Wprowadzenie do ACT-R

Konrad Miazga

Poznan University of Technology

Poznań, 2018

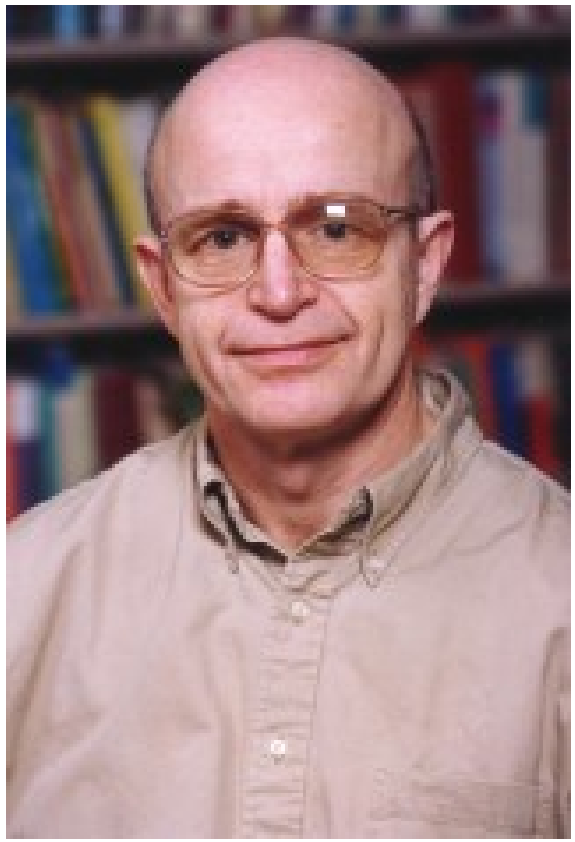
Materiały

- tutorial ACT-R (w folderze programu; osiem jednostek, 230 stron A4)
- te wykłady (materiał obejmujący pierwsze trzy / cztery jednostki tutorialu)

Informacje wstępne

ACT-R (Adaptive Control of Thought—Rational) jest architekturą kognitywną opracowaną przez Johna Roberta Andersona (Carnegie Mellon University).

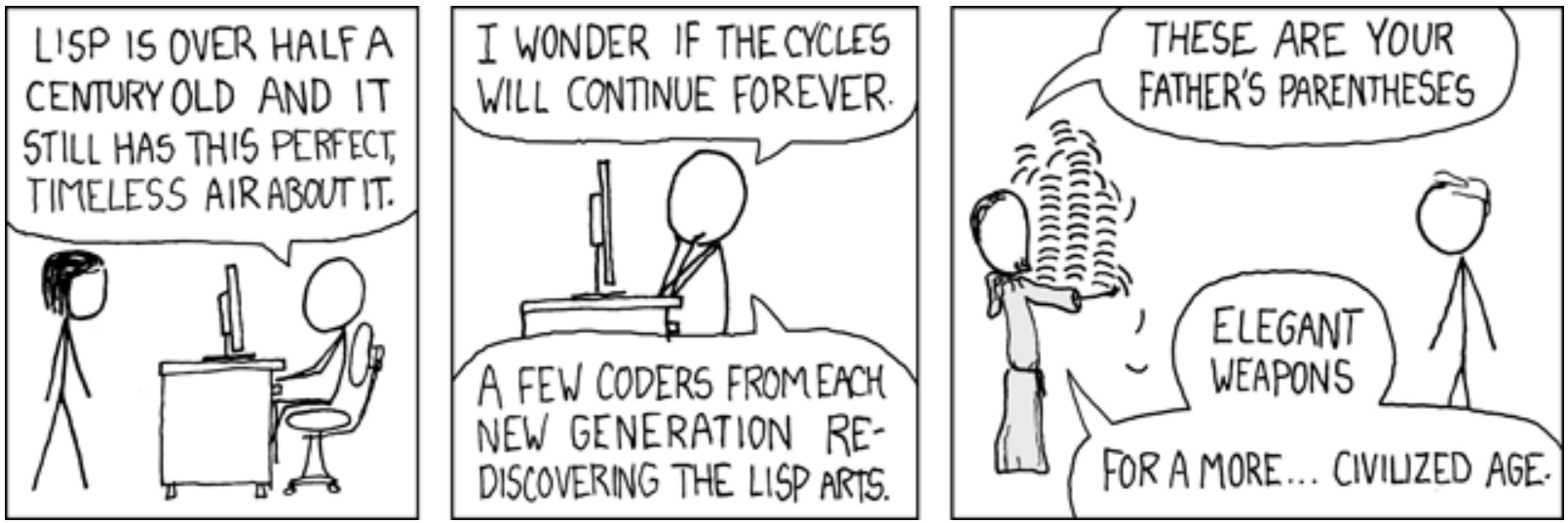
- Początki programu sięgają lat siedemdziesiątych.
- ACT-R wpisuje się w nurt teorii symbolicznych (w przeciwieństwie do teorii konekcjonistycznych których przedstawicielem jest np. Nengo), aczkolwiek niektóre jego elementy są inspirowane konekcjonizmem.



