Konstrukcja spójnego obrazu stanu globalnego

Proces rozproszony

Proces rozproszony Π, będący współbieżnym wykonaniem zbioru $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ procesów sekwencyjnych P_i, opisuje uporządkowana czwórka

$$\Pi = \langle \boldsymbol{\Sigma}, \boldsymbol{\Sigma}^0, \boldsymbol{\Lambda}, \boldsymbol{\Phi} \rangle, \tag{7.1}$$

gdzie:

- $\boldsymbol{\Sigma}$ – jest zbiorem stanów globalnych procesu rozproszonego, $\boldsymbol{\Sigma} \subseteq \boldsymbol{\mathcal{S}}_1 \times \boldsymbol{\mathcal{S}}_2 \times ... \times \boldsymbol{\mathcal{S}}_n$,
- $\boldsymbol{\Sigma}^0$ – jest zbiorem stanów początkowych, $\boldsymbol{\Sigma}^0 \subseteq \boldsymbol{\mathcal{S}}_1^0 \times \boldsymbol{\mathcal{S}}_2^0 \times ... \times \boldsymbol{\mathcal{S}}_n^0$,
- $\boldsymbol{\Lambda}$ – jest zbiorem zdarzeń, $\boldsymbol{\Lambda} = \boldsymbol{\mathcal{E}}_1 \cup \boldsymbol{\mathcal{E}}_2 \cup ... \cup \boldsymbol{\mathcal{E}}_n$;
- $\boldsymbol{\Phi}$ – jest funkcją tranzycji, taką że $\boldsymbol{\Phi} \subseteq \boldsymbol{\Sigma} \times \boldsymbol{\Lambda} \times \boldsymbol{\Sigma}$.

(2)

Wykonanie częściowe procesu

Częściowe wykonanie procesu rozproszonego

$$\boldsymbol{\Pi} = \langle \boldsymbol{\Sigma}, \boldsymbol{\Sigma}^0, \boldsymbol{\Lambda}, \boldsymbol{\Phi} \rangle$$

utożsamia się z ciągiem $\Sigma^0, E^1, \Sigma^1, E^2, \ldots, \Sigma^s, E^{s+1}, \Sigma^{s+1},$ składającym się naprzemiennie ze stanów i zdarzeń, takim że dla każdego u, $0 \leqslant u \leqslant s$,

$$\langle \Sigma^u, E^{u+1}, \Sigma^{u+1} \rangle \in \boldsymbol{\Phi}. \qquad (7.2)$$

(3)

Wykonanie procesu

Przez **wykonanie** (**realizację**) Υ procesu $\boldsymbol{\Pi}$ rozumieć będziemy natomiast częściowe wykonanie rozpoczynające się stanem początkowym $\Sigma^0 \in \boldsymbol{\Sigma}^0$.

(4)

Stan osiągalny

Powiemy, że stan Σ' procesu jest **osiągalny** ze stanu Σ, co oznaczymy przez

$$\boldsymbol{\Sigma \rightsquigarrow \Sigma'}, \qquad (7.3)$$

jeżeli istnieje częściowe wykonanie $\Sigma^0, E^1, \Sigma^1, E^2, \ldots, \Sigma^s, E^{s+1}, \Sigma^{s+1}$ procesu Π, takie że $\Sigma = \Sigma^0$ a $\Sigma' = \Sigma^{s+1}$.

(5)

Globalny stan osiągalny

Oznaczmy przez Υ zbiór wszystkich możliwych wykonań (realizacji) procesu Π. Jeżeli istnieje wykonanie procesu Π, takie że Σ jest stanem końcowym, to stan ten nazwiemy **globalnym stanem osiągalnym** (**spójnym**) procesu rozproszonego Π (ang. *reachable, consistent*).

(6)

Historia wykonania

Każdemu wykonaniu $\Upsilon \in \boldsymbol{\Upsilon}$ procesu Π , odpowiada pewien ciąg stanów Σ^0, Σ^1, Σ^2, ... , Σ^s, Σ^{s+1}, nazywany **śladem wykonania (realizacji) procesu** Π , oraz ciąg zdarzeń E^0, E^1, E^2, ..., E^s, E^{s+1}, nazywany **historią wykonania (realizacji) procesu**.

Historię E^0, E^1, E^2, ... , E^s, oznaczamy przez Ξ^s, a zbiór historii – przez $\boldsymbol{\Xi}$.

