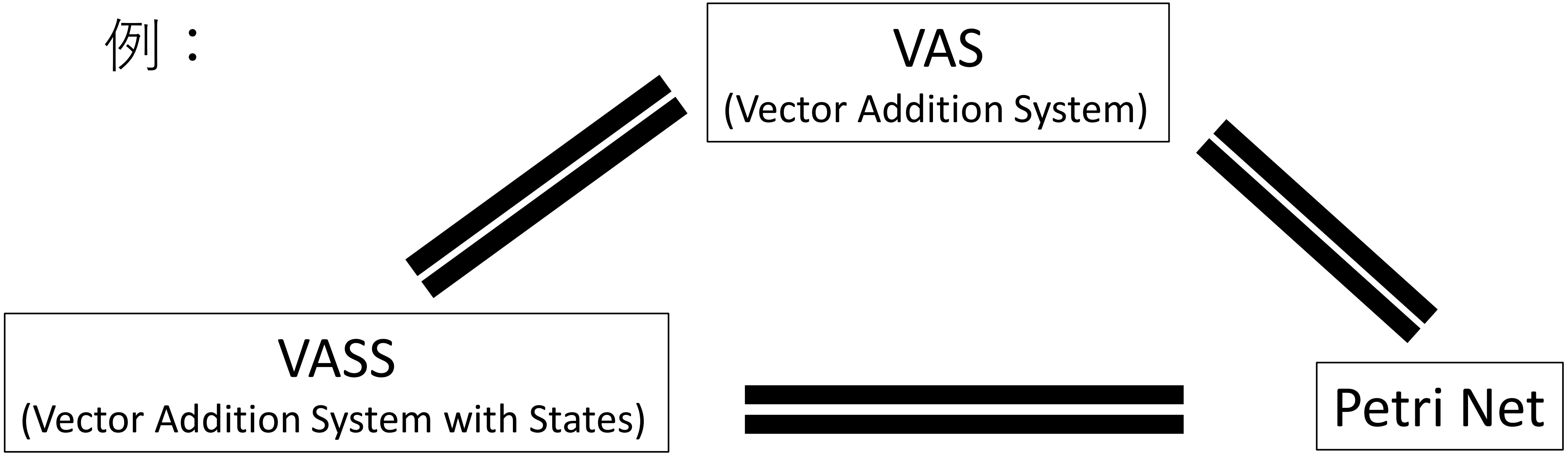
Coq/MathComp上の
VASSからVASへの変換の形式化

千葉大学大学院　脇坂勝大　山本光晴

概要

・状態遷移系…状態と状態間の遷移からなる

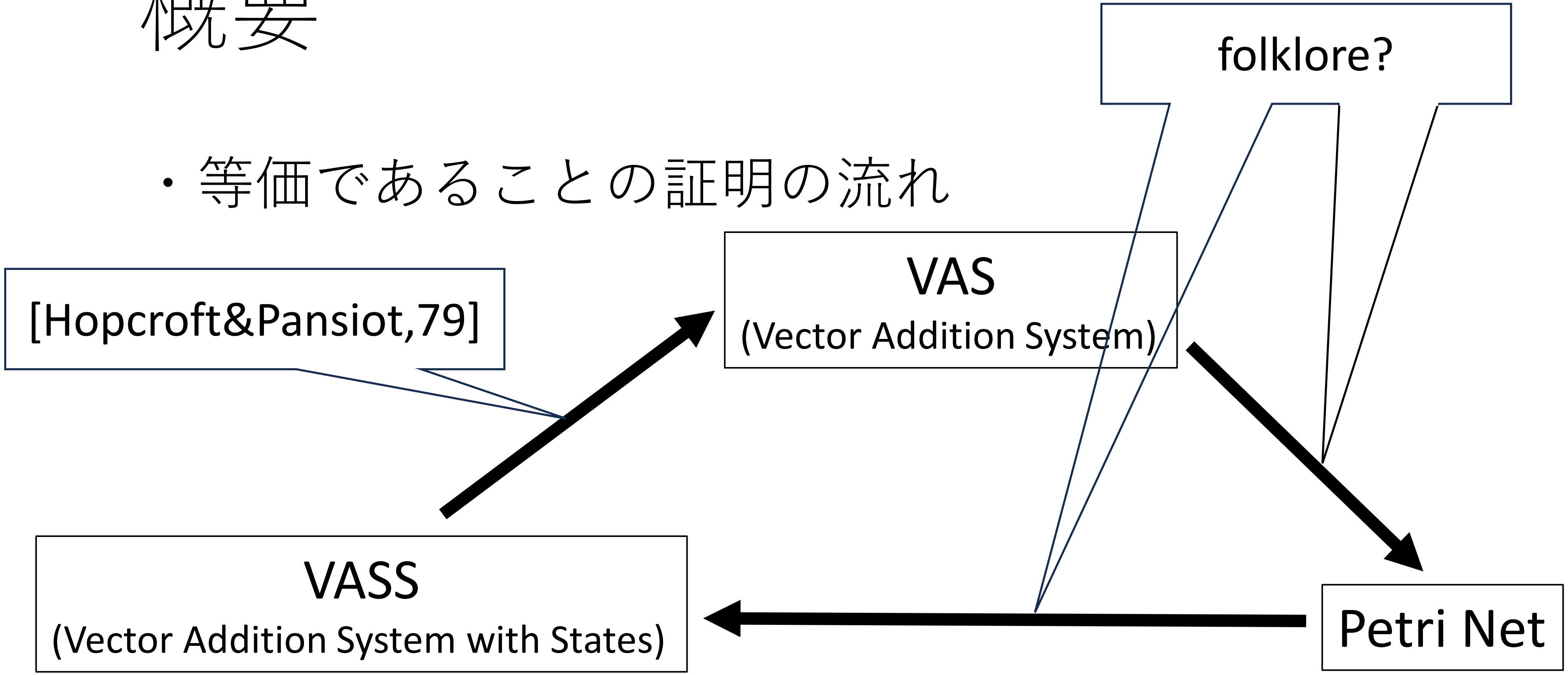
例：



・これらの状態遷移系は互いに等価であることが知られている

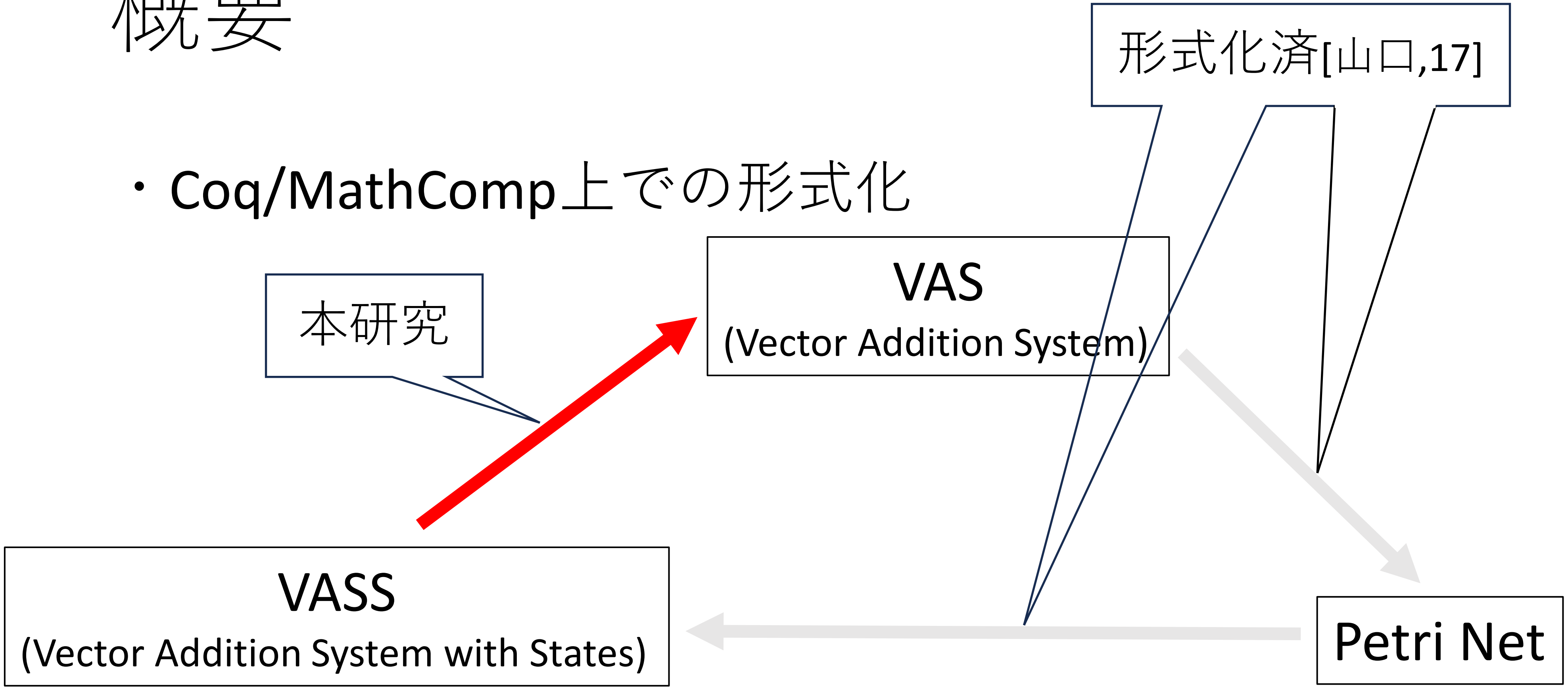
概要

・等価であることの証明の流れ



概要

・Coq/MathComp上での形式化



・さらにHopcroftらの変換に改良を与えた

１．VAS・VASSの定義とその形式化
２．VASSからVASへの変換とその形式化
３．変換の改良

