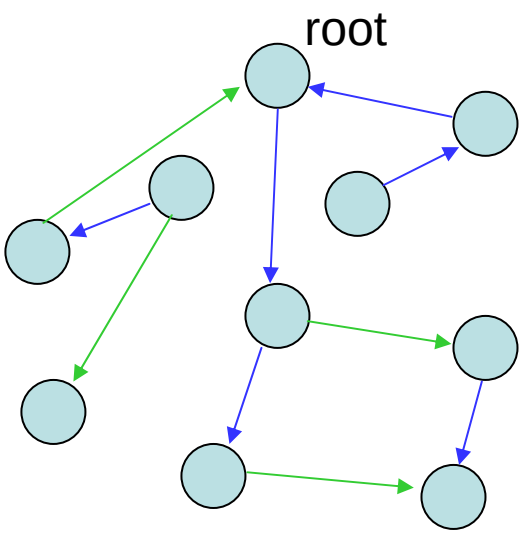
述語抽象化によるアルゴリズムの検証
～ シェープ解析を例として

PPL サマースクール
2006年9月12日

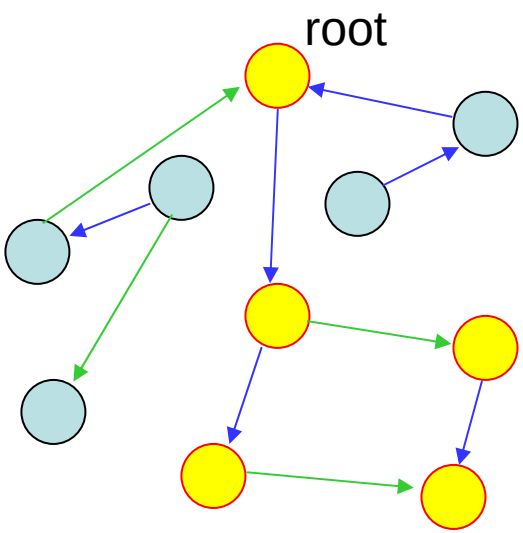
産業技術総合研究所 システム検証研究センター
東京大学大学院情報理工学系研究科
田辺良則

- 投影されるスライドは，お手元の配付資料から更新されたものになっています．
- このスライド (またはさらに更新された版) は，以下のURLからダウンロードできます:
  http://staff.aist.go.jp/tanabe.yoshinori/06/09/12/

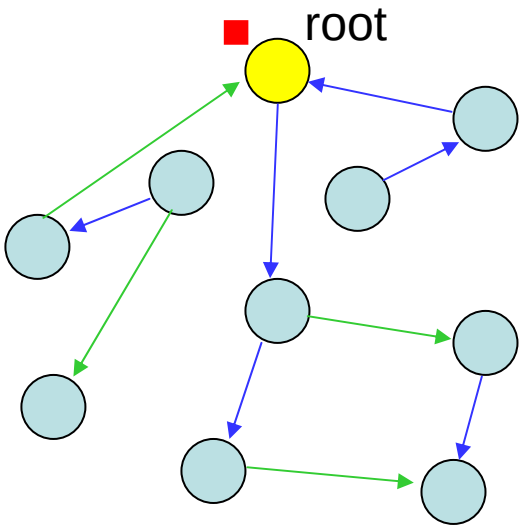
マーキングアルゴリズム



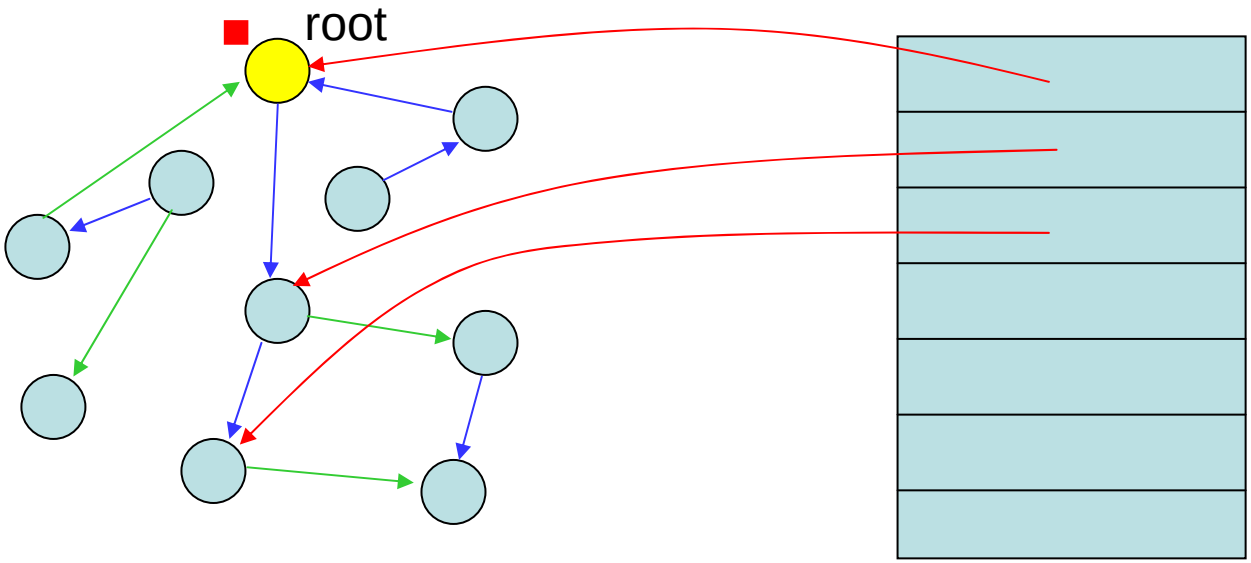
マーキングアルゴリズム



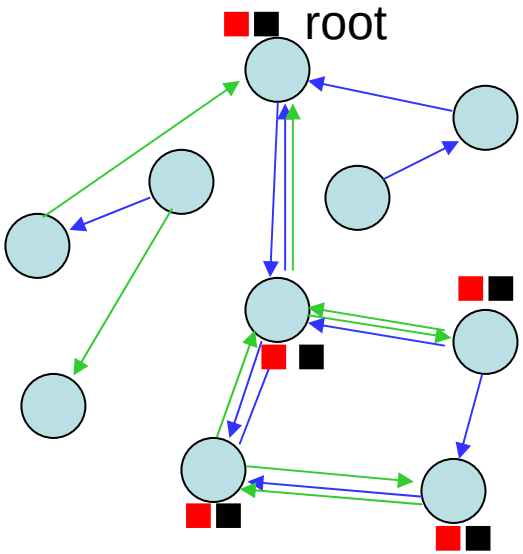
深さ優先探索マーキングアルゴリズム



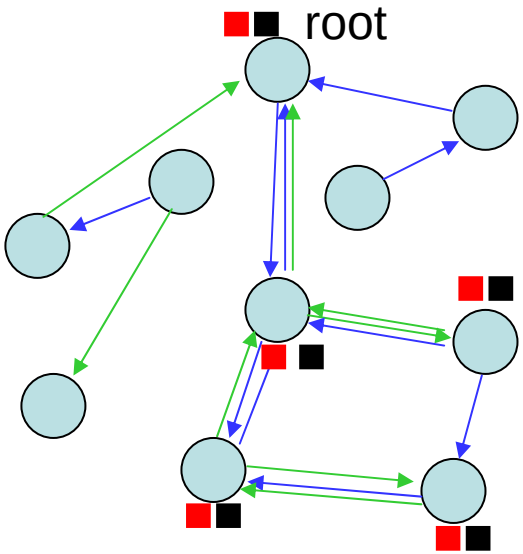
深さ優先探索マーキングアルゴリズム



Deutcsh-Schorr-Waiteマーキングアルゴリズム



Deutcsh-Schorr-Waiteマーキングアルゴリズム




```
void dsw(node root) {
  node t = root;
  node p = NULL;
  while (p != NULL || (t != NULL && ! t->m))
   {
    if (!(t == NULL || t->m)) {  /* push */
      node q = p;
      p = t; t = t->l;
      p->l = q; p->m = 1; p->c = 0;
    }else if (! p->c) {          /* swing */
      node q = t;
      t = p->r; p->r = p->l;
      p->l = q; p->c = 1;
    }else {                      /* pop */
      node q = t;
      t = p; p = p->r; t->r = q;
    }
  }
}
```

本日の内容

- 導入: DSWマーキングアルゴリズム
- 背景: 検証，モデル検査，...
- 述語抽象化によるソースコード検証
  - 遷移空間と抽象化
  - 述語抽象化とその正当性
  - 反例に基づく抽象構造の詳細化
- シェープ解析
  - 3値論理
  - 抽象化とその正当性
  - ツールTVLA

対象: 一般的なCプログラム

対象: DSWのような「リンク構造」

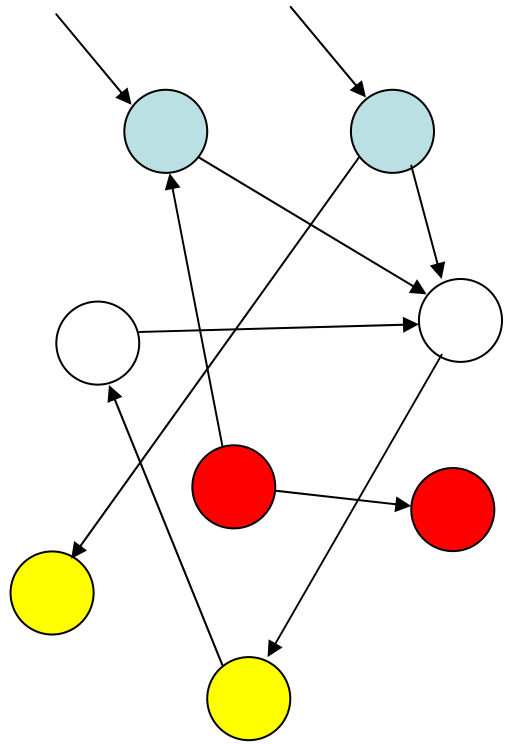
背景: 検証，モデル検査，ソースコード検証...

「検証」へのアプローチ

- 対話型検証
  - 検証すべき問題を数学的に定式化.
  - 定理証明器による支援
  - 高い専門的知識が要求される.
- 自動検証
  - 検証すべき問題を，有限個の空間の探索問題に落とす.
  - 代表例: モデル検査
  - 適用できる範囲に限りはあるが，「ボタン一つで」答が出る.
- 混合

モデル検査

- 状態遷移系: 有限個 (でも，巨大かも) の「状態」とそれらの間の「遷移関係」，「初期状態」が指定されている.
- 遷移に関する性質の検証.
  - 赤には決して到達しない.
  - 黄色に行くまでには白を必ず通る
- 全数探索
- 効率的なアルゴリズムが発達
  - 状態数10^{120}くらいは実績がある.



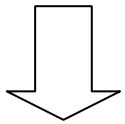
自動検証方式によるソースコード検証

- 基本的には，モデル検査に持ち込む．
- 状態数爆発
  - int型の変数が10個あったら，状態数は$(2^{32})^{10}$
- 抽象化によって状態数を減らす．
  - たくさんの状態をひとつの状態にまとめて，小さな状態遷移系にする．

- まとめた後の遷移関係は?
- 「小さな」遷移系で検査をして，ソースコードを検査したことになる?

アルゴリズムの検証

- アルゴリズムをコード / 擬似コードで書く.



- ソースコード検証

実は必ずしもイコールではないが....

述語抽象化によるソースコード検証

Running Example

```
Example() {
  do {
    lock();
    old = new;
    q = q->next;
    if (q != NULL) {
      q->data = new;
      unlock();
      new++;
    }
  } while (new != old);
  unlock();
  return;
}
```

- ロック済みのときに再度ロックしないか?
- ロックしていないときにロック解除をしないか?

Running Example

```
Example() {
  LOCK = 0
  do {
    lock();
    old = new;
    q = q->next;
    if (q != NULL) {
      q->data = new;
      unlock();
      new++;
    }
  } while (new != old);
  unlock();
  return;
}
```

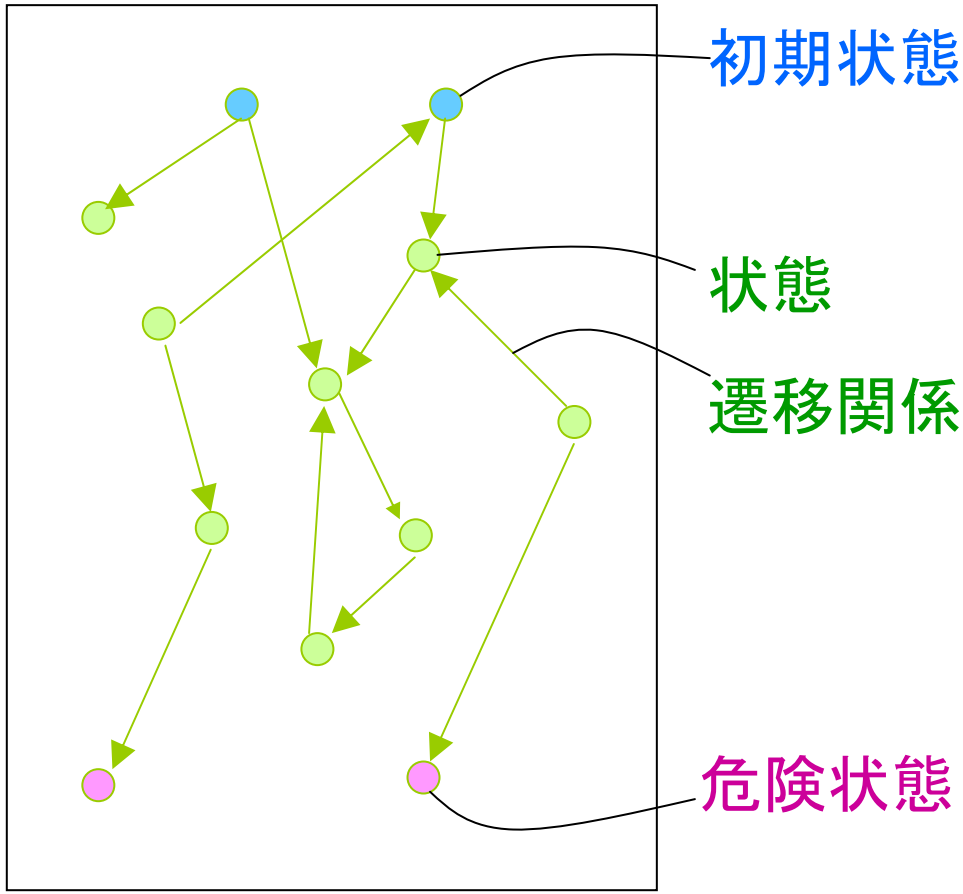
```
void lock() {
  if (LOCK == 1) {
ERROR:
  }
  LOCK = 1;
}

void unlock() {
  if (LOCK == 0) {
ERROR:
  }
  LOCK = 0;
}
```

