Ćwiczenia 5 6.11

1. Modele równoważne: sieci Petriego, VAS, VASS i automaty licznikowe bez testów 0.

Pomysł: sieć Petriego $\rightsquigarrow$ VAS

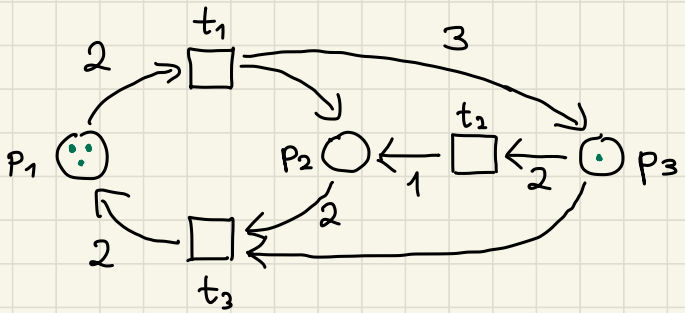
$\uparrow$ automat licznikowy $\leftsquigarrow$ VASS

$\downarrow$

równoważność w znaczeniu osiągalności oraz pokrywalności

sieć Petriego $\rightsquigarrow$ VAS

* zakładamy, że nie ma ciasnych pętli

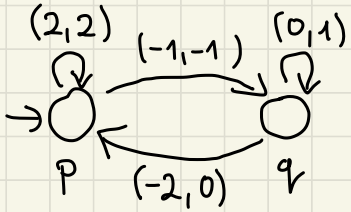


* symulacja VASem:
  - wektor początkowy $u = (3,0,1)$ (początkowa konfiguracja)
  - zbiór wektorów $V = \{t_1, t_2, t_3\}$, gdzie $t_1 = (-2,1,3)$, $t_2 = (0,1,-2)$, $t_3 = (2,-2,1)$

VAS $\rightsquigarrow$ VASS

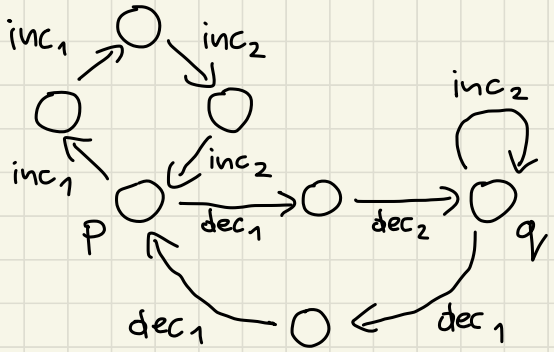
* dany VAS: (u, V)
* definiujemy VASS tak, żeby miał jeden stan q i dajemy mu przejścia (q, v, q) dla każdego $v \in V$
* czy w osiągalne z u? $\rightsquigarrow$ czy (q, w) osiągalne z (q, u)?

VASS $\rightsquigarrow$ automat licznikowy bez testów 0



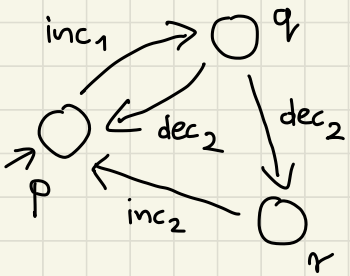
$q_1(1,2)$ osiągalne z $p_1(0,0)$

$q_1(0,0)$ nieosiągalne z $p_1(0,0)$

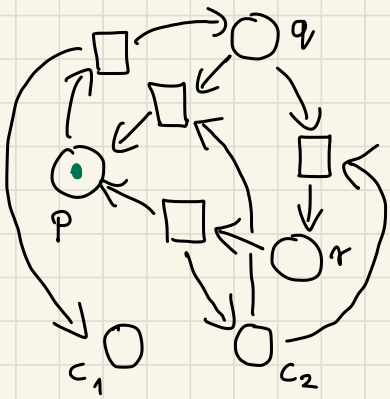


automat licznikowy bez testów 0 $\rightsquigarrow$ sieć Petriego

* definiujemy po jednym miejscu dla każdego stanu automatu i po jednym dla liczników
* tranzycja (w sieci Petriego) reprezentująca przejście (p, inc_i, q) bierze jeden żeton z p i kładzie po jednym żetonie na q i miejscu c_i reprezentującym licznik i