１. VAS・VASSの定義とその形式化
２. VASSからVASへの変換とその形式化
３. 変換の改良

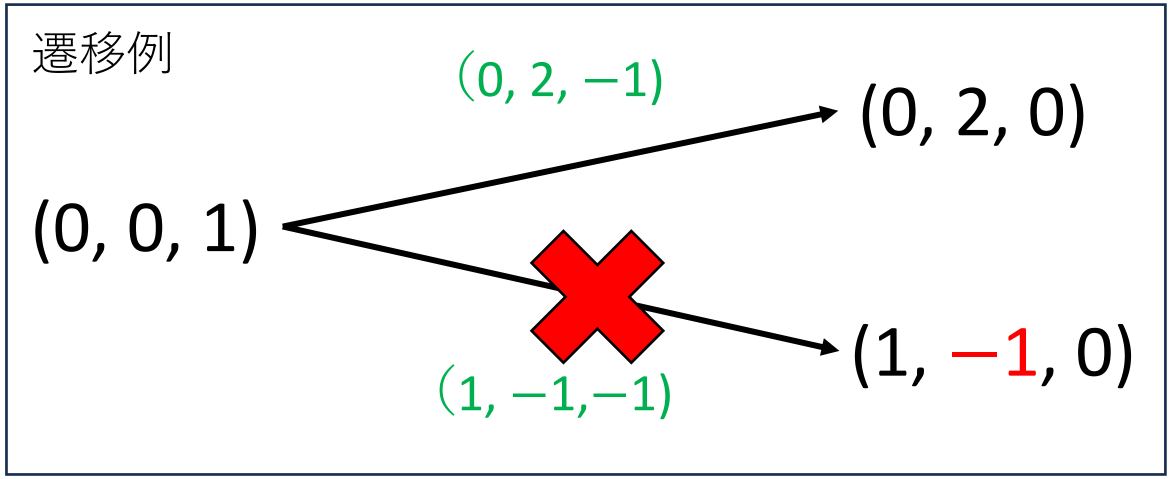
VAS(ベクトル加算系)の定義

・d次元VAS V ：$V \subseteq \mathbb{Z}^d,\ \#V < \infty$

状態： $\boldsymbol{m} \in \mathbb{N}^d$

遷移：$\boldsymbol{m} \xrightarrow{\boldsymbol{v}} \boldsymbol{m} + \boldsymbol{v}$　(条件：$\boldsymbol{v} \in V,\ \boldsymbol{m} + \boldsymbol{v} \in \mathbb{N}^d$)