ERRORラベルに到達しない

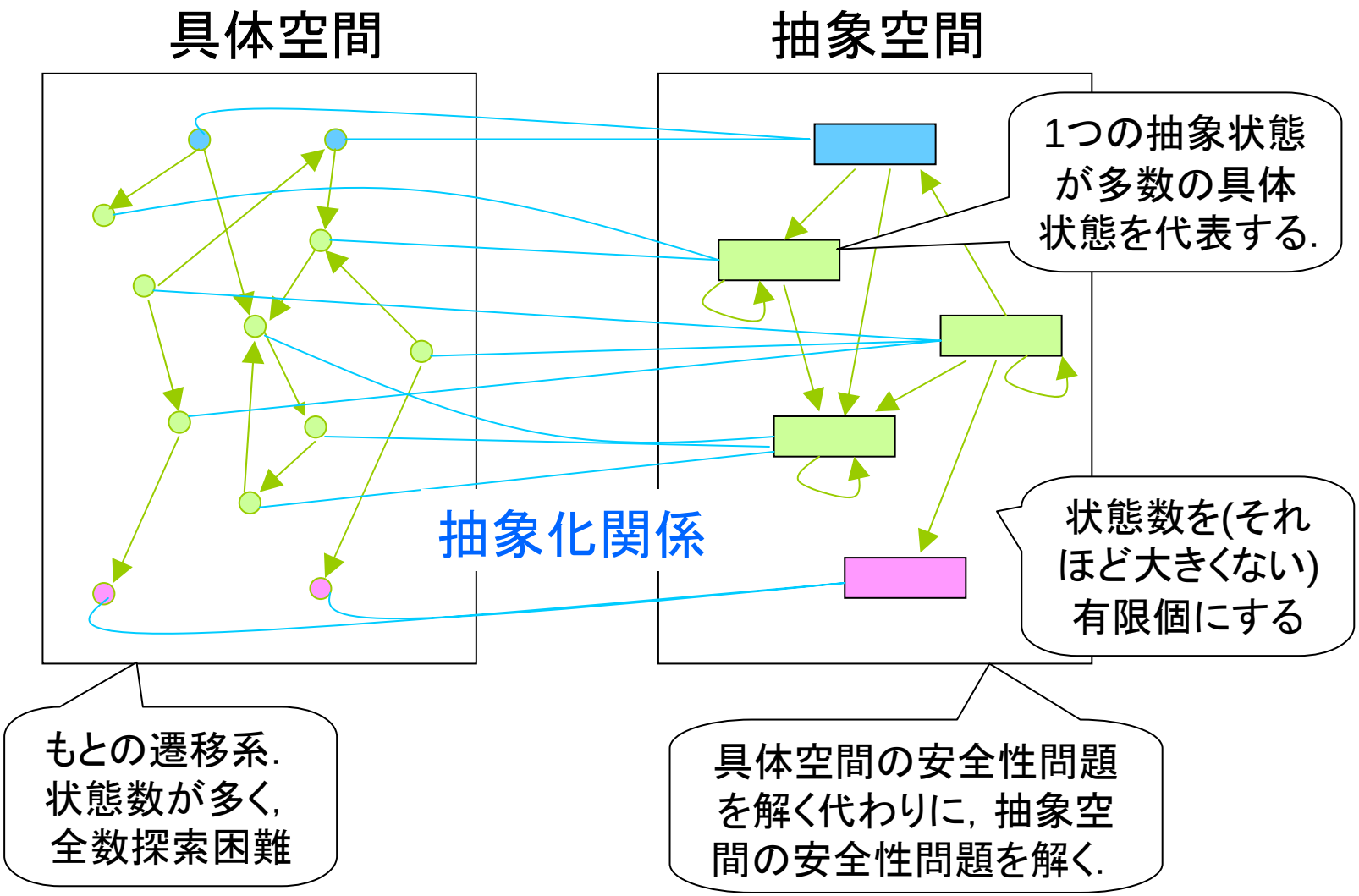
遷移系の安全性問題

遷移系

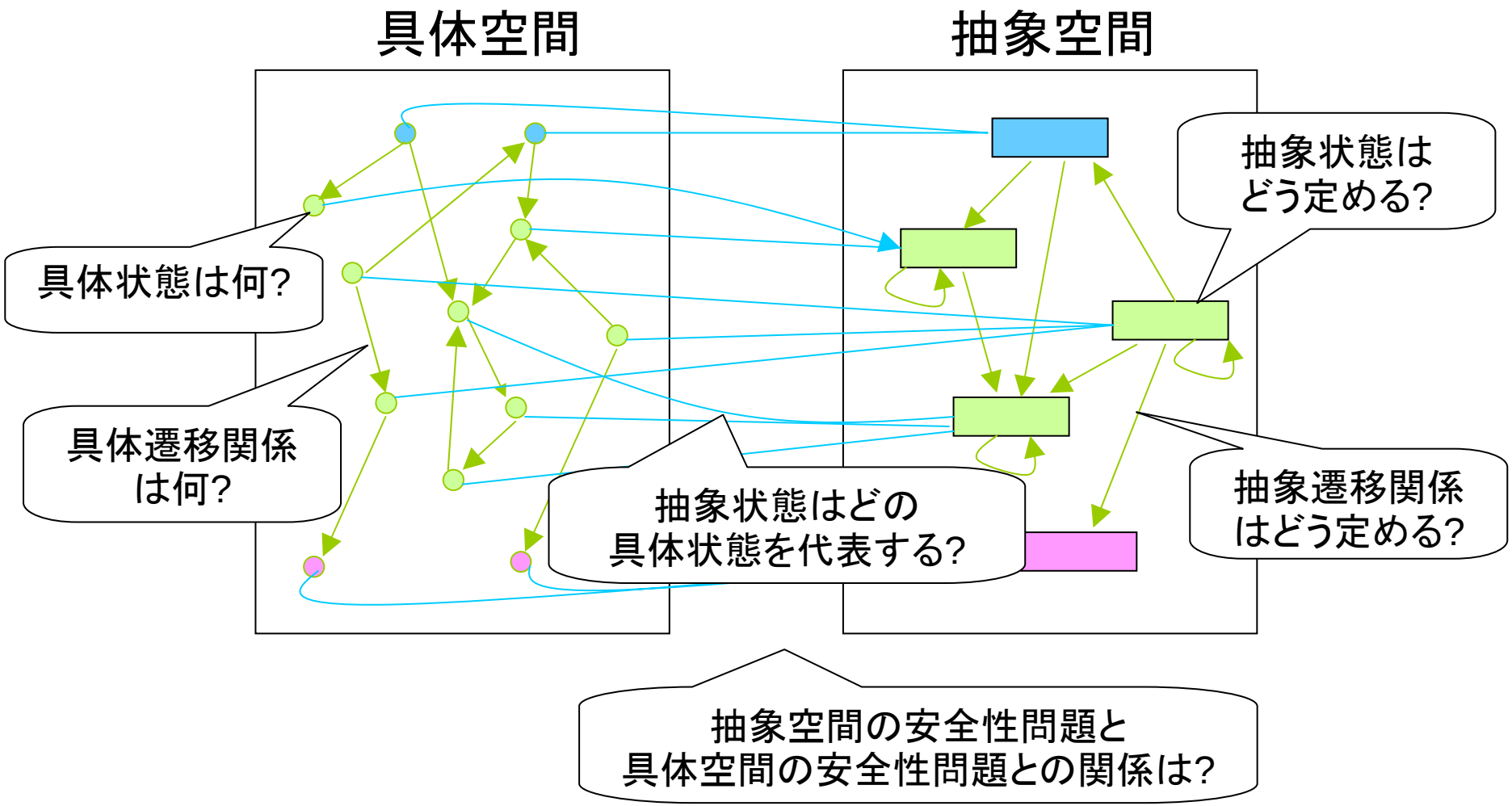


- 安全性問題:
  初期状態から危険状態に達するか?
- 多数/無限個の状態 ===> 全数調査は不可能

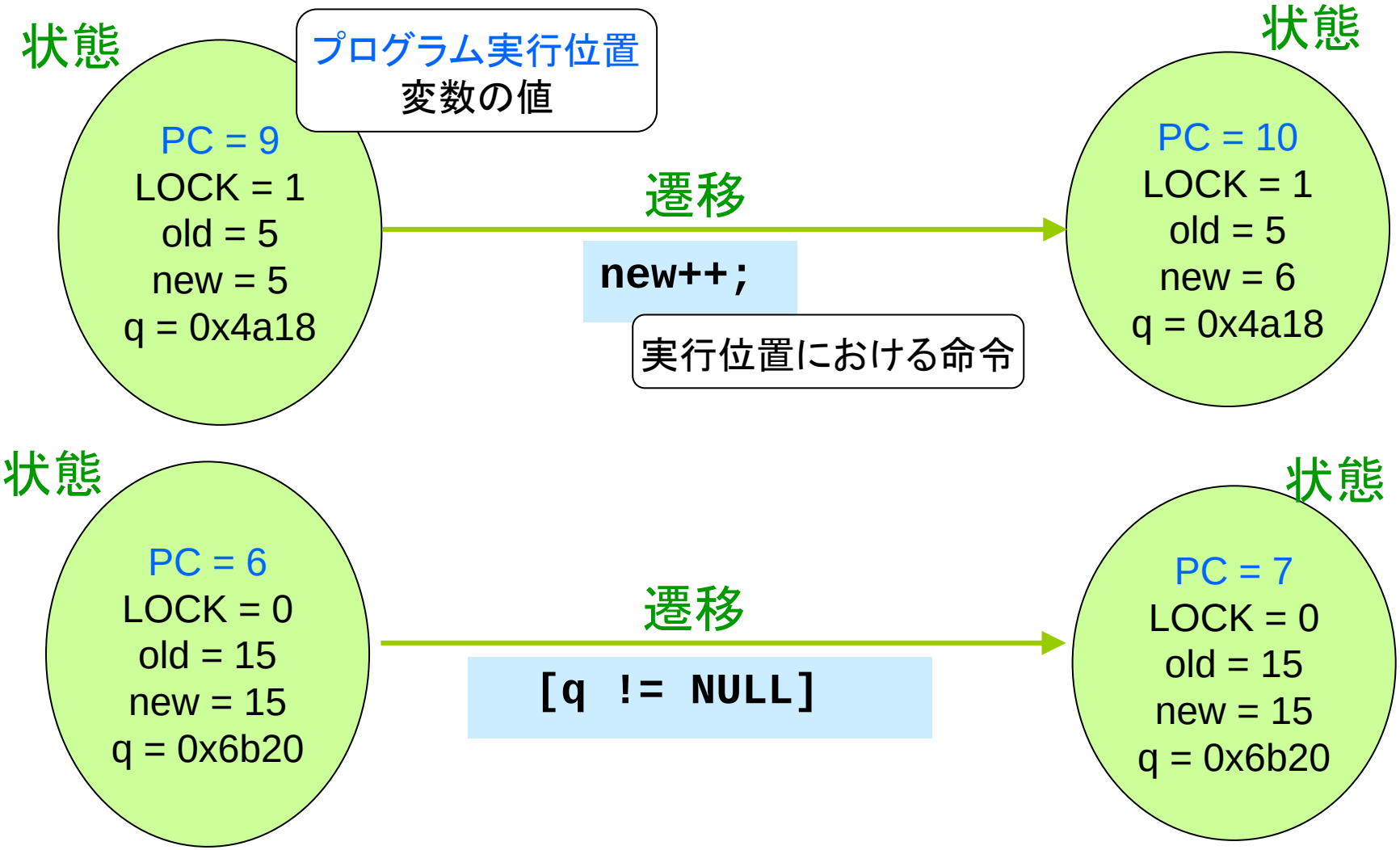
安全性問題の抽象化による解



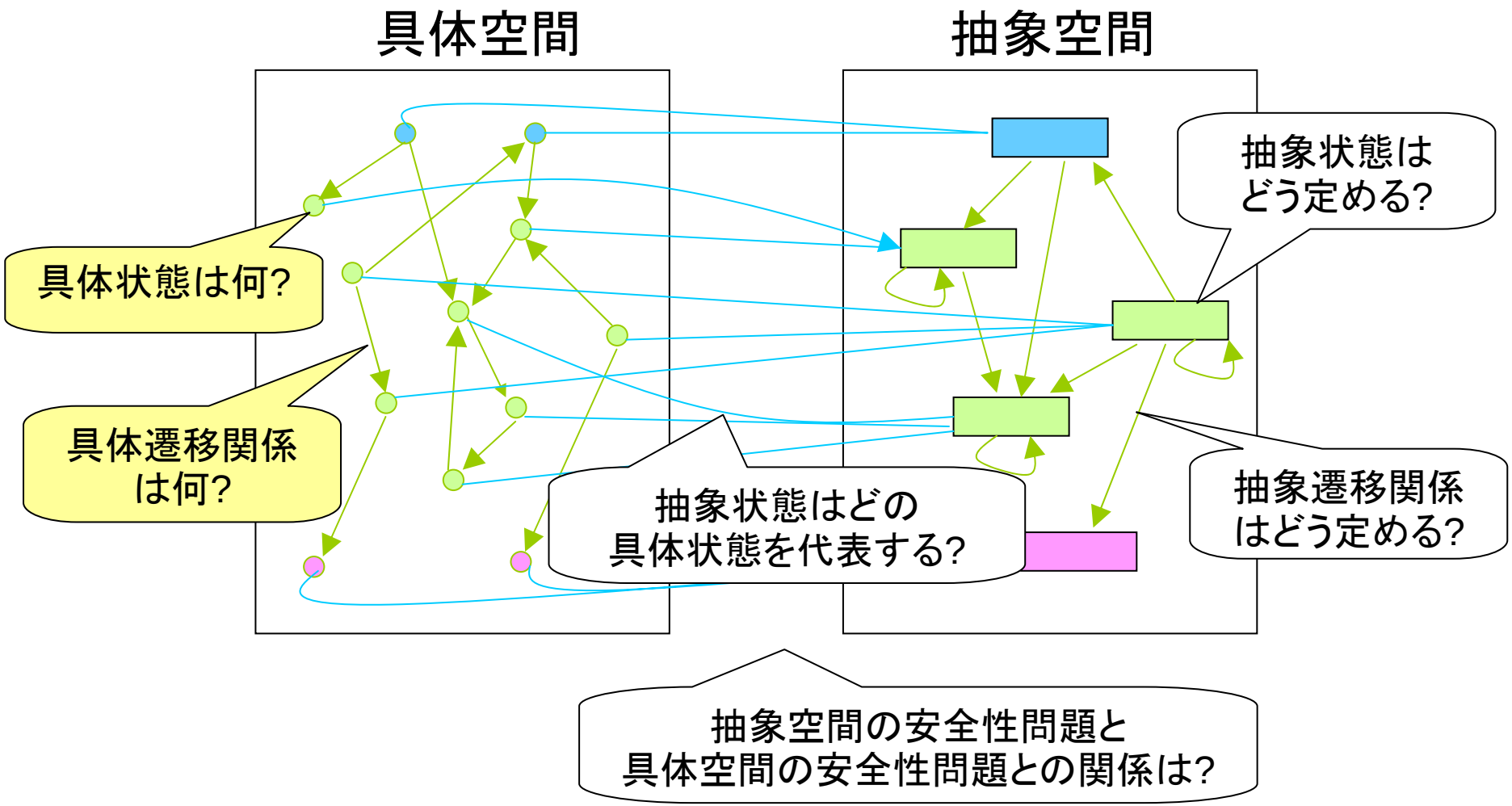
適用するには....


具体空間
抽象空間
具体状態は何?
具体遷移関係
は何?
抽象状態はどの
具体状態を代表する?
抽象状態は
どう定める?
抽象遷移関係
はどう定める?
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

具体状態と遷移


状態
プログラム実行位置
変数の値
PC = 9
LOCK = 1
old = 5
new = 5
q = 0x4a18
遷移
new++;
実行位置における命令
状態
PC = 10
LOCK = 1
old = 5
new = 6
q = 0x4a18
状態
PC = 6
LOCK = 0
old = 15
new = 15
q = 0x6b20
遷移
[q != NULL]
状態
PC = 7
LOCK = 0
old = 15
new = 15
q = 0x6b20

適用するには....

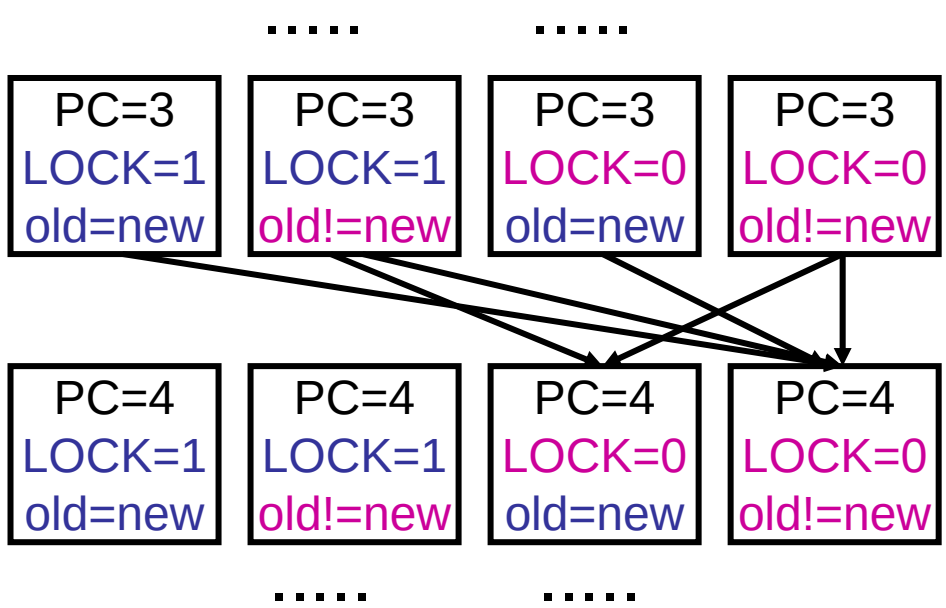

具体空間
抽象空間
具体状態は何?
具体遷移関係は何?
抽象状態はどの具体状態を代表する?
抽象状態はどう定める?
抽象遷移関係はどう定める?
抽象空間の安全性問題と具体空間の安全性問題との関係は?

述語による抽象化

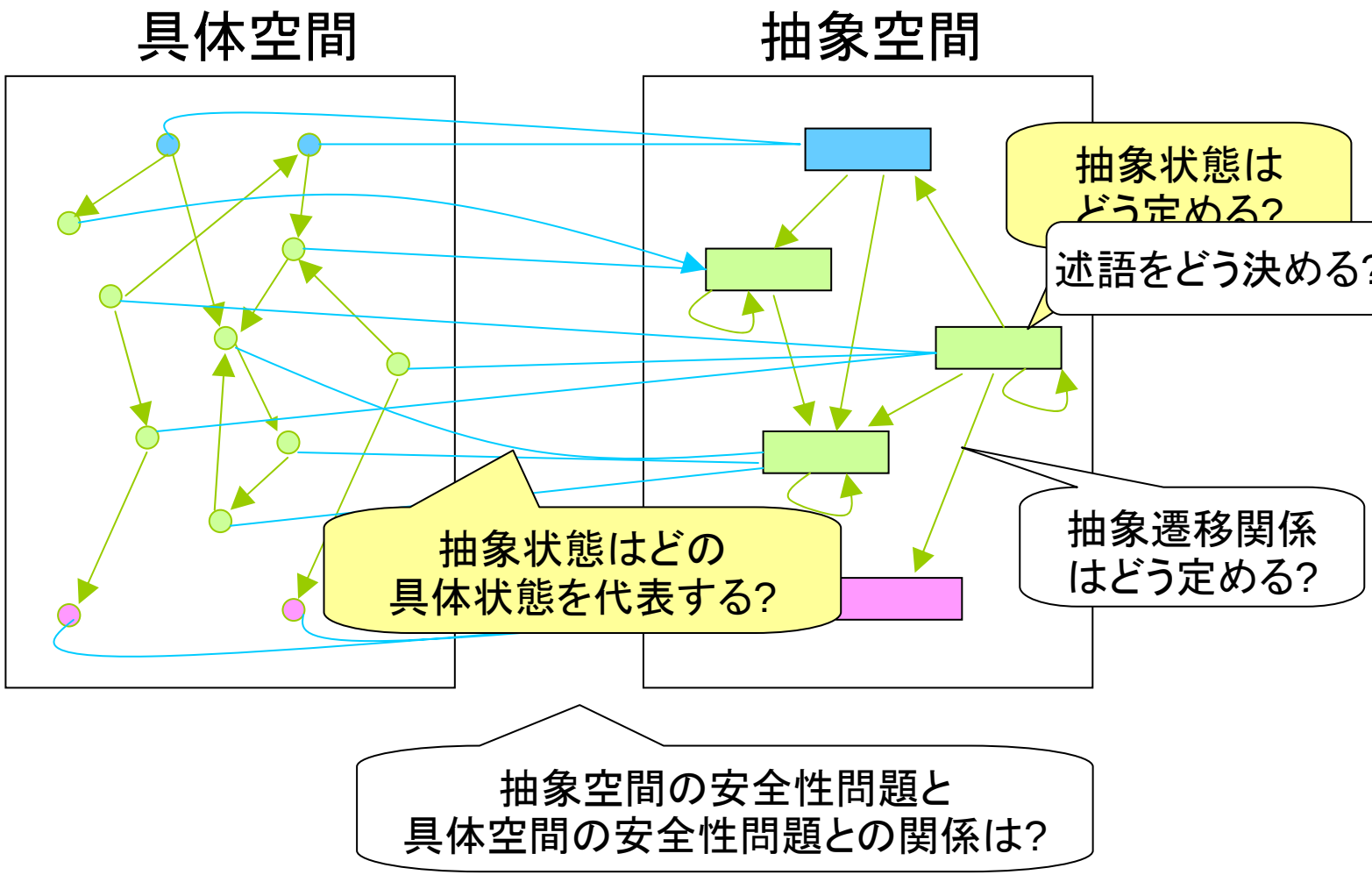
- 有限個の述語 P1, ..., Pn を決める.
- 抽象状態:
  - PC (プログラム実行位置)
  - P1の真偽
  - ...
  - Pn の真偽

(例):

- P1: LOCK = 1
- P2: old = new

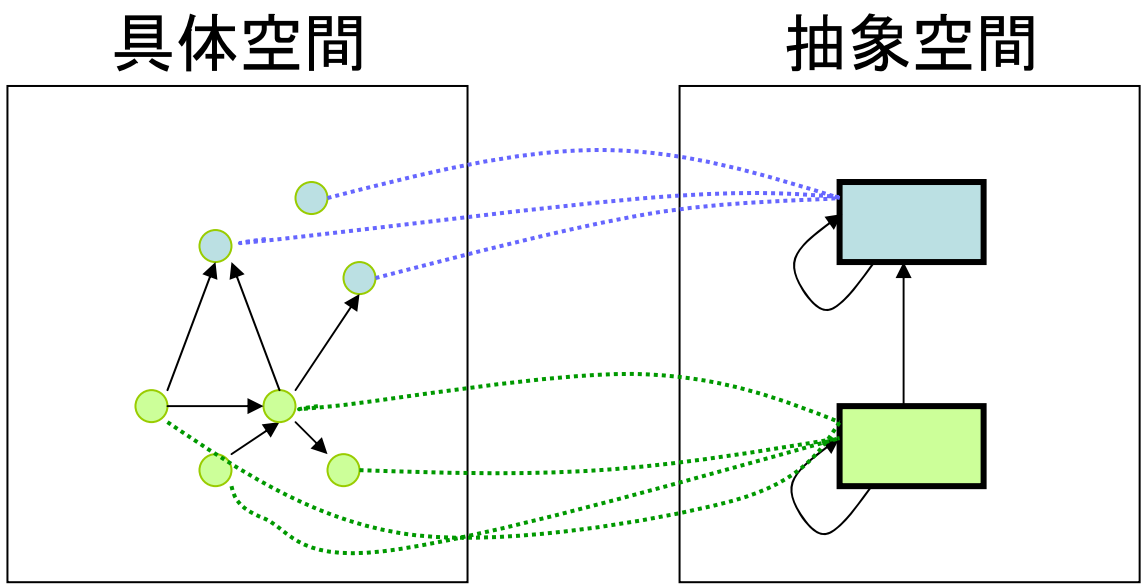


適用するには....


具体空間
抽象空間
抽象状態は
どう定める?
述語をどう決める?
抽象遷移関係
はどう定める?
抽象状態はどの
具体状態を代表する?
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

抽象化の正当性

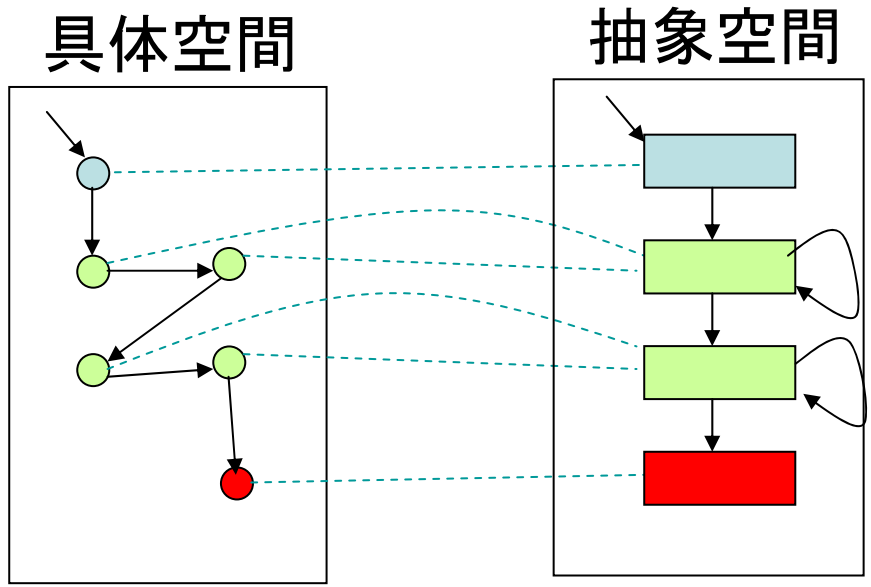
- Existential Abstraction
  具体空間の2状態間に遷移があれば,
  対応する抽象空間の2状態間にも遷移がある.



抽象化の正当性

Existential Abstraction においては，抽象空間において対応する安全性が検証できれば，具体空間における安全性が検証できたことになる.

対偶: 具体空間において危険状態に達し得るのであれば，抽象空間でもそうなる.