automat licznikowy



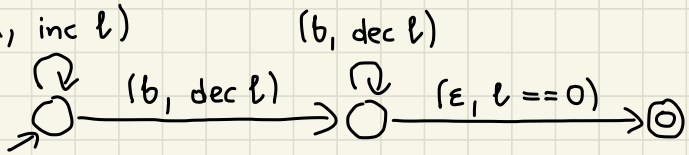
odpowiadająca sieć Petriego

* gdy automat jest w stanie p i istnieje krawędź $t = (p, dec_i, q)$, a obecnie $c_i = 0$, to t jest nieaktywne (nie można go użyć)
* istnieje rodzaj sieci Petriego pozwalający na testy na zero - dodajemy krawędzie „inhibitor arcs"

Pytanie: Co dają nam testy na 0 w automacie?
Czy można za ich pomocą rozpoznać
coś więcej niż języki regularne?

* tak, można na przykład rozpoznać język
$\{ a^n b^n \mid n \in \mathbb{N} \}$

* tworzymy dla niego automat z jednym
licznikiem ℓ zainicjowanym na zero oraz
trzema stanami:



* etykieta $\ell == 0$ oznacza sprawdzenie, czy licznik
ℓ ma wartość zero i umożliwia przejście tylko,
jeśli odpowiedź to „tak"

* sprawdzimy teraz, jaką dokładnie moc mają
automaty z testami na 0 ↪

2. Pokaż, jak zasymulować taśmę maszyny Turinga przy użyciu automatu licznikowego z testami na 0?

Pomysł 1

* BSO załóżmy, że mamy tylko jedynki i zera na taśmie (możemy zakodować dowolny alfabet za pomocą dwuliterowego alfabetu)
* najpierw symulujemy binarny stos przy użyciu automatu z dwoma licznikami i testami na 0
* interpretujemy taśmę maszyny jako dwa stosy i symulujemy ją przy użyciu automatu z czterema licznikami

Pomysł 2

* można użyć jednego licznika na zawartość taśmy, drugiego na pozycję głowicy i pomocniczych liczników do wykonywania operacji na taśmie

przy tym podejściu analiza może być bardziej skomplikowana

Dokładniejszy zarys pierwszego pomysłu

* stos reprezentujemy jako liczbę binarną – na górze stosu trzymamy najmniej znaczący bit
* jak wrzucić 0 albo 1 na górę stosu?
  - wrzucenie 0 to mnożenie przez 2
  - wrzucenie 1 to pomnożenie przez 2 i dodanie 1
* jak mnożyć wartość licznika, jeżeli mamy do dyspozycji tylko dodawanie i odejmowanie?
* poza podstawowym licznikiem ℓ mamy jeszcze pomocniczy licznik ℓ_p i wykonujemy program

```
def double(ℓ):
    while ℓ ≠ 0
    |   dec ℓ
    |   inc ℓp
    |   inc ℓp
    swap(ℓ, ℓp)
```

definicja:

```
while ℓp ≠ 0:
    |   dec ℓp
    |   inc ℓ
```

* Stąd wrzuceniu 0 albo 1 odpowiada kod

```
def push_0(ℓ):
    double(ℓ)
```

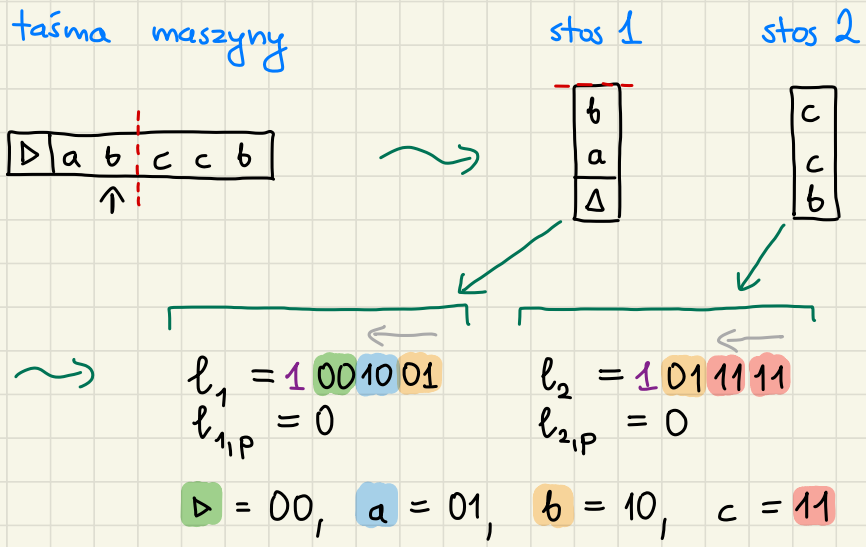
```
def push_1(ℓ):
    double(ℓ)
    inc ℓ
```