VASの例：
V = {(0, 2, −1), (1, −1, −1), (0, 0, 2)}



VASの形式化

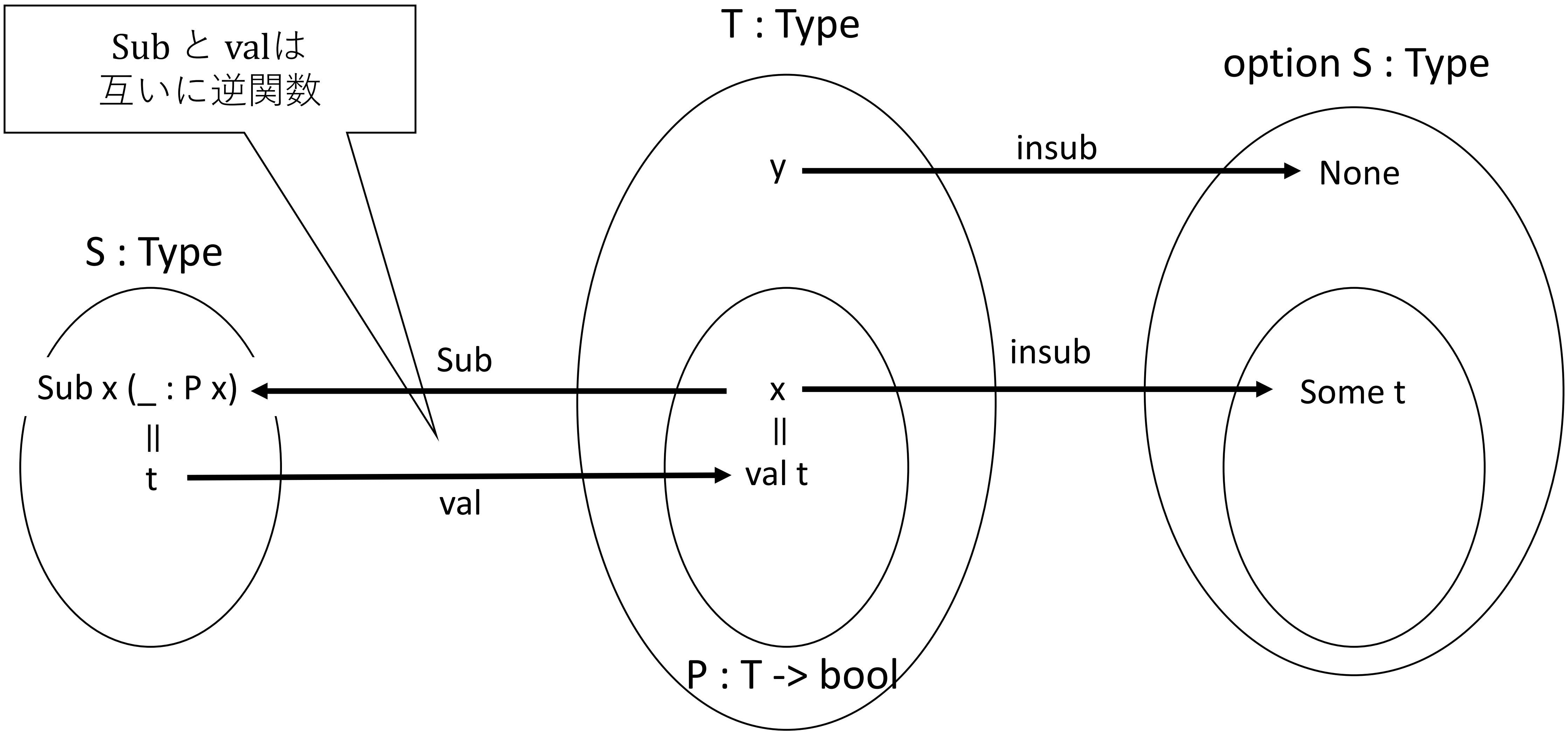
d次元VAS V ：$V \subseteq \mathbb{Z}^d$, $\#V < \infty$

状態：$\boldsymbol{m} \in \mathbb{N}^d$

遷移：$\boldsymbol{m} \xrightarrow{\boldsymbol{v}} \boldsymbol{m} + \boldsymbol{v}$ （条件：$\boldsymbol{v} \in V$, $\boldsymbol{m} + \boldsymbol{v} \in \mathbb{N}^d$）

