Omówienie systemu Soar

- Architektura kognitywna
  - Struktura ”pętli głównej” i moduły agenta
  - Mechanizmy uczenia: chunking, RL
- Model obliczeń
  - System regułowy, mechanizm wyboru operatora
- Język programowania i system monitorowania / symulacji
  - IDE, Debugger, CLI

Reguły vs Produkcje

- Jeśli uznamy Soar za system regułowy jak poniżej:
  - *system regułowy* to system w którym wiedza / program jest reprezentowany w postaci par Przesłanki→Wnioski nasycających zbiór konsekwencji (pamięć roboczą), ”w przybliżeniu” zgodnie z pewną logiką,
  - a *system produkcji* to system regułowy działający na parach Warunki→Akcje tzn. interpretujący wnioski jako nieodwracalne działania i dyskryminujący ”pechowe” dopasowane reguły
- To Soar **nie jest** systemem produkcji
  - Ale zachowuje terminologię, np. *production memory*

”Logika” Soara

- Niemonotoniczność: negacja w warunkach, wnioski mogą odrzucać wcześniej stwierdzone fakty, percepty znikają
- Dwie modalności: konsekwencja *I-support* oraz rezultat *O-support*
  - Rezultat zostaje wycofany tylko jeśli jawnie odrzucony
  - Konsekwencja zostaje wycofana również jeśli jej przesłanka wycofana lub odrzucona
  - Paradoks kłamcy: konsekwencja ”P → nie P” lub ”nie P → P”

    np. `{sp (state <s> ^skasuj mnie) → (<s> ^skasuj mnie -)}`
- Wszystkie dopasowania reguł odpalają naraz (nawet kilka dopasowań dla jednej reguły)

Fakty, pamięć robocza

- Fakty to trójki `(v1 ^relacja v2)` lub `(v1 ^relacja stała)` gdzie `v1, v2, o1, ...` to obiekty spoza języka
- Budują WM: graf skierowany z krawędziami oraz liśćmi etykietowanymi przez wspólne stałe z języka
- Można patrzeć jak na obiekty / struktury atrybutowe
  - Ale zawsze mamy zbiór wartości atrybutu, etykiety krawędzi mogą się powtarzać
- Kawałki WM nieosiągalne z wyróżnionego obiektu s1 `(s1 ^superstate nil)` są odśmiecane
- (Porównania operatorów nie są dostępne w WM)

Reguły

- Zmienne z warunku są <uniwersalne>, odnoszą się do obiektów/stałych w WM, a tylko z wniosku są <egzystencjalne>, generują nowy obiekt w WM
- Każda reguła musi dopasowywać stan s1 lub podstan – obiekt, z którego s1 jest osiągalny po atrybutach `^superstate` – pod <s> z `(state <s> ...)`
- Zmienna dopasowana w pozycji atrybutu lub liścia grafu przechwytuje stałą, a w pozycji węzła wewnętrznego przechwytuje obiekt (ten węzeł), obiekty i stałe są rozłączne

Pętla główna

- Pojawiają się percepty `(<s> ^io.input-link ...)`
- Pętla wyciągania konsekwencji *elaboration*:
  - Wycofaj fakty, dopasuj reguły, odpal naraz reguły
- System na podstawie preferencji wybiera jeden z zaproponowanych operatorów `(<s> ^operator ...)`
  - Jeśli coś nie tak, podprocedura (*substate)* impasu
- Pętla wyciągania (też) rezultatów *application*
  - Rezultaty to wnioski reguł które w przesłankach mają wybrany operator; mogą pisać na `^io.output-link`

Prosta pętla główna w detalach

- Zalecany jest następujący schemat:
  - *1. Operator proposal*: reguły dodają propozycje wszystkich możliwych operatorów (akcji)
    - `(<s> ^operator <o> +)` to atrybut inny niż `^operator`
  - *2. Operator comparison*: reguły wyboru operatora dodają porównania zaproponowanych operatorów
  - Powiedzmy, że systemowi udało się wybrać operator `(<s> ^operator <o>)`
  - *3. Operator application*: reguły dopasowywują `(<s> ^operator <o>)` i generują rezultaty O-support
  - W tle *State elaboration*: dodaje struktury pomocnicze