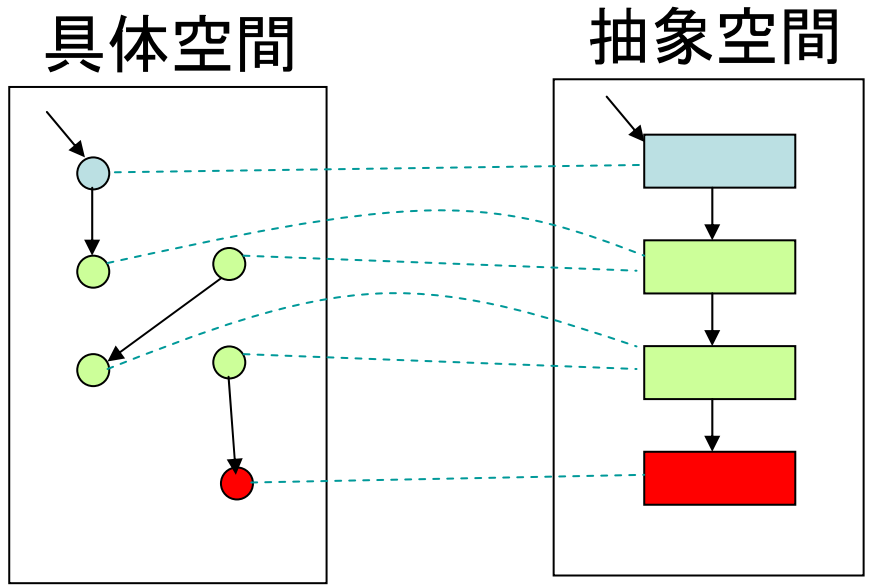


抽象化の正当性

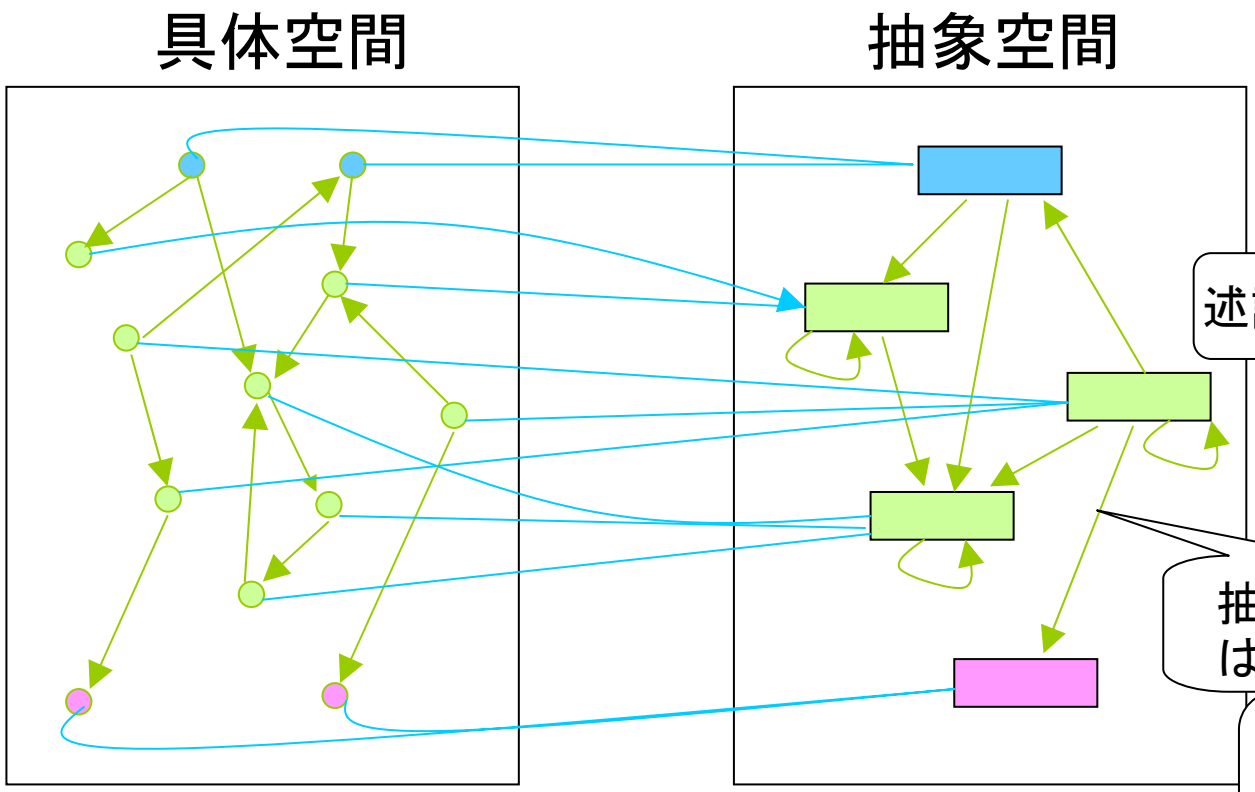
Existential Abstraction においては，抽象空間において対応する安全性が検証できれば，具体空間における安全性が検証できたことになる.

注意:

逆は成立しない.



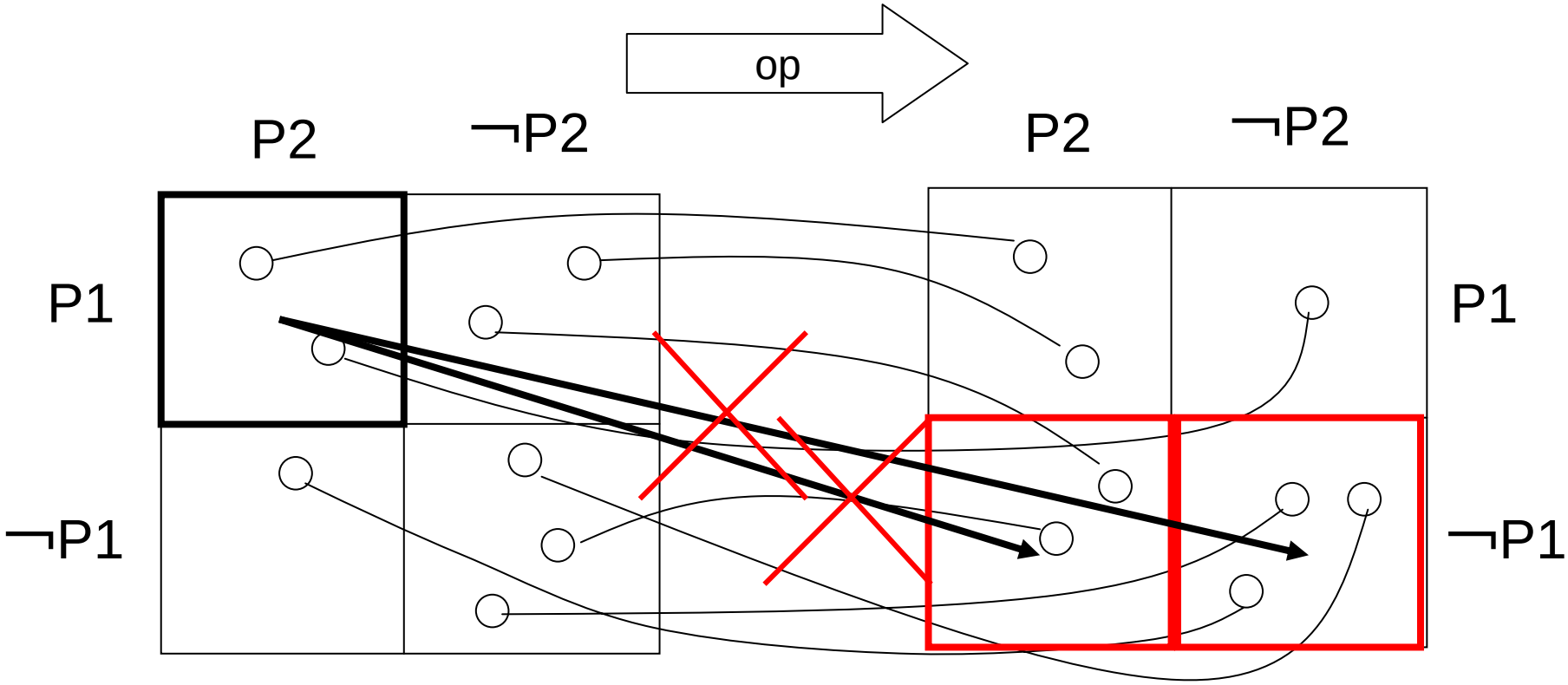
適用するには....


具体空間
抽象空間
述語をどう決める?
抽象遷移関係
はどう定める?
Existential
Abstraction
になるように
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

最弱事前条件

- P: 条件，OP: 命令
- WP(P, OP): 以下を満たす最も弱い条件P':
  「OPを実行する前にP'が成り立っていれば，
  OPの実行後にPが成り立つ．」
- 例: WP(P, x=e) = P[e/x]
  WP(new == old，new = new+1)
  = new+1 == old

遷移関係の計算(1)

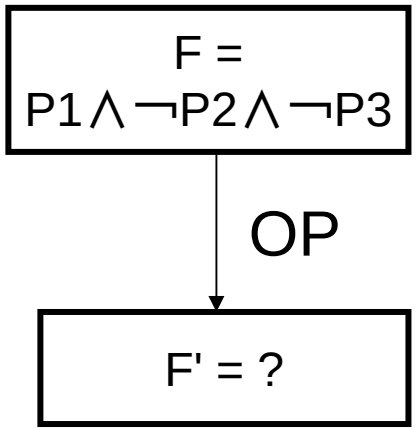


(P1∧P2) => WP(P1, op) が恒真

P1∧P2 なら，op のあと，必ず P1 になる．

P1∧P2 から，¬P1 への遷移はない．

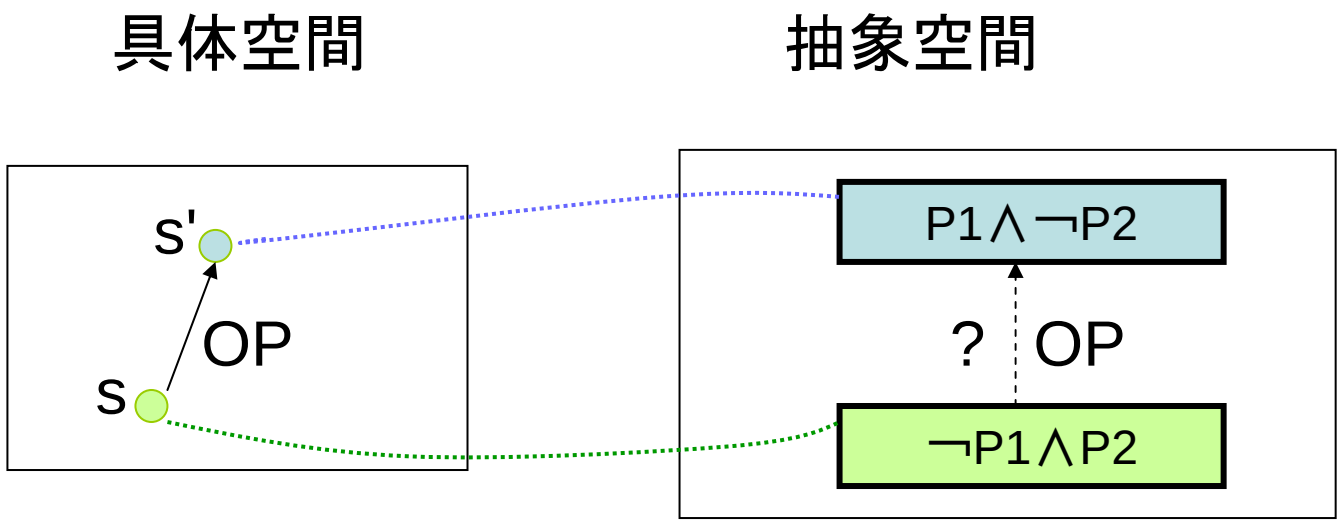
遷移関係の計算 (2)



- (F,PC)から遷移できる状態はどこか?
- PCからOPを得る.
- 各Pn について, WP(Pn, OP), WP(¬Pn, OP) を求める.
- 自動定理証明器に,
  F => WP(Pn, OP)
  が恒真であるかどうか問い合わせる.
- Yes なら, ¬Pnを含む状態へは遷移できない. (Noなら遷移できる.)
- 同様に F => WP(¬Pn, OP) が更新なら, Pnを含む状態へは遷移できない.

述語抽象化の正当性

- 述語抽象化は，Existential Abstraction である．



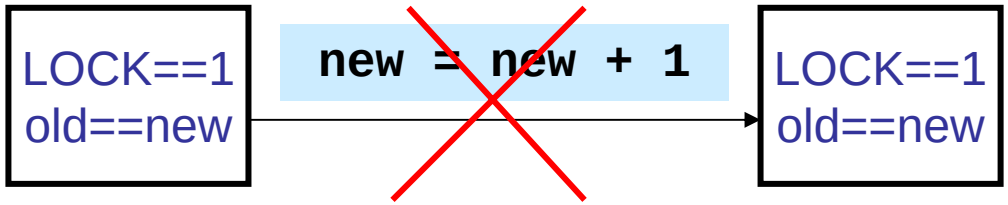
上図のような状況を考える．

¬P1∧P2 => WP(¬P1, OP) が恒真だとする．s は ¬P1∧P2 をみたすから，sは WP(¬P1, OP)をみたすことになる．したがって，s' は¬P1をみたすはず．だが，s' に対応する抽象空間ではP1が成り立つ．したがってこれは恒真ではない．

同様に，¬P1∧P2 => WP(P2, OP) も恒真でない．

したがって，抽象遷移がある．

遷移関係の計算例 (1)



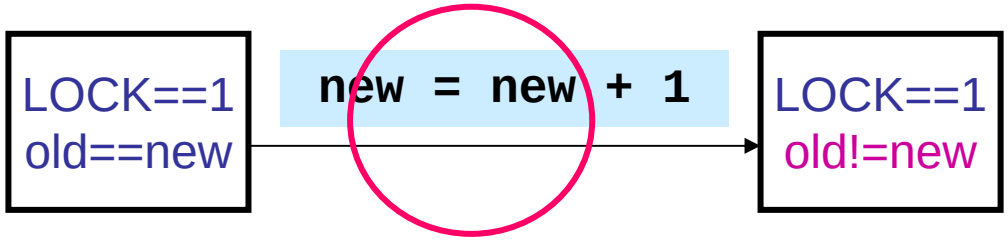
WP(LOCK != 1, OP) = LOCK != 1

LOCK==1∧old==new => LOCK != 1 ： 恒真でない

WP(old != new, OP) = old != new + 1

LOCK==1∧old==new => old != new+1 ： 恒真

遷移関係の計算例 (2)



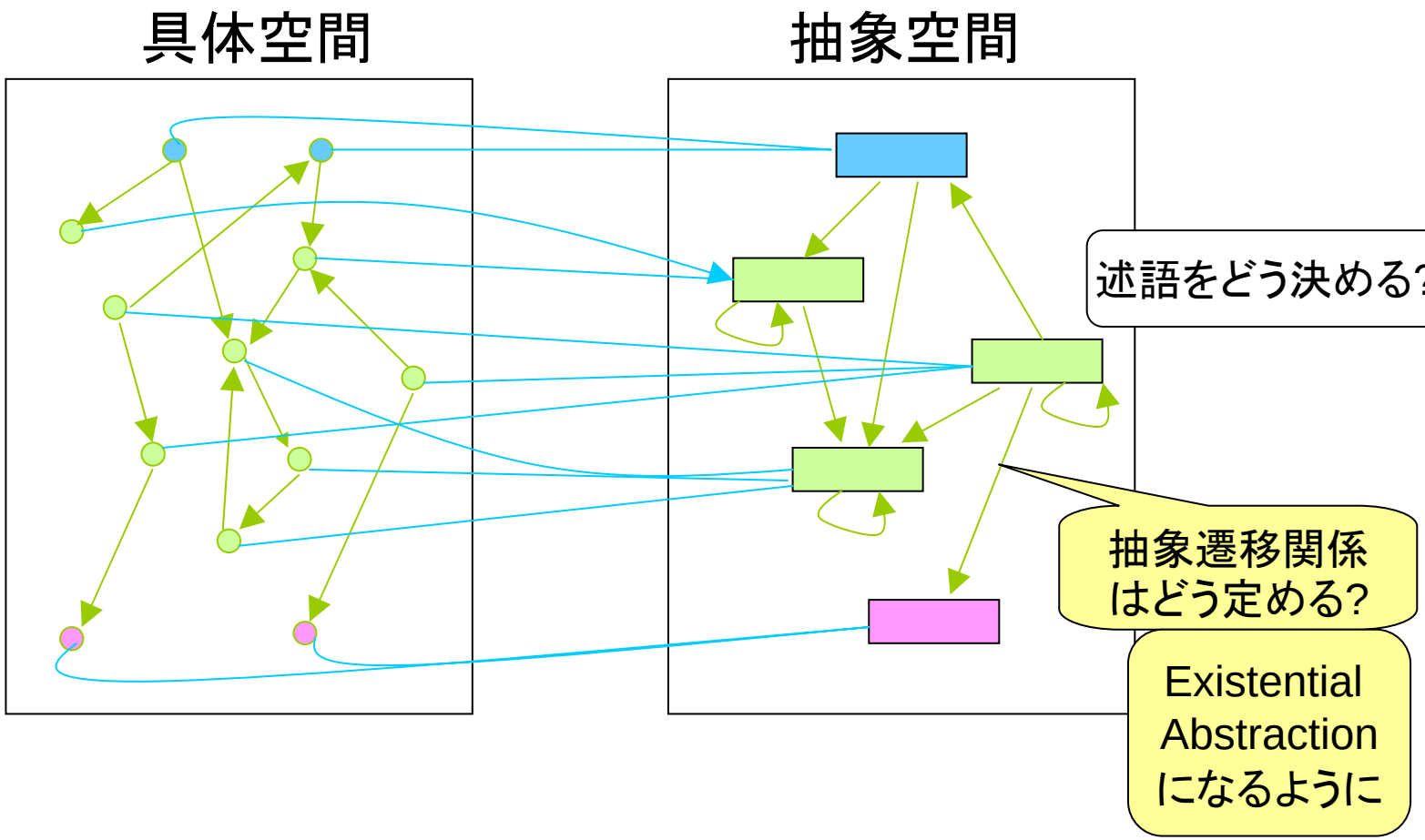
WP(LOCK != 1, OP) = LOCK != 1

LOCK==1∧old==new => LOCK != 1 ： 恒真でない

WP(old == new, OP) = old == new + 1

LOCK==1∧old==new => old == new+1 :恒真でない

適用するには....

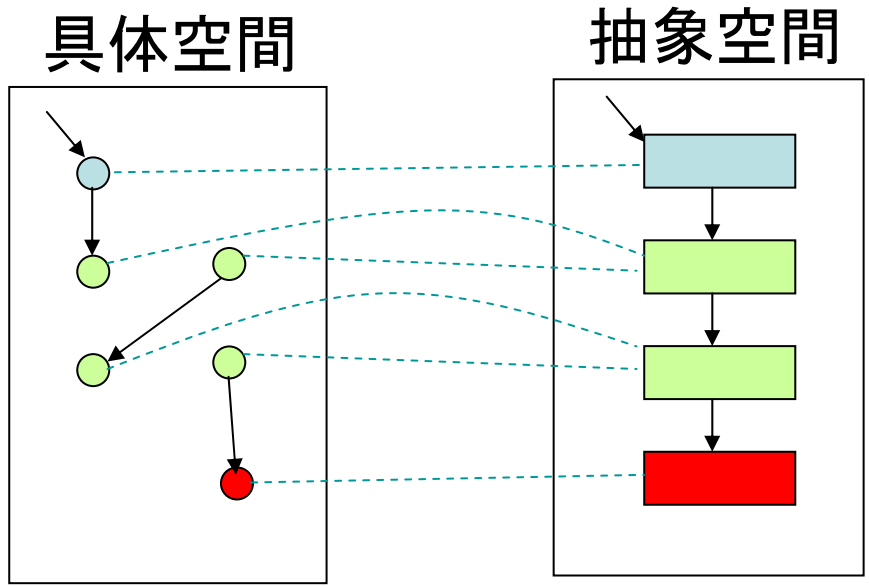

具体空間
抽象空間
述語をどう決める?
抽象遷移関係
はどう定める?
Existential
Abstraction
になるように

抽象化の正当性 (再掲)

Existential Abstraction においては，抽象空間において対応する問題が検証できれば，具体空間における安全性が検証できたことになる.

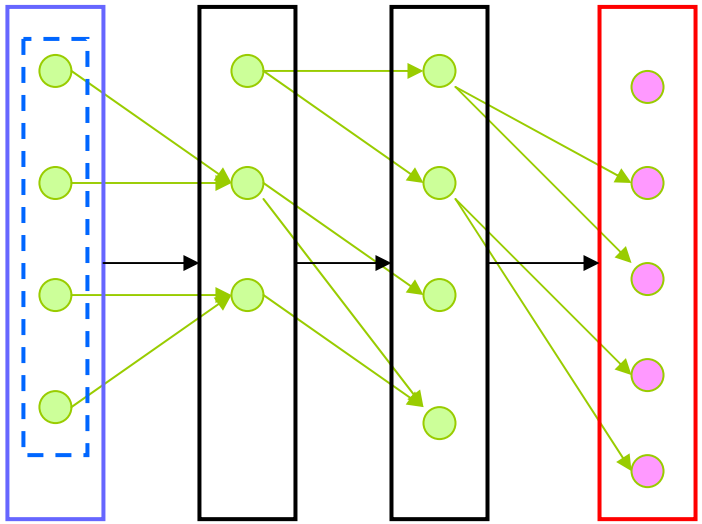
注意:

逆は成立しない.



