古典的プランニング問題と
グラフ探索アルゴリズム

陣内　佑
総合文化研究科広域科学専攻

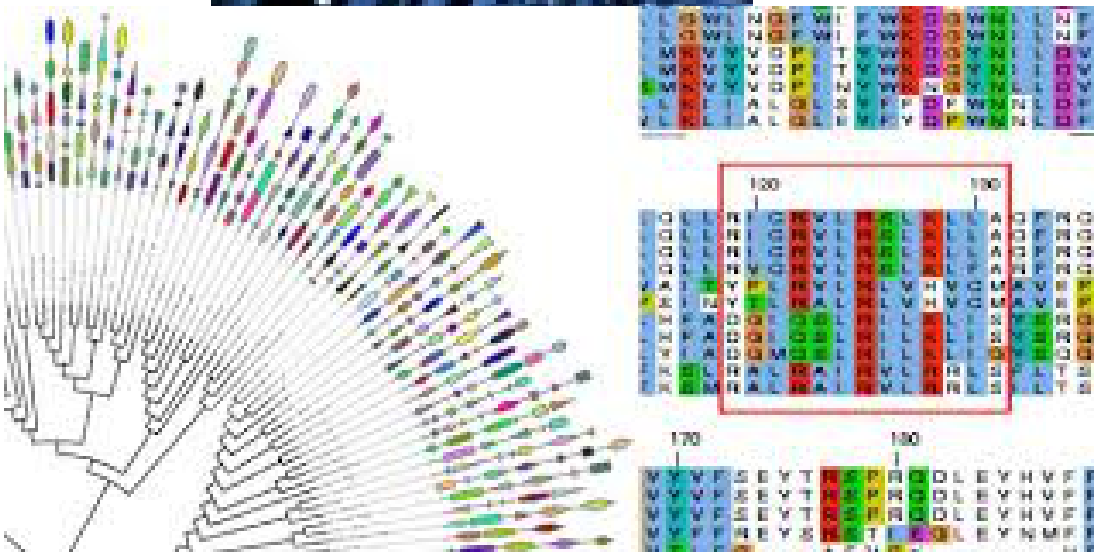
自己紹介

- 陣内　佑
- 所属：東京大学総合文化研究科 (３月卒業予定)
- ３月から理研 AIP で働く予定
- ９月から PhD プログラムに入学予定 (どこかは未定)

おはなし

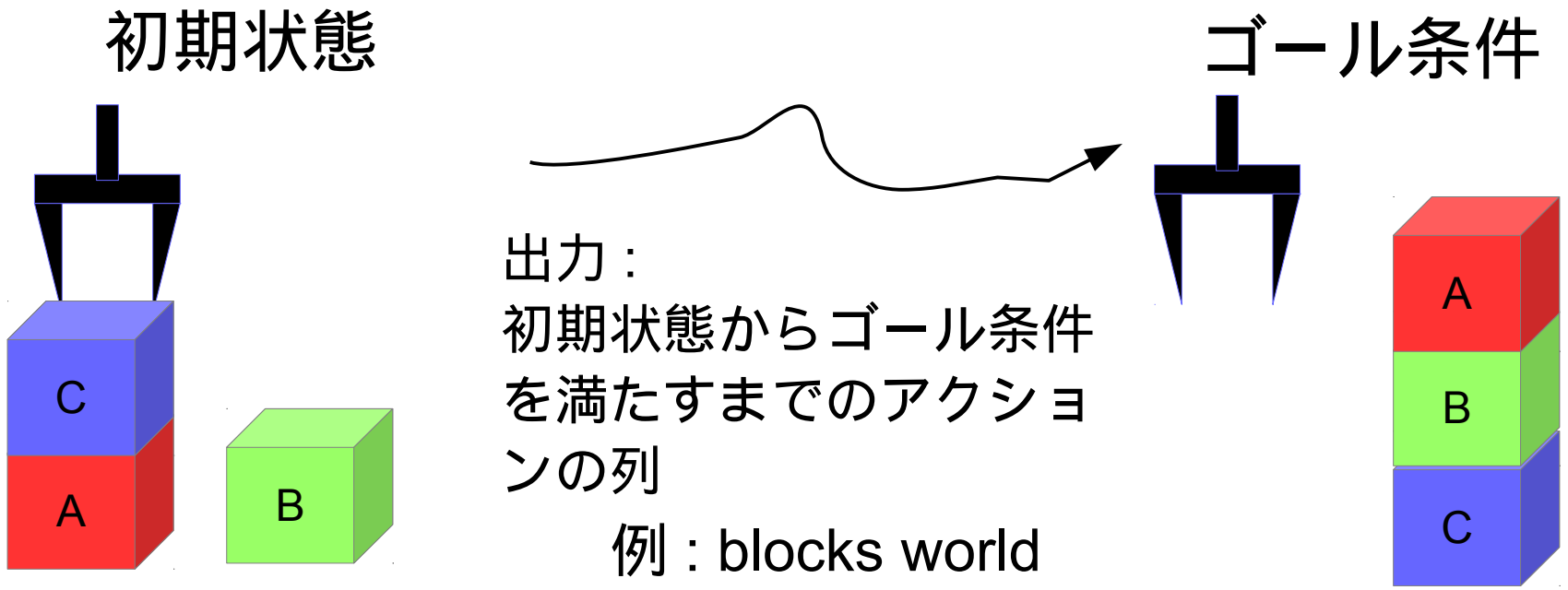
- **古典的プランニング問題**
- A* 探索
- BDD を用いたグラフ探索
- 並列 A* 探索 (Hash Distributed A*)
- Black-box Planning (ビデオゲーム AI)