Podprocedury: podstany aka. podcele

- Typy impasów: *tie* – kilka równie dobrych operatorów bez =; *conflict* – przeczące sobie porównania; *state no-change* – brak propozycji operatorów; *operator no-change* – brak aplikacji operatora
- W podstanie stosujemy ”lokalne” operatory aż uda się nam zmienić sytuację w nadstanie, wtedy podstan jest kasowany i przeżywają rzeczy podczepione bezpośrednio do nadstanu
- Podstan będzie skasowany też gdy sytuacja w nadstanie zmieni się z innego powodu, np. input

Chunking

- SLAJD DO ZROBIENIA

Porady: ”reguła pięciu”

- Ogranicz liczbę warunków i akcji w każdej regule do co najwyżej pięciu
- Licząc jako warunki te dopasowania które coś testują (np. do stałej), a nie tylko rozbierają strukturę
- Wyjątek: reguły inicjalizacji mogą mieć dużo akcji
- Przykład:

```
sp {propose*recharge*health-BAD
   (state <s> ^name tanksoar
              ^io.input-link <il>)
   (<il> ^radar.tank.distance > 0
         ^health < 300
         -^smell.distance < 4
         -^sound
         -^incoming)
-->
   (<s> ^operator <o> +)
   (<o> ^name recharge-health)
}
```

```
sp {elaborate*in-danger
   (state <s> ^name tanksoar
              ^io.input-link <il>)
   (<il> ^radar.tank.distance > 0
         -^smell.distance < 4
         -^sound
         -^incoming)
-->
   (<s> ^in-danger yes)
}
sp {propose*recharge*health
   (state <s> ^name tanksoar
              ^in-danger yes
              ^io.input-link <il>)
   (<il> ^health < 300)
-->
   (<s> ^operator <o> +)
   (<o> ^name recharge-health)
}
```

Porady

- Sprawdzaj tylko warunki potrzebne do sprawdzenia
- Nie obliczaj zawczasu (np. na etapie proponowania jeśli wystarczy na etapie aplikacji)
- Proponuj wszystkie możliwe operatory, selekcją martw się w regułach wybierania
- Jedna akcja na operator, nie uogólniaj za bardzo znaczenia operatorów
- Nie używaj operatorów do ”programowania imperatywnego” i ”ręcznego zarządzania pamięcią”
  - Jeśli coś I-supported znika, to ma do tego powód!

Porady

- Unikaj domyślnych założeń o stanie: precyzyjne reguły
- Staraj się odnosić do stanów poprzez ich nazwy
  `(state <s> ^name mystate)`
- Używaj VisualSoar, utrzymuj aktualną *data map*
- Działaj zgodnie z wytycznymi architektury Soar (podział na proponowanie, wybieranie, zastosowywanie operatorów)

Soar-RL

- Nagroda RL zbierana jest z wartości `(<s> ^reward-link.reward.value *)` dla aktualnego stanu <s>
- Jeśli środowisko dostarcza nagrody, agent powinien ją skopiować z `^io.input-link` na `^reward-link...`
- O-supported nagrody będą doliczane w każdym cyklu aż zostaną ręcznie usunięte; lepiej I-supported
- Q-learning z abstrakcją stanów i akcji: w dziedzinie Q są tzw. RL-reguły wyboru operatora
  - musi mieć postać `{... → (<s> ^operator <o> = 0)}` – zamiast 0 dowolna stała liczbowa: wartość początkowa

Wybór operatora

- Gdy kilka reguł zaproponowało wybrany operator, Q-wartość jest obliczana jako suma (domyślnie) lub średnia z wartości reguł numeric-indifferent-mode --avg
- Podobnie jak ze zwykłym =, symboliczne preferencje <, > mają pierwszeństwo
- Aktywacja RL zmienia metodę sekelcji z softmax – proporcjonalnie do wartości, na epsilongreedy
- Możemy aktywować liniowe lub wykładnicze malenie epsilona (przełącz: indifferent-selection --autoreduce)
  - indifferent-selection –reduction-policy epsilon exponential
    indifferent-selection –reduction-rate epsilon exponential 0.9