* operacji pop odpowiada z kolei kod

```
def pop(ℓ):
    while ℓ ≠ 0:
        dec ℓ
        if ℓ == 0, return 1
        dec ℓ
        if ℓ == 0, return 0
        inc ℓ_p
    swap(ℓ, ℓ_p)
```

* haczyk: jak sprawdzić, czy symulowany stos jest pusty? Skąd mamy wiedzieć, czy na stosie jest 0, czy nie ma nic?

* rozwiązanie: używamy 1, żeby reprezentować pusty stos (to znaczy: teraz licznik 1001 będzie oznaczał, że na stosie jest 001)
* dwa liczniki pozwalają symulować jeden stos, czyli cztery wystarczą nam do symulacji taśmy maszyny Turinga:



* głowica wskazuje na szczyt pierwszego stosu
* szczegółowy opis jest tutaj:

https://www.cs.usfca.edu/~galles/cs411/lecture/lecture14.pdf

3. Zaproponuj transformację n-wymiarowego VASSu do równoważnego $(n+s)$-wymiarowego VASu. Jak małe może być s?

VASS: graf (Q, T), konfiguracja (p, u)

Podejście 1 – 1 dodatkowy wymiar

* każdemu stanowi z Q przypisujemy inną etykietę ze zbioru $K = \{1, 2, \dots, |Q|\}$ (funkcja $\ell : Q \to K$)
* konfigurację (p, u) VASSu reprezentujemy w tworzonym VASie jako $(\ell(p), u) =: c_{p,u}$
* narzuca to tłumaczenie tranzycji w następujący sposób: $(p, v, q) \rightsquigarrow (\ell(q) - \ell(p), v) =: m_{p,v,q}$
* **problem:** $m_{p,v,q}$ można dodać do każdego $c_{r,u}$ dla r spełniającego $\ell(r) \geq \ell(p)$, czyli są przejścia niewystępujące w VASSie ↯
* trzeba poszukać takiego kodowania stanów, które pozwoli poprawnie zdefiniować tranzycje

Podejście 2 - $|Q| := k$ dodatkowych wymiarów

* nowy wymiar dla każdego stanu z Q
* wymiar reprezentujący $q \in Q$ zawiera 1 wtw gdy opisywana konfiguracja VASSu jest w stanie q i wpp zawiera 0
* formalnie dostajemy transformację

	VASS		VAS
konfiguracja	(p, u)	$\rightsquigarrow$	$(0, 0, \dots, 0, \underset{\uparrow p}{1}, 0, \dots, 0, u)$
tranzycja	(p, v, q)	$\rightsquigarrow$	$(0, \dots, 0, \underset{\uparrow q}{1}, 0, \dots, 0, \underset{\uparrow p}{-1}, 0, \dots, 0, v)$

* wektor reprezentujący (p, v, q) można dodać tylko do wektora odpowiadającego konfiguracji $(p, \_)$, więc kodujemy tylko i wyłącznie biegi w VASSie

* problem: rozmiar Q może znacznie przekraczać wymiar wektora u - chcemy znaleźć bardziej oszczędną transformację

Podejście 3 – 4 dodatkowe wymiany

pomysł: na dwóch z nowych miejsc kodujemy obecny stan, a pozostałe dwa używamy przy kodowaniu tranzycji (aby umożliwić wpisanie stanu, do którego przechodzimy)

* jak zakodować stan na dwóch miejscach, aby uzyskać łatwe kodowanie tranzycji?