(7)

Konfiguracja

- **Zbiór konfiguracji** (obrazów stanu globalnego) $\boldsymbol{\Gamma}$ jest iloczynem kartezjańskim stanów procesów składowych procesu rozproszonego

$$\boldsymbol{\Gamma} = \boldsymbol{\mathcal{S}}_1 \times \boldsymbol{\mathcal{S}}_2 \times \ldots \times \boldsymbol{\mathcal{S}}_n$$

- **Konfiguracja** – wektor stanów lokalnych wszystkich składowych procesów sekwencyjnych procesu rozproszonego

$$\Gamma = \langle S_1, S_2, \ldots, S_n \rangle$$

(8)

Konfiguracja spójna (1)

Konfigurację Γ nazwiemy **konfiguracją spójną** lub **obrazem spójnym** jeżeli $\forall E,\ \forall E'$ zachodzi:

$$(E' \in \Gamma \wedge E \mapsto E') \Rightarrow (E \in \Gamma) \qquad (7.4)$$

(9)

Konfiguracja spójna (2)

Twierdzenie 7.1
Konfiguracja $\Gamma = \langle S_1^{k1}, S_2^{k2}, \ldots, S_n^{kn} \rangle$, reprezentująca stan osiągalny przetwarzania rozproszonego Π jest konfiguracją spójną.

(10)

Linia odcięcia, odcięcie spójne

Linia odcięcia – dzieli zbiór zdarzeń na **przeszłość** oraz **przyszłość**

Odcięciem Ψ zbioru zdarzeń Λ nazwiemy skończony zbiór $\Psi \subseteq \Lambda$, taki że:

$$(E' \in \Psi \wedge E \mapsto_i E') \Rightarrow (E \in \Psi) \qquad (7.5)$$

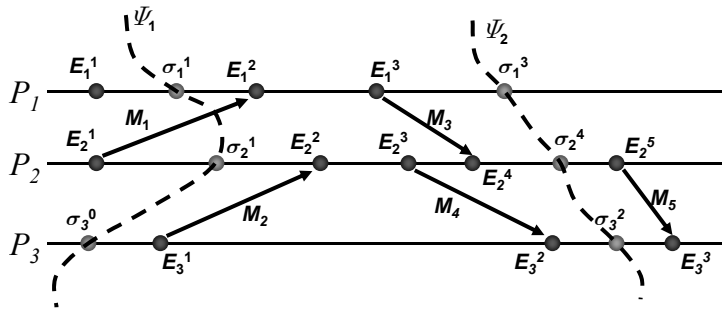
Odcięcie Ψ zbioru zdarzeń Λ nazwiemy ***odcięciem spójnym***, gdy:

$$(E' \in \Psi \wedge E \mapsto E') \Rightarrow (E \in \Psi) \qquad (7.6)$$

Odcięcie Ψ_2 jest **późniejsze** od Ψ_1, jeżeli $\Psi_1 \subseteq \Psi_2$

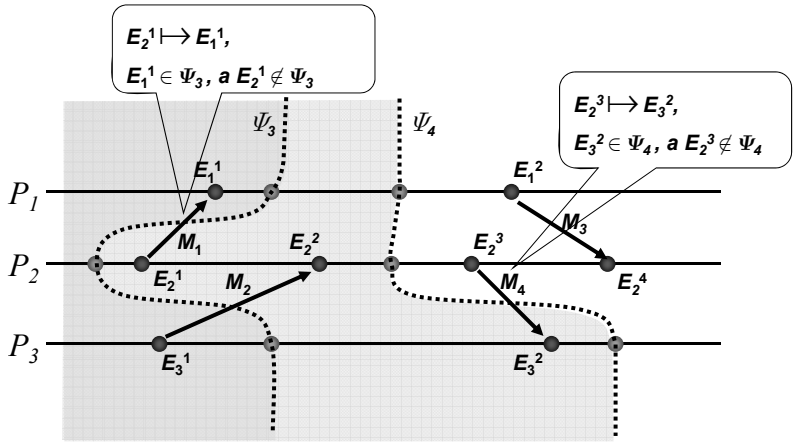
(11)

Odcięcie spójne – przykład



(12)

Odcięcie niespójne – przykład



(13)

Odcięcie spójne a konfiguracja spójna

Zgodnie z definicją, każdemu odcięciu Ψ opisanemu przez linię odcięcia $\sigma_1^{k1}, \sigma_2^{k2}, ..., \sigma_n^{kn}$ odpowiada konfiguracja

$$\Gamma = \langle S_1^{k1}, S_2^{k2}, ..., S_n^{kn} \rangle. \qquad (7.7)$$

Twierdzenie 7.5

Niech Γ będzie konfiguracją a Ψ odpowiadającym jej odcięciem. Konfiguracja Γ jest konfiguracją spójną, wtedy i tylko wtedy, gdy Ψ jest odcięciem spójnym.