偽反例

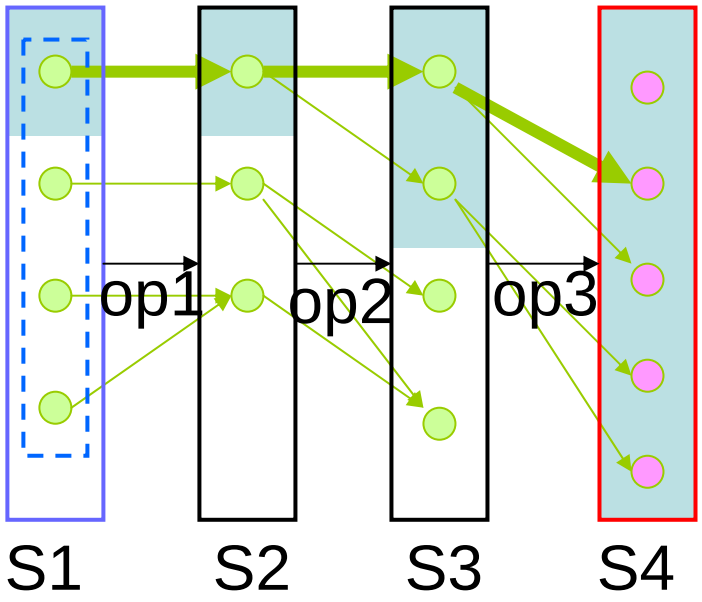
- 具体空間では安全性が成立しているのに，抽象空間では危険状態に達する場合がある.「偽反例(spurious counterexample)」



初期状態

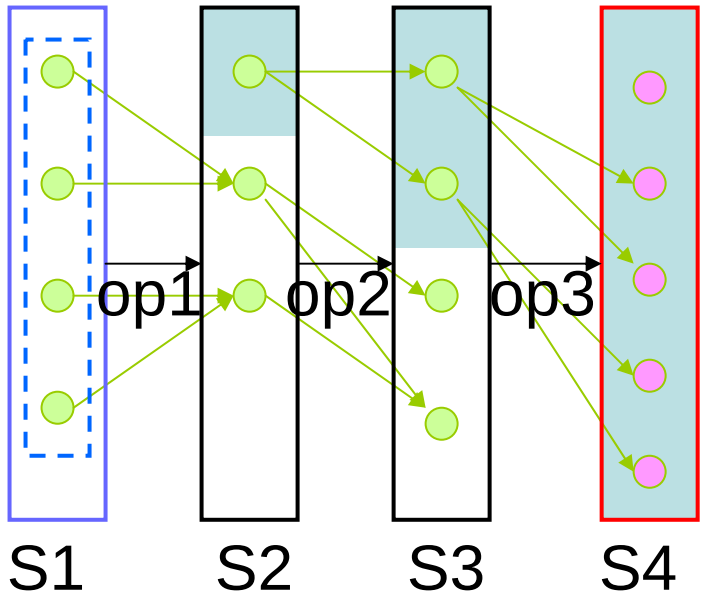
偽反例の判定 (1)

C1=WP(op1, C2)≠false
C2 = WP(op2, C3)
C3 = WP(op3,C4)
C4=true



真の反例

C1=WP(op1, C2)=false
C2 = WP(op2, C3)
C3 = WP(op3,C4)
C4=true



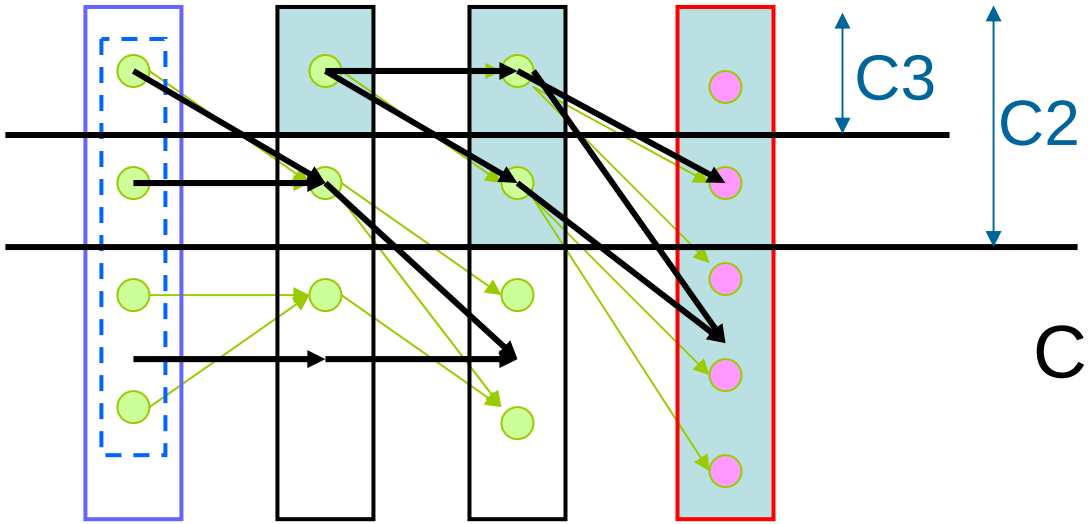
偽反例

偽反例の判定 (2)

- 抽象空間における反例 (初期状態から危険状態にいたる列 $S_1, \ldots, S_n$ が偽反例かどうかを判定:
- S_i から S_{i+1} への命令を OP_i とする.
- C_n = true
- $C_{i-1} = WP(C_i, OP_{i-1})$　(i = n, ..., 2)
- C_1 が false でなければ, 真の反例である.
- C_1 が false であれば, 偽反例である.

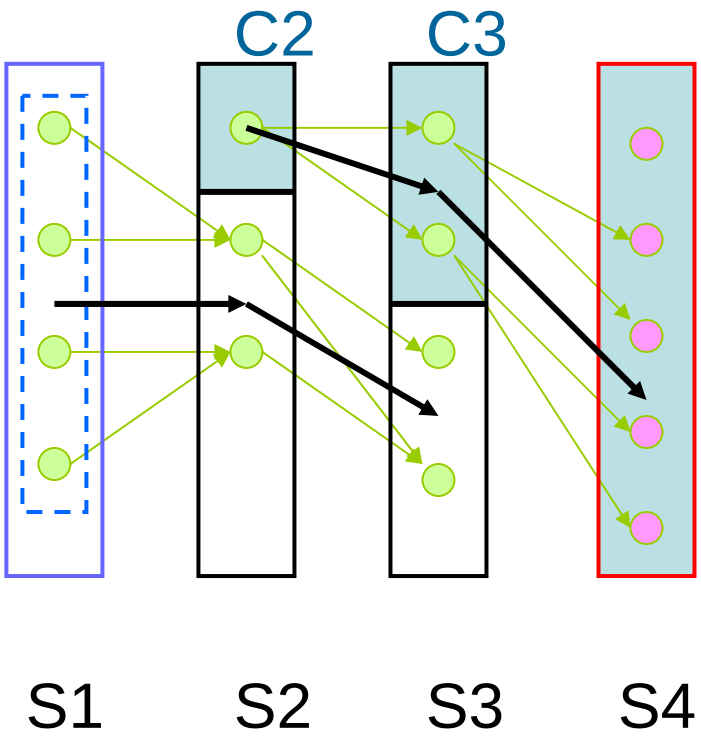
反例による詳細化

- 偽反例の原因: 抽象化の粒度が粗すぎる.
- 偽反例を排除するように，抽象化の精度を上げる.
  反例による詳細化 (CEGAR = CounterExample-Guided Abstraction Refinement)



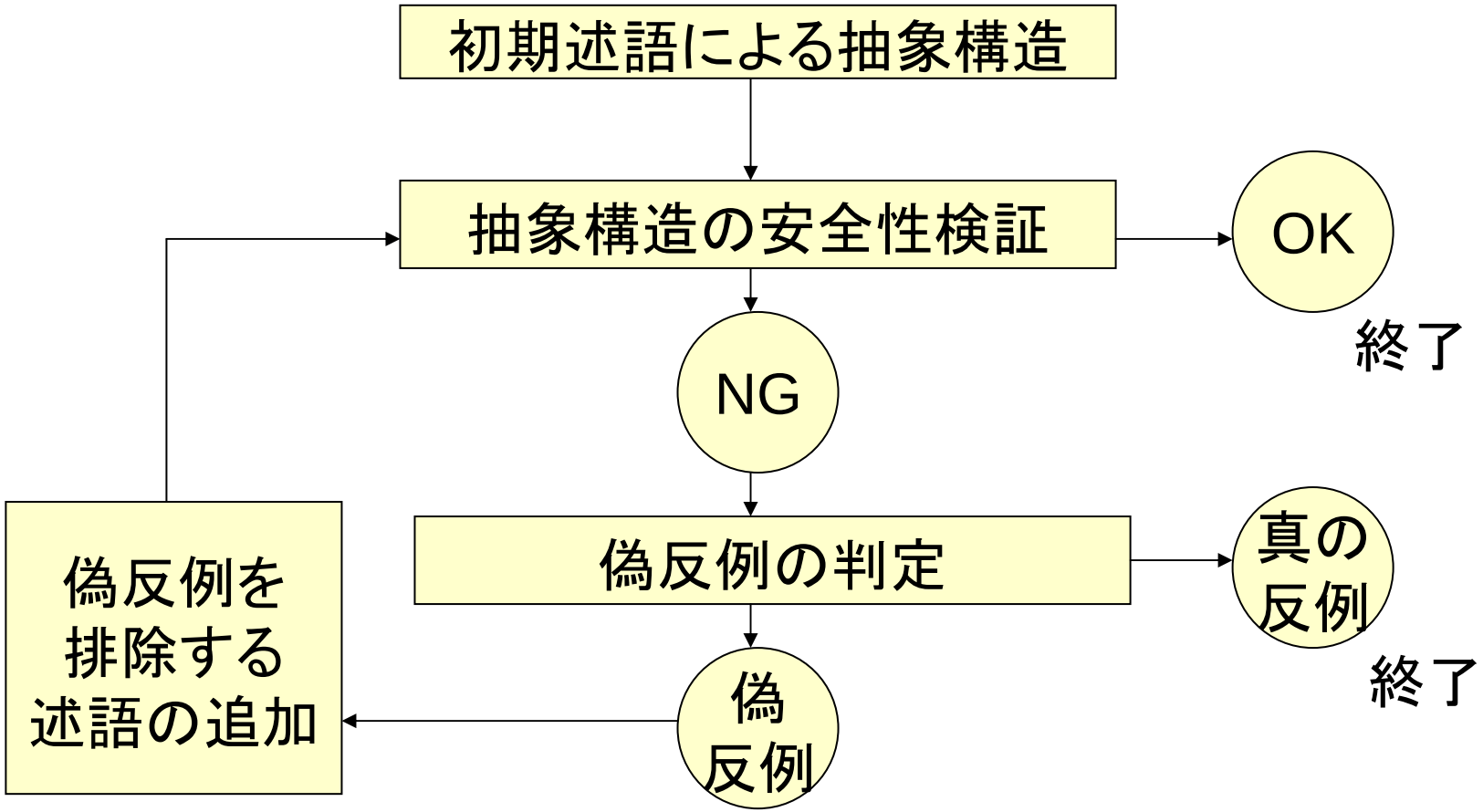
C2とC3を述語として追加

効率化: 述語を局所的に定義



- C2はS2でのみ述語として採用.
- C3はS3でのみ述語として採用

述語抽象化による安全性検証: 全体像



ツール

述語抽象化の手法を(も)使っているソースコード検証ツール

- SLAM (Microsoft)
- BLAST (UC Berkeley)
- Bandera (Kansas State Univ)
- Java PathFinder (NASA)
- MAGIC (CMU)
- CBMC (CMU)

抽象化によるシェープ解析

シェープ解析

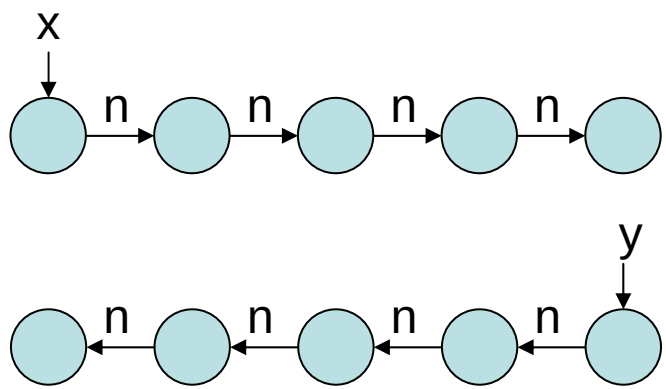
- 「リンク」を持つデータの「形状」に関する解析
  - 一方向リスト，双方向リスト，木，DAG, ...
  - DSWの検証は，ある意味典型的な問題.
- 自動検証アプローチ
  - TVLA (Sagiv, Reps, Wilhelm, ...)
  - PALE (Møller, Schwartzbach, ...)
- 対話検証アプローチ
  - Separation Logic (Reynolds, O'Hearn, ...)

TVLA

- Three-Valued Logic Analysis engine
- Tel-Aviv University
- M. Sagiv, T. Reps, R. Wilhelm, ...
- http://www.cs.tau.ac.il/~tvla/
- ヒープ上に構築されたデータに関する性質の検証を，抽象化の手法で行うツール．

Running Example

- 「リストを逆転させるアルゴリズム」
- 入力: xから始まる一方向リスト



- 出力: yから始まる一方向リスト
- 性質
  - 実行中，NULLポインタ参照外しが起こらない.
  - 入力と出力のリストの要素は一致している.
  - 入力と出力で「指す」という関係 （リンク関係) が反転している.

Running Example

```
/* list.h */
typedef struct node {
  struct node *n;
  int data;
} *List;
```

```
/* reverse.c */
#include "list.h"
List reverse(List x) {
  List y, t;
  y = NULL;
  while (x != NULL) {
    t = y;
    y = x;
    x = x->n;
    y->n = t
  }
  return y;
}
```

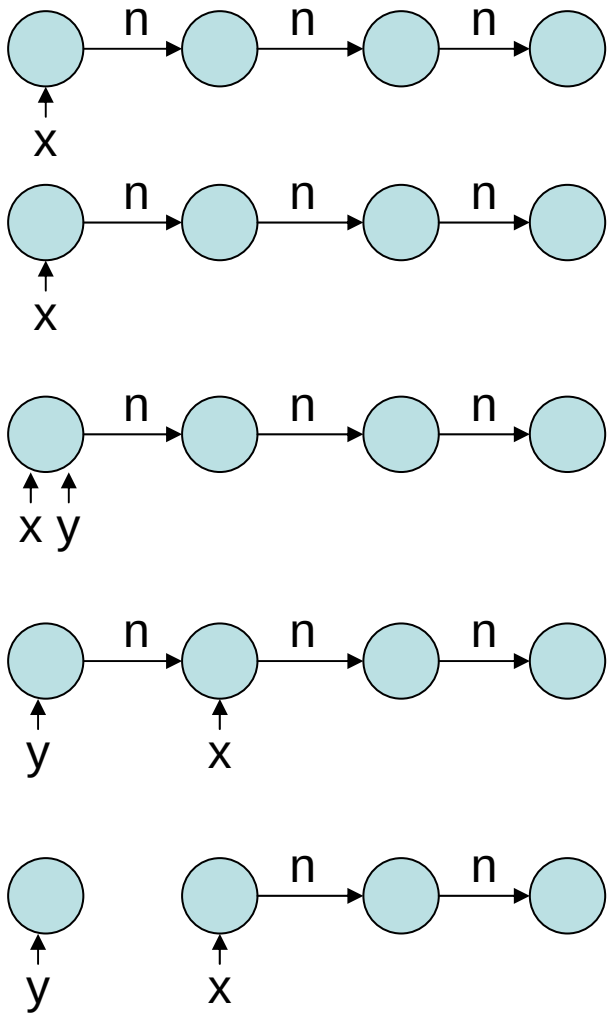
Running Example

t=y

y=x

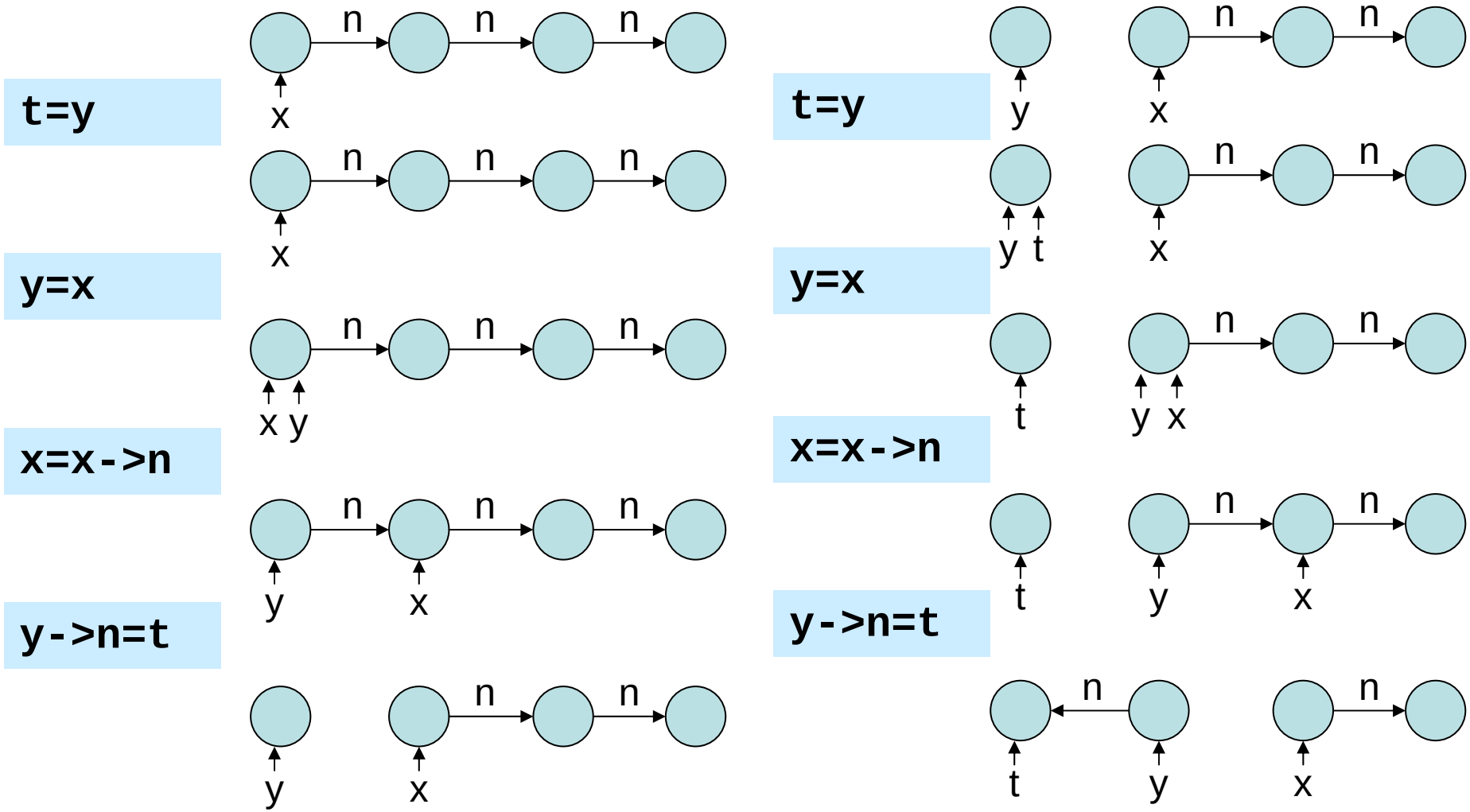
x=x->n

y->n=t

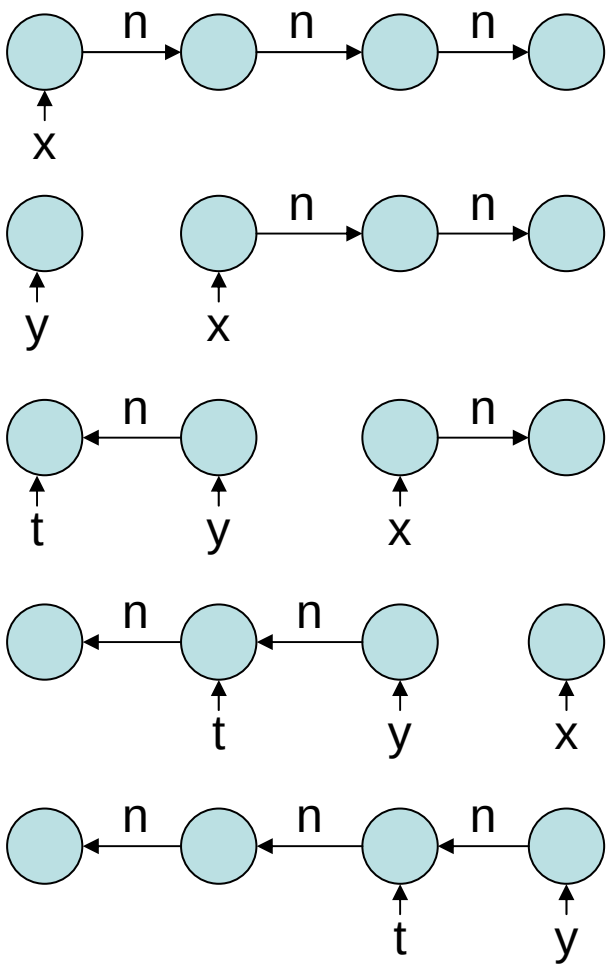



```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

Running Example



Running Example




```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

準備: Kleene の3値論理

- 真偽値: 0(偽), 1(真), 1/2(不明)

でない

	0	1/2	1
¬	1	1/2	0

かつ

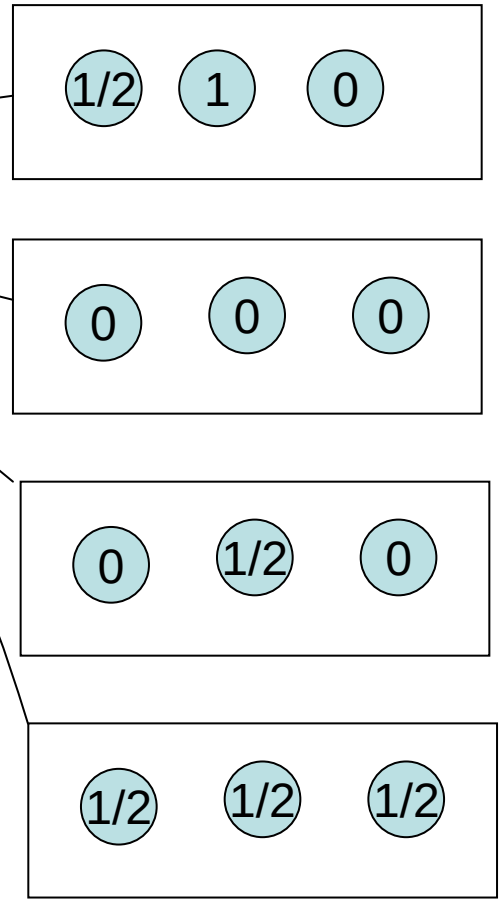
∧	0	1/2	1
0	0	0	0
1/2	0	1/2	1/2
1	0	1/2	1

または

∨	0	1/2	1
0	0	1/2	1
1/2	1/2	1/2	1
1	1	1	1

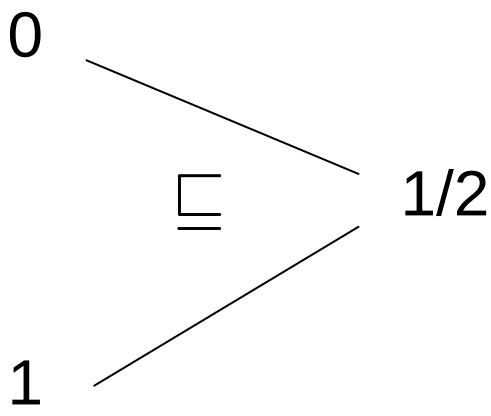
準備: Kleene の3値論理

- ∃x. p(x) の値は
  - 1:　p(u)の値が1となるuがある.
  - 0:　p(u)の値がすべて0.
  - 1/2: それ以外.
- ∀x. p(x) の値は
  - 1:　p(u)の値がすべて1.
  - 0:　p(u)の値が0となるuがある.
  - 1/2: それ以外.