* szkic modelu:

kodowanie p na dwóch miejscach przy użyciu pewnych funkcji a i b

$$(p, u) \rightsquigarrow (a(p), b(p), 0, 0, u) =: c_{p,u}$$

$$(p, v, q) \rightsquigarrow (-a(p), -b(p), a(q), b(q), v) =: m_{p,v,q}$$

wpisanie kodowania stanu q

* chcemy, żeby fragment $-a(p), -b(p)$ uniemożliwiał dodanie $m_{p,v,q}$ do $c_{r,u}$ dla każdego $r \neq p$ (innymi słowy dla stanu $r \neq p$ ma zachodzić $a(r) - a(p) < 0$ lub $b(r) - b(p) < 0$)

$k := |Q|$

* rozwiązanie: niech $a(p) = \ell(p)$, $b(p) = k + 1 - \ell(p)$

* dla $r \neq p$ mamy $a(r) < a(p)$, jeśli $\ell(r) < \ell(p)$ oraz $b(r) < b(p)$, gdy $\ell(r) > \ell(p)$

przykład:

stan p	$\ell(p)$	$a(p)$	$b(p)$
p_1	1	1	5
p_2	2	2	4
p_3	3	3	3
p_4	4	4	2
p_5	5	5	1

m_{p_2, v, p_3} zaczyna się od $-2, -4$, które można dodać tylko do $m_{p_2, -}$ bez „spadania poniżej zera"

* przy obecnym zapisie po dodaniu $m_{p,v,q}$ do wektora $c_{p,u}$ kodowanie stanu q znajdzie się na 3. i 4. pozycji, więc formalnie trzeba zaznaczyć, że każdy stan ma też reprezentację

$$c'_{p,u} = (0, 0, a(p), b(p), u),$$

a każda tranzycja musi jeszcze mieć drugie możliwe kodowanie (aby umożliwić aplikację do $c'_{\dots}$)

$$m'_{p,v,q} = (a(q), b(q), -a(p), -b(p), v)$$

* podana transformacja generuje $2|T|$ wektorów przejścia ($m_{p,v,q}$ i $m'_{p,v,q}$) - czy da się lepiej?
* tak, można zachowywać kodowanie stanu na pierwszych dwóch miejscach i dla każdej tranzycji z T generować tylko jeden wektor przejścia
* koszt: $|Q|+|T|$, kodowanie:

$$(p,u) \rightsquigarrow (a(p), b(p), 0, 0, u)$$

pomocnicze przejście:
$$\downarrow$$
$$(-a(p), -b(p), a(p), b(p), 0)$$

pomocnicza konfiguracja:
$$\downarrow$$
$$(0, 0, a(p), b(p), u)$$

$$(p, v, q) \rightsquigarrow (a(q), b(q), -a(p), -b(p), v)$$
$$\downarrow$$
$$(q, u+v) \rightsquigarrow (a(q), b(q), 0, 0, u+v)$$

* czy da się zejść do trzech dodatkowych wymiarów? jeśli tak, w jaki sposób?

Podejście 4 – 3 dodatkowe wymiary

* podobnie jak w poprzednim podejściu zaczniemy od sposobu, który nie jest optymalny, ale w tym przypadku jest zdecydowanie prostszy do analizy
* każdej konfiguracji (p,u) VASSu będą odpowiadały trzy możliwe konfiguracje tworzonego VASu:

$$c_{p,u} = (a(p),\ b(p),\ 0,\ u),$$
$$c'_{p,u} = (0,\ a(p),\ b(p),\ u),$$
$$c''_{p,u} = (b(p),\ 0,\ a(p),\ u)$$

dla pewnych nowych funkcji a i b

* każdej tranzycji będą z kolei odpowiadały trzy wektory przejścia, np. na zaaplikowanie (p, v, q) do $c_{p,u}$ będzie pozwalał wektor

$$m_{p,v,q} = (-a(p),\ -b(p)+a(q),\ b(q),\ v),$$

co spowoduje przejście z reprezentacji $c_{-,-}$ do $c'_{-,-}$

* podobnie z $c'_{-,-}$ będziemy przechodzili do $c''_{-,-}$, a następnie do $c_{-,-}$ (tak definiujemy $m'_{p,v,q}$ i $m''_{p,v,q}$)