Rysunek: John Robert Anderson

Informacje wstępne

- ACT-R opiera się na języku programowania Lisp. Znajomość języka jest przydatna do definiowania eksperymentów, ale same modele definiowane są za pomocą ograniczonego zbioru poleceń Lispa (swego rodzaju język w języku).



Rysunek: https://xkcd.com/297/

Informacje wstępne

- Używany w ponad 1000 różnych publikacjach naukowych.
- We współczesnej wersji posiada wiele modułów percepcyjnych i motorycznych.
- We współczesnej wersji pozwala przewidywać wzory pobudzeń w mózgu podczas wykonywania modelowanych zadań.

ARCHITEKTURA

Rodzaje wiedzy

W środowisku ACT-R, wiedza dzieli się na dwa rodzaje:

- deklaratywna (CO, np. „bitwa pod Grunwaldem miała miejsce w 1410 roku”),
- proceduralna (JAK, reguły produkcji, np. „jeżeli dodajesz 1 i 2 to jako wynik przyjmij 3”).

Wiedza deklaratywna

Wiedza deklaratywna jest w ACT-R reprezentowana za pomocą *chunków* (ang. chunk – kawałek).

- Każdy chunk posiada nazwę oraz atrybuty (sloty).
  - Nazwa chunka nie jest istotna, służy ona do prostszego odwoływania się do różnych chunków.
- Sloty składają się z nazwy oraz wartości.
  - Wartości slotów interpretowane są jako symbole, a więc nie można przeprowadzać na nich np. operacji matematycznych.

Wiedza deklaratywna

Przykład - chunk

Fact3+4

addend1 three

addend2 four

result seven

Fact3+4

addend1 three

addend2 four

result seven

chunk name, slots, slot name, value

Wiedza proceduralna

Wiedza proceduralna jest w ACT-R reprezentowana za pomocą reguł produkcji (IF ... THEN ...).

Przykład

IF

the goal is to add two digits d1 and d2 in a column and d1 + d2 = d3

THEN

create a goal to write d3 in the column

W regułach ACT-R'a, w prawej części produkcji (THEN) zazwyczaj będą występować zapytania do reszty systemu (prośba o dane, zlecenie podjęcia akcji itp.)

Moduły

Architektura ACT-R opiera się na modułach.
Moduły to niezależne jednostki odpowiadające za pewne konkretne funkcje kognitywne. W uproszczeniu można przyjąć, że każdy moduł odpowiada innej części mózgu, ale w praktyce niektóre z modułów będą służyły jedynie uproszczeniu modelów, np. moduł do generowania liczb pseudo-losowych.

Bufory

Każdy z modułów może posiadać jeden bądź więcej buforów.

- Każdy bufor posiada swoją nazwę.
- Bufory służą za interfejs modułów - to poprzez nie moduły mogą się ze sobą komunikować.
- Moduł może „dzielić się” z resztą systemu swoją wiedzą poprzez wstawienie do bufora jednego ze swoich chunków. Będzie on widoczny dla wszystkich modułów.
- W każdym buforze może znajdować się jednocześnie tylko jeden chunk!
- Moduły mogą kierować do buforów zapytanie (np. prośbę o wstawienie do bufora pewnych chunków).
- W momencie odebrania przez bufor zapytania, bufor jest czyszczony (usuwany jest znajdujący się w nim chunk).

Moduły

Na dzisiejszych zajęciach będziemy korzystać jedynie z trzech, najbardziej podstawowych modułów:

- Moduł celu (Goal)
- Moduł deklaratywny (Declarative)
- Moduł proceduralny (Procedural)

Opóźnienia

Opóźnienia

Aby poprawnie modelować opóźnienia związane z procesami kognitywnymi, niektóre z modelowanych akcji (np. dostęp do pamięci) będą zajmowały więcej czasu niż jest to potrzebne komputerowi do przeprowadzenia obliczeń. Czasy podawane w logach produkowanych przez ACT-R'a związane są właśnie z tymi symulowanymi opóźnieniami!

