszechobecne problemy: założenie o zamkniętym świecie i problem ramek. Jak zdefiniowano wcześniej, założenie zamkniętego świata mówi, że model świata zawiera wszystko, co robot musi wiedzieć: nie może być niespodzianek. Jeśli założenie zamkniętego świata zostanie naruszone, robot może nie być w stanie działać poprawnie. Ale z drugiej strony bardzo łatwo zapomnieć o umieszczeniu wszystkich niezbędnych szczegółów w modelu świata. W rezultacie sukces robota zależy od tego, jak dobrze programista potrafi myśleć o wszystkim.

ZAŁOŻENIE OTWARTEGO ŚWIATA

Ale nawet zakładając, że programista wymyślił wszystkie przypadki, wynikowy model świata prawdopodobnie będzie ogromny. Zastanów się, jak duży i nieporęczny był model świata tylko do przemieszczania się między dwoma pokojami. I nie było przeszkód! Ludzie zaczęli zdawać sobie sprawę, że liczba faktów (lub aksjomatów), które program będzie musiał przeanalizować przy każdym przejściu przez tabelę różnic, stanie się niewykonalna dla jakiegokolwiek realistycznego zastosowania. Problem reprezentowania sytuacji w świecie rzeczywistym w sposób, który można było obliczyć obliczeniowo, stało się znane jako problem ramy. Przeciwieństwo założenia zamkniętego świata znane jest jako założenie otwartego świata. Kiedy robotycy mówią, że „robot musi funkcjonować w otwartym świecie", twierdzą, że założenia o zamkniętym świecie nie można realistycznie zastosować do tej konkretnej dziedziny. Powyższy przykład, choć banalny, pokazuje, jak żmudne są Strips (chociaż komputery są dobre w żmudnych algorytmach). W szczególności nieintuicyjna jest potrzeba formalnego reprezentowania świata, a następnie utrzymywania w nim każdej zmiany. Strips ilustruje również zaletę założenia o zamkniętym świecie: wyobraź sobie, jak trudno byłoby zmodyfikować algorytm planowania, gdyby model świata mógł się nagle zmienić. Algorytm może się zgubić między rekursjami. Ten przykład powinien również uświadomić znaczenie problemu z ramą: wyobraź sobie, co stanie się z rozmiarem modelu świata, jeśli doda się trzecie pomieszczenie z pudełkami, do których robot może się przemieścić i podnieść! A to tylko dla świata pokoi i pudeł. Oczywiście aksjomaty, które kształtują świat, staną się zbyt liczne, aby można je było wykorzystać do opisania jakiejkolwiek realistycznej dziedziny. Jednym z wczesnych rozwiązań był ABStrips, który próbował podzielić problem

na wiele warstw abstrakcji, to znaczy najpierw rozwiązać problem na zgrubnym poziomie. Miało to swoje wady i wkrótce wielu ludzi, którzy zaczynali w robotyce, zaczęło pracować w obszarze sztucznej inteligencji zwanym planowaniem. Te dwie dziedziny stały się odrębne, a do lat 80. badacze zajmujący się planowaniem i robotyką mieli oddzielne konferencje i publikacje. Wielu robotyków w latach 70. i 80. pracowało albo nad kwestiami związanymi z wizją komputerową, próbując sprawić, by roboty były w stanie lepiej wyczuwać świat, albo nad planowaniem ścieżki, obliczaniem najbardziej wydajnej trasy omijania przeszkód i tak dalej, do lokalizacji docelowej .

Problem z uziemieniem symbolu

SYMBOL FIZYCZNY UZIEMIENIE

Strips opiera się na modelu światowym. Model świata to struktura wiedzy, w której regiony świata odczuwanego są oznaczone symbolami, takimi jak korytarz, drzwi, moje biuro i tak dalej. Proces przypisywania symboli do obiektów lub pojęć nazywany jest w kognitywistyce ugruntowaniem symboli. Robotyka koncentruje się na fizycznym ugruntowaniu symboli, gdzie dane pozyskane z czujników o świecie fizycznym są przypisywane do symboli.52 Symbole reprezentują obiekty lub stany świata, które mają pewną trwałość, a zatem są trwałe. Na przykład robot może minąć pomieszczenie, w którym są otwarte drzwi, a następnie wrócić później, gdy drzwi są zamknięte - Skąd robot wie, że minął pomieszczenie z drzwiami lub że jest to ten sam pokój, ale drzwi są teraz zamknięte lub że ktoś musiał zamknąć drzwi? Ogólne uziemienie symboli staje się jeszcze ważniejsze, jeśli system robota przyjmuje polecenia z interfejsów w języku naturalnym („dostarcz to Johnowi"), a następnie wykorzystuje je do zadań nawigacyjnych i procedur rozpoznawania percepcyjnego. Teraz robot musi przełożyć „ten" symbol na koncepcję, a następnie na namacalną fizyczną obecność w świecie; „to" oznacza paczkę na stole przed człowiekiem, a nie stół czy filiżankę kawy. Takie tłumaczenia symboli na obiekty fizyczne i zrozumienie, co zrobić z tymi obiektami fizycznymi, są często przedmiotem rozważań na temat ludzkich przekonań, pragnień i intencji. Zrozumienie fizycznego uziemienia symboli jest priorytetem społeczności uziemiającej symbole, ale pozostaje nieuchwytne przez dziesięciolecia. Jest to priorytetowe, ponieważ inne rodzaje uziemienia symboli, takie jak język naturalny ze smartfonami, mają więcej możliwości rozwiązania niejasności („nie zrozumiałem cię") i mogą ustrukturyzować interakcję. Istnieje stosunkowo mniej opcji ujednoznacznienia w uziemianiu symboli do nawigacji i rozumienia świata. Inny powód, dla którego fizyczne uziemienie symbolu jest trudne, wynika z wariantu problemu z ramą: jak budować i opisywać modele świata z właściwą ilością szczegółów.

KOTWICZENIE

Fizyczne uziemienie symbolu obejmuje zakotwiczenie: przypisanie etykiet do wejść percepcyjnych, a tym samym zakotwiczenie percepcji we wspólnym układzie odniesienia, który może być wykorzystany przez funkcje deliberacyjne robota. Kotwiczenie jest zwykle realizowane za pomocą jednej z dwóch ogólnych strategii:

* Odgórne rozpoznawanie obiektów, które pasuje do istniejących symboli w reprezentacjach wiedzy robota. Na przykład robot jest zaprogramowany do wyszukiwania drzwi, pokoi, korytarzy, ślepych zaułków, „mojej filiżanki kawy" i tak dalej, i ma opisy, jak je rozpoznać na podstawie sygnałów percepcyjnych. Zaletą tej odgórnej strategii uziemiania symboli jest to, że filtruje ona niepotrzebne lub nieistotne informacje percepcyjne i identyfikuje tylko to, co jest ważne, to znaczy narzuca de facto rozwiązanie problemu ramki. Jest to podejście „znajdź fizyczną manifestację symbolu w wejściach percepcyjnych". Wadą strategii odgórnej w otwartym świecie jest to, że może przegapić coś ważnego, ponieważ czegoś nie szuka się lub nie ma w bazie wiedzy. Inną wadą jest to, że koncepcja lub cecha na mapie może być trudna do wiarygodnego dostrzeżenia przez robota. W ten sposób istnieje rozdźwięk między reprezentacją wewnętrzną a światem zewnętrznym.