* przyjmijmy $a(p) = \ell(p)$, szukamy dobrego kandydata na $b(p)$

* musimy zapewnić, że wektor

$$m_{p,v,q} = (-a(p),\ -b(p)+a(q),\ b(q),\ v)$$

nie będzie dodany do konfiguracji

$$c_{r,u} = (a(r),\ b(r),\ 0,\ u)$$

dla $r \neq p$

* w wyniku dostalibyśmy

$$(a(r)-a(p),\ b(r)-b(p)+a(q),\ b(q),\ u+v)$$

* dla $\ell(r) < \ell(p)$ pierwsza współrzędna będzie < 0

* dla $\ell(r) > \ell(p)$ chcemy, żeby na drugim miejscu pojawiła się wartość ujemna, czyli

$$b(r)-b(p) < -a(q),$$

$$b(p)-b(r) > a(q)$$

* wystarczy $b(p)-b(r) > k$, zatem możemy przyjąć $b(p) = (k+1)(k+1-\ell(p))$

* jako formalność pozostaje sprawdzić resztę przejść

* podana transformacja generuje $3|T|$ wektorów przejścia – podobnie jak poprzednio, można poprawić złożoność do $|T| + 2|Q|$

* tym razem generujemy przejścia następująco:

$$(p, u) \rightsquigarrow (a(p), b(p), 0, u)$$

pomocnicze przejścia ↓ (będzie omówione później)

pomocnicza konfiguracja

$$(b(p), 0, a(p), u) =: c_{p,u}$$

$$(p, v, q) \rightsquigarrow (a(q) - b(p), b(q), -a(p), v) =: m_{p,v,q}$$

$$(q, u+v) \rightsquigarrow (a(q), b(q), 0, u+v)$$

* musimy zagwarantować, że $m_{p,v,q}$ nie będzie dodane do konfiguracji innej niż $c_{p,u}$

* proste rachunki pokazują, że wystarczy przyjąć $a(p) = \ell(p)$ (co będzie blokowało dodawanie dla $\ell(r) < \ell(p)$) oraz $b(p) = (k+1)(k+1-\ell(p))$ (blokujące $\ell(r) > \ell(p)$)

Pełne rozwiązanie:

Lemma 2.1. *An n-dim VASS can be simulated by an* $(n+3)$*-dim VAS.*

Proof. We give the construction of the VAS. The last three coordinates encode the state while the first n coordinates are as in the VASS. Assume that the VASS has k states $q_1, \ldots, q_k$. Let $a_i = i$ and $b_i = (k+1)(k+1-i)$ for $i = 1$ to k. If the VASS is at v in state q_i then the VAS will be at $(v, a_i, b_i, 0)$. For each i the VAS has two dummy transitions t_i and t_i' defined so that t_i goes from $(v, a_i, b_i, 0)$ to $(v, 0, a_{k-i+1}, b_{k-i+1})$ and t_i' goes from $(v, 0, a_{k-i+1}, b_{k-i+1})$ to $(v, b_i, 0, a_i)$. Note that t_i and t_i' modify only the last three components. In addition there is a transition t_i'' for each transition $i \to (j, w)$ of the VASS, defined by

$$t_i'' = (w, a_j - b_i, b_j, -a_i).$$

Clearly any path of the VASS can be mimicked by the VAS. It remains to be shown that the VAS cannot do something unintended. We will only show that t_i'' can only be applied if the last three components are b_i, 0 and a_i respectively. The other cases are similar. Observe that for each i and j, $a_i < a_{i+1}$, $b_i > b_{i+1}$, $a_i < b_j$ and $b_i - b_{i+1} = k+1 > a_j$. Let v_i'' be the vector $(w, a_j - b_i, b_j, -a_i)$ which accomplishes the transition t_i''. Note that the $n+1$st and last components are negative. Hence t_i'' cannot be applied when the last three coordinates are $(a_i, b_i, 0)$ or $(0, a_{k-i+1}, b_{k-i+1})$ since either the first or third components are 0. Let the last three coordinates be $(b_m, 0, a_m)$. Then if $m < i$, t_i'' cannot be applied since $a_m - a_i < 0$. If $m > i$, then t_i'' cannot be applied since $b_m + a_j - b_i \leq a_j - (k+1) < 0$. □