Abstrakcja, tworzenie RL-reguł

- Czym ogólniejsze reguły, tym mniej dokładne ale szybsze uczenie
- Ogólna komenda gp Soara generująca zestaw reguł:
  - gp {... [a b c] ... [d e] ...} ==> sp {... a ... d}, sp {... b ... d}, ...
- Wzorcowe RL-reguły :template
  - Każdemu zestawowi dopasowanych stałych odpowiada osobna reguła (tworzona ”on demand”)
  - Tutaj wartość operatora może być zmienną, wartość pocz. wygenerowanej reguły będzie dopasowaną liczbą
  - Sprawdzanie unikalności reguły spowalnia działanie

TD, aktualizacja wartości

- Po podjęciu nowej decyzji aktualizowana jest wartość poprzedniej decyzji
- TD: learning-rate*(nagrody - suma/średnia wartości reguł uaktualnianych)
- Jeśli sumowanie, to podziel TD pomiędzy reguły
- Nagrody: nagroda + discount-rate*suma/średnia wartości nowych reguł
- Schemat eksploracji epsilongreedy jest naiwny, możesz go wyłączyć i zaprogramować własny modyfikując wartości na ^reward-link (np. by umożliwić planowanie eksploracyjne)

Kroki nie-RL

- Nie musimy w każdym kroku wybierać operator przez RL-reguły
  - Ważne, ponieważ reprezentują one pary stan-akcja, łatwo zgubić informację o stanie
- Włączanie: rl –set –temporal-extension on
- W ramach TD paruje kroki rozseparowane krokami nie-RL, odpowiednio przecenia wartość TD

Eligibility traces

- Uaktualniać można w jednym kroku wartości nie tylko bezpośrednio poprzedzających reguł, ale też wcześniejszych: cenne, gdy nagroda pojawia się rzadko
- Watkins Q(λ) różni się od Sarsa(λ) tym, że po kroku eksploracyjnym wszystkie ślady są czyszczone
- rl –set –eligibility-trace-decay-rate 0.5

Sarsa(λ)

- Initialize reward r = 0
- Repeat (for each Soar cycle):
- Select operator o
- Repeat (for each o in e):
- if (e[o] < [eligibility trace tolerance])
- remove e[o]
- increment = 1/n
- if (e[o])
- e[o]+=increment
- else
- e[o]=increment
- Repeat (for each o in e):
- Q[o]+=(learning rate) (update) e[o]
- e[o]*=(discount rate) (eligibility trace decay rate)
- Apply operator o
- Observe reward r

Uczenie się hierarchiczne

- Soar-RL jest w naturalny sposób hierarchiczny, podstany są osobnymi problemami RL, każdy stan ma swój ^reward-link
- Przy *operator no-change*, wartość w nadstanie jest akumulowana z przecenianiem do czasu, gdy ”sterowanie” powróci do nadstanu
- Przy innych impasach, nadstan zachowuje się jakby nie było dziury (tylko jednokrotnie dyskontuje)
- Przy aktualizacji ostatniej reguły z podstanu nie ma kolejnej reguły, więc ”nagrody=ostatnia nagroda”

Zadanie domowe (niepunktowane!)

- Uruchom wersję RL agenta water-jug
- Zrób kilka przebiegów z wyłączonym RL
- Następnie włącz RL, zrób kilka przebiegów oglądając wartości reguł
- Zresetuj Soara, włącz RL z eligibility-trace-decay-rate odpowiednio wysokim i powtórz eksperyment
- Porównaj ilości kroków do osiągnięcia celu w różnych wariantach

Literatura

- *The Soar User's Manual Version 9.0* (John E. Laird and Clare Bates Congdon)
- *Soar Design Dogma* (Andrew Nuxoll and John Laird)
- *Soar-RL Manual Version 1.0* (Nate Derbinsky, Nick Gorski, John Laird, Bob Marinier, Yongjia Wang)
- *Part VII: Soar-RL - Reinforcement Learning* (Soar Tutorial Authors)