プランニング問題



プランニング問題

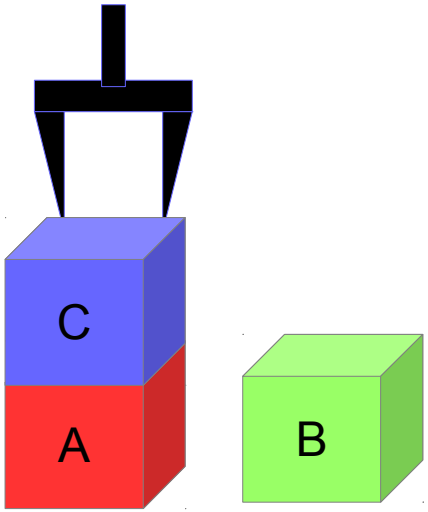
- 入力：初期状態とゴール条件、可能なアクション集合
- 出力：初期状態からゴール条件を満たすためのアクションの列



色々なプランニング

- 古典的プランニング
  - 命題からなる世界
- モーションプランニング
  - ロボットなどの行動計画
- Black-box プランニング
  - シミュレーターを用いた行動計画

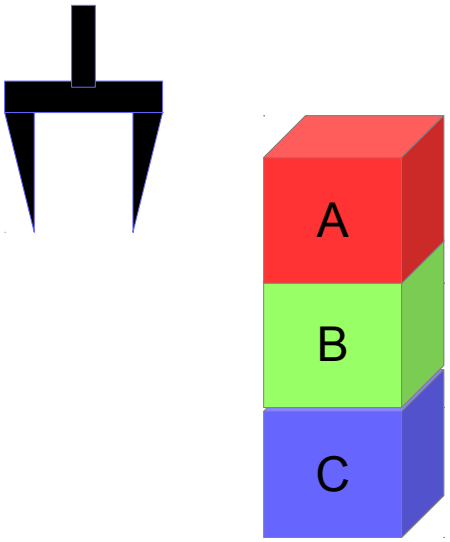
初期状態



出力：
初期状態からゴール条件を満たすまでのアクションの列

例：blocks world

ゴール条件



古典的プランニング (Classical planning)

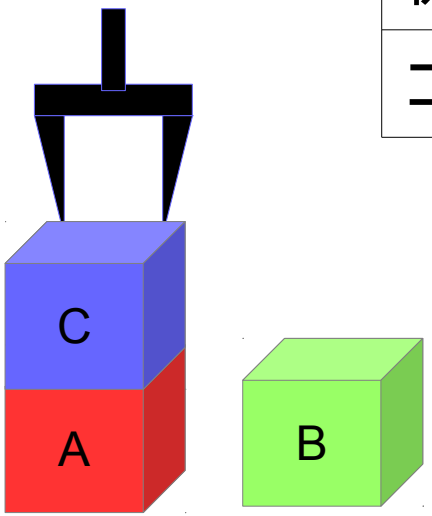
(Fikes&Nilsson'71)