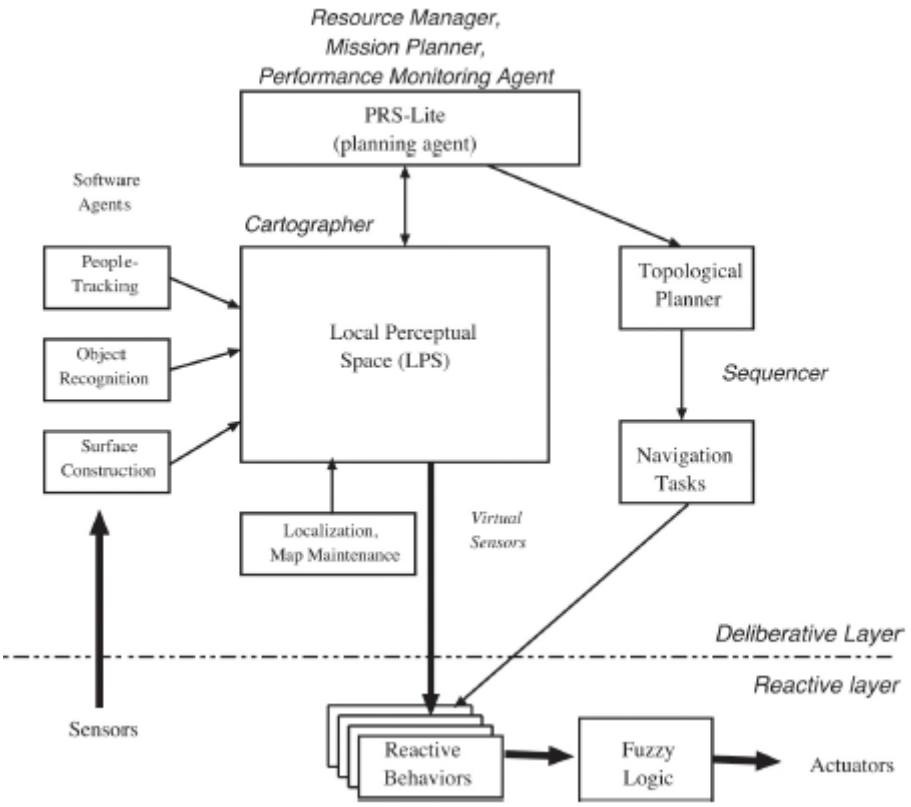
* Oddolne rozpoznawanie obiektów, w którym robot wyczuwa nieznany obiekt, określa, że jest on interesujący, deklaruje go jako obiekt i przypisuje mu etykietę Object-23. Etykieta może później zostać zmieniona na CoffeeCup (My). Jednym z intrygujących mechanizmów zmiany etykiety jest uruchomienie przez robota wyszukiwania w sieci WWW, aby sprawdzić, czy może znaleźć dowolny obraz z etykietą, który pasuje do tego, co widzi. Jest to „Co jest w wejściach percepcyjnych?" zbliżać się. Zaletą podejścia oddolnego jest to, że nie wyklucza nieoczekiwanych obiektów. Wadą jest określenie, jak percepcyjnie pogrupować cechy w spójny obiekt - na przykład zrozumienie, gdzie kończy się filiżanka kawy, a zaczyna stół. Chociaż zakotwiczenie jest konieczne do uziemienia symbolu, nie jest ono wystarczające. Może identyfikować, czym jest przedmiot i gdzie się znajduje, ale nie zapewnia semantyki przedmiotu. Zastanów się na przykład nad ustaleniem, czy percepcyjnym sygnałem wejściowym jest „filiżanka kawy Donny" lub „została tutaj po spotkaniu i powinna być z powrotem w jej biurze". Nie ma wystarczających informacji percepcyjnych, aby oznaczyć „filiżankę kawy" tymi informacjami; Wymagane są zdolności deliberatywne, takie jak pamięć, rozumowanie i wnioskowanie. W rezultacie polecenie robotowi „Podnieś filiżankę Donny i odłóż ją" może zostać utrudnione, jeśli w zasięgu wzroku są dwie filiżanki kawy, chyba że wykonywana jest bardziej rozważna praca w celu rozwiązania niejednoznaczności.

Globalne modele świata

Jeśli uziemienie symbolu jest trudne dla pojedynczego obiektu, takiego jak CoffeeCup(My), to w jaki sposób konstruowane są globalne modele świata do celów deliberatywnych? Jedną z metod jest posiadanie globalnego modelu świata składającego się ze zbioru ograniczonych globalnych modeli świata zwanych lokalnymi przestrzeniami percepcyjnymi. Inną, bardziej popularną metodą jest stworzenie modelu świata z warstwami abstrakcji na temat świata fizycznego, zwanych hierarchicznymi globalnymi modelami świata. Zagnieżdżony kontroler hierarchiczny jest najbardziej wszechobecnym hierarchicznym globalnym modelem świata i dlatego zostanie omówiony w osobnej sekcji. Oba typy modeli świata mogą być wykorzystywane przez warstwę reaktywną jako wirtualne czujniki. Jednak lokalne przestrzenie percepcyjne i hierarchiczne modele globalne skupiają się na zakotwiczeniu, określeniu, gdzie znajdują się obiekty, ignorując jednocześnie włączenie innej wiedzy, idealnie zawartej w globalnym modelu świata.