(14)

Stan globalny systemu

Wiele problemów istniejących w systemach rozproszonych można sprowadzić do problemu oceny stanu globalnego:

- Śledzenie i sterowanie wykonywaniem programu rozproszonego (monitoring, debugging)
- Detekcja stanów awaryjnych (utraty wiadomości, zakleszczenia) lub oczekiwanych (zakończenia obliczeń rozproszonych)
- Dostosowywanie konfiguracji i funkcji systemu do zmieniającego się obciążenia

(15)

Modele stanów globalnych (1)

- Problem: wzajemne wykluczanie trzech procesów współdzielących pewien zasób
- Warunkiem uzyskania dostępu do zasobu jest posiadanie znacznika (ang. *token*)
- Znacznik krąży między procesami połączonymi w logiczny pierścień

(16)

Modele stanów globalnych (2)

Stan $S_i(\tau)$ procesu P_i w każdej chwili τ czasu globalnego (rzeczywistego) zdefiniowany jest przez trzy zmienne: $present_i(\tau)$, $outLog_i(\tau)$, $inLog_i(\tau)$.

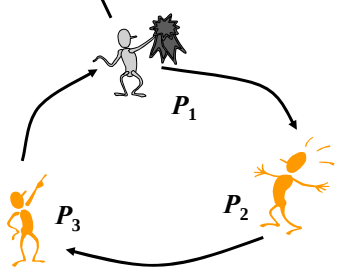
$$S_i(\tau) = \langle\, present_i(\tau),\, outLog_i(\tau),\, inLog_i(\tau) \,\rangle \qquad (7.9)$$

- $present_i(\tau)$ - przyjmuje wartość *True*, tylko wówczas, gdy znacznik typu TOKEN znajduje się w chwili τ w procesie P_i.
- $outLog_i(\tau)$ - kolejka znaczników wysłanych do chwili τ przez proces P_i.
- $inLog_i(\tau)$ - kolejka znaczników odebranych przez proces P_i do chwili τ.

(17)

Modele stanów globalnych – przykład 1 (2)

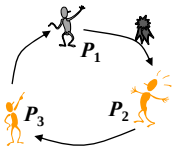
$$\Sigma(\tau_0) = \langle\, present_1(\tau_0) = True \quad outLog_1(\tau_0) = \varnothing \quad inLog_1(\tau_0) = \varnothing \,\rangle$$
$$\langle\, present_2(\tau_0) = False \quad outLog_2(\tau_0) = \varnothing \quad inLog_2(\tau_0) = \varnothing \,\rangle$$
$$\langle\, present_3(\tau_0) = False \quad outLog_3(\tau_0) = \varnothing \quad inLog_3(\tau_0) = \varnothing \,\rangle \,\rangle$$



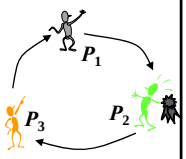
(18)

Modele stanów globalnych – przykład 1 (5)

$$\Sigma^1 = \langle \langle present_1(\tau_1) = False \;\; outLog_1(\tau_1) = \{token_1\} \;\; inLog_1(\tau_1) = \varnothing \rangle$$
$$\langle present_2(\tau_1) = False \;\; outLog_2(\tau_1) = \varnothing \;\; inLog_2(\tau_1) = \varnothing \rangle$$
$$\langle present_3(\tau_1) = False \;\; outLog_3(\tau_1) = \varnothing \;\; inLog_3(\tau_1) = \varnothing \rangle \rangle$$

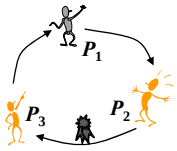


$$\Sigma^2 = \langle \langle present_1(\tau_2) = False \;\; outLog_1(\tau_2) = \{token_1\} \;\; inLog_1(\tau_2) = \varnothing \rangle$$
$$\langle present_2(\tau_2) = True \;\; outLog_2(\tau_2) = \varnothing \;\; inLog_2(\tau_2) = \{token_1\} \rangle$$
$$\langle present_3(\tau_2) = False \;\; outLog_3(\tau_2) = \varnothing \;\; inLog_3(\tau_2) = \varnothing \rangle \rangle$$