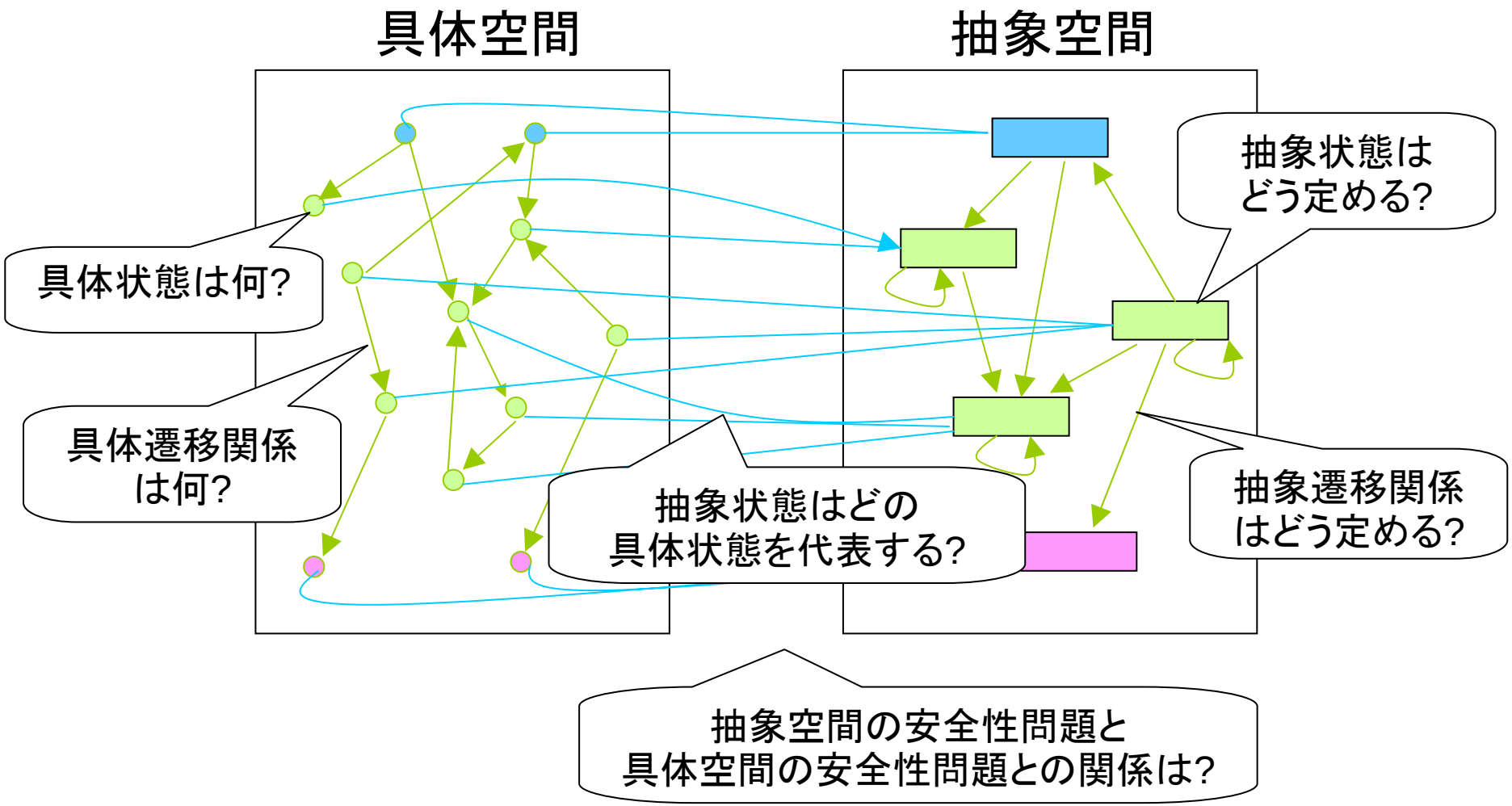


準備: Kleene の3値論理

- 情報量順序 $a \sqsubseteq b$ $(a, b = 0, 1, 1/2)$
  「aとbは矛盾せず，aの方がより詳しいか等しい」
- $0 \sqsubseteq 1/2$, $1 \sqsubseteq 1/2$, $0 \sqsubseteq 0$, $1 \sqsubseteq 1$, $1/2 \sqsubseteq 1/2$

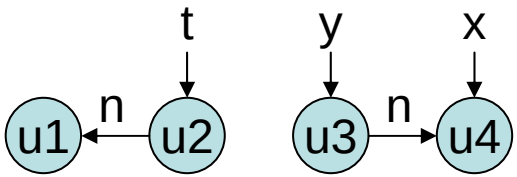


抽象化


具体空間
抽象空間
具体状態は何?
具体遷移関係
は何?
抽象状態はどの
具体状態を代表する?
抽象状態は
どう定める?
抽象遷移関係
はどう定める?
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

具体構造

- 具体構造(2値構造): 具体空間の状態



- 述語 x(·), y(·), t(·), n(·,·) に対する真偽値の割当とみることができる.

	x	y	t
u1	0	0	0
u2	0	0	1
u3	0	1	0
u4	1	0	0

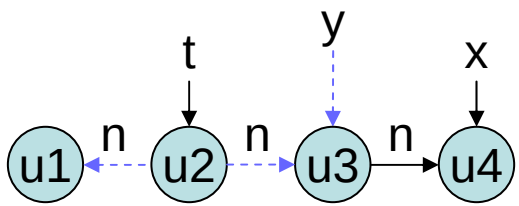
n	u1	u2	u3	u4
u1	0	0	0	0
u2	1	0	0	0
u3	0	0	0	1
u4	0	0	0	0

抽象構造

- 抽象空間の状態
- 抽象構造 (3値構造) : 述語に対して3値論理の真偽値 (0, 1/2, 2) を割り当てる.

	x	y	t
u1	0	0	0
u2	0	0	1
u3	0	1/2	0
u4	1	0	0

n	u1	u2	u3	u4
u1	0	0	0	0
u2	1/2	0	1/2	0
u3	0	0	0	1
u4	0	0	0	0



述語

- core predicates: 構造を定める基本の述語.
  - (ポインタ型の) 変数 x に対して，単項述語 x(・):「xが表している」
  - (ポインタ型の) フィールド n に対して，2項述語 n(・,・) :「nによるリンク関係がある」
  - 単項述語 sm(・):「2つ以上の具体ノードを表す」: 2値構造では常に値0, 3値構造では値0または1/2. (SuMmary)
- instrumentation predicates: core predicate を用いて定義する述語. 検証したい性質に応じて導入する. (しかし，以下の3つは，たいてい必要となる.)
  - 単項述語 is_n(・) : リンク関係 n によって，2つの異なるノードから指されている. (IsShared)
  - 単項述語 $r_{x,n}$(・) : リンク関係 n をたどることによって，変数xから到達できる. (Reachable)
  - 単項述語 c_n(・) : リンク関係 n をたどることによって，自分自身に到達する. (Cyclic)

2値構造の例

core述語

instrumentation 述語

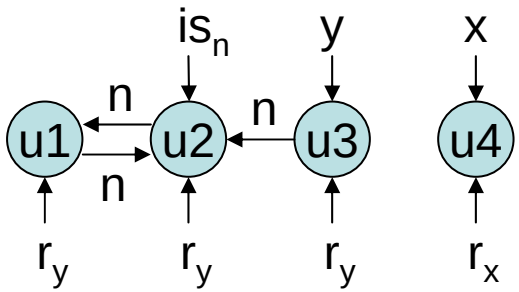
	sm	x	y	is_n	r_x	r_y
u1	0	0	0	0	0	1
u2	0	0	0	1	0	1
u3	0	0	1	0	0	1
u4	0	1	0	0	1	0

2値構造では，常にsmの値は0

core述語

n	u1	u2	u3	u4
u1	0	1	0	0
u2	1	0	0	0
u3	0	1	0	0
u4	0	0	0	0

2値構造では，instrumentation述語の値はcore述語の値から決まる．



3値構造の例

core述語

instrumentation 述語

	sm	x	y	is_n	r_x	r_y
v2	1/2	0	0	1/2	0	1
v3	0	0	1	0	0	1
v4	0	1	0	0	1	0

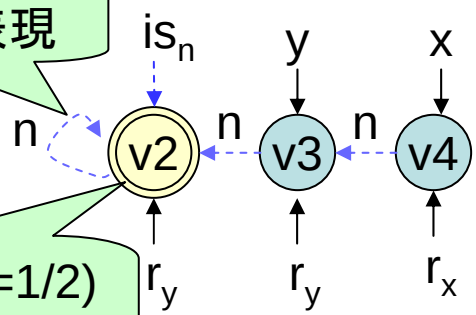
core述語

n	v2	v3	v4
v2	1/2	0	0
v3	1/2	0	0
v4	0	1/2	0

smは0か1/2

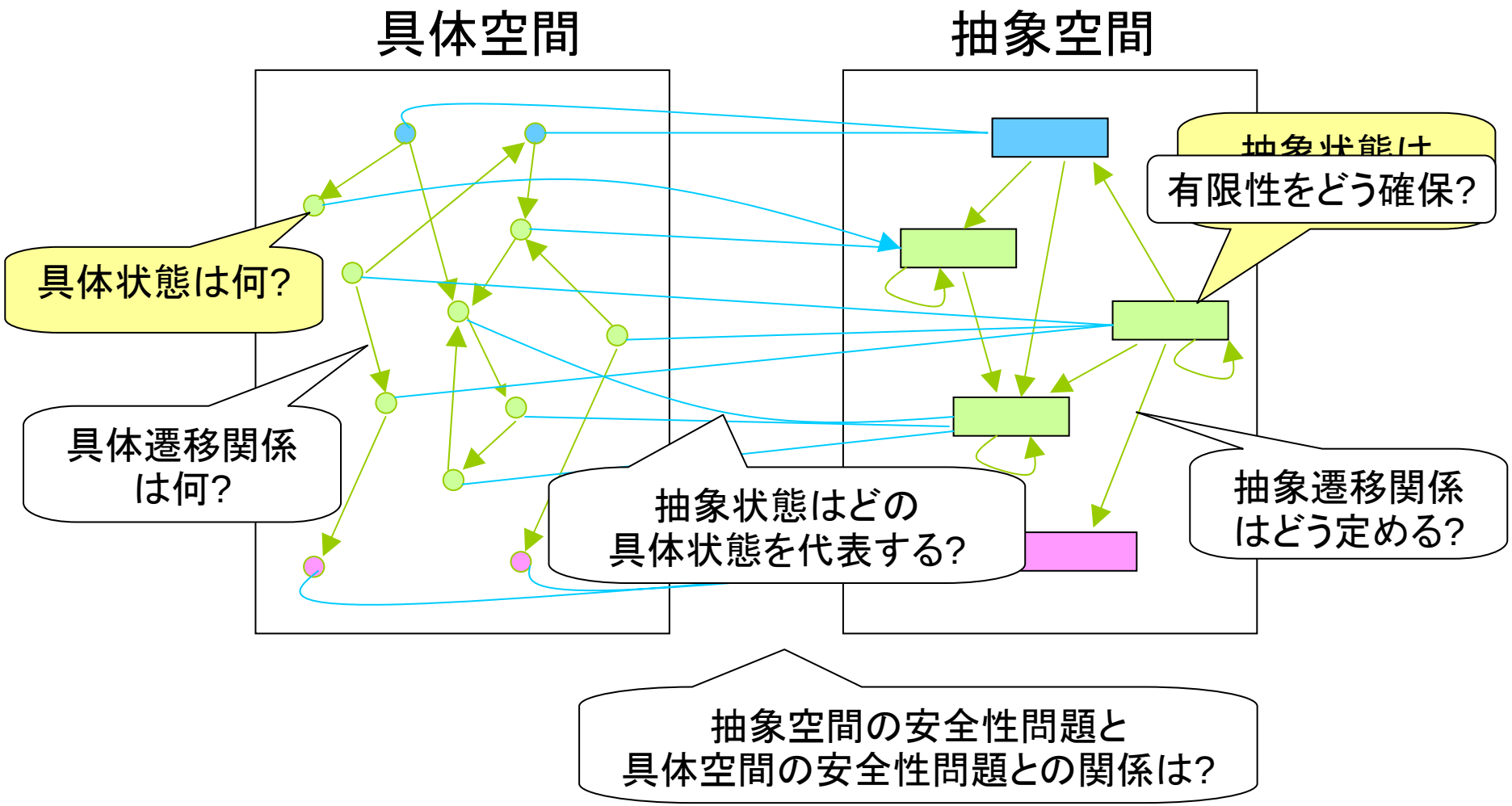
3値構造では，
instrumentation述語の値
はcore述語の値から一意
に決まるわけではない．

点線で1/2を表現

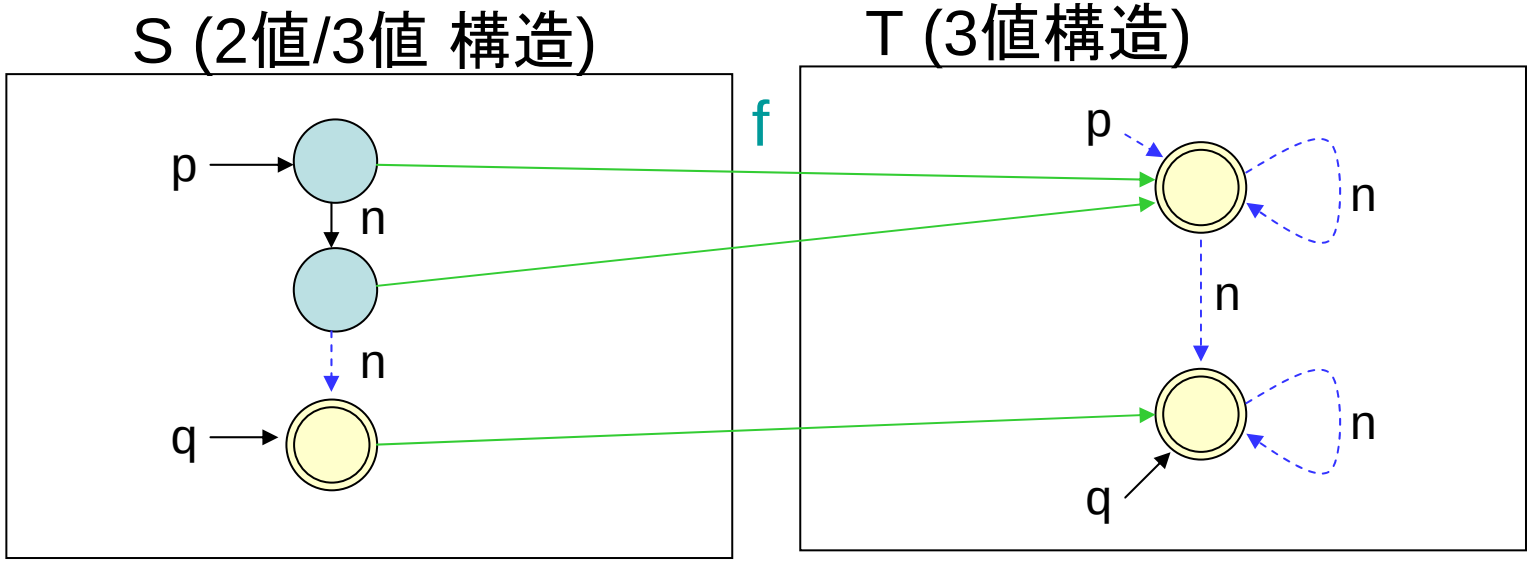


サマリノード(sm=1/2)
は2重丸で表現

抽象化

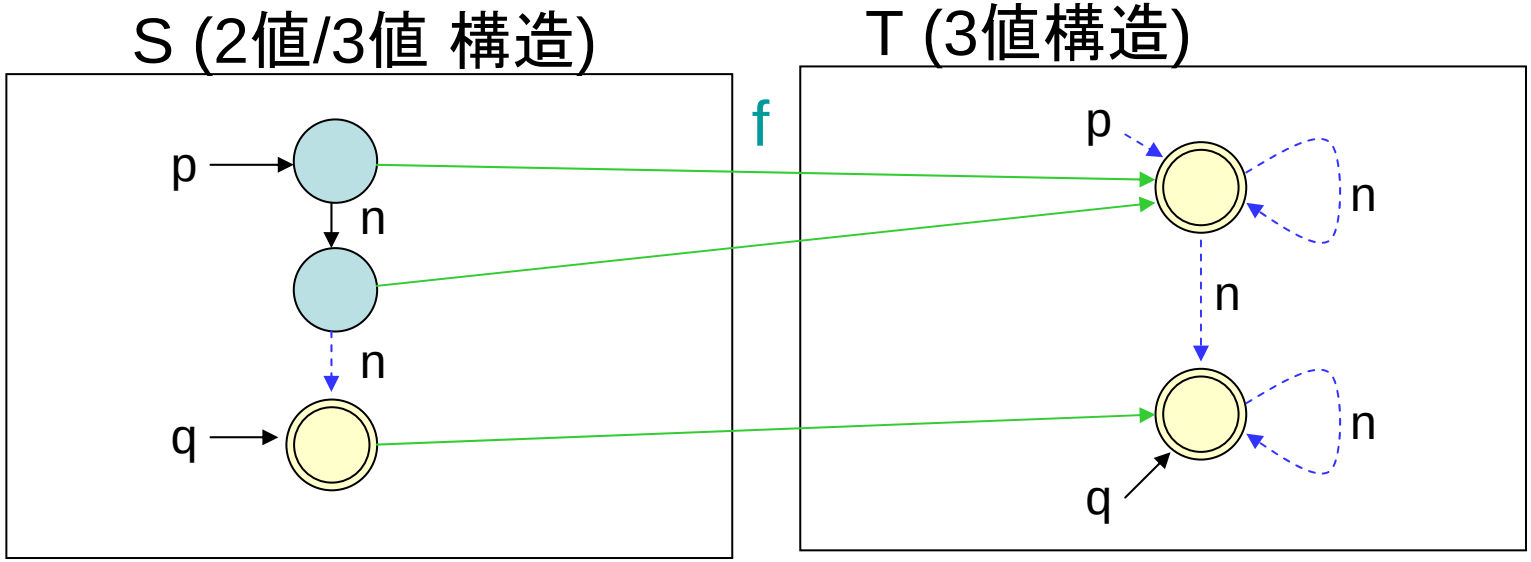

具体空間
抽象空間
具体状態は何?
有限性をどう確保?
具体遷移関係
は何?
抽象状態はどの
具体状態を代表する?
抽象遷移関係
はどう定める?
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

埋込



- f: S → T が埋込 iff
  - f は全射
  - 各述語 p に対し，$p^S(u,..) \sqsubseteq p^T(f(u),...)$
  - v∈T に対し，f(u)=v なる u が2個以上あるとき，$sm^T(v) = 1/2$

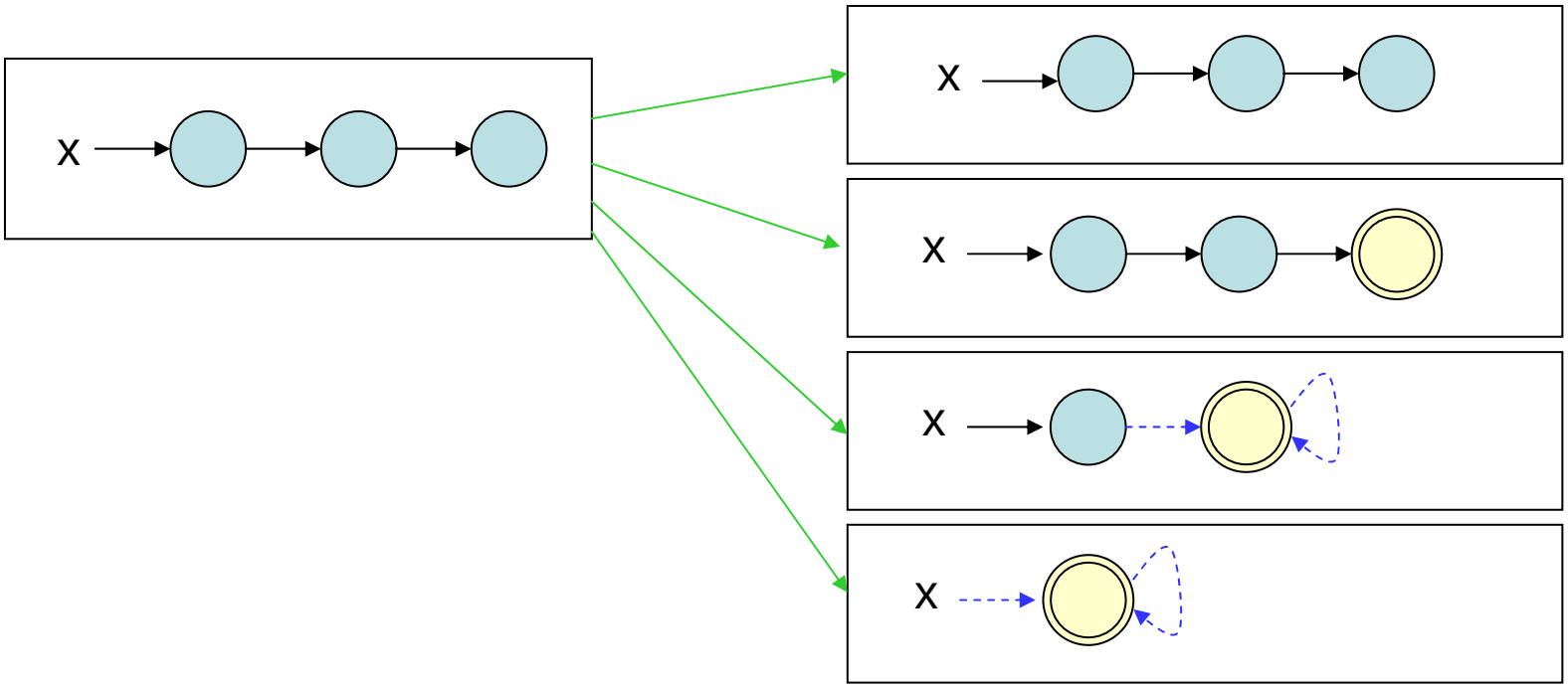
抽象化関係



- 埋込 f: S→T が存在する時，抽象構造T は具体構造S を代表する. (抽象化関係)

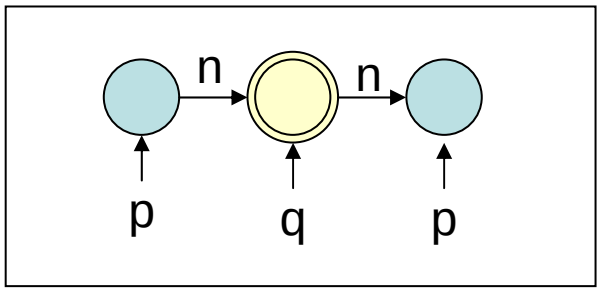
抽象化関係 (2)

- 注意: 具体構造を決めても，それを代表する抽象構造は複数個ある.
- 注意: 抽象構造の全体は，無限集合.