Lokalne przestrzenie percepcyjne

Lokalne przestrzenie percepcyjne zostały wprowadzone przez Konolige i Myersa jako część Erratic, wnuka Shakeya. Pomysł polega na tym, że robot tworzy i utrzymuje model regionu, w którym działa. Na przykład robot może wtoczyć się do pokoju i stworzyć skan 3D pokoju oraz wykryć i zakotwiczyć wszelkie interesujące obiekty w tym pokoju. Następnie może przetoczyć się do innego pokoju i stworzyć model tego pokoju, prowadząc do globalnego modelu świata dwóch bardzo bogatych lokalnych modeli konkretnych pokoi, które są połączone modelem korytarza. Kotwiczenie może być ograniczone do wszystkiego, co ma znaczenie dla zadania robota, rozwiązując w ten sposób problem z ramą. To, co było bardzo innowacyjne w lokalnych przestrzeniach percepcyjnych, to możliwość ich tworzenia i udostępniania przez zespół robotów. Jeden robot mógł wyczuć jeden pokój, podczas gdy inny robot jednocześnie wyczuwał inny pokój, a następnie obaj mogli podzielić się swoimi spostrzeżeniami i stworzyć połączony, jeden globalny model świata. Układ architektoniczny Erratic, zwany architekturą Saphira.



Lokalne przestrzenie percepcyjne są przeciwieństwem większości symultanicznych algorytmów lokalizacji i mapowania opisanych w rozdziale 15, które starają się stworzyć pojedynczą trójwymiarową reprezentację wszystkiego na świecie bez zakotwiczania określonych obiektów. Trójwymiarowa mapa środowiska jest zubożałym globalnym modelem świata, ponieważ nie ma uziemienia symbolu.

Wielopoziomowe lub hierarchiczne modele świata

Wielopoziomowe lub hierarchiczne modele świata tworzą model świata składający się z warstw abstrakcji. Najniższym poziomem może być przestrzenna reprezentacja świata 2D lub 3D. Następna wyższa lub bardziej abstrakcyjna warstwa może być topologiczną reprezentacją przestrzenną, która określa części świata jako „pokoje", „drzwi" i „korytarze". Następna warstwa może oznaczać obiekty w pomieszczeniu lub opisywać kolor ścian. Warstwy są rejestrowane przestrzennie, więc jeśli funkcja deliberatywna przeszukuje model świata w poszukiwaniu filiżanek do kawy i stwierdza, że istnieje ich duża kolekcja, może powiązać to znalezisko z „kuchnią" i wiedzieć, gdzie jest kuchnia. Ta metodologia jest podobna do hierarchii percepcyjnych lub piramid percepcyjnych w wizji komputerowej, w której obraz biura jest abstrakcjonowany na warstwy i etykiety reprezentujące region, dopóki nie pojawi się jedno oznaczone jako „biuro Murphy'ego". Warstwy, jako ścisła hierarchia abstrakcji, od reprezentacji przestrzennej do symbolu, okazały się niepraktyczne ze względu na wiele rodzajów funkcji rozważania, które były potrzebne do przechowywania i wykorzystywania informacji o świecie. Rzeczywiście, termin „globalny" jest używany prawie jako synonim słowa „deliberatywne", a „lokalny" z „reaktywnym". Ta synonimiczna terminologia może prowadzić do znacznego zamieszania, ponieważ „globalny" nie zawsze jest prawdziwie globalny. Deliberatywna część architektury hybrydowej zawiera moduły i funkcje dla rzeczy, które nie są łatwe do przedstawienia w zachowaniach reaktywnych. Niektóre z tych funkcji wyraźnie wymagają globalnego modelu świata; Planowanie ścieżek i tworzenie map to prawdopodobnie najlepsze przykłady. Ale inne działania wymagają globalnej wiedzy innego rodzaju. Zarządzanie behawioralne (planowanie, jakich zachowań użyć) wymaga znajomości aktualnej misji i aktualnego (i przewidywanego) stanu środowiska. Jest to wiedza globalna, ponieważ wymaga, aby

moduł wiedział coś na zewnątrz siebie w porównaniu z reaktywnym zachowaniem, które może funkcjonować bez jakiejkolwiek wiedzy o tym, czy inne zachowania są aktywnie wykonywane. Podobnie monitorowanie wydajności, aby sprawdzić, czy robot rzeczywiście robi postępy w realizacji swojego celu, oraz rozwiązywanie problemów to działania globalne. Rozważ napisanie programu, który diagnozuje, czy robot nie porusza się do przodu, ponieważ jest problem z terenem (utknął w błocie) lub z czujnikiem (enkodery wału nie zgłaszają prawidłowo obrotów kół). Aby przeprowadzić diagnostykę, program musi wiedzieć, jakie zachowania próbowały osiągnąć, czy istnieją inne czujniki lub źródła wiedzy potwierdzające hipotezy i tak dalej. Dlatego funkcja deliberatywna może nie potrzebować globalnego modelu świata, ale może potrzebować wiedzy o wewnętrznym działaniu robota w skali globalnej, choćby po to, by wiedzieć, z jakimi innymi modułami lub możliwościami deliberatywnymi program powinien wchodzić w interakcje.

Wirtualne czujniki

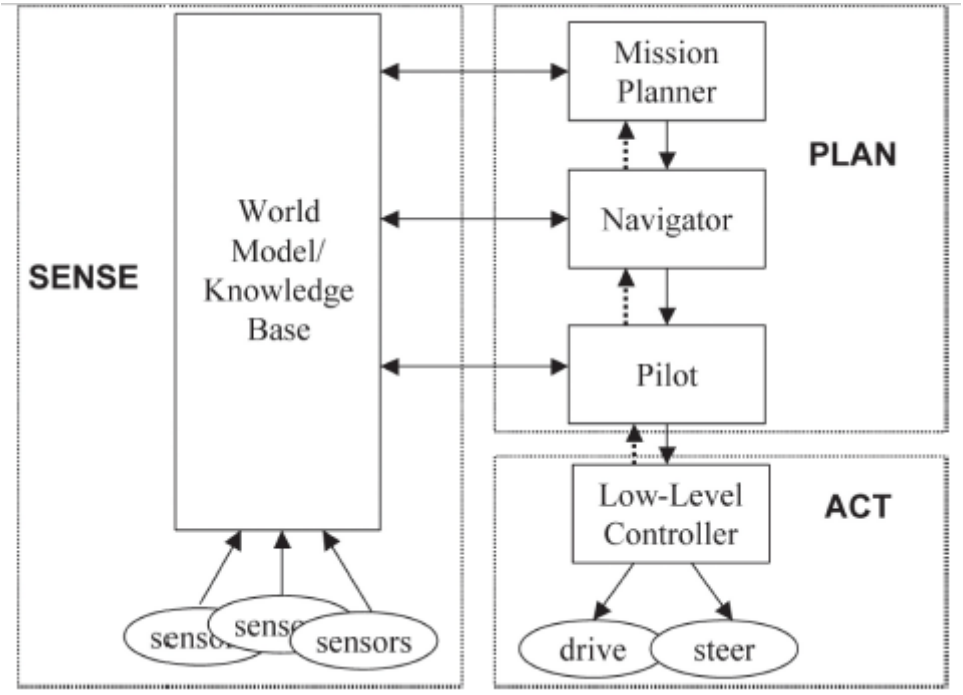
Globalny model świata może również służyć do postrzegania zachowań, w którym to przypadku globalny model świata służy jako wirtualny czujnik. Zamiast schematów percepcyjnych operujących na wejściu czujnika, schemat percepcyjny może wyodrębnić spostrzeżenia z modelu świata. Czujnik logiczny traktuje model świata tak, jakby był wejściem czujnika. Jako przykład czujników logicznych rozważ robota próbującego nawigować w zaśmieconym, trudnym do wyczucia środowisku. Robot posiada schemat behawioralny polegający na unikaniu przeszkód, który wykorzystuje schemat percepcyjny „wyciągnij najbliższą przeszkodę" połączony ze skanerem 3D. Robot może również budować bardzo dokładne odwzorowanie świata w 3D, które ma lepsze informacje o głębi niż pojedynczy skan 3D świata. Globalny model świata można przekonwertować na wirtualny skan 3D, który będzie używany jako dane wejściowe do schematu percepcyjnego. Zaletą jest to, że błędy i niepewność czujnika mogą być filtrowane przez model światowy za pomocą fuzji czujników w czasie. Może to radykalnie poprawić wydajność robota bez zmiany żadnego ze schematów behawioralnych. Ta metoda zachowuje modułowość i filozofię projektowania polegającą na budowaniu zachowań, a nie ich zastępowaniu.