Źródło: John Hopcroft, Jean-Jacques Pansiot.
On the reachability problem for 5-dimensional vector addition systems

* można pomyśleć nad tym, jakie problemy pojawią się przy użyciu tylko 2 dodatkowych wymiarów

4. For every $m > 0$ create a Petri net of size $O(n+m)$ that is bounded by $F_m(n)$ and not bounded by any smaller number, where

i) $F_1(n) = 2n$

ii) $F_{m+1}(n) = F_m^n(1) = \overbrace{F_m(F_m(\ldots(F_m(1))\ldots))}^{n \text{ times}}$

* we can construct a VASS instead of a Petri net (since the two models are equivalent and we defined how to simulate one with another solving the first problem; however, here it will be easier to create a Petri net based on the VASS itself)
* for a given n-dimensional VASS (Q, T) with initial configuration (p, u) we define its size as

$$|Q| + |T| + \sum_{i=1}^{n} u(i) + \sum_{t \in T} \sum_{i=1}^{n} |t(i)|$$

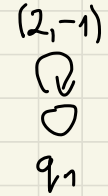
* how does a closed-form expression for $F_m(n)$ look like?

$F_2(n) = F_1(F_1(\dots(F_1(1))\dots)) = 2^n$

$F_3(n) = F_2(F_2(\dots(F_2(1))\dots)) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ (n twos)

and so on

* first, compute $F_1(1)$

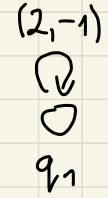


initial config.

$(q_1, (0,1))$

interpretation: first coordinate stores the result of the current computation, second coordinate is the argument to F_1

thus, for $F_1(n)$ we have

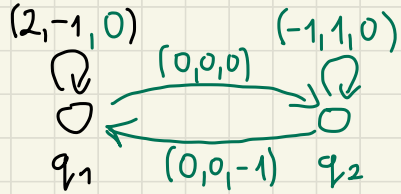


initial config.

$(q_1, (0,n))$

* now, we can build a VASS for $F_2(n)$ using the construction above

* VASS for $F_2(n) = F_1(F_1(\dots(F_1(1))\dots))$



initial config.

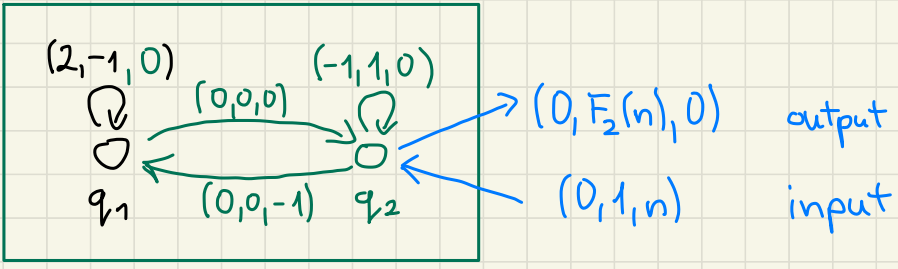
$(q_2, (0,1,n))$

interpretation: two first coordinates serve the same purpose as previously, the third one can be interpreted as the argument for F_2 – the number of times we have to compose F_1 with itself

the initial configuration says that we apply the n-th composition of F_1 to 1

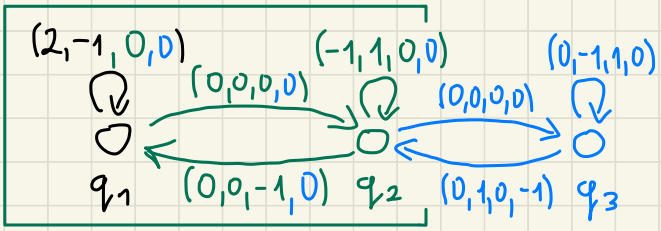
loop in state q_2 updates the parameter for F_1 to the current computation result