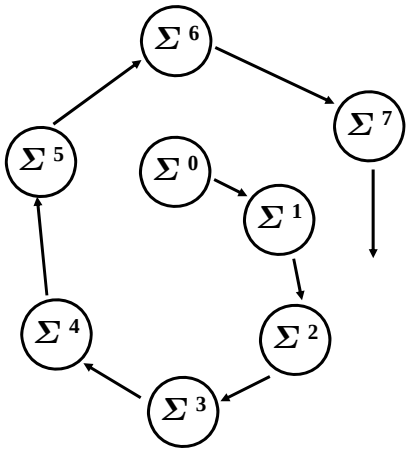
$$\Sigma^3 = \langle \langle present_1(\tau_3) = False \;\; outLog_1(\tau_3) = \{token_1\} \;\; inLog_1(\tau_3) = \varnothing \rangle$$
$$\langle present_2(\tau_3) = False \;\; outLog_2(\tau_3) = \{token_2\} \;\; inLog_2(\tau_3) = \{token_1\} \rangle$$
$$\langle present_3(\tau_3) = False \;\; outLog_3(\tau_3) = \varnothing \;\; inLog_3(\tau_3) = \varnothing \rangle \rangle$$

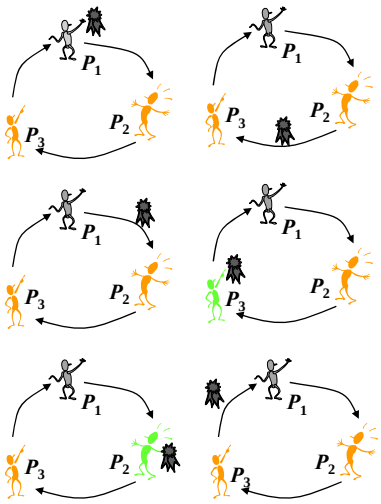
$\vdots$ itd...



(19)

Graf stanów osiągalnych





(20)

Model stanów globalnych (3)

Zalety:

- Ogranicza liczbę składowych stanu globalnego do liczby procesów

Wady:

- podejście jest adekwatne tylko dla systemów z kanałami niezawodnymi
- rosnący koszt związany z zapamiętywaniem wysłanych i odebranych wiadomości

(21)

Modele stanów globalnych (4)

- Podejście alternatywne: stan globalny jako złożenie stanów lokalnych i stanów kanałów komunikacyjnych
- $n+m$ składowych odpowiadających stanom lokalnym wszystkich n procesów i wszystkich m kanałów
- Prostsza reprezentacja stanu procesu kosztem większej liczby składowych

(22)

Modele stanów globalnych (5)

- Stan procesu P_i jest w każdej chwili określony przez zmienną logiczną $present_i$, oraz przez liczniki $sentNo_i$ (ang. *sent number*) i $recvNo_i$ (ang. *received number*)
- Stan kanału $L_{i,j}$ określany przez zbiór znaczników znajdujących się aktualnie w kanale $C_{i,j}$

(23)

Model stanów globalnych (6)

•Prostsza reprezentacja:

- stan procesu określony przez zmienną $present_i$ (posiadanie znacznika)
- stan kanału określony przez zmienną $present_{i,j}$ (znacznik w kanale)

(24)

Modele stanów globalnych – przykład 2

$\Sigma(\tau) = \langle S_1(\tau), S_2(\tau), S_3(\tau), L_{1,2}(\tau), L_{2,3}(\tau), L_{3,1}(\tau) \rangle$

$\Sigma^0 = \langle 1, 0, 0, 0, 0, 0 \rangle$

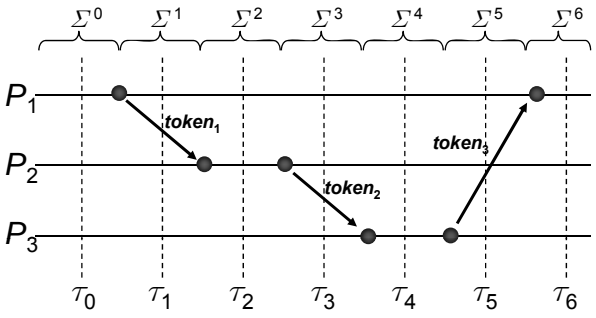
$\Sigma^1 = \langle 0, 0, 0, 1, 0, 0 \rangle$

$\Sigma^2 = \langle 0, 1, 0, 0, 0, 0 \rangle$

$\Sigma^3 = \langle 0, 0, 0, 0, 1, 0 \rangle$

$\Sigma^4 = \langle 0, 0, 1, 0, 0, 0 \rangle$

$\Sigma^5 = \langle 0, 0, 0, 0, 0, 1 \rangle$



(25)

Ocena stanów globalnych

Przykłady oceny stanów globalnych

- **Zaginięcie znacznika** – może prowadzić do blokady całego systemu
- **Zwielokrotnienie znaczników** – może prowadzić do obecności kilku procesów w sekcji krytycznej

