# Programowanie współbieżne i rozproszone

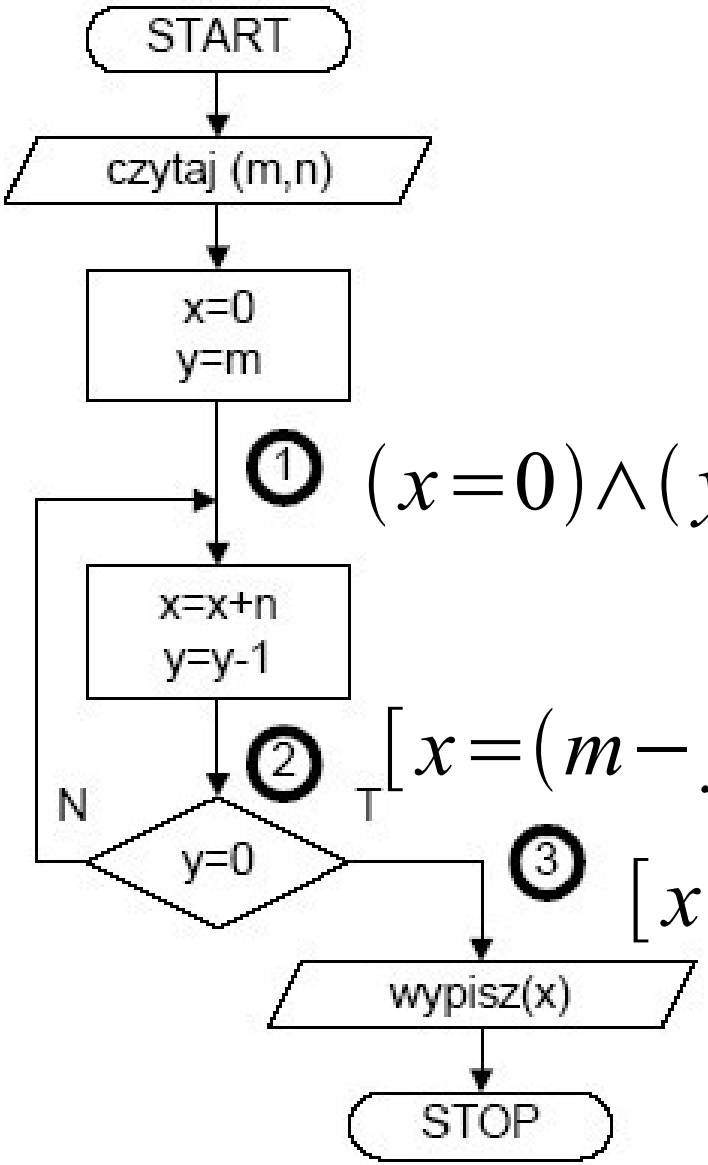
## WYKŁAD 13

# Niezmienniki

- Niezmiennik – formuła *A* taka, że:
  - jest ona spełniona na początku programu,
  - jest ona spełniona we wszystkich stanach programu aż do końcowego.
- Niezmiennik dowodzi się przez indukcję:
  - udowodnienie, że formuła *A* jest prawdziwa w stanie początkowym,
  - udowodnienie, że formuła *A* jest prawdziwa w stanie następnym przy założeniu, że jest ona prawdziwa we wszystkich stanach aż do bieżącego.

# Niezmienniki

## Przykład




$$(x=0)\wedge(y=m)\wedge(m>0)\wedge(n>0)\wedge(y>0)$$

$$[x=(m-y)n]\wedge[y\geqslant 0\ ]$$

$$[x=(m-y)n]\wedge[y=0\ ]\Rightarrow[x=mn]$$

# Algorytm Petersona

- Algorytm Petersona rozwiązuje problem wzajemnego wykluczania.
- Ustawienia początkowe:

```
flagi[0]=false;
flagi[1]=false;
```

„true” oznacza, że dany proces chce wejść do sekcji krytycznej

```
ostatni=0;
```

definiuje, który proces ostatni zażądał wejścia do sekcji krytycznej

# Algorytm Petersona

- Proces 1:

```
while true
    <sekcja niekrytyczna>
    flagi[0]=true;
    ostatni=0;                          start1
    while flagi[1] and ostatni=0 do
        <nie rób nic>                   test1
    end while
    <sekcja krytyczna>                  sk1
    flagi[0]=false;                     koniec1
end while
```

# Algorytm Petersona

- Proces 2:

```
while true
    <sekcja niekrytyczna>
    flagi[1]=true;                          start2
    ostatni=1;
    while flagi[0] and ostatni==1 do        test2
        <nie rób nic>
    end while
    <sekcja krytyczna>                      sk2
    flagi[1]=false;                         koniec2
end while
```