Globalny model świata i rozważania

Wykorzystywanie jednego globalnego modelu świata do wykrywania wydaje się być powrotem do paradygmatu hierarchicznego i koncepcyjnie tak jest, ale ma zasadnicze znaczenie dla rozważań. Nie zawsze jest tak, że osoba rozważająca generuje zestaw zachowań, włącza je i pozwala na wykonanie, dopóki nie ukończy podzadania lub nie zakończy się niepowodzeniem. W przypadku wyczuwania może być pożądane, aby osoba rozważająca udostępniła niektóre z konstruowanych globalnych modeli zachowań. Rozważmy na przykład pojazd robota jadący drogą. Załóżmy teraz, że szuka dużego drzewa, ponieważ ma skręcić w prawo na skrzyżowaniu za drzewem. Aby zachować prawidłową sekwencję zachowań, deliberator ma globalny model świata, w którym odnotowuje się drzewa. Zachowania reaktywne polegające na podążaniu drogą i omijaniu przeszkód nie wymagają wyraźnej reprezentacji drzewa. Ale co, jeśli drzewo rzuca cień, który może dezorientować zachowanie ścieżki i spowodować, że zachowanie będzie wydawać robotowi nieprawidłowe polecenia kierowania? W takim przypadku dobrze by było, gdyby informacja o obecności drzewa została przyswojona przez zachowanie. Jednym ze sposobów, aby to zrobić, jest umożliwienie metodom modelu świata, aby działały jako wirtualne czujniki lub schemat percepcyjny. Następnie zachowanie podążania za drogą może korzystać z wirtualnego czujnika, który informuje o zachowaniu, gdy granice drogi wyodrębnione przez czujnik wizyjny są prawdopodobnie zniekształcone i ignorują dotknięte obszary na obrazie, które byłyby normalne

Zagnieżdżony kontroler hierarchiczny

Nested Hierarchical Controller (NHC) ma wpływ na projektowanie globalnych modeli świata od czasu jego stworzenia przez Alexa Meystela w 1987 roku. Wykorzystuje hierarchiczny globalny model świata, w którym każda warstwa obsługuje określoną zdolność lub agenta. Organizacja jest intuicyjna i łatwa do wdrożenia; dotyczy to jednak tylko nawigacji. Jak pokazano na rysunku



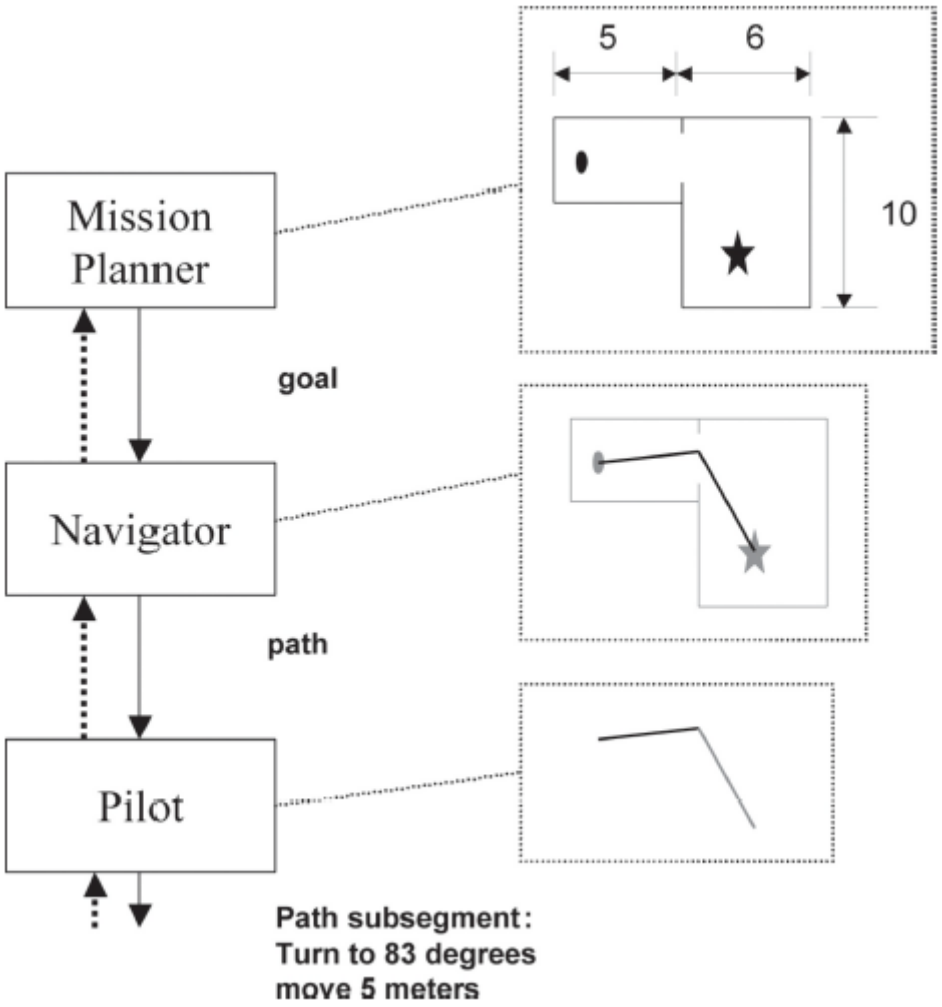
, architektura Nested Hierarchical Controller jest klasycznym systemem hierarchicznym. Zawiera komponenty, które można łatwo zidentyfikować jako SENSE, PLAN lub ACT, używa ich w kolejności i opiera się na globalnym modelu świata. Robot zaczyna od zebrania obserwacji ze swoich czujników i połączenia tych obserwacji w celu utworzenia struktury danych Modelu Świata za pomocą działania SENSE. Model świata może również zawierać a priori wiedzę o świecie — na przykład mapy budynku, zasady dotyczące trzymania się z dala od foyer podczas rozpoczynania i kończenia godzin pracy i tak dalej. Po utworzeniu lub zaktualizowaniu Modelu Świata, robot może PLANOWAĆ, jakie działania powinien wykonać. Planowanie nawigacji ma procedurę lokalną składającą się z trzech kroków wykonywanych przez Mission Planner, Navigator i Pilot. Każdy z tych modułów ma dostęp do Modelu Świata w celu obliczenia ich części planowania. Ostatnim krokiem w planowaniu jest generowanie przez moduł Pilot określonych czynności do wykonania przez robota (na przykład skręcanie w lewo, skręcanie w prawo, poruszanie się prosto z prędkością 0,6 metra na sekundę). Działania te są tłumaczone na sygnały sterujące siłownika (na przykład profil prędkości zapewniający płynny obrót) przez kontroler niskiego poziomu. Kontroler niskiego poziomu i siłowniki tworzą razem część architektury ACT.