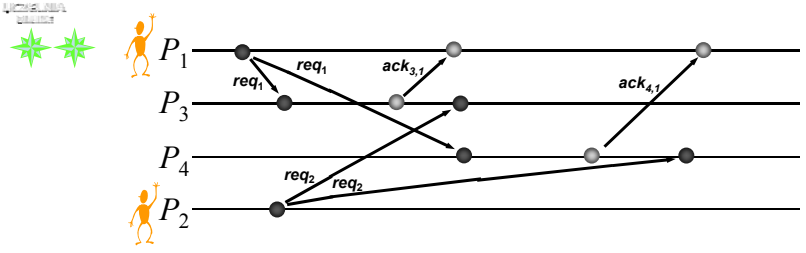
(26)

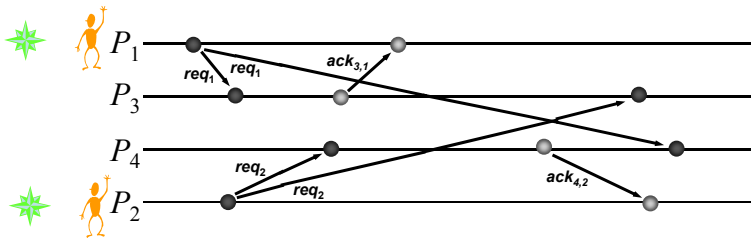
Porównanie reprezentacji stanów globalnych

- Pierwsza reprezentacja:
  operuje na informacji najpełniejszej: unikalnych znacznikach i całej historii komunikacji.
- Druga reprezentacja:
  utożsamia wszystkie znaczniki i wyróżnia tylko stany ich obecności oraz nieobecności w poszczególnych procesach i kanałach

(27)

Niedeterminizm przetwarzania rozproszonego





(28)

Metoda wyznaczania stanu globalnego

Proces chcący wyznaczyć stan globalny wysyła żądania wyznaczania stanów lokalnych, a następnie konstruuje na ich podstawie stan globalny.

(29)

Problemy związane z wyznaczaniem stanu globalnego

Otrzymane składowe stany lokalne procesów mogą być:

- przestarzałe,
- niekompletne,
- odpowiadać konfiguracjom niespójnym (reprezentującym stany nieosiągalne).

Problem konstrukcji stanu globalnego

Dodatkowe założenia upraszczające

- dostępny jest dla wszystkich procesów globalny zegar czasu rzeczywistego
- znane jest maksymalne opóźnienie komunikacji
- względne prędkości przetwarzania w poszczególnych węzłach są ograniczone

(31)

Koncepcja konstrukcji obrazu spójnego (1)

- Proces inicjatora Q_α wysyła do monitorów wszystkich procesów przetwarzania rozproszonego wiadomość: „zapamiętaj stan lokalny w chwili τ_s”
- Monitory Q_i procesów zapamiętują stan lokalny S_i w chwili τ_s i natychmiast wysyłają wiadomość kontrolną do wszystkich monitorów. Zapamiętują też stan $L_{k,i}$ kanałów wejściowych jako zbiorów wszystkich wiadomości aplikacyjnych, które zostały wysłane przed τ_s a dotarły do Q_i po chwili τ_s.

(32)

Koncepcja konstrukcji obrazu spójnego (2)

- Gdy monitor Q_i otrzyma przez kanał $C_{j,i}$ pierwszą wiadomość o etykiecie czasowej większej od τ_s, aktualną wartość $L_{j,i}$ uznaje jako stan tego kanału w chwili τ_s
- Monitor Q_i, po otrzymaniu wiadomości o etykietach czasowych większej od τ_s ze wszystkich kanałów wejściowych, przesyła stan lokalny i wyznaczone stany kanałów wejściowych do inicjatora Q_α

(33)

Koncepcja konstrukcji obrazu spójnego (3)

Po odebraniu od wszystkich monitorów wiadomości zawierających stany lokalne oraz stany ich kanałów wejściowych w chwili τ_s, inicjator Q_α konstruuje obraz stanu globalnego Γ

(34)

Założenia dotyczące środowiska przetwarzania

- Niezawodne kanały zachowują uporządkowanie wiadomości (niezawodne kanały FIFO)
- Stan reprezentowany jest w postaci złożenia lokalnych stanów procesów i stanów kanałów
- Pełen asynchronizm komunikacji i przetwarzania
- Brak zegara globalnego

(35)

Algorytm Chandy – Lamporta: Koncepcja (1)