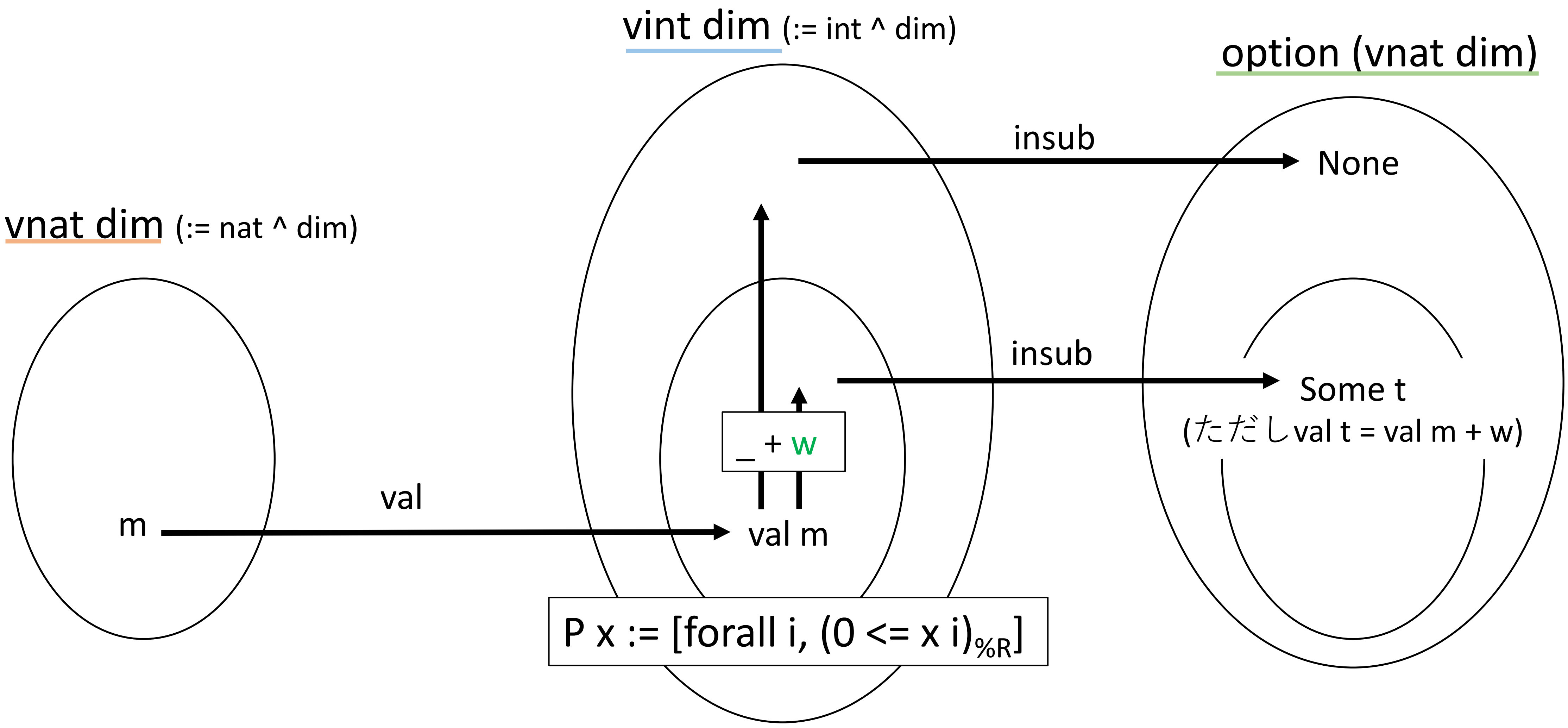
```
Variable dim : nat.
Definition VAS := {fset (vint dim)}.          (* vint dim := int ^ dim *)
Definition markingVAS := vnat dim.            (* vnat dim := nat ^ dim *)
Definition vtrans (m : vnat dim) (w : vint dim) : option (vnat dim)
 := insub (val m + w)%R.                      (* vnatはvintのSubTypeとして定義*)
Definition nextVAS {vas : VAS} (m : markingVAS) (v : vas)
 : option markingVAS := vtrans m (val v).
```

MathCompにおけるSubTypeのイメージ



SubTypeの例

Definition vtrans (m : vnat dim) (w : vint dim) : option (vnat dim)
:= insub (val m + w)%R.



VASSの定義

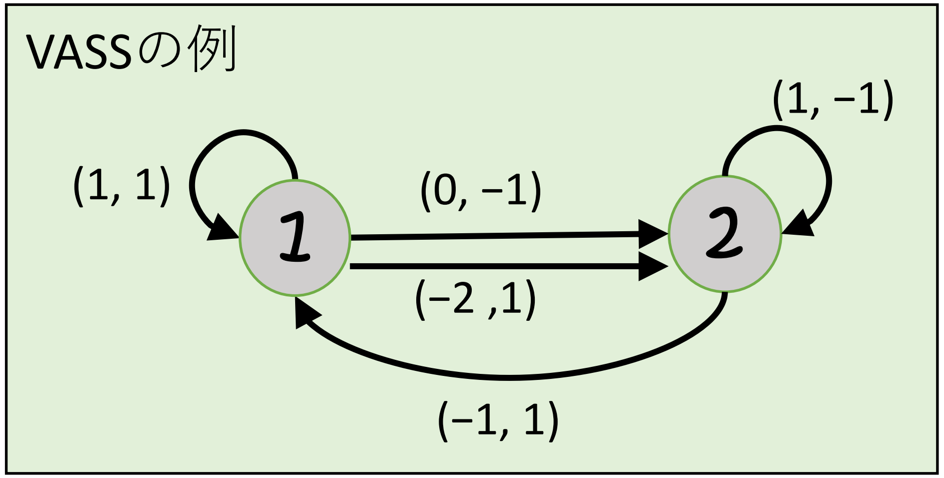
・d次元VASS (Q,T) ：$T \subseteq Q \times \mathbb{Z}^d \times Q,\ \#Q < \infty,\ \#T < \infty$

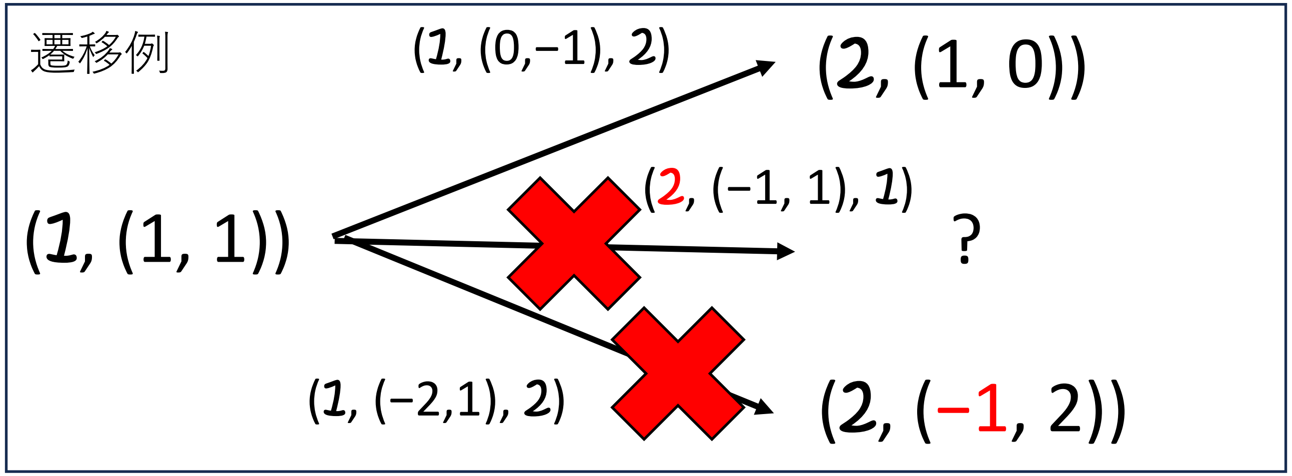
状態： $(q,\boldsymbol{m}) \in Q \times \mathbb{N}^d$