PLANOWANIE MISJI

NAWIGATOR

PILOT

Głównym wkładem NHC był sprytny podział planowania na trzy różne funkcje lub podsystemy mające na celu wspieranie nawigacji: Planer misji, Nawigator i Pilot. Jak pokazano na rysunku



, Mission Planner albo otrzymuje misję od człowieka, albo generuje misję dla siebie - na przykład podnosi pudełko w sąsiednim pokoju. Planista misji jest odpowiedzialny za operacjonalizację lub przełożenie tej misji na terminy zrozumiałe dla innych funkcji: box=B1; rm=POKÓJ311. Następnie Mission Planner uzyskuje dostęp do mapy budynku i lokalizuje, gdzie znajduje się robot i gdzie jest cel. Nawigator pobiera te informacje i generuje ścieżkę z bieżącej lokalizacji do celu. Generuje zestaw punktów orientacyjnych lub linii prostych, którymi ma podążać robot. Ścieżka jest przekazywana Pilotowi. Pilot wybiera pierwszy odcinek linii prostej lub ścieżki i określa, jakie czynności musi wykonać robot, aby podążać za tym segmentem ścieżki. Na przykład robot może potrzebować odwrócić się, aby zmierzyć się z punktem orientacyjnym, zanim zacznie jechać do przodu. Co się stanie, jeśli Pilot udzieli wskazówek dotyczących długiego odcinka ścieżki (powiedzmy 50 metrów) lub jeśli osoba nagle przejdzie przed robotem? W przeciwieństwie do Shakeya pod NHC robot niekoniecznie chodzi z zamkniętymi oczami. Po tym, jak Pilot wyda polecenia kontrolera niskiego poziomu, a kontroler wyśle sygnały do siłownika, robot ponownie odpytuje swoje czujniki. Zaktualizowano model świata. Jednak cały cykl planowania się nie powtarza. Ponieważ robot ma plan, nie musi ponownie uruchamiać Planera misji ani Nawigatora. Zamiast tego Pilot sprawdza Model Świata, aby zobaczyć, czy robot zboczył z podsegmentu ścieżki (w tym przypadku generuje nowy sygnał sterujący), osiągnął punkt orientacyjny lub czy pojawiła się przeszkoda. Jeśli robot osiągnął swój punkt orientacyjny, Pilot informuje o tym Nawigatora. Jeśli punkt orientacyjny nie jest celem, oznacza to, że robot musi podążać innym podsegmentem, a więc Nawigator przekazuje nowy podsegment Pilotowi. Jeśli celem jest punkt orientacyjny, Nawigator informuje Planistę, że robot osiągnął cel. Planista misji może wtedy wystawić

nowy cel, na przykład Powrót do miejsca startu. Jeśli robot napotka przeszkodę na swojej drodze, Pilot przekazuje kontrolę z powrotem do Nawigatora. Nawigator musi obliczyć nową ścieżkę i podsegmenty na podstawie zaktualizowanego modelu świata. Następnie Nawigator przekazuje zaktualizowany podsegment ścieżki Pilotowi do wykonania. NHC ma kilka zalet. Różni się od Strips tym, że przeplata planowanie i działanie. Robot wymyśla plan, zaczyna go realizować, a następnie zmienia go, jeśli świat jest inny niż się spodziewał. Zauważ, że rozkład jest z natury hierarchiczny pod względem inteligencji i zakresu. Planista misji jest „mądrzejszy" niż Nawigator, który jest mądrzejszy niż Pilot. Planista misji odpowiada za wyższy poziom abstrakcji niż Nawigator i tak dalej. Zobaczymy, że inne architektury, zarówno w paradygmacie hierarchicznym, jak i hybrydowym, będą wykorzystywać organizację NHC. Wadą rozkładu funkcji planowania przez NHC jest to, że jest ona odpowiednia tylko do zadań nawigacyjnych. Podział obowiązków wydaje się mniej pomocny lub jasny w przypadku zadań takich jak podnoszenie pudełka, a nie tylko przechodzenie do niego. Rola pilota w kontrolowaniu efektorów końcowych nie jest jasna. W momencie początkowego rozwoju, NHC nigdy nie został wdrożony i przetestowany na prawdziwym robocie mobilnym; koszty sprzętu w okresie hierarchicznym zmusiły większość robotników do pracy w symulacji.