古典的プランニング問題 $P = (S, A, s_0, T)$

- 世界は proposition の集合 AP として表せられる

  e.g. `on(a,table),on(b,table),clear(a),clear(b)`

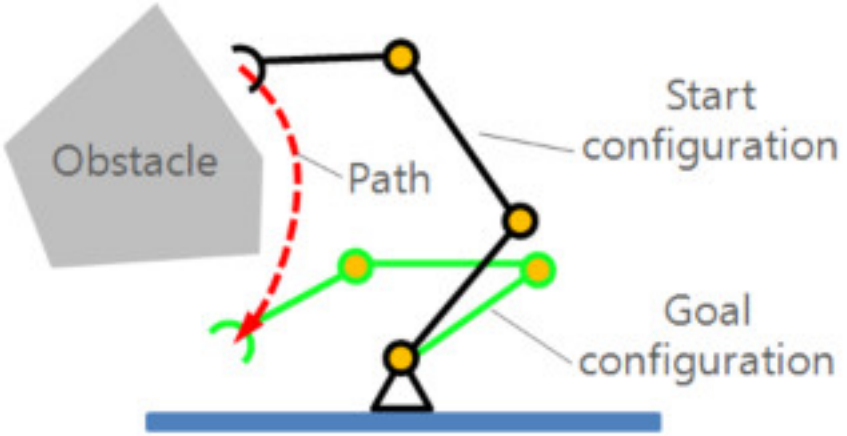
状態の集合	$S = \Omega^{AP}$
アクションの集合	$a \in A : S \rightarrow S$
初期状態	$s_0 \in S$
ゴール状態集合	$T \subseteq S$



モーションプランニング

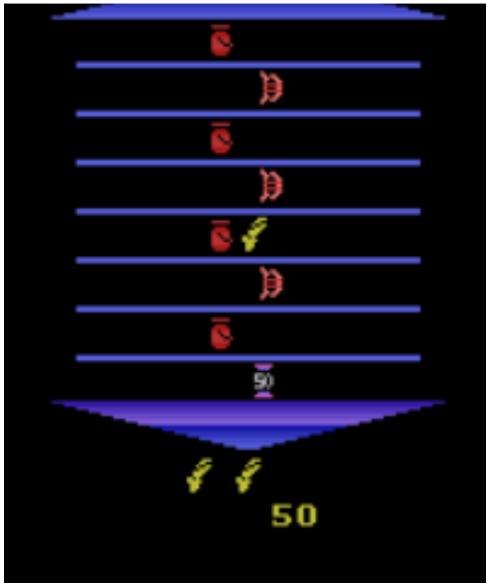
- ロボットのある状態 (configuration) から目的の状態へ物理的制約を満たしながら遷移するプラン

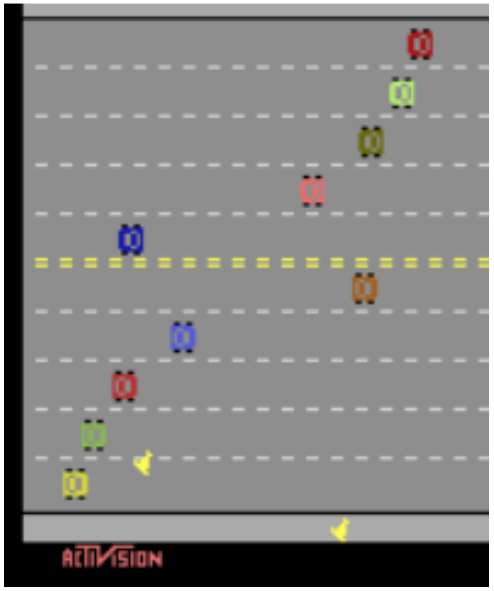
Google Robot Dog



Black-box Planning
(e.g. Bellemare et al.'13)

- ブラックボックスのシミュレーターを用いてプランニングを行う





Arcade Learning Environment
(Bellemare et al. 2013)

古典的プランニング (Classical planning)

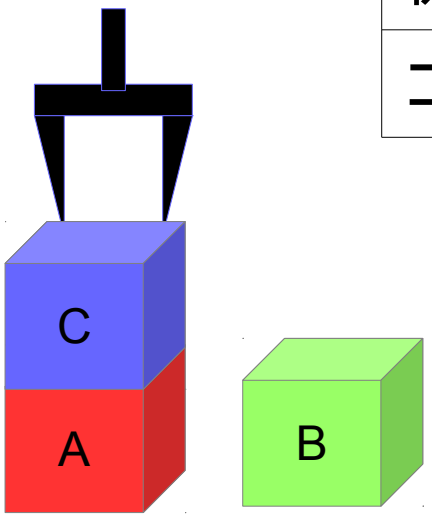
(Fikes&Nilsson'71)