Configurationと呼ぶ

遷移： $(q,\boldsymbol{m}) \xrightarrow{(q,\boldsymbol{v},q')} (q',\boldsymbol{m}+\boldsymbol{v})$

(条件：$(q,\boldsymbol{v},q') \in T, \boldsymbol{m}+\boldsymbol{v} \in \mathbb{N}^d$)





VASSの形式化

d次元VASS (Q,T) ：$T \subseteq Q \times \mathbb{Z}^d \times Q$, $\#Q < \infty$, $\#T < \infty$

状態：$(q, \boldsymbol{m}) \in Q \times \mathbb{N}^d$

遷移：$(q, \boldsymbol{m}) \xrightarrow{(q, \boldsymbol{v}, q')} (q', \boldsymbol{m} + \boldsymbol{v})$ (条件：$(q, \boldsymbol{v}, q') \in T, \boldsymbol{m} + \boldsymbol{v} \in \mathbb{N}^d$)

```
Variables (dim : nat) (state : finType).
Definition VASS  := {fset (state * vint dim * state)}.
Definition confVASS : Type := state * vnat dim.
Definition nextVASS {vass : VASS} (c : confVASS) (w : vass)
 : option confVASS := let: (q, m) := c in let: (q1, v, q2) := val w in
  if q != q1 then None
  else if vtrans m v isn't Some t then None
  else (* (q == q1) && (vtrans m v == Some t) *) Some (q2, t).
```

１．VAS・VASSの定義とその形式化
２．VASSからVASへの変換とその形式化
３．変換の改良

VASSからVASへの変換[Hopcroft&Pansiot,79]

・ d次元VASSはある$a, b : Q \to \mathbb{N}$を用いてd+3次元VASで模倣可能

VASSの遷移

$$(q, \boldsymbol{m}) \xrightarrow{(q, \boldsymbol{v}, q')} (q', \boldsymbol{m} + \boldsymbol{v})$$

$Q = \{1, \ldots, k\}$,
$a_i = i, b_i = (k+1)(k - i + 1)$,
$\tilde{q} = k + 1 - q$ と定める

VASの遷移

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\boldsymbol{0} \mid (-a_q, a_{\tilde{q}} - b_q,\ b_{\tilde{q}}))} (\boldsymbol{m} \mid (0, a_{\tilde{q}}, b_{\tilde{q}}))$$

$$\xrightarrow{(\boldsymbol{0} \mid (b_q, -a_{\tilde{q}}, a_q - b_{\tilde{q}}))} (\boldsymbol{m} \mid (b_q, 0, a_q))$$

$$\xrightarrow{(\boldsymbol{v} \mid (a_{q'} - b_q,\ b_{q'}, -a_q))} (\boldsymbol{m} + \boldsymbol{v} \mid (a_{q'}, b_{q'}, 0))$$

ベクトルの連結

Hopcroftらの証明

・[Hopcroft&Pansiot,79]より

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 $b_m + a_j - b_i \leq a_j - (k+1) < 0$. $\square$

Since an n-dim VASS can trivially simulate an n-dim VAS, the reachability problem for VAS is solvable if and only if the reachability problem for VASS is solvable.

なぜこの変換？

$$Q=\{1,\dots,k\},$$
$$a_i = i, b_i = (k+1)(k-i+1),$$
$$\tilde{q} = k+1-q$$

・適用できる遷移を1種類のみにするため

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\mathbf{0} \mid (-a_q, a_{\tilde{q}} - b_q, b_{\tilde{q}}))} (\boldsymbol{m} \mid (b_q, 0, a_q))$$

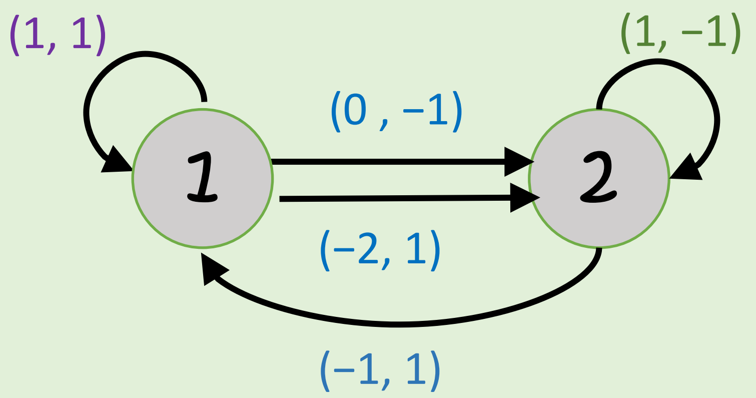
$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\mathbf{0} \mid (b_q, -a_{\tilde{q}}, a_q - b_{\tilde{q}}))} (\boldsymbol{m} \mid (a_q + b_q, b_q - a_{\tilde{q}}, a_q - b_{\tilde{q}}))$$

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\boldsymbol{v} \mid (a_{q'} - b_q, b_{q'}, -a_q))} (\boldsymbol{m} + \boldsymbol{v} \mid (a_q + a_{q'} - b_q, b_q + b_{q'}, -a_q))$$

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow[(q' \neq q)]{(\mathbf{0} \mid (-a_{q'}, a_{\tilde{q}} - b_{q'}, b_{\tilde{q}}))} (\boldsymbol{m} \mid (a_q - a_{q'}, a_{\tilde{q}} + b_q - b_{q'}, b_{\tilde{q}}))$$