- Pewien monitor Q_α inicjuje **proces konstrukcji (detekcji)** spójnego obrazu stanu globalnego
- Q_α zapamiętuje stan lokalny skojarzonego z nim procesu aplikacyjnego P_α i wysyła wiadomość kontrolną (znacznik), do wszystkich incydentnych monitorów

(36)

Algorytm Chandy – Lamporta: Koncepcja (2)

- Każdy monitor Q_i po odebraniu znacznika z kanału $C_{j,i}$, sprawdza, czy jest to pierwszy znacznik odebrany w danym procesie detekcji
- Jeżeli tak jest, monitor Q_i zapamiętuje stan S_i procesu P_i, uznaje stan kanału $C_{j,i}$ za pusty, i propaguje znacznik przez wszystkie swoje kanały wyjściowe

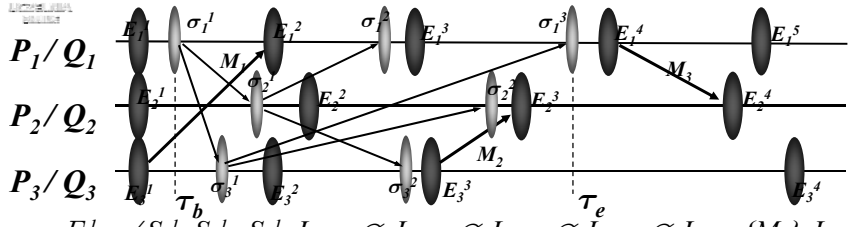
(37)

Algorytm Chandy – Lamporta: Koncepcja (3)

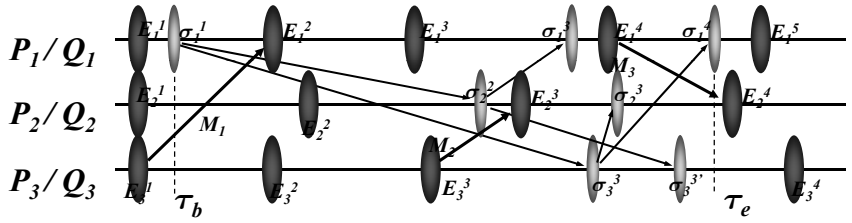
- Jeżeli stan procesu P_i został już wcześniej zapamiętany, to Q_i uznaje za stan kanału $C_{j,i}$ zbiór tych wszystkich wiadomości aplikacyjnych, które dotarły tym kanałem po zapamiętaniu stanu a przed otrzymaniem znacznika
- Po odebraniu znaczników ze wszystkich kanałów wejściowych, monitor Q_i przesyła zapamiętany stan lokalny $procState_i$ oraz stan $chanState_i$ do monitora Q_β

(38)

Ilustracja działania algorytmu Chandy-Lamporta



$\Gamma^1 = \langle S_1^1, S_2^1, S_3^1, L_{1,2}=\varnothing, L_{1,3}=\varnothing, L_{2,1}=\varnothing, L_{2,3}=\varnothing, L_{3,1}=\{M_1\}, L_{3,2}=\varnothing \rangle$



$\Gamma^2 = \langle S_1^1, S_2^2, S_3^3, L_{1,2}=\varnothing, L_{1,3}=\varnothing, L_{2,1}=\varnothing, L_{2,3}=\varnothing, L_{3,1}=\{M_1\}, L_{3,2}=\{M_2\} \rangle$

(39)

Algorytm Chandy – Lamporta (1)

```
type PACKET extends FRAME is record of
  data : MESSAGE
end record

type MARKER extends FRAME

type STATE extends FRAME is record of
  procState : PROCESS_STATE
  chanState : array [1..n] of set of MESSAGE
end record
```

(40)

Algorytm Chandy – Lamporta (2)

```
msgIn       : MESSAGE
pcktOut     : PACKET
markerOut   : MARKER
stateOut    : STATE
recvMark_i  : array [1..n] of BOOLEAN := False
chanState_i : array [1..n] of set of MESSAGE := ∅
procState_i : PROCESS_STATE
C_i^IN      : set of CHANNEL_ID
C_i^OUT     : set of CHANNEL_ID
involved_i  : BOOLEAN := False
```

(41)

Algorytm Chandy – Lamporta (3)

```
1.  procedure RECORDSTATE() do
2.      procState_i := S_i
3.      for all C_j,i ∈ C_i^IN do
4.          chanState_i[j] := ∅
5.      end for
6.      involved_i := True
7.      send(Q_i, Q_i^OUT, markerOut)
8.  end procedure

9.  procedure SENDSTATE(i) do
10.     stateOut.procState := procState_i
11.     stateOut.chanState := chanState_i
12.     send(Q_i , Q_β , stateOut)
13. end procedure
```