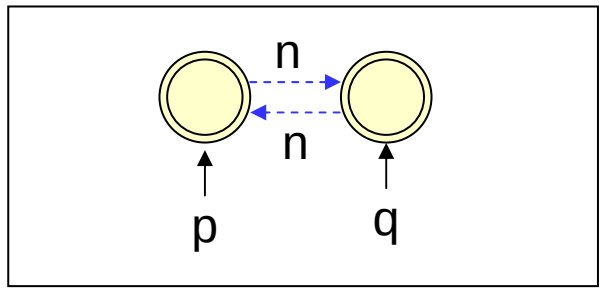


有界構造

- 3値構造Tが有界 (bounded)　iff
  v1,v2∈T, v1≠v2　ならば，　単項述語 p があって
  $p^T(v1) \neq p^T(v2)$.
- 述語集合を決めれば，有界構造は有限個しかない.
- 任意の2値/3値構造は，標準抽象化 (次スライド) によって，有界な3値構造に埋め込める.
- したがって，抽象空間を有界構造のみに限ることによって，全数探索による安全性検査が可能である.



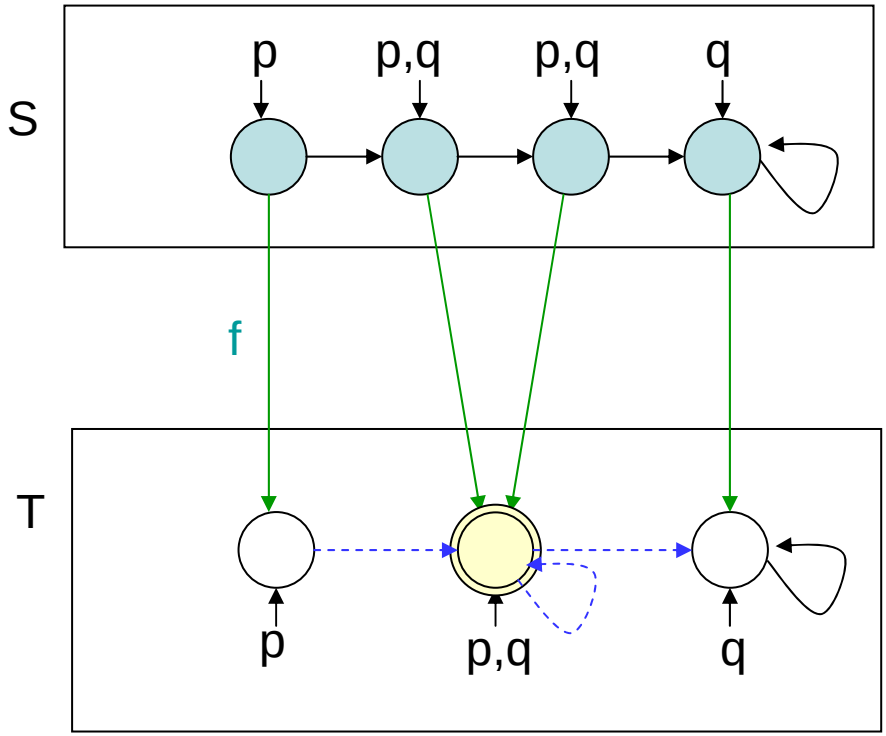
有界でない



有界

標準抽象化 (1)

- 3値構造 S に対して，以下の方法で，有界な3値構造 T と，埋込 f : S → T が作れる．これを，標準抽象化 (canonical abstraction) と呼ぶ．

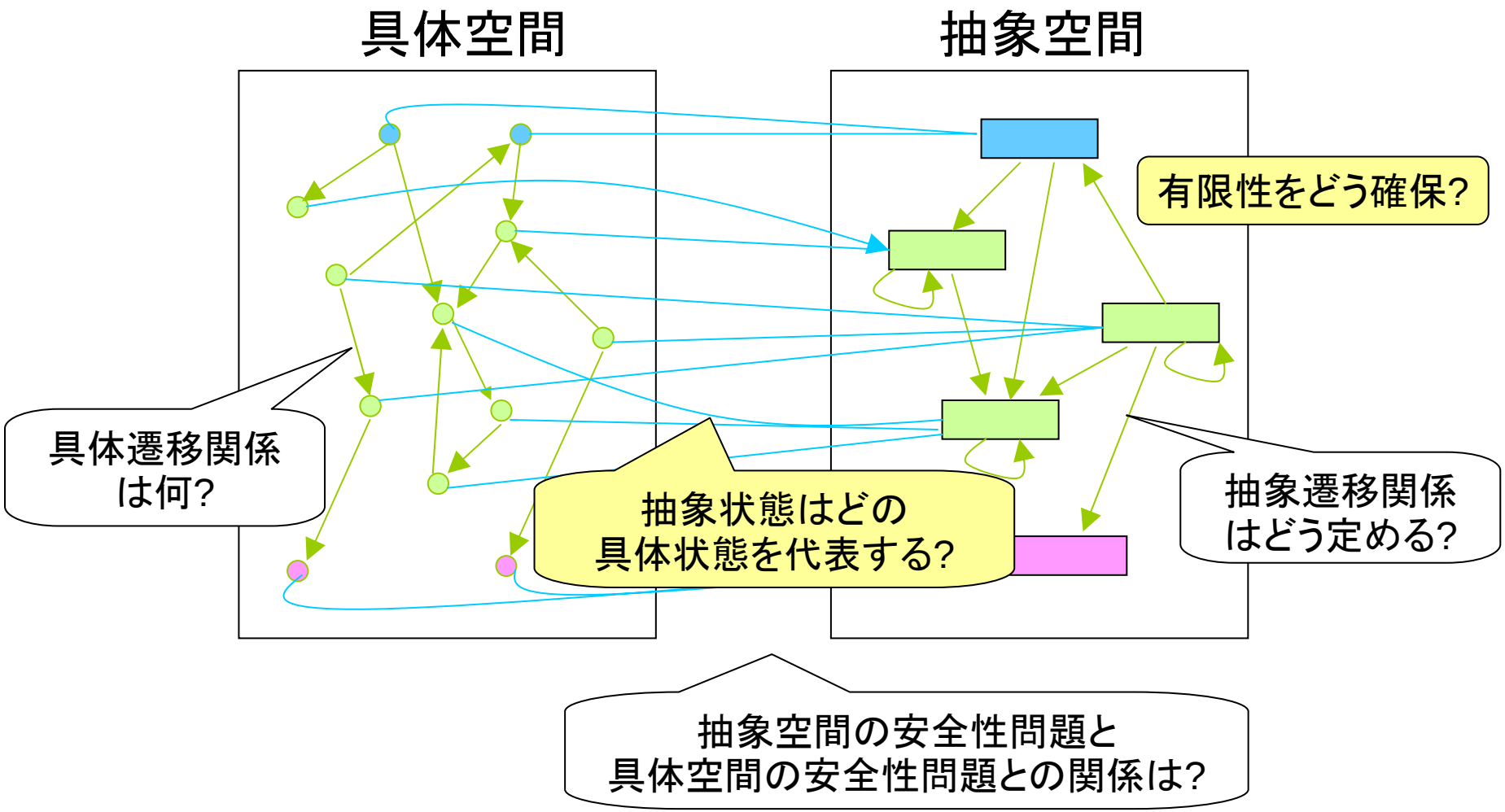


標準抽象化 (2)

- 3値構造 S に対して，以下の方法で，有界な3値構造 T と，埋込 f : S → T が作れる．これを，標準抽象化 (canonical abstraction) と呼ぶ．
- 単項述語の全体を p1,...,pn とする．
- 各 u ∈ S に対して，長さ n の {0, 1/2, 1} の列を対応させる．i番目の値が $pn^S(u)$.
- 長さ n の {0, 1/2, 1} の列vで，対応するu∈Sが存在するもののの全体を T とする．f(u) = v．対応するuが1個で，$sm^S(u)=0$のときには $sm^T(v) = 0$, そうでないときには $sm^T(v) = 1/2$.
- その他の述語については，たとえば

$$n^T(v,v') = \bigvee \{ n^S(u,u') \mid f(u) = v, f(u') = v' \}$$

抽象化


具体空間
抽象空間
有限性をどう確保?
具体遷移関係
は何?
抽象状態はどの
具体状態を代表する?
抽象遷移関係
はどう定める?
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

pre: 抽象遷移の計算準備 (1)

- 述語 p と命令 op に対して，論理式 pre(p, op) を定める: 命令 op によって，2値構造 S が S' に変化する時，

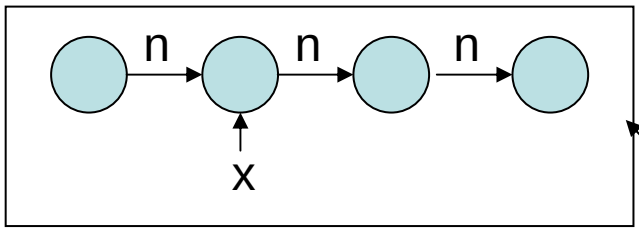
$$pre(p, op)^{S}(v,...) = p^{S'}(v,...)$$

- 例:
  - pre(y, **y = x->n**)(v) = $\exists u\,(\, x(u) \wedge n(u, v)\,)$
  - pre($r_{y,n}$, **y = x->n**)(v) = $r_{x,n}(v) \wedge (c_n(v) \vee \neg x(v))$

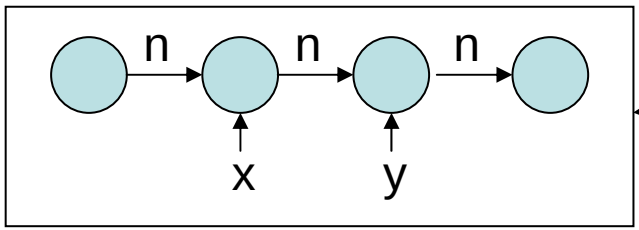
pre: 抽象遷移の計算準備 (2)

pre(y, **y = x->n**)(v) = ∃u (x(u) ∧ n(u, v))

vはxがなりたっている
ところのとなり



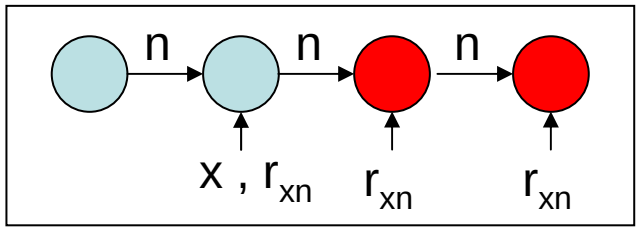
y = x->n

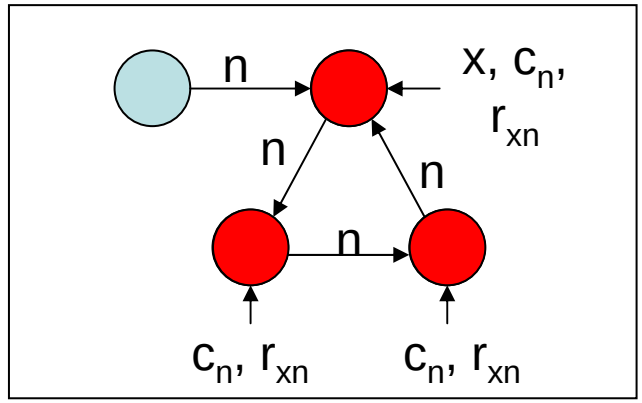


pre(y,y=x->n)(v)
<=>
y(v)

pre: 抽象遷移の計算準備 (3)

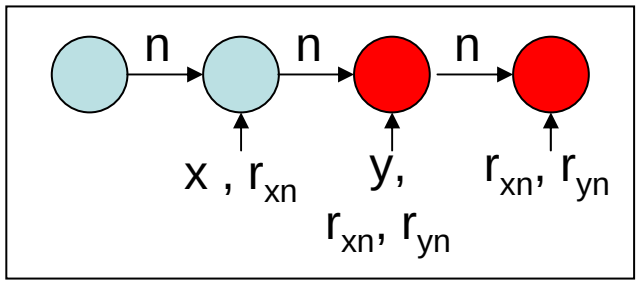
$$pre(r_{y,n},\ \mathbf{y = x->n})(v) = r_{x,n}(v) \wedge (c_n(v) \vee \neg x(v))$$

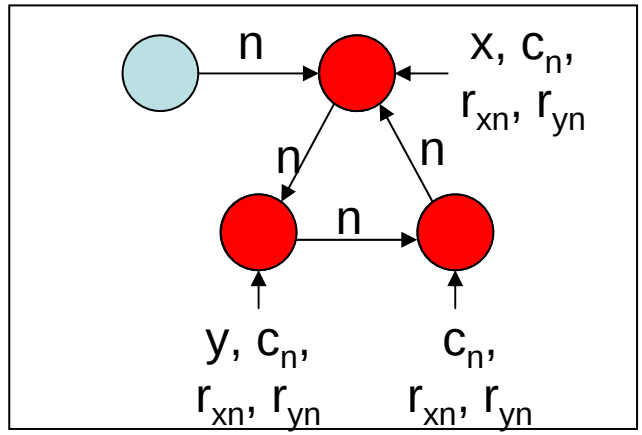




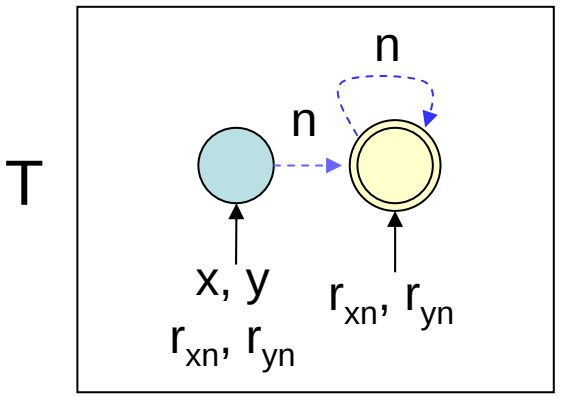
y = x->n

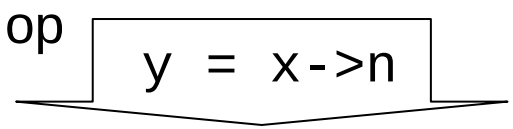
y = x->n

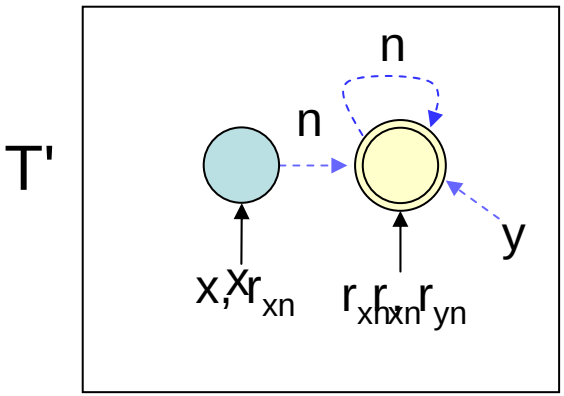




抽象遷移の計算





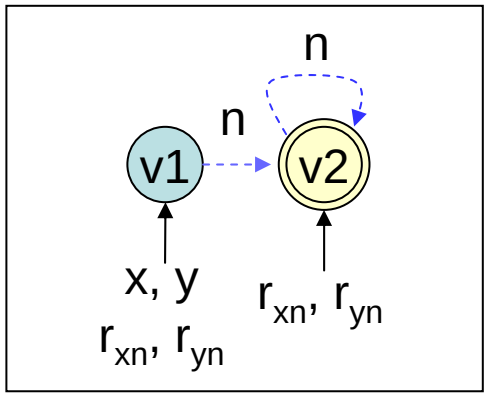


- 3値構造 T に 命令 op を施した結果の3値構造 T' を求める計算
- ノードは変えない．smも同じ．
- p の値は，pre(p, op) を用いて計算する:
  $p^{T'}(v, ...) = (pre(p, op))^{T}(v, ...)$

- pre(x, op) (v) = x(v)
- pre(n, op)(v,v') = n(v,v')
- pre(r_{xn}, op) (v) = r_{xn} (v)
- pre(y, op) (v) = $\exists u.\ x(u) \wedge n(u,v)$
- pre(r_{yn}, op)(v) = $r_{xn}(v) \wedge (c_n(v) \vee \neg x(v))$

くどいようですが．．．．

T

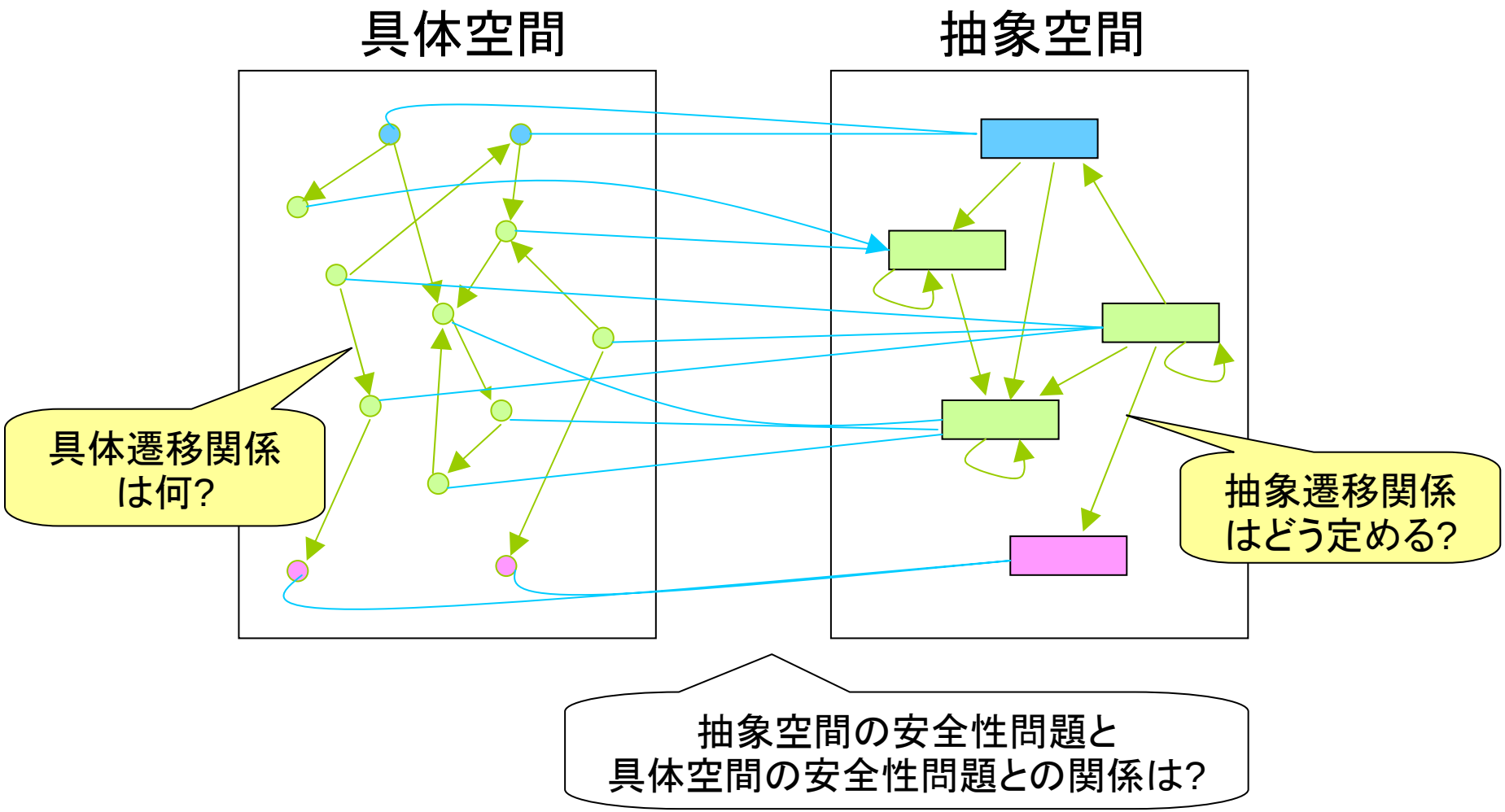


- pre(y, op) (v) = ∃u. x(u)∧n(u,v)