VASSからVASへの変換例

$Q = \{1, \dots, k\}$,
$a_i = i, b_i = (k+1)(k-i+1)$,
$\tilde{q} = k+1-q$ と定める



変換

$a_i = i, b_i = 3(3-i), \tilde{q} = q$
{(0,0,−1,−5,6), (0,0,−2,−1,3),
(0,0,6,−1,−5), (0,0,3,−2,−1),
(1,1,−5,6,−1),
(0, −1,−4,3,−1), (−2, 1,−4,3,−1),
(1,−1,−1,3,−2), (−1,1,−2,6,−2)}

VASSの遷移

$(1, (1, 1)) \xrightarrow{(1, (0, -1), 2)} (2, (1, 0))$

VASの遷移

$(1, 1, \underset{a_1}{1}, \underset{b_1}{6}, 0) \xrightarrow{(0, 0, -1, -5, 6)} (1, 1, 0, \overset{a_1}{1}, \overset{b_1}{6})$

$\xrightarrow{(0, 0, 6, -1, -5)} (1, 1, \overset{b_1}{6}, 0, \overset{a_1}{1})$

$\xrightarrow{(0, -1, -4, 3, -1)} (1, 0, \underset{a_2}{2}, \underset{b_2}{3}, 0)$

形式化の準備

ここではa,bを具体的に定めず、
$\tilde{q} = q$ とする

```
Variables (dim : nat) (state : finType) (a b : state –> nat). (* vrotrは自分で定義したベクトル右回転関数 *)

Definition vs (p : state) (i : 'Z_3) : vnat 3 :=
vrotr i [ffun j : 'Z_3 => if j == 0%R then a p else if j == 1%R then b p else (* j == 2%R *) 0].

Definition vst (p q : state) (i : 'Z_3) : vint 3 :=
vrotr i [ffun j : 'Z_3 => if j == 0%R then – (a p)%:Z
                          else if j == 1%R then (a q)%:Z – (b p)%:Z
                          else (* j == 2%R *) (b q)%:Z]%R.
```

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\boldsymbol{0} \mid (-a_q,\, a_{\tilde{q}} - b_q,\, b_{\tilde{q}}))} (\boldsymbol{m} \mid (0, a_{\tilde{q}}, b_{\tilde{q}}))$$

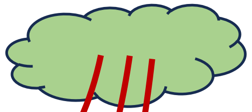
$$\xrightarrow{(\boldsymbol{0} \mid (b_q,\, -a_{\tilde{q}},\, a_q - b_{\tilde{q}}))} (\boldsymbol{m} \mid (b_q, 0, a_q))$$

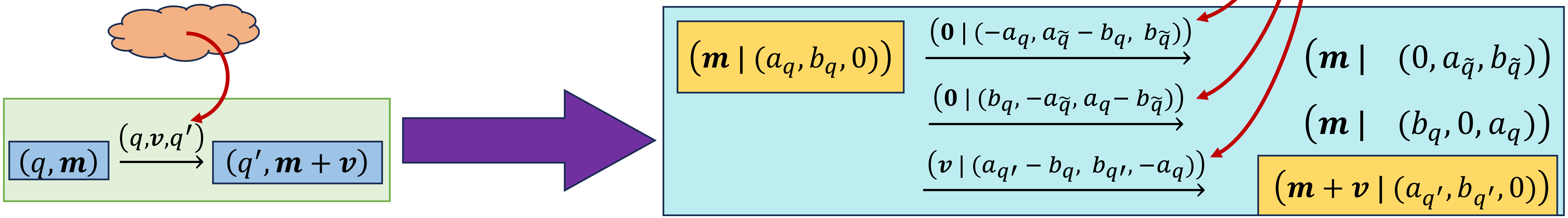
$$\xrightarrow{(\boldsymbol{v} \mid (a_{q'} - b_q,\, b_{q'},\, -a_q))} (\boldsymbol{m}+\boldsymbol{v} \mid (a_{q'}, b_{q'}, 0))$$

i = 0
i = 0
i = 1
i = 1
i = 2
i = 2
i = 0

VASSからVASへの変換の形式化

```
Definition VAS_of_VASS_m (c : confVASS dim state)  : markingVAS (dim + 3)  :=
 let: (q, m) := c in vcat m (vs q 0%R).                (* vcatはベクトルの連結 *)
Definition VAS_of_VASS_t (vass : VASS dim state) : VAS (dim + 3) :=
 [fset vcat (0%R : vint dim) (vst q q (x%:R)%R) | x : bool, q : state]
 `|` [fset let: (q,v,q') := t in vcat v (vst q q' 2%R) | t in vass].
```



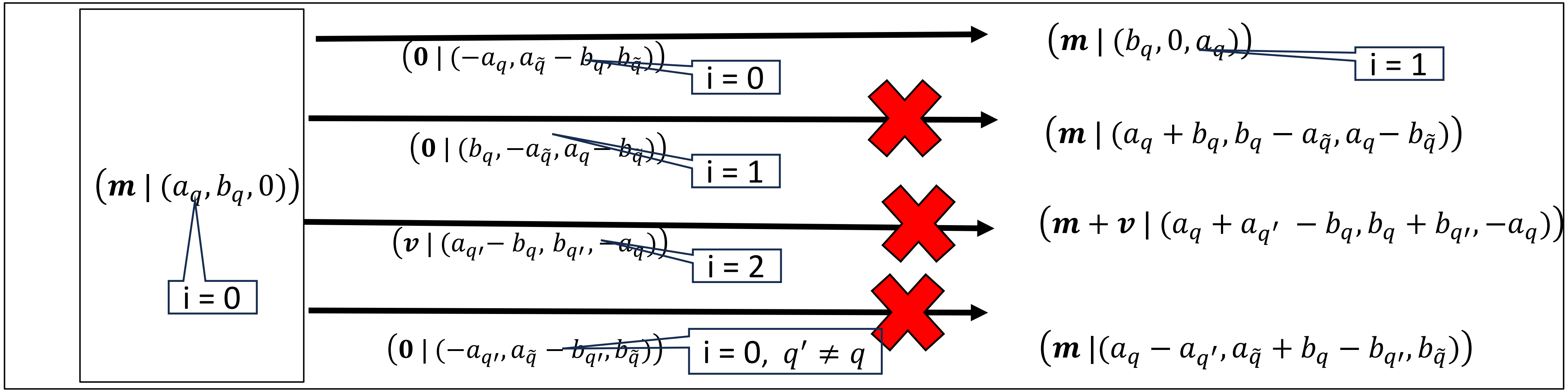


遷移の制御のためのa,bに関する条件

Definition ab_consistent (a b : state -> nat) := ∀(q q' q'' : state) (i i' : 'Z_3),

vtrans (vs a b q i) (vst a b q' q'' i') =

if (q' == q) && (i' == i) then Some (vs a b q'' $(i + 1)_{\%R}$) else None.