RAPS i 3T

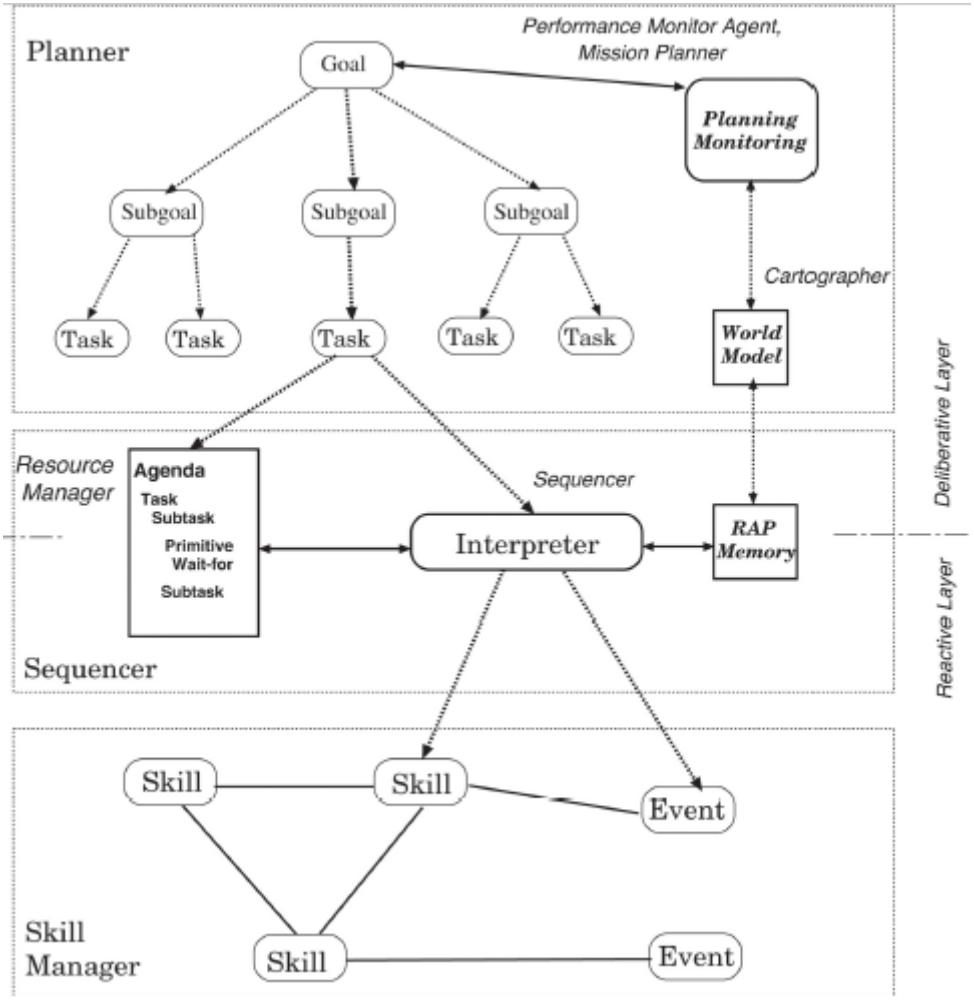
PLANOWANIE REAKTYWNE

Shakey rozważał generalnie deliberację, oferując algorytm rozwiązywania problemów ogólnych dla wszystkich form deliberacji. Planował ciężki. Zagnieżdżony kontroler hierarchiczny uprościł planowanie, skupiając się na planowaniu nawigacyjnym, ignorując, dlaczego robot musiał udać się w to miejsce lub co miał zrobić, gdy tam dotrze. To było zbyt ciężkie w nawigacji. Na szczęście w latach 90. członkowie ogólnej społeczności AI zostali narażeni na działanie robotów reaktywnych. Koncepcja postrzegania inteligentnego systemu lub agenta jako znajdującego się w jego środowisku, w połączeniu z dowodem istnienia, że szczegółowe, podobne do Shakeya reprezentacje świata nie zawsze są konieczne, doprowadziły do nowego stylu planowania. Ta zmiana w planowaniu została nazwana planowaniem reaktywnym. Wielu badaczy, którzy pracowali w tradycyjnej sztucznej inteligencji, zaangażowało się w robotykę. Jeden rodzaj reaktywnego planowania dla robotów, pakiety reakcji reaktywnych Jima Firby'ego (RAP)76, zostały zintegrowane jako warstwa w architekturze 3T25, gdzie aspekty systemu NaT firmy Slack196, architektura ATLANTIS w stylu subsumpcji Gat80 i system RAPs firmy Firby76 zostały połączone w NASA Jet Propulsion Laboratory pod początkowym kierownictwem Davida Millera, a później udoskonalone przez Pete'a Bonasso i Dave'a Kortenkampa w NASA Johnson Space Center. Idea reaktywnego planowania praktycznego zadania robota, opisana poniżej, ma fundamentalne znaczenie. Jak sama nazwa wskazuje, 3T dzieli się na trzy warstwy, jedną wyraźnie reaktywną, jedną wyraźnie deliberatywną i jedną służącą jako interfejs między nimi. 3T był używany głównie w łazikach planetarnych, pojazdach podwodnych i asystentach robotów dla astronautów.

SEKWENSER

UMIEJĘTNOŚCI

Górna warstwa 3T to Planner



Wypełnia obowiązki modułów Mission Planner i Cartographer poprzez wyznaczanie celów i generowanie planów strategicznych. Cele te są przekazywane do warstwy środkowej, zwanej sekwencerem. Sekwencer wykorzystuje technikę planowania reaktywnego zwaną RAPS, aby wybrać zestaw prymitywnych zachowań z biblioteki i opracowuje sieć zadań określającą sekwencję wykonywania zachowań dla konkretnego celu cząstkowego. Sequencer jest odpowiedzialny za funkcje sekwencjonowania i monitorowania wydajności ogólnej architektury hybrydowej. Warstwa Sequencer tworzy instancję zestawu zachowań (nazywanych umiejętnościami) w celu wykonania planu. Te zachowania tworzą dolną warstwę, zwaną kontrolerem lub menedżerem umiejętności. Aby uniknąć pomyłek z konotacjami czysto odruchowych zachowań pozostałych po paradygmacie reaktywności, 3T nie nazywa swoich zachowań „zachowaniami". Nazywa je „umiejętnościami" odróżniania ich od konotacji zachowań spopularyzowanych przez architekturę subsumpcji. Umiejętność jest często zbiorem prymitywnych umiejętności; rzeczywiście, jednym z interesujących aspektów 3T jest jego podstawa jako narzędzie do uczenia się asamblaży.

WYDARZENIA

Potężnym atrybutem niższego poziomu jest to, że umiejętności mają powiązane zdarzenia, które służą jako punkty kontrolne, aby jednoznacznie zweryfikować, czy akcja przyniosła właściwy efekt. Pod pewnymi względami zdarzenia są równoważne wrodzonym mechanizmom uwalniania; obie pozwalają światu być jego najlepszą reprezentacją.