Moduł celu (Goal)

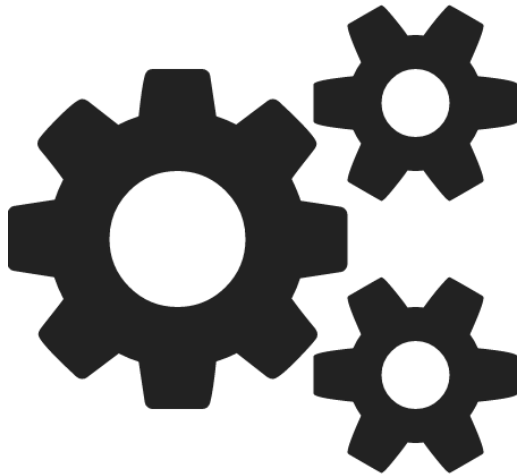
- Posiada bufor Goal, w którym znajduje się chunk opisujący aktualny cel.
- Jedyne zapytania które przyjmuje bufor Goal, to prośby o zmianę chunku celu na inny, dołączony do zapytania (z efektem natychmiastowym).

Moduł deklaratywny (Declarative)

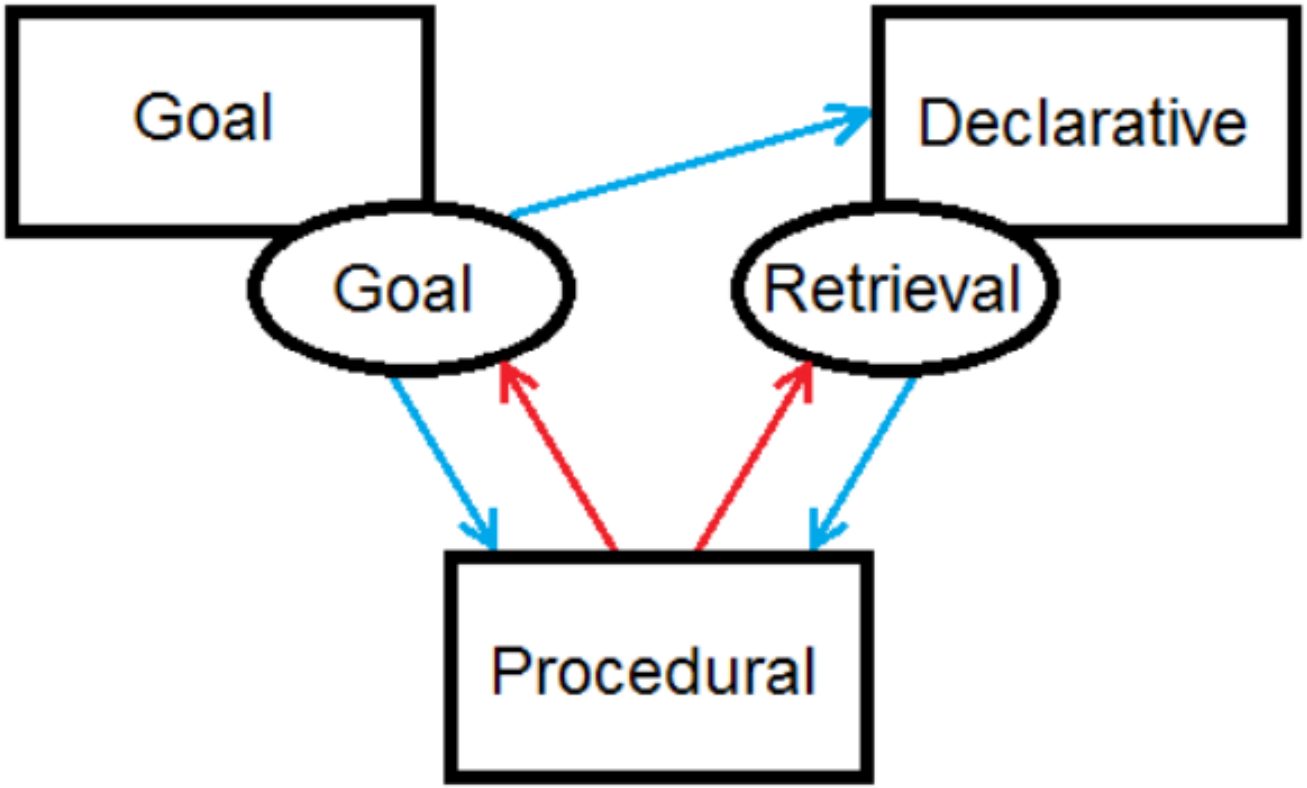
- Posiada bufor Retrieval.
- Do bufora Retrieval można kierować zapytania o udostępnienie chunku, który najlepiej pasuje do zadanego wzorca.
- Moduł deklaratywny wyszukuje najlepiej pasujący chunk który zostaje wstawiony do buforu z pewnym opóźnieniem (na razie załóżmy, iż jest ono stałe i wynosi 50ms).
- Moduł deklaratywny gromadzi wewnątrz wszystkie predefiniowane chunki oraz te które były w czasie działania modelu widoczne w innych buforach (są one zapisywane w momencie czyszczenia bufora).

