Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych

Reprezentacja wiedzy

Programy działań z efektami domyślnymi

Autorzy:

Dragan Łukasz
Flis Mateusz
Fusiara Marcin
Izert Piotr
Pielat Mateusz
Rząd Przemysław
Siry Roman
Waszkiewicz Piotr
Zawadzka Anna

19 marca 2016

1 Opis zadania

Zadaniem projektu jest opracowanie i zaimplementowanie języka akcji dla specyfikacji podanej klasy systemów dynamicznych oraz odpowiadający mu język kwerend.

System dynamiczny spełnia podane założenia:

1. Prawo inercji
2. Niedeterminizm i sekwencyjność działań
3. Pełna informacja o wszystkich akcjach i wszystkich ich skutkach bezpośrednich
4. Z każdą akcją związany jest:
   (a) Warunek początkowy (ew. true)
   (b) Efekt akcji
   (c) Jej wykonawca
5. Skutki akcji:
   (a) Pewne (zawsze występują po zakończeniu akcji)
   (b) Domyślne (preferowane. Zachodzą po zakończeniu akcji, o ile nie jest wiadomym, że nie występują)
6. Efekty akcji zależą od jej stanu, w którym akcja się zaczyna i wykonawcy tej akcji
7. W pewnych stanach akcje mogą być niewykonalne przez pewnych (wszystkich) wykonawców

Programem działań nazywać będziemy ciąg $((A_1, w_1), (A_2, w_2), \ldots, (A_n, w_n))$, gdzie A_i jest akcją, zaś w_i jej wykonawcą lub ϵ (ktokolwiek).

Język kwerend zapewnia uzyskanie odpowiedzi na następujące pytania:

1. Czy podany program działań jest wykonywalny zawsze/kiedykolwiek?
2. Czy wykonanie podanego programu działań z dowolnego stanu spełniającego warunek π prowadzi zawsze/kiedykolwiek/na ogół do stanu spełniającego warunek celu γ ?
3. Czy z dowolnego stanu spełniającego warunek π cel γ jest osiągalny zawsze/kiedykolwiek/na ogół?
4. Czy wskazany wykonawca jest zaangażowany w realizację programu zawsze/kiedykolwiek?

2 Język akcji Ω

2.1 Definicja języka

Ω jest rodziną języków, w której każdy język $\mathcal{L}$ określony jest nad sygnaturą

$$\Upsilon = (F, A, W)$$

gdzie:

- F - niepusty zbiór zmiennych (fluenty)
- A - niepusty zbiór akcji
- W - niepusty zbiór wykonawców (aktorów), przy czym $\epsilon \in W$, gdzie ϵ oznacza kogokolwiek

2.2 Syntaktyka języka

W języku Ω występują następnujące typy zdań:

- `initially` α
  formuła α zachodzi w stanie początkowym
- α `after` $(A_1, w_1), ..., (A_n, w_n)$
  formuła α zachodzi po wykonaniu sekwencji $(A_1, w_1), ..., (A_n, w_n)$, gdzie A_i jest akcją, zaś w_i jej wykonawcą
- (A, w) `causes` α
  skutkiem wykonania akcji A przez wykonawcę w jest stan, w którym spełniona jest formuła α
- (A, w) `causes` α `if` π
  skutkiem wykonania akcji A przez wykonawcę w w stanie spełniającym warunek π jest stan, w którym spełniona jest formuła α
- `observable` α `after` $(A_1, w_1), ..., (A_n, w_n)$
  po wykonaniu sekwencji $(A_1, w_1), ..., (A_n, w_n)$, gdzie A_i jest akcją, zaś w_i jej wykonawcą, w stanie początkowym może (ale nie musi) zachodzić formuła α
- `impossible` (A, w) `if` π
  niemożliwe jest wykonanie akcji A przez wykonawcę w w stanie spełniającym warunek π
- (A, w) `releases` f `if` π
  wykonanie akcji A przez wykonawcę w w stanie spełniającym warunek π może (ale nie musi) zmienić wartość zmiennej f
- (A, w) `typically causes` α `if` π
  skutkiem wykonania akcji A przez wykonawcę w w stanie spełniającym warunek π na ogół jest stan, w którym spełniona jest formuła α

- `always` α
  formuła α jest spełniona w każdym stanie

gdzie α jest dowolną kombinacją zmiennych (fluentów):

$$\alpha = f|\alpha|\neg\alpha|\alpha_1 \wedge \alpha_2|\alpha_1 \vee \alpha_2|\alpha_1 \rightarrow \alpha_2|\alpha_1 \leftrightarrow \alpha_2$$

2.2.1 Przykład 1

Farmer Bill i indyk Fred pracują razem nad pewnym projektem programistycznym. Zakładamy, że początkowo kod jest czytelny i kompilowalny. Bill to niedoświadczony programista, więc gdy dopisze on jakiś fragment cały kod przestaje być czytelny, a nierzadko przestaje się też kompilować. Indyk Fred jest z kolei weteranem branży IT, więc jego kod kompiluje się zawsze (gdy pracuje on z czytelnym kodem) lub prawie zawsze (gdy kod jest nieczytelny). W razie potrzeby Fred refaktoryzuje cały kod, dzięki czemu poprawia się jego czytelność. Bill i Fred zgodnie ustalili, że nie będą dopisywać nowych fragmentów kodu jeżeli dotychczasowy się nie kompiluje. W takim wypadku któryś z nich musi go najpierw zdebugować (co potrafi każdy programista mając odpowiednio dużo czasu).

```
initially compiles ∧ cleanCode
(Code, Bill) causes ¬cleanCode
(Code, Bill) releases compiles if compiles
(Code, Fred) causes compiles if cleanCode
(Code, Fred) typically causes compiles if ¬cleanCode
(Refactor, Fred) causes cleanCode
(Debug, ϵ) causes compiles
impossible (Code, ϵ) if ¬compiles
```