KIEROWNIK UMIEJĘTNOŚCI

Trzy warstwy reprezentują prawdziwą deliberację, reaktywne planowanie i reaktywną kontrolę. Reprezentują również filozofię zorganizowaną według zakresu państwa, a nie zakresu odpowiedzialności. Warstwa Skill Manager składa się z umiejętności, które działają tylko w Teraźniejszości (chociaż z pewnymi dodatkami, aby umożliwić utrzymywanie się niektórych zachowań, gdy bodziec czasowo zanika). Komponenty w warstwie Sequencer operują na informacjach o stanie odzwierciedlających wspomnienia dotyczące Przeszłości, jak również Teraźniejszości. Dlatego sekwencjami zachowań można zarządzać, pamiętając, co robot już zrobił i czy to się udało, czy nie. Daje to dużą niezawodność i wspiera monitorowanie wydajności. Warstwa Planner współpracuje z informacjami o stanie, aby przewidywać przyszłość. Może również wykorzystywać informacje z Przeszłości (co robot zrobił lub próbował) i Teraźniejszości (co robot robi teraz). Aby zaplanować misję, planista musi przewidzieć, jakie będzie środowisko i inne czynniki.

CZĘSTOTLIWOŚĆ AKTUALIZACJI

W praktyce 3T nie organizuje ściśle swoich funkcji w warstwy według stanu (przeszłość, teraźniejszość, przyszłość). Zamiast tego często używa szybkości aktualizacji. Algorytmy, które aktualizują się powoli, są umieszczane w Planerze, podczas gdy algorytmy szybkiej aktualizacji trafiają do warstwy Skill Manager. Wydaje się, że jest to sytuacja, w której pragmatyczne względy obliczeniowe wpłynęły na zasady projektowania; na początku lat 90. zachowania były bardzo szybkie, planowanie reaktywne (zwłaszcza RAP i Universal Plans) było szybkie, a planowanie misji było bardzo powolne. Jednak wiele algorytmów czujników wykorzystujących widzenie komputerowe było również powolnych, więc zostały umieszczone w Planerze, pomimo ich funkcji wykrywania niskiego poziomu. Podczas gdy architektury systemów robotyki zwykle mają pięć wspólnych podsystemów: Planowanie, Kartograf, Nawigacja, Schemat silnika i Percepcja, 3T dodało sekwencer, aby połączyć generowanie planów z wyborem i wdrażaniem określonych schematów oraz aby pomóc w monitorowaniu postępu planu, zwany Agenda. 3T miał również wpływ na grupowanie zachowań w umiejętności lub meta-zachowania oraz umiejętności w sieci zadań.

IDENTYFIKACJA I ODZYSKIWANIE USZKODZEŃ (FDIR)

Poprzednie sekcje przedstawiały rozważania jako klasyczny problem planowania, w jaki sposób robot może osiągnąć określony cel i, jak widać z 3T, jak monitorować postępy. Rozważanie jest również niezbędne do rozwiązywania problemów, zwłaszcza co zrobić, jeśli robot nie osiąga określonego celu. W inżynierii ten typ problemu jest często określany jako identyfikacja i naprawa wykrywania usterek (FDIR). Wstępne prace w FDIR zakładały, że system musiał wykryć usterkę, zidentyfikować lub zdiagnozować na czym polega usterka, a następnie zastosować odpowiednie metody naprawy, przypominające cykl SENSE-PLAN-ACT. Nowsze prace w FDIR są bardziej reaktywne; najpierw wykryj, że występuje usterka, zastosuj ogólne metody odzyskiwania, aby zachować działanie systemu (np. bez awarii), próbując dokładnie zidentyfikować usterkę, a następnie zastosuj specjalistyczne metody odzyskiwania.

NIEPOWODZENIE W GÓRĘ

W robotyce AI istnieją dwa ogólne podejścia do rozwiązywania problemów. Jednym z nich jest „niepowodzenie w górę" lub strategia obsługi wyjątków, której przykładem jest zagnieżdżony kontroler hierarchiczny. W NHC założono, że brak postępu nawigacyjnego zostanie wykryty na najniższym poziomie przez Pilota, a jeśli nie będzie mógł zostać obsłużony przez pilota, to zawiedzie w górę do Nawigatora. Jeśli Nawigator nie może wygenerować objazdu, system przestanie działać w górę do Planera misji. Inny system, architektura SFX, również wykorzystuje to podejście do obsługi wyjątków. Załóżmy, że czujnik się psuje, a schemat percepcyjny nie może wytworzyć percepcji. Schemat nie może sam się zdiagnozować. Dlatego jego niepowodzenie spowodowałoby uruchomienie funkcji

deliberatywnej. Jeśli menedżer schematu nie może znaleźć zastępczego schematu percepcyjnego lub równoważnego zachowania, nie może już spełniać ograniczeń nałożonych przez Mission Planner. Planista misji prawdopodobnie ma inteligencję, aby złagodzić ograniczenia misji lub, w praktyce, zaalarmować ludzkiego przełożonego. Podejścia do obsługi wyjątków dystrybuują monitorowanie i wypychają je do skonkretyzowanych zachowań i funkcji. Tworzy to perspektywę lokalną, która może narazić robota na przeoczenie bardziej subtelnych problemów lub rozwiązań wynikających z większej interakcji podsystemów. Podejścia do obsługi wyjątków mają również tendencję do odkładania identyfikacji problemu na rzecz szybkiego odzyskania, co może być korzyścią lub spowodować jeszcze więcej problemów, jeśli nowy problem wywoła niewłaściwą reakcję.

POZNAWCZA NIEPOWODZENIE

Niepowodzenie w górę lub obsługa wyjątków prowadzi do wyzwania polegającego na stworzeniu świadomej awarii, co oznacza, że analiza awarii wymaga pewnej świadomości kontekstu, aby rozwiązać problem. Nie chodzi o to, że samo poznanie zawiodło. Erann Gat zidentyfikował potrzebę świadomych niepowodzeń, nazywając to problemem „Wesson Oil". W starej reklamie telewizyjnej Wesson Oil matka gotowała kurczaka w oleju, kiedy musiałaby zatrzymać się, by zawieźć syna na ostry dyżur na leczenie złamanej ręki. W optymalnym świecie matka nie tylko wyłączy kuchenkę, ale także usunie kurczaka z oleju. W reklamie matka zaniedbuje usunięcie kurczaka. Ponieważ używa Wesson Oil, kurczak nie jest zepsuty po powrocie, ale tak by było z każdym innym olejem. Istotą problemu Wesson Oil dla robotyki jest to, co się dzieje, gdy zestaw reaktywnych zachowań zostanie wyłączony w środku sekwencji? Nie możemy zakładać, że robotowi się poszczęści i nie zrujnuje misji. W wielu przypadkach rozwiązaniem nie będzie prosta czynność deinstancji zachowań i aktywowania nowych. W sekwencji mogą pojawić się zachowania, które trzeba wykonać (wyłączyć piec) lub dojdzie do katastrofy, a także zachowania, które muszą wystąpić, aby zapobiec jedynie niepożądanym skutkom ubocznym (kurczak staje się tłusty, jeśli nie zostanie usunięty z oleju). Oznacza to, że planista lub sekwencer musi wiedzieć, dlaczego wystąpiła awaria lub dlaczego istnieje potrzeba nagłej zmiany zachowań i jaki jest cel bieżącej sekwencji. Rozważanie na pewno nie jest trywialne!