到達可能性の保存

```
Definition reachable {S T : Type} (next : S –> T –> option S) (x0 x : S) :=
 ∃ s : seq T, foldm next x0 s = Some x.  (* foldmはモナド版foldl *)
Lemma reachable_VASS_VAS (vass : VASS dim state)  (c0 c : confVASS dim state)
(a b : nat –> state) :
 ab_consistent a b –>
   reachable (@nextVASS _ _ vass) c0 c <–>          (* ←を帰納法で示すには一般化が必要 *)
   reachable (@nextVAS _ (VAS_of_VASS_t a b vass))
                          (VAS_of_VASS_m a b c0) (VAS_of_VASS_m a b c).
```

一般化した補題

Lemma VASS_of_VAS_reachable' (c_0 : confVASS dim state) (vass : VASS dim state)
(vm : markingVAS (dim+3)) (a b : nat -> state) :
ab_consistent a b –>
reachable (@nextVAS _ (VAS_of_VASS_t a b vass)) (VAS_of_VASS_m a b c_0) vm ->
∃ q m i, vm = vcat m (vs a b q i) ∧ reachable (@nextVASS _ _ vass) c_0 (q,m).

元の命題

reachable (@nextVAS _ (VAS_of_VASS_t a b vass))
(VAS_of_VASS_m a b c_0) (VAS_of_VASS_m a b c)
–> reachable (@nextVASS _ _ vass) c_0 c.

1. VAS・VASSの定義とその形式化
2. VASSからVASへの変換とその形式化
3. 変換の改良

条件の改良

問：Hopcroftらの論文で定めたa,bがab_consistentを満たすのか？
→そのままだと証明が難しいので、同値な条件を用意する

```
Variable state : finType.
Definition ab_aligned (a b : state –> nat) :=
 injective a
 ∧ (∀ p q : state, a p > a q –> ∀ r : state, a r + b p < b q)
 ∧ ∀ p q : state, a p < b q.
```

条件の同値性

Lemma ab_iff (a b : state –> nat) :
#|state| > 1 –> ab_consistent a b <–> ab_aligned a b.

条件#|state| > 1 が必要

ab_consistent

∀ (p p' q : state) (i i' : 'Z_3),
vtrans (vs a b p i) (vst a b p' q i') =
if (p' == p) && (i' == i) then Some (vs a b q $(i + 1)_{\%R}$) else None.

ab_consistentの
回転数を１つ固定

ab_consistent0

∀ (p p' q : state) (i' : 'Z_3),
vtrans (vs a b p $0_{\%R}$) (vst a b p' q i') =
if (p' == p) && (i' == $0_{\%R}$) then Some (vs a b q $1_{\%R}$) else None.

ab_aligned

injective a
∧ (∀ p q : state, $a_p > a_q$ -> ∀ r : state, $a_r + b_p < b_q$)
∧ ∀ p q : state, ap < b_q

Hopcroftらの a, b の形式化

```
(* Hopcroft&Pansiot: Q = {1, … , k}, a_i = i, b_i = (k + 1)(k + 1 − i) *)
Definition a_HP (q : state) : nat := (enum_rank q).+1.
Definition b_HP (q : state) : nat := #|state|.+1 * (#|state| − enum_rank q).
Lemma HPab_prop : ab_aligned a_HP b_HP.
```

問：この a, b が（何らかの意味で）最良か？

参考図(stateはaに関して昇順)

```
Definition ab_aligned (a b : state –> nat) :=
 injective a
 ∧ (∀ p q : state, a_p > a_q –> ∀ r : state, a_r + b_p < b_q)
 ∧ ∀ p q : state, a_p < b_q.
```

Hopcroft&Pansiotの変換では<

$$
\begin{aligned}
0 &\le a_1 < \cdots < a_k < b_k \\
&\le b_k + a_1 < \cdots < b_k + a_k < b_{k-1} \\
&\le b_{k-1} + a_1 < \cdots < b_{k-1} + a_k < b_{k-2} \\
&\vdots \\
&\le b_2 + a_1 < \cdots < b_2 + a_k < b_1
\end{aligned}
$$

ab_alignedを満たすa,b

(* Hopcroft&Pansiot: $Q = \{1, \ldots, k\}$, $a_i = i$, $b_i = (k+1)(k + 1 - i)$ *)

```
Definition a_HP (q : state) : nat := (enum_rank q).+1.
Definition b_HP (q : state) : nat := #|state|.+1 * (#|state| − enum_rank q).
Lemma HPab_prop : ab_aligned a_HP b_HP.
```

a_HP, b_HP

$a_i = i, b_i = 3(3 - i)$
{(0,0,−1,−5,6), (0,0,−2,−1,3), (0,0,6,−1,−5), (0,0,3,−2,−1), (1,1,−5,6,−1), (0,−1,−4,3,−1), (−2,1,−4,3,−1), (1,−1,−1,3,−2), (−1,1,−2,6,−2)}

(* $Q = \{1, \ldots, k\}$, $a_i = i - 1$, $b_i = k(k + 1 - i)$ のとき最小*)

```
Definition mina (q : state) : nat := enum_rank q.
Definition minb (q : state) : nat := #|state| * (#|state| − enum_rank q).
Lemma minab_prop : ab_aligned mina minb.
```