2.3 Semantyka języka

2.3.1 Stan

Stanem będziemy nazywać dowolną fukcję $\sigma : F \rightarrow \{1, 0\}$, która przypisuje zmiennym wartości logiczne. Jeśli $\sigma(f) = 1$, to znaczy, że zmienna f zachodzi w stanie σ. Funkcję tę można rozszerzyć na zbiór wszystkich formuł nad zbiorem zmiennych F wedug zasad obowiązujących w klasycznej logice zdań.

2.3.2 Struktura

Strukturą nazywamy układ $S = (\Sigma, \sigma_0, ResAb, ResN)$, gdzie:

- Σ - zbiór stanów
- $\sigma_0 \in \Sigma$ - stan początkowy
- $ResAb, ResN : A \times W \times \Sigma \rightarrow 2^{\Sigma}$ są funkcjami przejść. $ResAb$ jest funkcją przejść nietypowych, $ResN$ jest funkcją przejść typowych oraz $ResAb \cap ResN = \emptyset$

2.3.3 Model dziedziny

W celu zdefiniowania pojęcia modelu dziedziny wprowadzone zostaną następujące funkcje pomocnicze:

1. $Res_0 : A \times W \times \Sigma \to 2^{\Sigma}$ konstruowane na podstawie zdań efektów akcji.

   $\forall_{a \in A, w \in W, \sigma \in \Sigma} Res_0(a, w, \sigma) = \{\sigma' \in \Sigma : ((a, w)$ **causes** α **if** $\pi) \in D \wedge (\sigma \models \pi) \Rightarrow (\sigma' \models \alpha)\}$

   Oznacza to, że Res_0 konstruuje się bez minimalizacji zmian.

2. Funkcję Res^- wyznacza się stosując minimalizację zmian.

3. Funkcję $Res^+ : A \times W \times \Sigma \to 2^{\Sigma}$ spełniającą warunek $\forall_{a \in A, w \in W, \sigma \in \Sigma}$:

   $$Res_0^+(a, w, \sigma) =$$

   $\{\sigma' \in Res_0(a, w, \sigma) : ((a, w)$ **typically causes** β **if** $\pi) \in D \wedge (\sigma \models \varphi) \Rightarrow (\sigma' \models \beta)\}$

Niech D będzie dziedziną akcji języka Ω i niech $S = (\Sigma, \sigma_0, ResAb, ResN)$ będzie strukturą dla Ω. Mówimy, że S jest modelem D $\leftrightarrow$ spełnione są warunki:

- Σ jest zbiorem stanów z dziedziny D
- każde zdanie obserwacji i każde zdanie wartości z dziedziny D jest prawdziwe w S
- $\forall_{a \in A, w \in W, \sigma \in \Sigma} ResN(a, w, \sigma)$ jest zbiorem tych wszystkich stanów $\sigma' \in Res_0^+(a, w, \sigma)$, dla których zbiory $New(a, w, \sigma, \sigma')$ są minimalne
- $\forall_{a \in A, w \in W, \sigma \in \Sigma} ResAb(a, w, \sigma) = Res^-(a, w, \sigma) | ResN(a, w, \sigma)$

Warto zwrócić uwagę na to, że skutki *pewne* dla akcji traktowane są jak *typowe*.

2.3.4 Funkcja przejścia

Niech $S = (\Sigma, \sigma_0, ResAb, ResN)$ będzie strukturą dla języka. Konstrukcja funkcji $\Psi_S : (A \times W)^* \times \Sigma \to \Sigma$ wygląda następująco:

- $\Phi_S(a, \epsilon, \sigma) = \sigma$ gdzie ϵ oznacza ciąg pusty
- jeśli $\Phi_S(((a_1, w_1), \ldots, (a_n, w_n)), \sigma)$ jest określona to

  $\Phi_S(((a_1, w_1), \ldots, (a_n, w_n)), \sigma) \in ResAb((a_n, w_n), \Phi_S((a_1, w_1), \ldots, (a_{n-1}, w_{n-1})))$

  $$\cup ResN((a_n, w_n), \Phi_S((a_1, w_1), \ldots, (a_{n-1}, w_{n-1})))$$

3 Język kwerend

- **Czy podany program działań jest wykonywalny zawsze/kiedykolwiek?**
  always/ever executable SC
- **Czy wykonanie podanego programu działań z dowolnego stanu spełniającego warunek π prowadzi zawsze/kiedykolwiek/na ogół do stanu spełniającego warunek celu γ ?**
  always/ever/typically accessible γ if π when SC
- **Czy z dowolnego stanu spełniającego warunek π cel γ jest osiągalny zawsze/kiedykolwiek/na ogół?**
  always/ever/typically accessible γ if π
- **Czy wskazany wykonawca jest zaangażowany w realizację programu zawsze/kiedykolwiek?**
  always/ever partakes w when SC