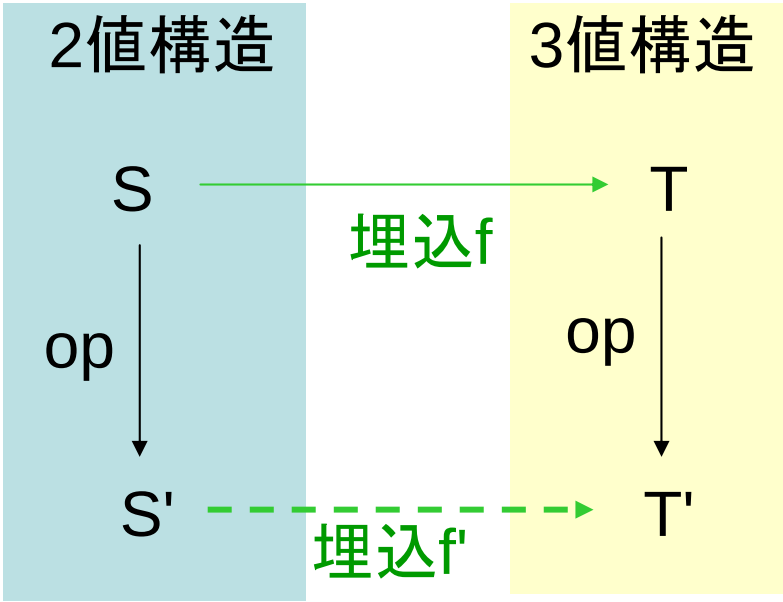
pre(y,op) (v1) = ∃u. x(u)∧n(u,v1)
= (x(v1)∧n(v1,v1)) ∨ (x(v2)∧n(v2,v1))
= (1 ∧ 0) ∨ (0 ∧ 0)
= 0

pre(y,op) (v2) = ∃u. x(u)∧n(u,v2)
= (x(v1)∧n(v1,v2)) ∨ (x(v2)∧n(v2,v2))
= (1 ∧ 1/2) ∨ (0 ∧ 1/2)
= 1/2

抽象化

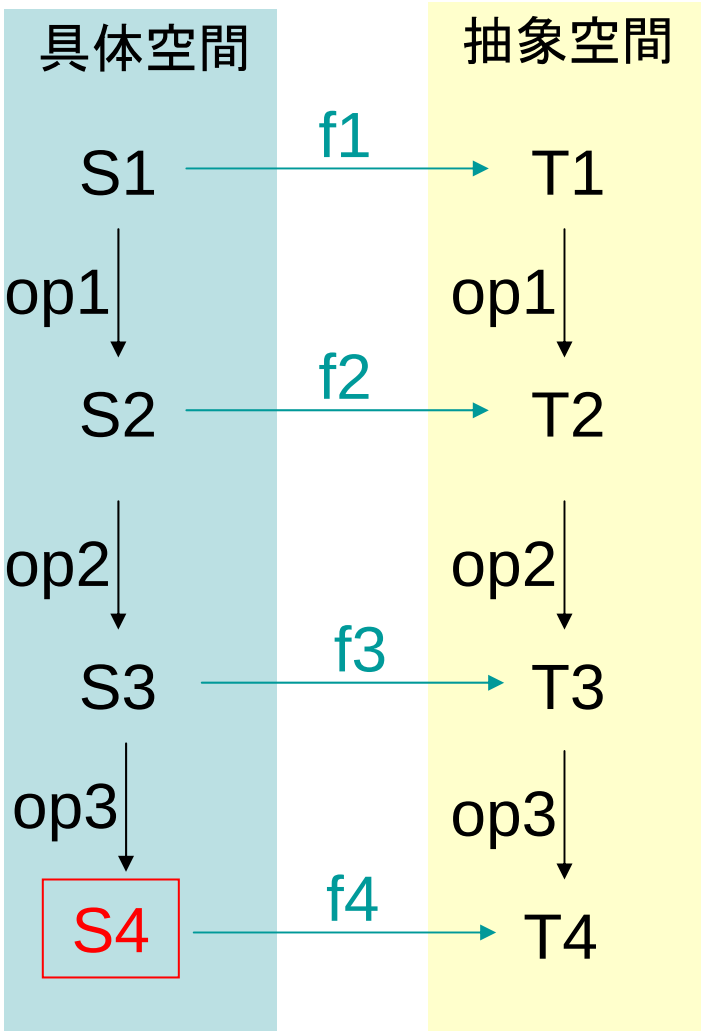

具体空間
抽象空間
具体遷移関係
は何?
抽象遷移関係
はどう定める?
抽象空間の安全性問題と
具体空間の安全性問題との関係は?

抽象化の正当性: 補題



⇒ S'からT'に埋込が存在する.

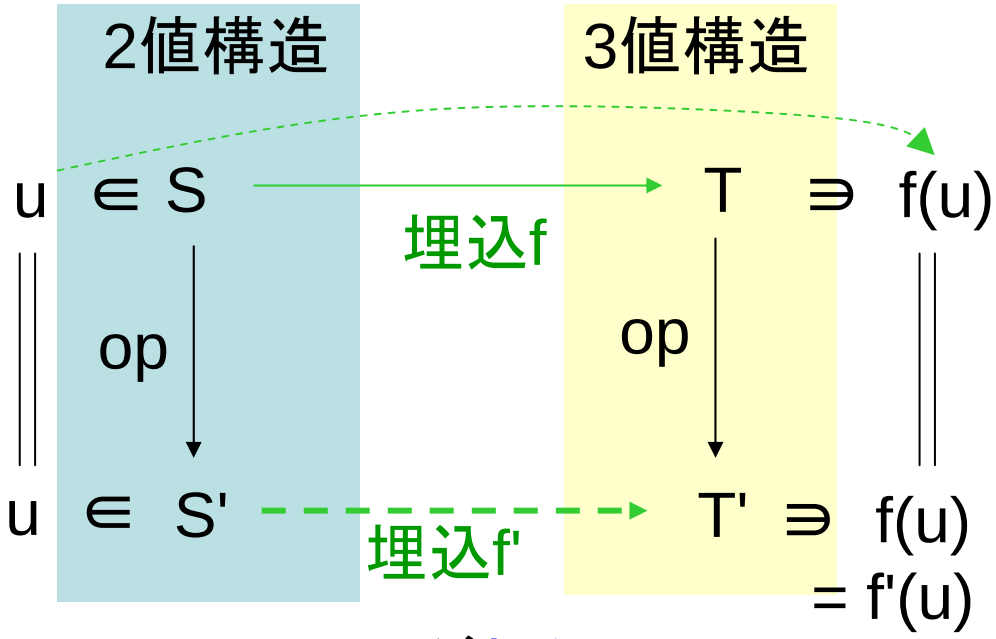
抽象化の正当性



具体空間で危険状態に到達するならば，抽象空間でも危険状態に到達する.

抽象空間で，安全性が確認できれば，具体空間でも，安全性が保障される.

抽象化の正当性: 補題の証明



$p^{S'}(u,..)$

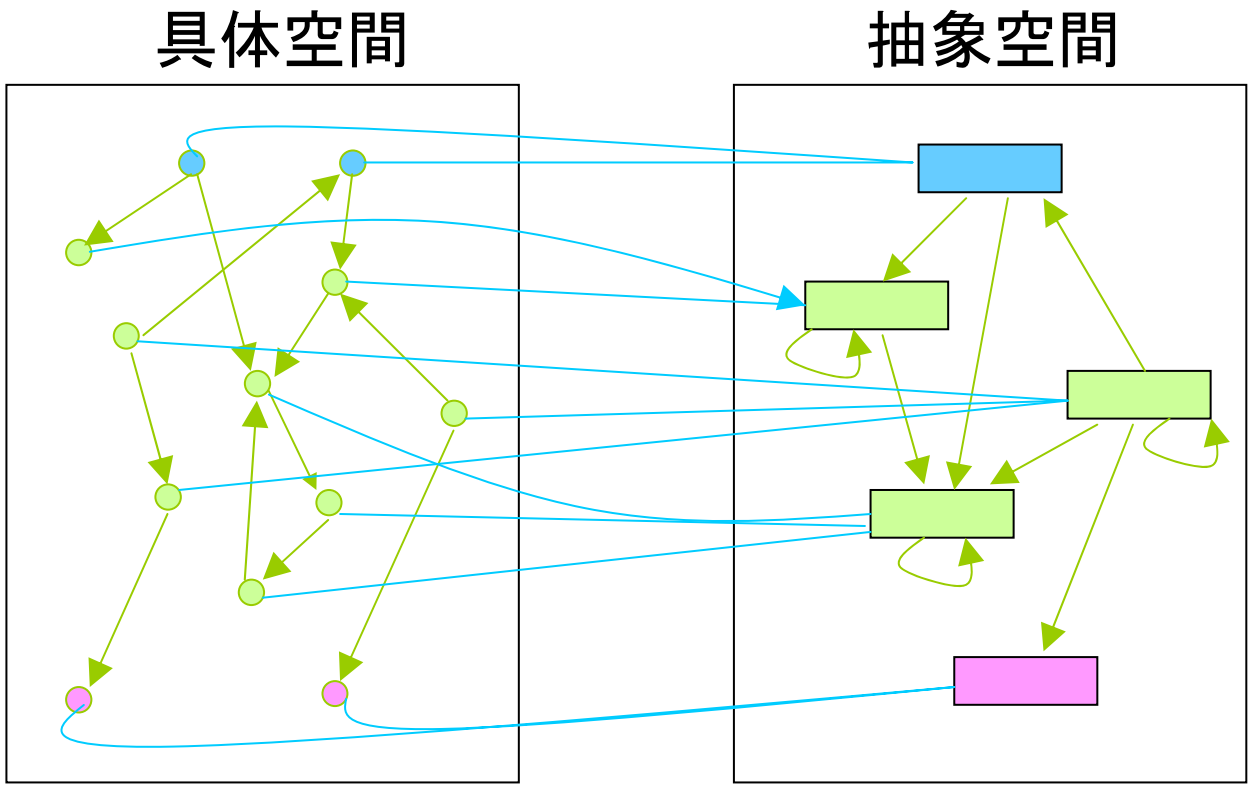
$= pre(p,op)^{S}(u,...)$

$\sqsubseteq pre(p,op)^{T}(f(u),...)$

$= p^{T'}(f'(u),...)$

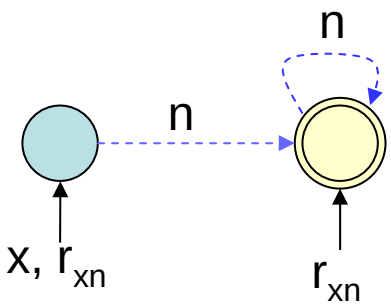
- f: S → T が埋込 iff
  - f は全射
  - v∈T に対し，f(u)=v なる u が2個以上あるとき，$sm^{T}(v) = 1/2$
  - 各述語 p に対し，$p^{S}(u,..) \sqsubseteq p^{T}(f(u),...)$

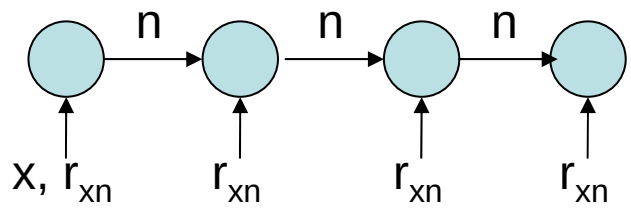
抽象化



抽象空間の安全性問題と
具体空間の安全性問題との関係は?

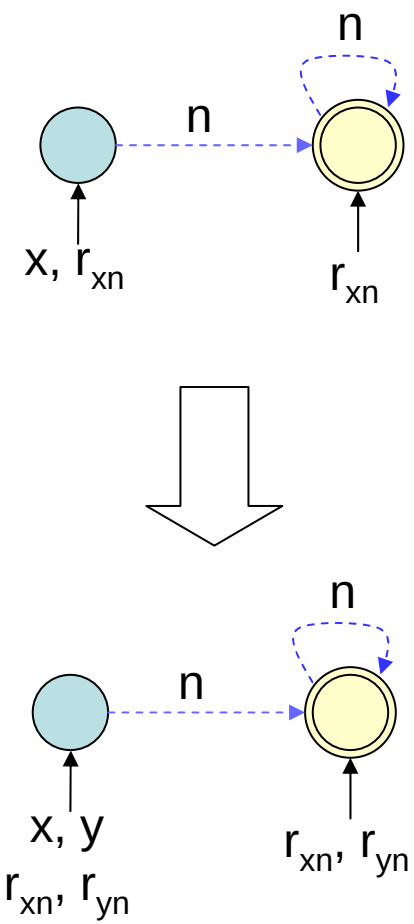
机上実験 (精度は十分?)





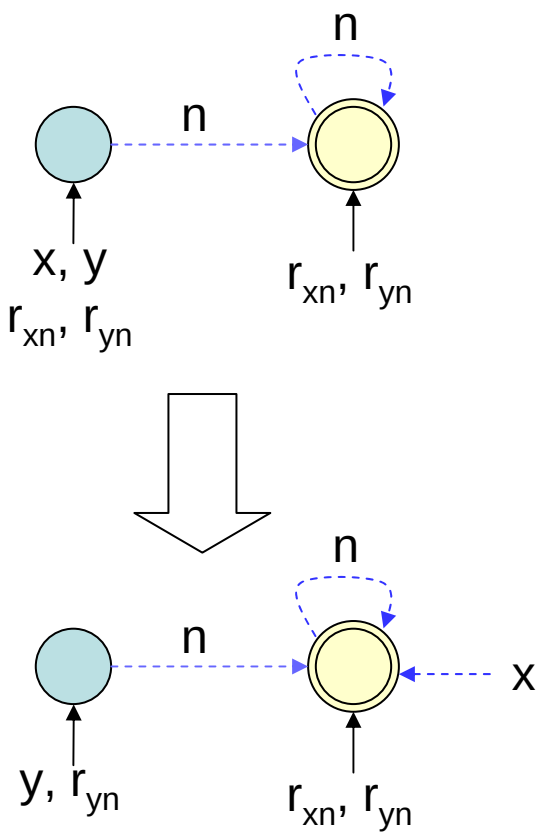

```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

机上実験 (精度は十分?)



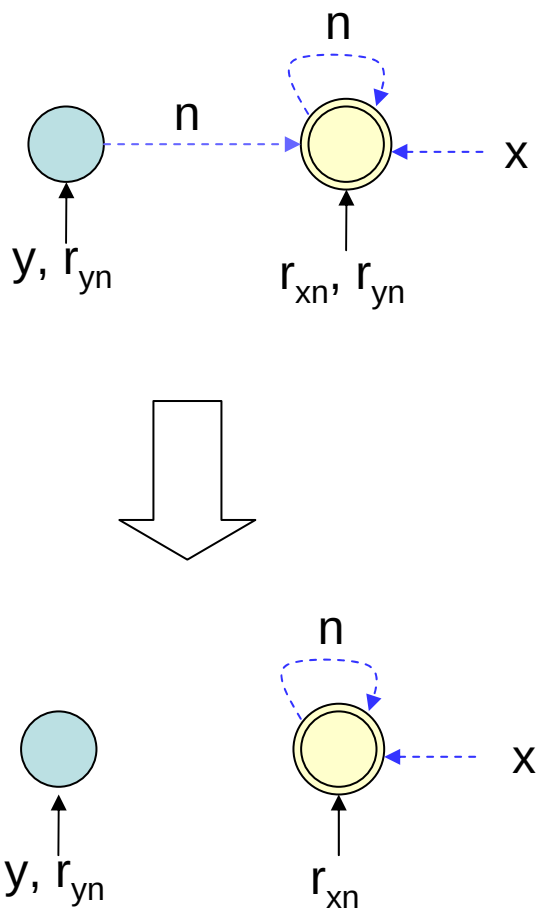

```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

机上実験 (精度は十分?)



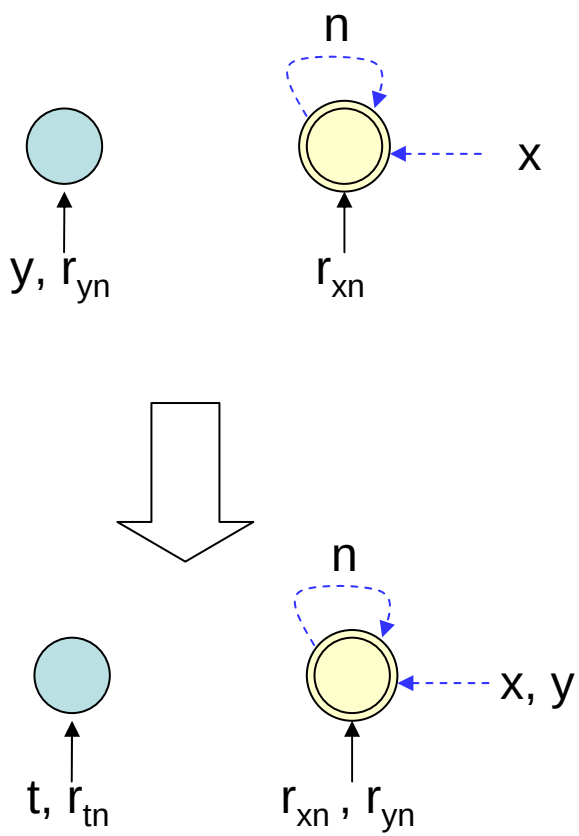

```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

机上実験 (精度は十分?)



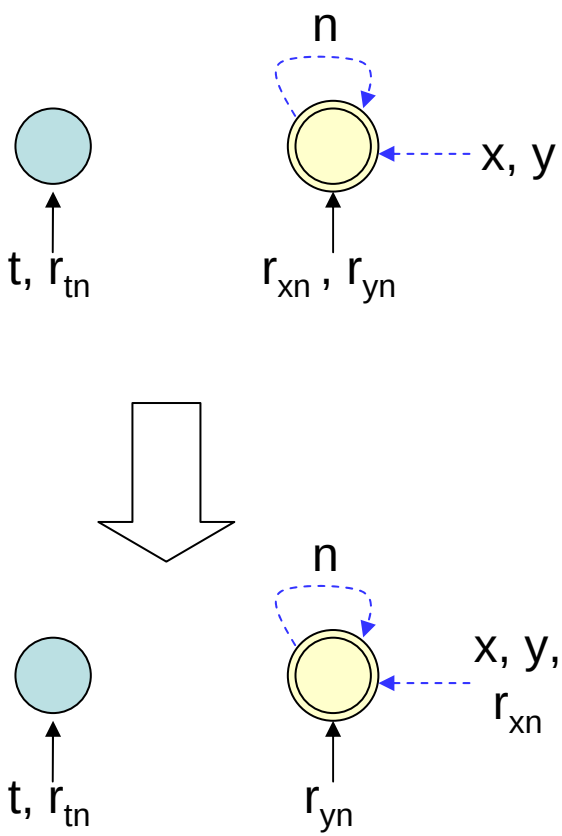

```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

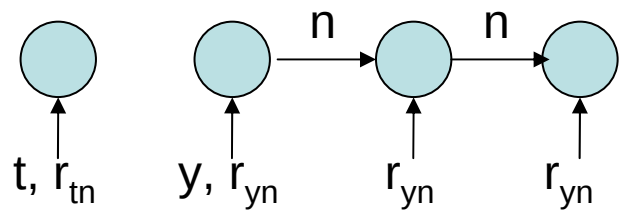
机上実験 (精度は十分?)




```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

机上実験 (精度は十分?)



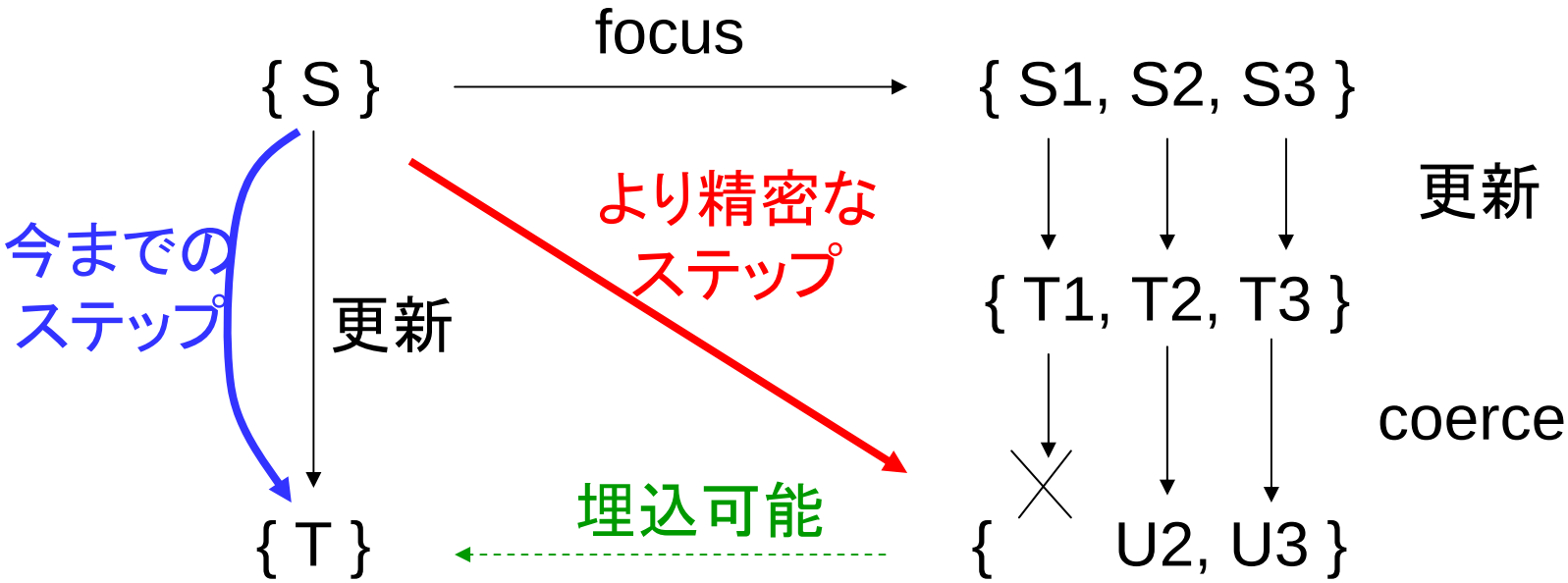