Moduł proceduralny (Procedural)

- Nie posiada żadnego bufora (a więc nie można kierować do niego zapytań).
- Gromadzi wiedzę proceduralną – wcześniej wspominane reguły produkcji.
- Aktywnie obserwuje bufory innych modułów i gdy znajdujące się w nich chunki spełniają wymagania którejś z reguł (lewa strona produkcji) wykonuje prawą stronę produkcji – zazwyczaj zapytania do innych buforów.
- Wykonanie prawej strony reguły produkcji następuje z pewnym opóźnieniem (na razie załóżmy, iż jest ono stałe i wynosi 50ms).
- Tylko jedna reguła produkcji może być wykonywana na raz.



Architektura ACT-R



Rysunek: Sprawdzanie chunków, Wysyłanie zapytań

MODELOWANIE

Typy chunków

ACT-R pozwala na zdefiniowanie typów chunków (chunk-type). Nie trzeba z nich korzystać, aczkolwiek warto ponieważ:

- ułatwiają one zrozumienie zdefiniowanego modelu,
- w razie niezgodności chunka ze zdefiniowanym dla niego typem, ACT-R ostrzeże nas o możliwym błędzie w modelu.

Definicja typu chunka

(chunk-type type-name slot-name-1 slot-name-2 . . . slot-name-n)

np.
(chunk-type action verb agent object)
(chunk-type addition-fact addend1 addend2 sum)

Tworzenie chunków

Aby dodać zdefiniować początkową wiedzę (deklaratywną) agenta, należy użyć metody add-dm („**add** to **d**eclarative **m**emory”):

Definiowanie wiedzy deklaratywnej

```
(add-dm
(b ISA count-order first 1 second 2)
(c ISA count-order first 2 second 3)
(d ISA count-order first 3 second 4)
(e ISA count-order first 4 second 5)
(f ISA count-order first 5 second 6)
(first-goal ISA count-from start 2 end 4))
```

Tworzenie chunków

Aby dodać zdefiniować początkową wiedzę (deklaratywną) agenta, należy użyć metody add-dm:

Definiowanie chunka

```
( b ISA count-order first 1 second 2 )
```

- b – nazwa chunka
- ISA count-order – definicja typu chunka (opcjonalne ale warto). Jeżeli nie podamy wartości któregoś ze slotów definiowanych przez typ, ACT-R nie zareaguje. Jeżeli podamy wartość dla slotu który nie był zdefiniowany, ACT-R wygeneruje ostrzeżenie.
- first 1 – lista slotów i ich wartości.

Tworzenie reguł produkcji

Definiowanie reguł

(p *nazwa* *"opcjonalna dokumentacja"*
testy buforów
==>
zapytania i zmiany buforów
)

Tworzenie reguł produkcji

Definiowanie reguł – przykład

```
(P counting-example
"example production"
=goal>
ISA count
state incrementing
number =num1
=retrieval>
first =num1
second =num2
==>
=goal>
ISA count
number =num2
+retrieval>
ISA count-order
first =num2
)
```

Tworzenie reguł produkcji

Definiowanie testów (lewa strona produkcji)

1) =goal>	1) sprawdź chunk w buforze Goal
2) ISA count	2) załóż, iż jest typu *count*
3) state incrementing	3) czy slot *state* ma wartość *incrementing*?
4) number =num1	4) zapamiętaj wartość slotu *number* jako *num1*

5) =retrieval>	5) sprawdź chunk w buforze Retrieval
6) first =num1	6) czy slot *first* ma wartość zapamiętaną jako *num1*?
7) second =num2	7) zapamiętaj wartość slotu *second* jako *num2*

Jeżeli odpowiedzi na pytania 3) i 6) były twierdzące, oraz możliwe było przeprowadzenie wszystkich operacji (tj. wszystkie odpytywane chunki i sloty istniały), to warunek zostaje spełniony: możemy przejść do prawej strony produkcji.