(42)

Algorytm Chandy – Lamporta (4)

```
14. when e_start(Q_α , TakeSnapshot) do
15.       RECORDSTATE()
16.       recvMark_α[α] := True
17.       if C_α^IN = ∅ then
18.         SENDSTATE(α)
19.     end if
20.   end when
```

(43)

Algorytm Chandy – Lamporta (5)

```
21.   when e_send(P_i, P_j, msgOut: MESSAGE) do
22.      pcktOut.data := msgOut
23.     send(Q_i, Q_j, pcktOut)
24.   end when
```

(44)

Algorytm Chandy – Lamporta (6)

```
25. when e_receive(Q_j, Q_i, markerIn: MARKER) do
26.  if ¬ involved_i then
27.      RECORDSTATE( )
28.  end if
29.  recvMark_i[j] := True
30.  if ∀ C_j,i ∈ C_i^IN :: recvMark_i[j] then
31.     SENDSTATE(i)
32.   end if
33. end when
```

(45)

Algorytm Chandy – Lamporta (7)

```
34. when e_receive (Q_j, Q_i, pcktIn : PACKET) do
35.      msgIn := pcktIn.data
36.      if involved_i ∧ ¬ recvMark_i[j] then
37.       chanState_i[j] :=chanState_i[j] ∪{msgIn}
38.      end if
39.      deliver (P_j, P_i, msgIn)
40. end when
```

(46)

Twierdzenie

Twierdzenie

Algorytm Chandy-Lamporta wyznacza w skończonym czasie konfigurację spójną.

(47)

Wyznaczanie obrazu spójnego: Kanały non-FIFO

Lai i Yang przedstawili algorytm, który nie wymaga by kanały były typu FIFO.

W algorytmie zakłada się, że reprezentacje stanów lokalnych obejmują ***historię komunikacji****, a więc odpowiednie zbiory wiadomości dotychczas wysłanych i odebranych.*

$outLog_{i,j}(\tau) = inLog_{i,j}(\tau')$

P_i P_j

τ – *wysłanie znacznika*

τ' – *odebranie znacznika*

(48)

Algorytm Lai-Yang: Koncepcja (1)

- Jeśli po zapamiętaniu stanu proces P_i już nigdy nie wyśle wiadomości do P_j, to nie ma potrzeby przesłania do Q_j znacznika
- Jeśli wiadomości będą w dalszym ciągu wysyłane, to dołączany jest do nich znacznik w formie etykiety

(49)

Algorytm Lai-Yang: Koncepcja (2)

- Wyróżnione pakiety ze znacznikiem przechwytywane są przez monitor Q_j odbiorcy i powodują zapamiętanie stanu procesu P_j, a przed przekazaniem wiadomości aplikacyjnej procesowi P_j
- Algorytm przydziela procesom (kontrolerom) oraz wiadomościom jeden z dwóch kolorów *White* albo *Red*

(50)

Algorytm Lai-Yang (1)

```
type PACKET extends FRAME is record of
   colour    : enum  {White, Red}
   data      : MESSAGE
  end record

type STATE extends FRAME is record of
   procState : PROCESS_STATE
   sentLog   : set of MESSAGE
   recvLog   : set of MESSAGE
  end record
```

(51)

Algorytm Lai-Yang (2)

```
msgIn      : MESSAGE
pcktOut    : PACKET
dummyOut   : PACKET
stateOut   : STATE
procState_i : PROCESS_STATE
procColour_i: enum {White, Red} := White
sentLog_i  : set of MESSAGE := ∅
outLog_i   : set of MESSAGE := ∅
recvLog_i  : set of MESSAGE := ∅
inLog_i    : set of MESSAGE := ∅
```

(52)

Algorytm Lai-Yang (3)

```
1.  procedure RECORDSTATE() do
2.      procState_i := S_i
3.      sentLog_i := outLog_i
4.      recvLog_i := inLog_i
5.  end procedure

6.   procedure SENDSTATE(i) do
7.      stateOut.procState := procState_i
8.      stateOut.sentLog := sentLog_i
9.      stateOut.recvLog := recvLog_i
10.     send(Q_i, Q_β, stateOut)
11.  end procedure
```

(53)

Algorytm Lai-Yang (4)

```
12.  when e_start(Q_α, TakeSnapshot) do
13.       RECORDSTATE()
14.       procColour_α := Red
15.       dummyOut.pcktColour := Red
16.       dummyOut.data := ∅
17.       send(Q_α, 𝒬\{Q_α}, dummyOut)
18.       SENDSTATE(α)
19.  end when
```