```
y = NULL;
while (x != NULL) {
  t = y;
  y = x;
  x = x->n;
  y->n = t
}
```

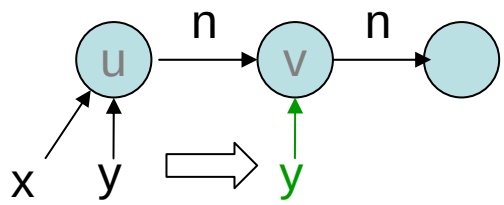
より精密な抽象化

- ここまで述べた抽象化では，検証に必要な精度が得られないことが多い．
- 理由1: ノードの操作は，変数の場所で行われる．変数が，サマリーノードを指していると，情報が失われる．　⇨「focus」
- 理由2: 述語が成り立つことで，ありうる形状に制約がかかる場合がある．この制約が反映できていない．
  ⇨「coerce」

より精密な抽象化(2)


focus
{ S }
{ S1, S2, S3 }
更新
今までの
ステップ
更新
より精密な
ステップ
{ T1, T2, T3 }
coerce
埋込可能
{ T }
{ U2, U3 }

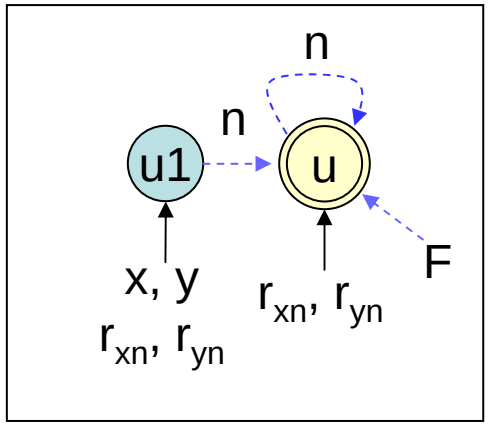
focus



- 例: **y = y->n**
- focus 論理式: F(v) = ∃u. y(u)∧n(u, v)
  = pre(y, y=y->n)
- 実行後に左辺yが指すノードで，実行前にF(v) が成り立つ.
- focus 論理式の値が 1/2 にならないようにノードを分割

focus (2)

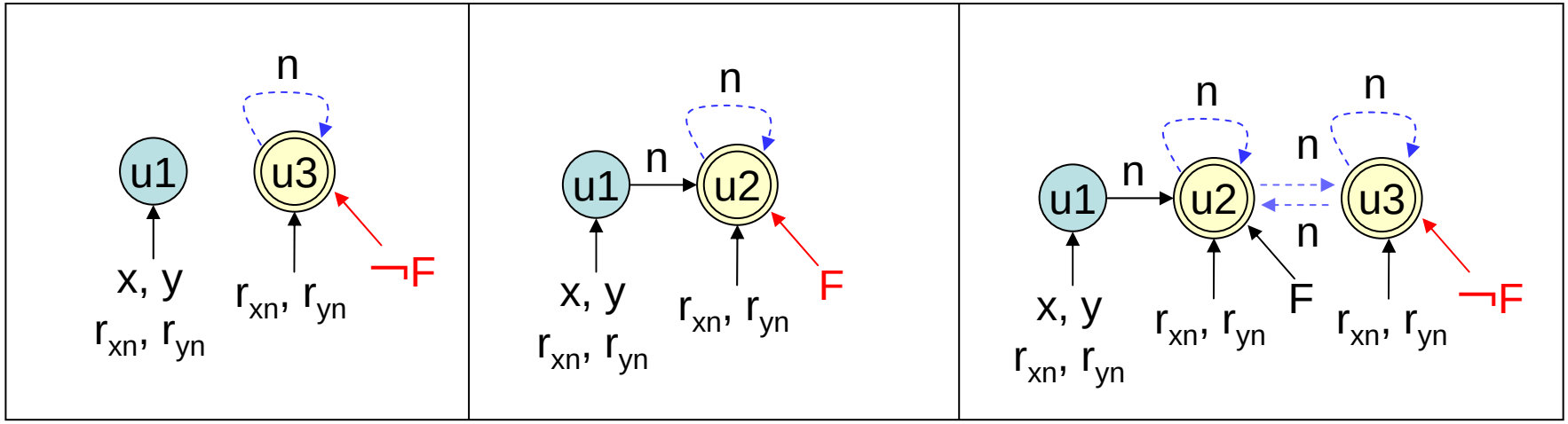
focus前



$F(\cdot) = \exists u_1.\ y(u_1) \wedge n(u_1, \cdot)$

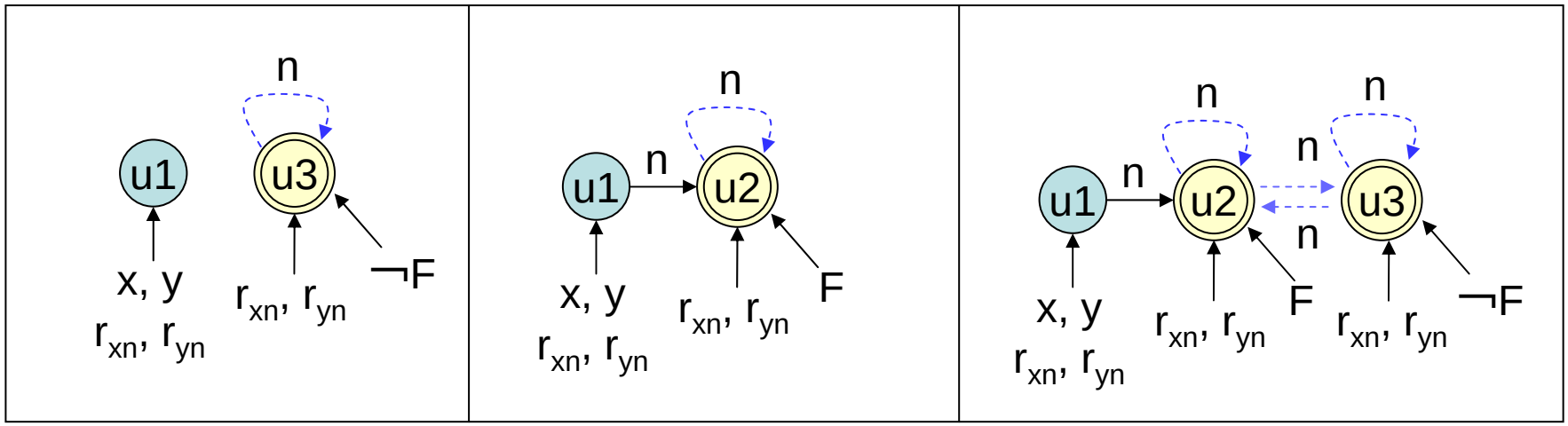
n(u1,u) = 1/2

focus後

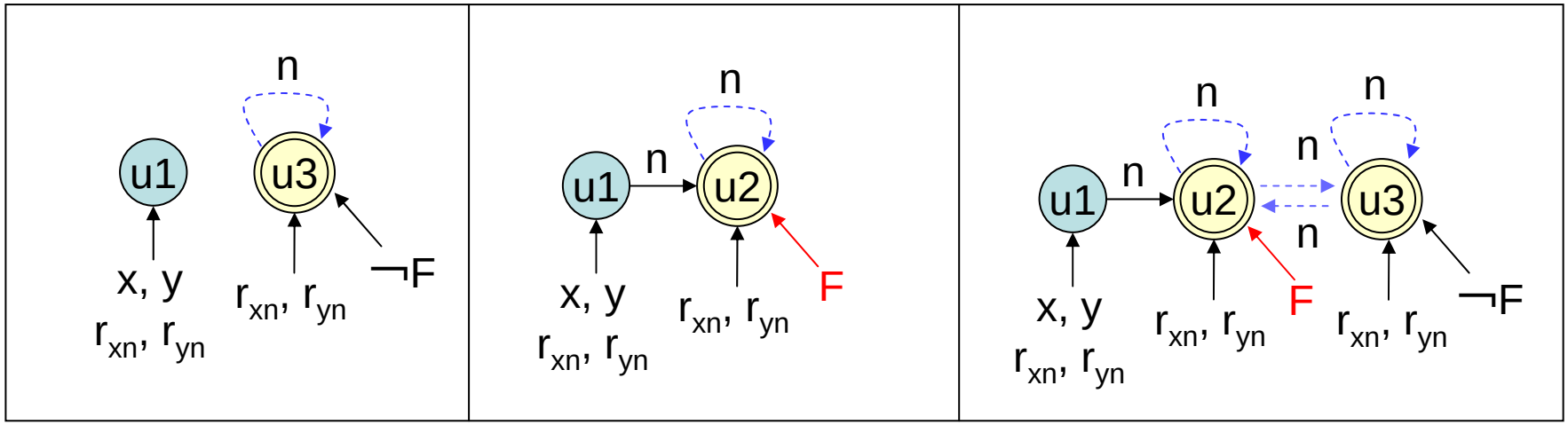


n(u1,u3) = 0,　n(u1, u2) = 1

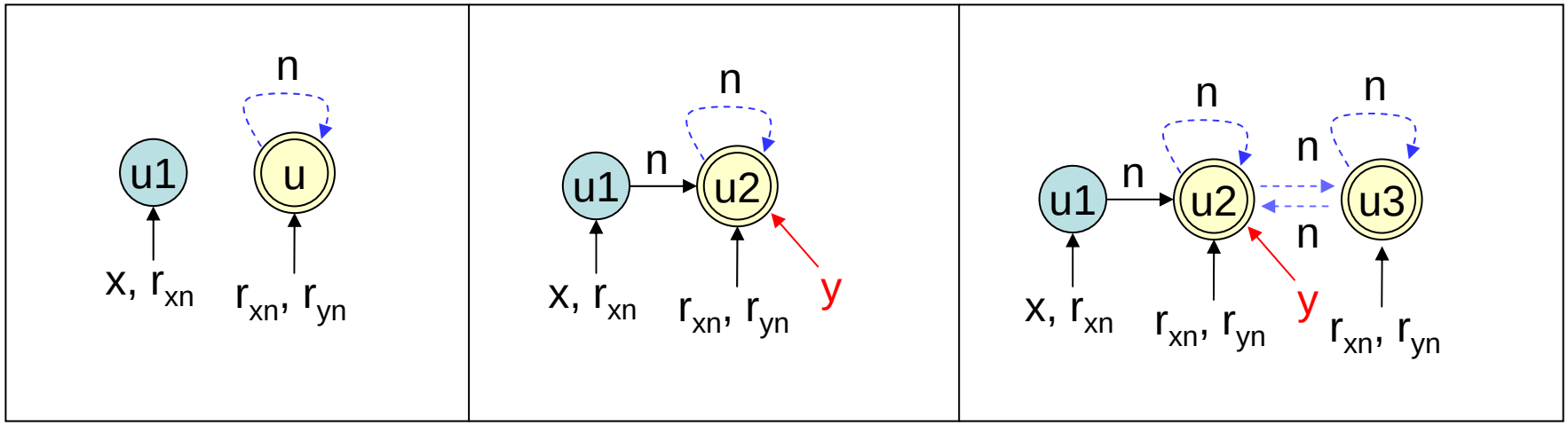
focus (2)


n
u1
u3
x, y
r_xn, r_yn
r_xn, r_yn
¬F
n
u1
n
u2
x, y
r_xn, r_yn
r_xn, r_yn
F
n
n
u1
n
u2
n
u3
n
x, y
r_xn, r_yn
r_xn, r_yn
F
r_xn, r_yn
¬F

更新


n
u1
u3
x, y
r_{xn}, r_{yn}
r_{xn}, r_{yn}
¬F
n
u1
n
u2
x, y
r_{xn}, r_{yn}
r_{xn}, r_{yn}
F
n
n
u1
n
u2
n
u3
n
x, y
r_{xn}, r_{yn}
F
r_{xn}, r_{yn}
r_{xn}, r_{yn}
¬F

y = y->n

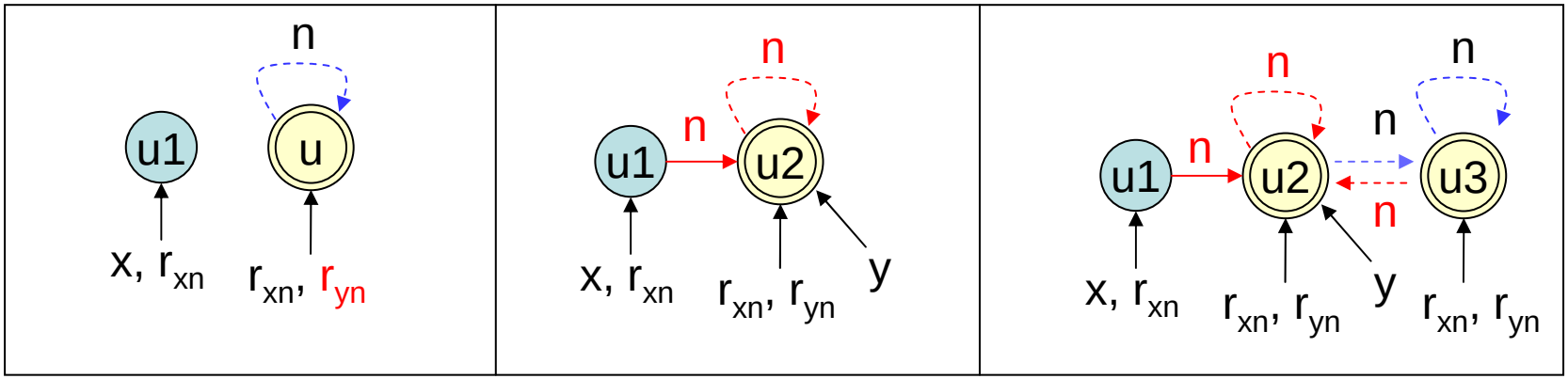

n
u1
u
x, r_{xn}
r_{xn}, r_{yn}
n
u1
n
u2
x, r_{xn}
r_{xn}, r_{yn}
y
n
n
u1
n
u2
n
u3
n
x, r_{xn}
r_{xn}, r_{yn}
y
r_{xn}, r_{yn}

coerce

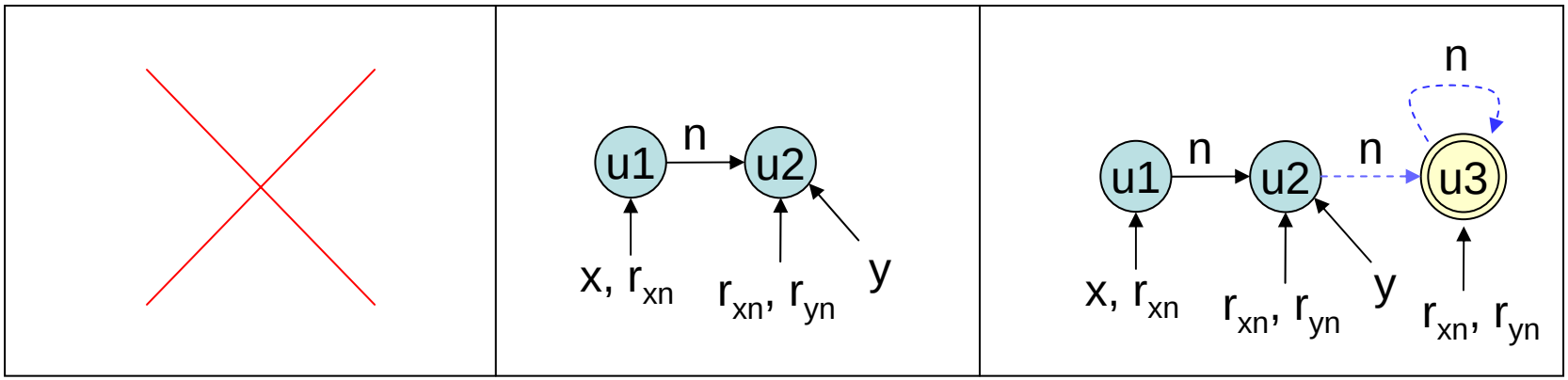
- 存在し得ない構造体を削除する.
  - yが存在しないのに, $r_{y,n}$が成立するノードがある.
  - c_nが成立しているノードで, nのリンクがない.
- 真偽値1/2が割り当てられている命題を0または1に割り当てる.
  - $is_n(v) = 0$, $n(u,v) = 1$, $n(w,v) = 1/2$ ならば, $n(w,v) = 0$ に割り当て直せる.
  - $r_{xn}(u) = 1$, $n(u,v) = 1$, $r_{xn}(v) = 1/2$ ならば, $r_{xn}(v) = 1$ に割り当て直せる.

coerce (2)

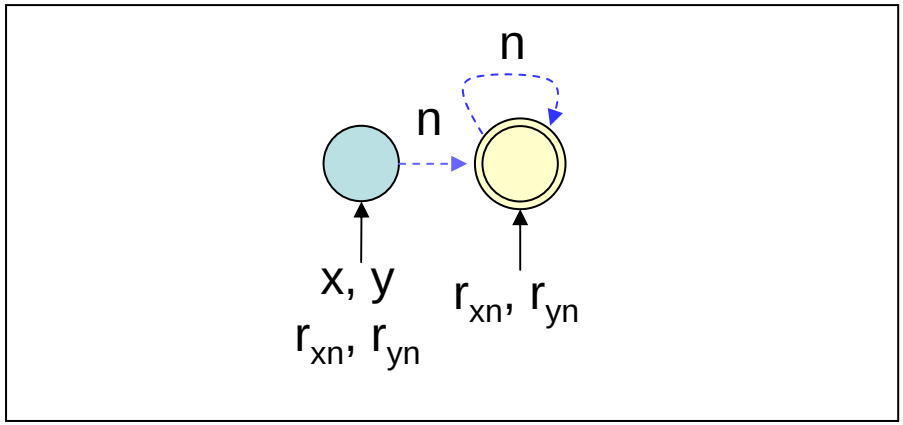
coerce前



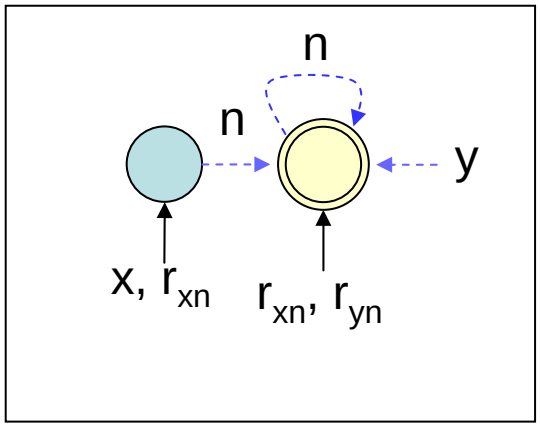
coerce後



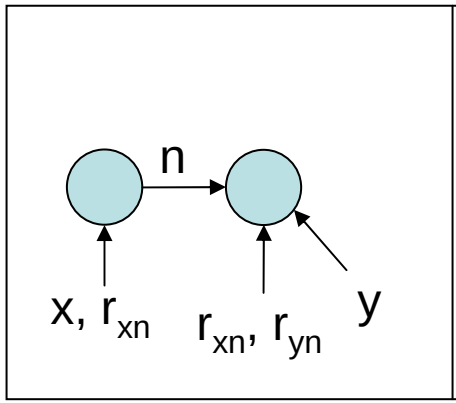
より精密な抽象化(3)

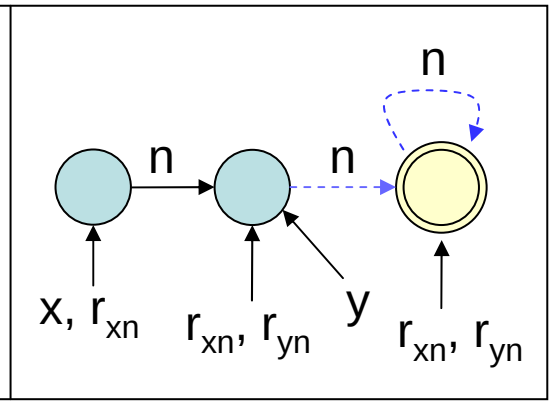


更新



focus + 更新 + coerce





検証例 (TVLA)

…別ウィンドウ…

復習

- アルゴリズムの検証?
- ソースコード検証
- 抽象化による有限遷移系の全数探索
- 述語抽象化
- 反例を用いた抽象構造の詳細化
- シェープ解析
- 3値論理を用いた抽象構造
- focus/coerceによる詳細化

参考文献

E.M. Clarke, O.Grumberg, and D.Peled: Model Checking. MIT Press, 1999
　　モデル検査の (よく参照される) 教科書. 抽象化についても記述されている.
Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar and Gregoire Sutre: Lazy Abstraction. In ACM SIGPLAN-SIGACT Conference on Principles of Programming Languages, pages 58-70, 2002.
　　BLASTの動作原理である遅延抽象化と述語発見法について.
Susanne Graf, Hassen Saidi: Construction of abstract state graphs with PVS. Conference on Computer Aided Verification CAV'97 (LNCS 1254) pp.72-83, 1997
　　述語抽象化について. (下の論文の方がわかりやすいか?)
Thomas Ball, Rupak Majumdar, Todd Millstein, Sriram K. Rajamani: Automatic Predicate Abstraction of C Programs. Conference on Programming Language Design and Implementation 2001, SIGPLAN Notices 36(5), pp. 203-213
　　BLASTと同様の(こちらの方が古い)考え方で設計されているツールSLAMにおける述語抽象化について.
Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu and Helmut Veith: Counterexample-Guided Abstraction Refinement. Computer Aided Verification, 12th International Conference (CAV 2000)
　　反例による抽象構造の詳細化.
Sagiv M., Reps T, and Wilhelm R.: Parametric shape analysis via 3-valued logic TOPLAS, 24:3 (2002)
　　TVLAの動作原理である3値論理によるシェープ解析について.
Alexey Loginov, Thomas Reps and Mooly Sagiv: Automated Verification of the Deutsch-Schorr-Waite Tree-Traversal Algorithm. The 13th International Static Analysis Symposium (SAS 2006)
　　TVLAによるDeutsch-Schorr-Waiteアルゴリズムの検証. オリジナルのアルゴリズムとは若干異なる.

（スライド末尾)

このスライド(またはそのアップデート版)は，以下のURLからダウンロードできます:
http://staff.aist.go.jp/tanabe.yoshinori/06/09/12/