・実際に最小になることも示した

mina, minb

$a_i = i - 1, b_i = 2(3 - i)$
{(0,0,**0**,−4,4), (0,0,−1,−1,2), (0,0,4,**0**,−4), (0,0,2,−1,−1), (1,1,−4,4,**0**), (0,−1,−3,2,**0**), (−2,1,−3,2,**0**), (1,−1,−1,2,−1), (−1,1,−2,4,−1)}

minabの補題

aの昇順になるようstateを整列したもの

```
Definition sorted_state := sort (relpre a leq) (enum state).
Definition sorted_a := map a sorted_state.
Lemma a_ith_geq : ∀ i, i < #|state| -> i <= nth 0 sorted_a i.
Lemma b_ith_geq : ∀ i, i < #|state| ->
 #|state| * (#|state| - i) <= nth 0 sorted_b i.
```

気持ち： ∀ q : sorted_state, mina q <= a q

気持ち： ∀ q : sorted_state, minb q <= b q

なぜこの変換？（再掲）

$a_q = 0$ を許すことにより、
ここが負にならない可能性がある

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\mathbf{0} \mid (-a_q,\, a_{\tilde{q}} - b_q,\, b_{\tilde{q}}))} (\boldsymbol{m} \mid (b_q, 0, a_q))$$

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\mathbf{0} \mid (b_q,\, -a_{\tilde{q}},\, a_q - b_{\tilde{q}}))} (\boldsymbol{m} \mid (a_q + b_q,\, b_q - a_{\tilde{q}},\, a_q - b_{\tilde{q}}))$$

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow{(\boldsymbol{v} \mid (a_{q'} - b_q,\, b_{q'},\, -a_q))} (\boldsymbol{m} + \boldsymbol{v} \mid (a_q + a_{q'} - b_q,\, b_q + b_{q'},\, -a_q))$$

$$(\boldsymbol{m} \mid (a_q, b_q, 0)) \xrightarrow[(q' \neq q)]{(\mathbf{0} \mid (-a_{q'},\, a_{\tilde{q}} - b_{q'},\, b_{\tilde{q}}))} (\boldsymbol{m} \mid (a_q - a_{q'},\, a_{\tilde{q}} + b_q - b_{q'},\, b_{\tilde{q}}))$$

条件の同値性（再掲）

$a_q = 0$ を許すことにより、
ここの証明で場合分けが増える

Lemma ab_iff (a b : state –> nat) :

#|state| > 1 –> ab_consistent a b <–> ab_aligned

ab_consistent

∀ (p p' q : state) (i i' : 'Z_3),
vtrans (vs a b p i) (vst a b p' q i') =
if (p' == p) && (i' == i) then Some (vs a b q ($_{\%R}$) else None.

ab_consistent0

∀ (p p' q : state) (i' : 'Z_3),
vtrans (vs a b p $0_{\%R}$) (vst a b p' q i') =
if (p' == p) && (i' == $0_{\%R}$) then Some (vs a b q $1_{\%R}$) else None.

ab_aligned

injective a
∧ (∀ p q : state, $a_p > a_q$ -> ∀ r : state, $a_r + b_p < b_q$)
∧ ∀ p q : state, ap< b_q

$a_q = 0$を許すことによる証明の変化

```
Lemma ab_aligned_vs0 : ab_aligned -> ab_consistent0.
Proof.
 rewrite /ab_consistent0 /ab_aligned => -[inj_a [a_gt_bpbq a_gt_b]] p p' q i.
 case: ifP.
  move/andP => [/eqP <- /eqP ->].
  rewrite /vtrans; case: insubP=> [w|].
   move/forallP => H1 H2; congr Some.
   apply: val_inj; rewrite H2; apply/ffunP=
   by case: (Z3_cases k) => -> /=; lia.
  rewrite negb_forall; move/existsP=> -[j].
  rewrite !ffunE.
  by case: (Z3_cases j) => -> /=; lia.
 move/negP/negP; rewrite Bool.negb_andb; mo
 rewrite /vtrans insubN // negb_forall.
```

```
apply/existsP; move: H; case: (Z3_cases i) => [-> []|-> _|-> _] //.
   case: (ltngtP (a p) (a p')) => [h _|h _|h].
      by exists 0%R; rewrite !ffunE /=; lia.
     by exists 1%R; rewrite !ffunE /=; move: (a_gt_bpbq _ _ h q); lia.
    by move: (inj_a _ _ h)=> ->; rewrite eqxx.
  by exists 2%R; rewrite !ffunE /=; move: (a_gt_b q p'); lia.
case E: (a p') => [|n];last by exists 2%R; rewrite !ffunE /=; lia.
 exists 0%R; rewrite !ffunE /=.
 case E': (a q) => [|m]; first by move: (a_gt_b p p'); lia.
 have h : a p' < a q by rewrite E E'.
 by move: (a_gt_b q q) (a_gt_bpbq q p' h p); lia.
Qed.
```

今後の展望

・今回は被覆性の形式化を行わなかった

→Hierarchy Builder[HB,20]を用いることで簡潔に形式化？

・コードの改良

→$\tilde{q}$が任意の置換で成り立つことの証明

本研究のコード:
https://github.com/Wakisaka1205/VASS2VAS-1x

まとめ

VASSからVASへの変換を形式化した

- Hopcroft&Pansiotが示した変換に基づいている
- 遷移を模倣できる条件ab_consistentを定め、この条件下で到達可能性が保存されることを示した

変換の改良を行った

- ab_consistentを満たす a, b を不等式で特徴付けた
- 改良した a, b (a_q = 0を許す) を定め、その最小性を示した

参考文献

[Karp＆Miller,69] Richard M. Karp and Raymond E. Miller. “Parallel Program Schemata”. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[Hopcroft＆Pansiot,79] John Hopcroft and Jean-Jacques Pansiot. “On the reachability problem for 5-dimensional vector addition systems”. *Theoretical Computer Science*, 8(2):135–159, 1979.

[HB,20] Cyril Cohen, Kazuhiko Sakaguchi and Enrico Tassi. “Hierarchy Builder: Algebraic hierarchies Made Easy in Coq with Elpi”. *FSCD 2020*, volume 167 of LIPIcs, pages 34:1–34:21, 2020.

[山口, 17] 山口智也．「ベクトル加算系・状態付きベクトル加算系・ペトリネットの等価性の形式化」. 千葉大学修士論文, 2017.