when we reach deadlock, the second coordinate stores the final result, hence, we can interpret the construction above as a blackbox computing $F_2(n)$



we can use this observation to construct a VASS computing $F_3(n)$

* "induction" step – VASS for $F_3(n)$



initial configuration: $(q_3, (0,0,1,n))$

it says that we have to apply the n-th composition of F_2 to 1

first we go from q_3 to q_2: we subtract 1 from the fourth coordinate, which can be interpreted as the start of computation of the most inner call of F_2 in $F_3(n) = F_2(F_2(\dots(F_2(1))\dots))$

moreover, we add 1 to the second coordinate to obtain $0,1,1$ as the prefix of the current configuration – this starts the computation of $F_2(1)$ in the green box

the computation ends in state q_2 with configuration prefix equal to $0, F_2(1), 0$

next, we go through q_3 and approach green box again with a new request – to compute F_2 with the argument $F_2(1)$ (as the conf. prefix is $0, 1, F_2(1)$) ...

* it is easy to see how to continue the construction for $F_4(n)$, $F_5(n)$, ...

5. Prove that in the pessimistic case the maximum size of the coverability tree is not smaller than $Ack(n)$, where n is the size of the net.

* we can assume $Ack(n) \approx F_n(n)$
* formally:

$$Ack(m,n) = \begin{cases} n+1 & m=0 \\ Ack(m-1,1) & m>0,\ n=0 \\ Ack(m-1, Ack(m,n-1)) & m>0,\ n>0 \end{cases}$$

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n+1$
1	2	3	4	5	6	$n+2 = 2+(n+3)-3$
2	3	5	7	9	11	$2n+3 = 2\cdot(n+3)-3$
3	5	13	29	61	125	$2^{(n+3)}-3$
4	$13 = 2^{2^2}-3$	$65533 = 2^{2^{2^2}}-3$	$2^{65536}-3 = 2^{2^{2^{2^2}}}-3$	$2^{2^{65536}}-3 = 2^{2^{2^{2^{2^2}}}}-3$	$2^{2^{2^{65536}}}-3 = 2^{2^{2^{2^{2^{2^2}}}}}-3$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{n+3}-3$

* thus the proof follows the construction presented for the previous problem

6. Improve Lipton's construction presented during the lecture to avoid the exponential blowup of the instruction number.

* consider $Dec_{m+1}(x)$ and think about how to improve it (during tutorials), then analyze the whole construction once again (homework)

$Dec_{m+1}(x)$: ← we would like to have one universal Dec_i for all possible calls

loop

a_m++, $\hat{a}_m$--

loop

b_m++, $\hat{b}_m$--

x--, $\hat{x}$++ ← if there is one gadget for Dec_i, how does it know what to increase?

$Dec_m(b_m)$ ← how does it know where to return?

$Dec_m(a_m)$

reverse the side-effect on a_m, b_m and ensure that

this part is repeated $2^{2^m} \cdot 2^{2^m} = 2^{2^{m+1}}$ times

$Dec_{m+1}(x)$:

loop

| $a_m{+}{+}$, $\hat{a}_m{-}{-}$
| loop
| | $b_m{+}{+}$, $\hat{b}_m{-}{-}$
| | $x{-}{-}$, $\hat{x}{+}{+}$
| $Dec_m(b_m)$

$Dec_m(a_m)$

new variables with dashes representing the flags / semaphores for previously defined variables

$\bar{x}{-}{-}$
$x{-}{-}$, $\hat{x}{+}{+}$
$\bar{x}{+}{+}$
or
$\bar{y}{-}{-}$
$y{-}{-}$, $\hat{y}{+}{+}$
$\bar{y}{+}{+}$
or
...

← used only when $\bar{x} = 1$

← used only when $\bar{y} = 1$

$c_i{+}{+}$, $\overline{b_m{+}{+}}$, $Dec_m(b_m)$, $\overline{b_m{-}{-}}$, $c_i{-}{-}$

$c_i{+}{+}$ can be interpreted as a call statement and $\overline{b_m{+}{+}}$ as a parameter for the called function (it informs that some action have to be performed on b_m)

$c_i{-}{-}$ asserts that the execution goes back to the right place

* it remains to work on the details
* we update the other functions in a similar way