# Algorytm Petersona

- Wzajemne wykluczanie – należy udowodnić formułę:

$$G \neg (sk1 \wedge sk2)$$

- Żywotność - należy udowodnić formułę:

$$G(start1 \rightarrow F\ sk1) \wedge G(start2 \rightarrow F\ sk2)$$

# Algorytm Petersona

- Dedukcyjny dowód poprawności jest złożony, wymaga m.in. udowodnienia prawdziwości niezmienników:

$$G(ostatni=0)\vee(ostatni=1)$$

$$G(flagi[0]<=>(test1\vee sk1\vee koniec1))$$

$$G(flagi[1]<=>(test2\vee sk2\vee koniec2))$$

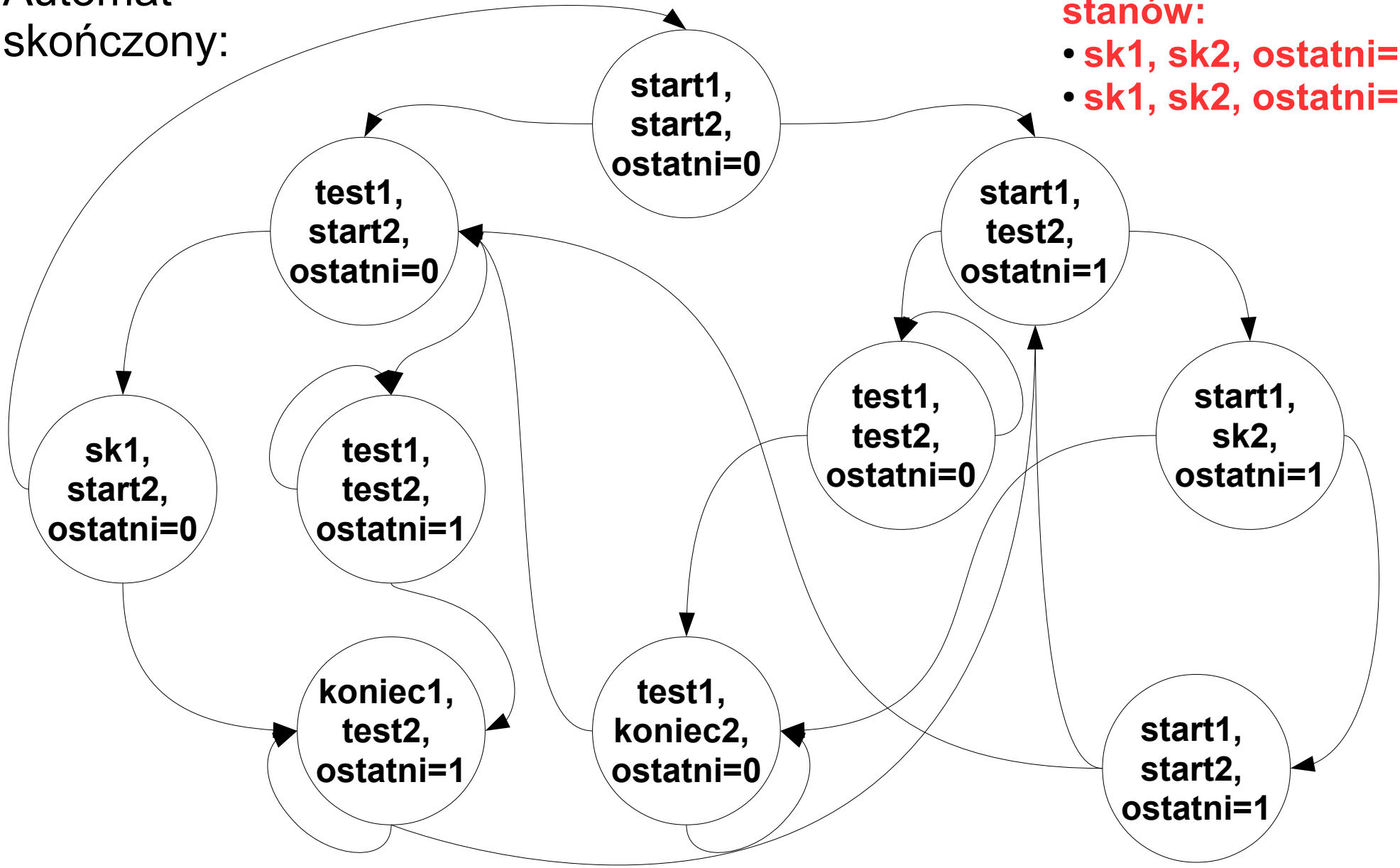
**Dowód poprawności algorytmu Petersona → *Ben-Ari, M.: Logika matematyczna w informatyce. WNT, Warszawa, 2005*.**

# Algorytm Petersona

- Weryfikacja poprzez model (ang. *model checking*):
  - W algorytmie Petersona można wyróżnić skończoną liczbę stanów, w których algorytm może przebywać.
  - Możliwe jest utworzenie automatu skończonego opisującego wszystkie stany i sprawdzenie na jego podstawie czy pewna własność zachodzi.

# Algorytm Petersona

Automat skończony:




**Automat nie zawiera stanów:**

- **sk1, sk2, ostatni=0**
- **sk1, sk2, ostatni=1**

# Algorytm Petersona

```
bool ostatni, flagi[2];
byte licznik;
active [2] proctype proces()
{
   assert(_pid == 0 || _pid == 1);
etykieta:
   flagi[_pid] = 1;
   ostatni = _pid;
   (flagi[1 - _pid] == 0 && ostatni == _pid)
   licznik++;
   assert(licznik == 1);  /* sekcja krytyczna */
   licznik--;
   flagi[_pid] = 0;
   goto etykieta
}
```

Specyfikacja
w języku Promela
→ weryfikacja w
narzędziu SPIN