(54)

Algorytm Lai-Yang (5)

```
20.  when e_send(P_i, P_j, msgOut: MESSAGE) do
21.        outLog_i := outLog_i ∪ {msgOut}
22.        pcktOut.colour := procColour_i
23.        pcktOut.data := msgOut
24.        send(Q_i, Q_j, pcktOut)
25.  end when
```

(55)

Algorytm Lai-Yang (6)

```
26.  when e_receive(Q_j, Q_i, pcktIn: PACKET) do
27.    if pcktIn.colour=Red ∧ procColour_i=White then
28.       procColour_i := Red
29.       RECORDSTATE(i)
30.       SENDSTATE(i)
31.    end if
32.    msgIn := pcktIn.data
33.    if msgIn ≠ ∅ then
34.       inLog_i := inLog_i ∪ {msgIn}
35.       deliver(P_j,P_i, msgIn)
36.    end if
37.  end when
```

(56)

Algorytm Lai-Yang: Złożoność czasowa

Dla grafu pełnego reprezentującego topologię przetwarzania rozproszonego złożoność czasowa algorytmu Lai-Yanga wynosi 1, a złożoność komunikacyjna, w sensie liczby przesyłanych znaczników, wynosi n-1, pomijając fazę przesyłania wiadomości o stanach procesów i kanałów.

(57)

Algorytm kolorujący procesy i wiadomości: Koncepcja (1)

- Każdy proces pierwotnie ma kolor *White*, a staje się *Red* po zapamiętaniu jego stanu lokalnego a przed przekazaniem mu pierwszej wiadomości z pakietem koloru *Red*.
- Inicjator detekcji stanu globalnego zapamiętuje stan lokalny procesu, staje się *Red* i wysyła pusty pakiet koloru *Red* do wszystkich monitorów.

(58)

Algorytm kolorujący procesy i wiadomości: Koncepcja (2)

- Wiadomości będące w wyznaczonym obrazie stanu globalnego w kanałach, to wiadomości w pakietach koloru *White* odebrane przez monitor koloru *Red*
- Za każdym razem, gdy monitor otrzymuje tego typu pakiet, przesyła zawartą w nim wiadomość do inicjatora

(59)

Algorytm kolorujący procesy i wiadomości (1)

```
type PACKET extends FRAME is record of
   colour : enum  {White, Red}
   data   : MESSAGE
end record

type PROC_STATE extends FRAME is record of
   procState : PROCESS_STATE
end record

type CHAN_STATE extends FRAME is record of
   chanState : MESSAGE
end record
```

(60)

Algorytm kolorujący procesy i wiadomości (2)

```
msgIn          : MESSAGE
pcktOut        : PACKET
dummyOut       : PACKET
procStateOut   : PROC_STATE
chanStateOut   : CHAN_STATE
procState_i    : PROCESS_STATE
procColour_i   : enum {White, Red} := White
```

(61)

Algorytm kolorujący procesy i wiadomości (3)

```
1.  when e_start(Q_α , TakeSnapshot) do
2.    procState_α := S_α
3.    procColour_i := Red;
4.    dummyOut.colour := Red;
5.    dummyOut.data := ∅
6.    send (Q_α, 𝒬\{Q_α}, dummyOut)
7.  end when
```

(62)

Algorytm kolorujący procesy i wiadomości (4)

```
8.  when e_send (P_i, P_j, msgOut: MESSAGE) do
9.    pcktOut.colour := procColour_i
10.   pcktOut.data := msgOut
11.   send (Q_i, Q_j, pcktOut)
12. end when
```

(63)

Algorytm kolorujący procesy i wiadomości (5)

```
13. when e_receive (Q_j, Q_i, pcktIn : PACKET) do
14.   msgIn := pcktIn.data
15.   if pcktIn.colour = White ∧ procColour_i = Red
16.      then
17.         chanStateOut.chanState := msgIn
18.         send(Q_i , Q_α, chanStateOut)
19.   end if
```

(64)

Algorytm kolorujący procesy i wiadomości (6)

```
20.    if pcktIn.colour=Red ∧ procColour_i= White then
21.          procColour_i := Red
22.          procState_i := S_i
23.          procStateOut.procState := procState_i
24.          send (Q_i ,Q_α, procStateOut)
25.    end if
27.    if msgIn ≠ ∅ then
28.           deliver (P_j, P_i, msgIn)
29.    end if
30.  end when
```

(65)