Tworzenie reguł produkcji

Definiowanie akcji (prawa strona produkcji)

1) =goal>	1) zmodyfikuj chunk w buforze Goal
2) ISA count	2) załóż, iż jest typu *count*
3) number =num2	3) ustaw wartość slotu *number* na wartość zapamiętaną jako *num2*

4) +retrieval>	4) wyślij zapytanie do buforu Retrieval o udostępnienie chunka
5) ISA count-order	5) załóż, iż szukany chunk jest typu *count-order*
6) first =num2	6) wymagaj aby w szukanym chunku slot *first* przyjmował wartość zapamiętaną jako *num2*

PROSTY MODEL LICZENIA

Prosty model liczenia

Zadanie 1

Przeanalizuj kod definiujący prosty model liczący od jednej do drugiej liczby (domyślnie od 2 do 4). Zwróć uwagę na zdefiniowane chunki (wiedza deklaratywna) oraz reguły produkcji (wiedza proceduralna). Na kolejnych slajdach znajdziesz komentarze do nietypowych fragmentów kodu.

Plik z modelem:
...\ACT-R Standalone Environment\tutorial\unit1\count.lisp

#1

```
(clear-all)
```

Czyszczenie środowiska po wykonaniach poprzednich modeli.

#3

```
(define-model count
```

Główne polecenie, definiujące cały model. Pierwszym argumentem jest nazwa modelu, następnie podajemy cały kod modelu.

#5

```
(sgp :esc t :lf .05 :trace-detail high)
```

Parametry modelu, na razie się nie musimy tym przejmować. *:trace-detail high* zapewnia nam pełną szczegółowość logów które będą wypisywane w konsolce.

#19

(goal-focus first-goal)

Ustalamy tu początkowy cel naszego modelu - jest on definiowany przez chunk *first-goal* (wcześniej dodany pod tą nazwą do pamięci deklaratywnej).

#25

count nil

Nil to specjalna wartość w lispie (a więc i w ACT-R) – odpowiednik **null** np. w Javie. Linijka ta oznacza, że testujemy czy chunk nie posiada zdefiniowanej wartości dla slotu *count*.

#39

- end =num1

Minus poprzedzający test wartości slotu oznacza negację, tj. test zostanie zaliczony gdy wartość slotu *end* będzie **różna** od wartości zapamietanej jako *num1*.

#51

!output! (=num1)

Polecenie to poskutkuje wypisaniem do konsoli wartości przechowywanej w zmiennej *num1*.

Prosty model liczenia

Zadanie 2

Załaduj do ACT-R model *count* i uruchom go za pomocą polecenia:
(run 1)
W konsoli powinny się pokazać logi przebiegu wykonania modelu. Spróbuj prześledzić zachowanie modelu. Czy przewidywane przez ciebie zachowanie modelu jest zgodne z logami?

Plik z modelem:
...\ACT-R Standalone Environment\tutorial\unit1\count.lisp

Analiza logów

Przykładowa linia logu

0.000 GOAL SET-BUFFER-CHUNK GOAL FIRST-GOAL REQUESTED NIL

- 0.000 – czas zdarzenia (symulowany),
- GOAL – moduł inicjujący zdarzenie,
- SET-BUFFER-CHUNK – zdarzenie, w tym wypadku akcja wstawienia chunka do bufora,
- GOAL – bufor z którym związane jest zdarzenie,
- FIRST-GOAL – nazwa użytego chunka,
- REQUESTED NIL – zdarzenie nie zostało zlecone przez żadną regułę produkcji.

Prosty model liczenia

Zadanie 3

Zedytuj model liczenia z zadania 2:

a) rozszerz model do liczb z zakresu [0; 10],

b) spraw aby model liczył wstecz (od pierwszej podanej liczby w dół do drugiej),

c) spraw aby model wypisywał jedynie liczby parzyste. W tym celu dodaj do pamieci deklaratywnej dodatkowe potrzebne fakty, rozszerz definicję celu o pole trzymające informację o fazie przetwarzania (liczenie / sprawdzanie parzystości) oraz dodaj reguły wypisujące liczbę na wyjście jedynie dla liczb parzystych.

Plik z modelem:
...\ACT-R Standalone Environment\tutorial\unit1\count.lisp

Prosty model liczenia

Zadanie 4

Zaimplementuj model dodający dwie liczby:

a) w zakresie [0; 10],

b) w zakresie [-5; 5].