ROZUMOWANIE MODELOWE

Drugie podejście do rozwiązywania problemów to rozumowanie oparte na modelach, które zostało użyte przez Deep Space One, jak omówiono w części 3. Deep Space One, sonda NASA, wykorzystała globalny model świata i kompletny model zwyczajny tego, jak wszystkie systemy współpracują ze sobą . Kiedy zawór pędnika utknął w pozycji zamkniętej, system rozumowania oparty na modelu odzyskał sprawność, przełączając się na tryb sterowania wtórnego. System rozumowania przeprojektował również obchodzenie zablokowanej kamery i naprawę instrumentu poprzez jego zresetowanie. Zastosowanie modeli uprościło identyfikację i odzyskiwanie, ponieważ system posiada model możliwych wyjaśnień awarii. Może być jednak tysiące wyjaśnień. W związku z tym, aby przyspieszyć odzyskiwanie, wielu wnioskujących na podstawie modelu wykorzystuje również prawdopodobieństwa, aby pomóc w ustaleniu priorytetów wyszukiwania wyjaśnienia. Wadą opartego na modelach FDIR jest to, że wymaga kompletnych modeli zarówno systemu, jak i jego reakcji na środowisko; jak widać w przypadku robotów reaktywnych, przewidywanie otoczenia jest trudne.

Uwagi dotyczące programowania

Warto zauważyć, że użycie logiki predykatów i rekurencji przez Strips faworyzuje języki takie jak Lisp i PROLOG. Języki te zostały opracowane przez naukowców zajmujących się sztuczną inteligencją specjalnie do wyrażania operacji logicznych, ale niekoniecznie mają dobre właściwości sterowania w czasie rzeczywistym, takie jak C lub C++. Jednak w latach 60. dominującym językiem naukowym i inżynieryjnym był FORTRAN IV, który nie wspierał rekurencji. Dlatego badacze robotyki AI często

wybierali mniejsze zło (słaba przydatność do kontroli w porównaniu z brakiem rekurencji) i program w Lispie. Wykorzystanie specjalnych języków sztucznej inteligencji, takich jak Lisp do robotyki, mogło pomóc w podziale podejścia inżynieryjnego i sztucznej inteligencji do robotyki, a także spowolnić napływ pomysłów między obie społeczności. Z pewnością zniechęciło to badaczy spoza AI do angażowania się w robotykę AI. Jednak inżynieryjny powód partycji pozostaje: rzeczy, które operują na symbolach i informacjach globalnych, powinny znajdować się w warstwie deliberatywnej; rzeczy, które działają bezpośrednio na czujnikach i aktuatorach, powinny znajdować się w warstwie reaktywnej. Przetwarzanie symboli faworyzuje języki takie jak Lisp, podczas gdy kontrola faworyzuje języki takie jak C++.

Podsumowanie

Część odpowiada na pytanie Jak myśli robot? ,poprzez przedstawienie przeglądu Warstwy Deliberatywnej. W warstwie deliberatywnej robot angażuje się w sześć z siedmiu obszarów sztucznej inteligencji przedstawionych w części 1. Jak widać w przypadku Shakeya, reprezentacja wiedzy jest podstawowym wyzwaniem. Shakey wybrał wiedzę reprezentacji świata, która wspiera użycie logiki i stylu planowania i rozwiązywania problemów General Problem Solver zawartego w Strips. Prezentacja wiedzy składała się z globalnego modelu świata z powiązaną bazą wiedzy zawierającą mapy a priori lub wiedzę związaną z zadaniem. Strips polega na reprezentowaniu proceduralnej wiedzy o świecie za pomocą pewnych informacji przestrzennych (np. CONNECTS). Ponadto metody planowania i rozwiązywania problemów zakładały, że robot był w stanie wykonać fizyczne uziemienie symboli w celu zbudowania tych reprezentacji wiedzy i połączenia wykonania planu do rzeczywistych działań w świecie i zakotwiczenia zmian w tym, co postrzegane. Oparcie się na globalnym modelu świata wprowadza problem ramowy. Fizyczne uziemienie symboli pozostaje otwartym wyzwaniem w robotyce i jest badane przez społeczności wizji i języka naturalnego. Płaskie lub „wszystko-w-jednym" modele świata są w praktyce rzadkością, a zamiast tego powszechny jest jakiś wariant lokalnych przestrzeni percepcyjnych lub hierarchii abstrakcji. Przestrzenne reprezentacje świata są popularnie nazywane „modelami świata", ale są one zubożone i często mają minimalne zastosowanie w zaawansowanych rozważaniach. Służą one jedynie jako „powłoki" i nie zawierają proceduralnej wiedzy o tym, jak robot sytuuje się w świecie ani jakie są symboliczne etykiety przedmiotów. Powłoki te mogą służyć jako wirtualne czujniki zachowań i nawigacji. Algorytmy planowania i rozwiązywania problemów były głównym wysiłkiem w tworzeniu możliwości deliberatywnych dla robotów. Planowanie często ogranicza się do planowania ścieżek nawigacyjnych, a wiele systemów naśladuje styl zagnieżdżonego kontrolera hierarchicznego, czyli Mission Planner, Navigator i Pilot, dzieląc obowiązki i zakres. Planowanie i rozwiązywanie problemów często intensywnie wykorzystują algorytmy wyszukiwania, gdzie Strips był zasadniczo wyszukiwarką rekurencyjną, opartą na wyraźnych warunkach wstępnych dla operacji kandydujących. W praktyce wnioskowanie jest używane rzadziej, prawdopodobnie ze względu na ograniczone zastosowania robotów, ale ważne jest wykrywanie, identyfikacja i usuwanie usterek. Robot może nie mieć czasu na jednoznaczne potwierdzenie przyczyny problemu i zamiast tego musi zgadywać wystarczająco blisko, aby szybko wygenerować strategię odzyskiwania, aby zapobiec katastrofie. Uczenie się to jedyny obszar w AI, który nie pojawia się często w pracy nad deliberacją. Jest to najprawdopodobniej spowodowane wszechobecnością uczenia się, która rozciąga się na cały cykl SENSE-PLAN-ACT. W kolejnych rozdziałach skupimy się na rozważaniach dotyczących nawigacji, badaniu zakresu przestrzennych reprezentacji wiedzy i algorytmach planowania ruchu robota.