古典的プランニング問題 $P = (S, A, s_0, T)$

- 世界は proposition の集合 AP を用いて表せられる
- 状態は AP の部分集合である

e.g. `on(a,table),on(b,table),clear(a),clear(b)`

状態の集合	$S = \Omega^{AP}$
アクションの集合	$A : S \rightarrow S$
初期状態	$s_0 \in S$
ゴール状態集合	$T \subseteq S$



STRIPS planning
(Fikes&Nilsson'71)

- ゴール状態集合 T はゴール条件 $g \subseteq AP$ がすべて真である状態の集合と定義される：

  $T = \{s;\ s \supseteq g\}$

- アクション a は precondition, add effect, delete effect の 3 つの proposition の集合によって定義される：
  - $pre(a)$:　アクションを実行するために状態が満たすべき命題
  - $add(a)$:　アクションの実行によって新たに追加される命題
  - $del(a)$:　アクションの実行によって消去される命題

- 状態 s にアクション a を適用した後の状態 $s' = a(s)$ は

  $s' = (\ s \cup add(a)\ ) \setminus del(a)$

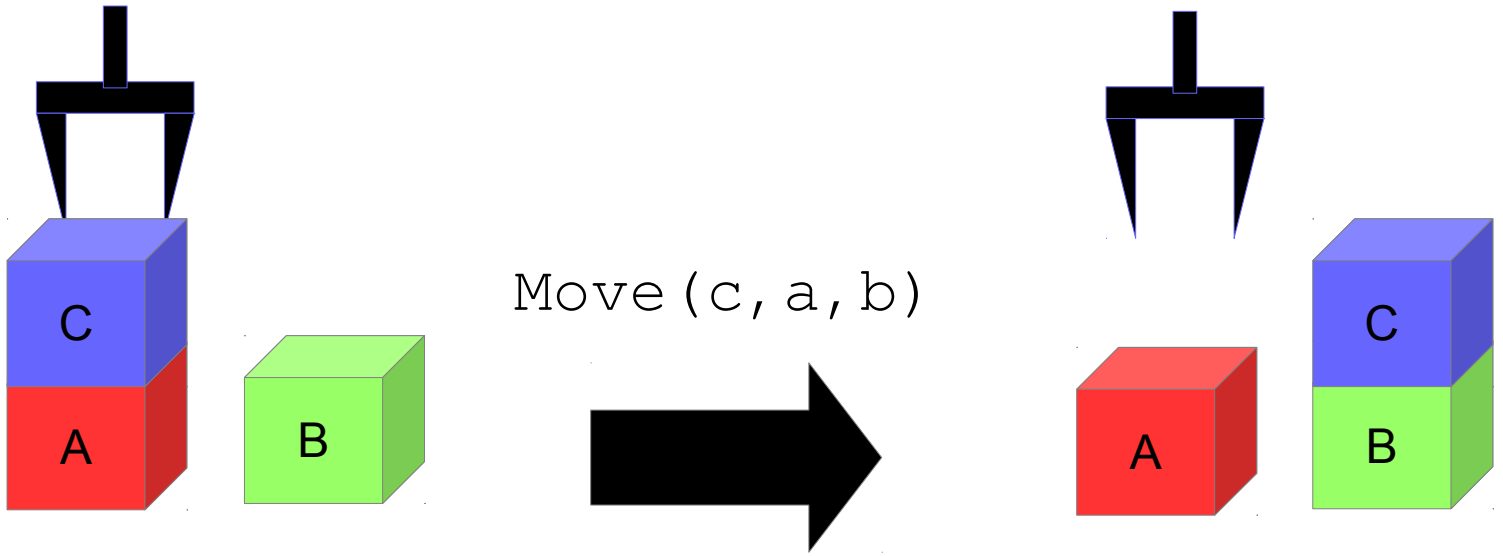
STRIPS planning
(Fikes&Nilsson'71)

- アクション a は precondition, add effect, delete effect の 3 つの proposition の集合によって定義される：
  - $pre(a)$: アクションを実行するために状態が満たすべき命題
  - $add(a)$: アクションの実行によって新たに追加される命題
  - $del(a)$: アクションの実行によって消去される命題

```
###Blocks-world-domain.pddl
Action:
   Move(t,x,y)
      Precond: on(x,t),clear(x),clear(y)
      Effect:  on(y,t),clear(x),¬on(x,t),¬clear(y)
```

STRIPS planning
(Fikes&Nilsson'71)

```
###Blocks-world-domain.pddl
Action:
   Move(f,x,t)
      Precond: on(x,f),clear(x),clear(t)
      Effect:   on(x,t),clear(f),¬on(x,f),¬clear(t)
```

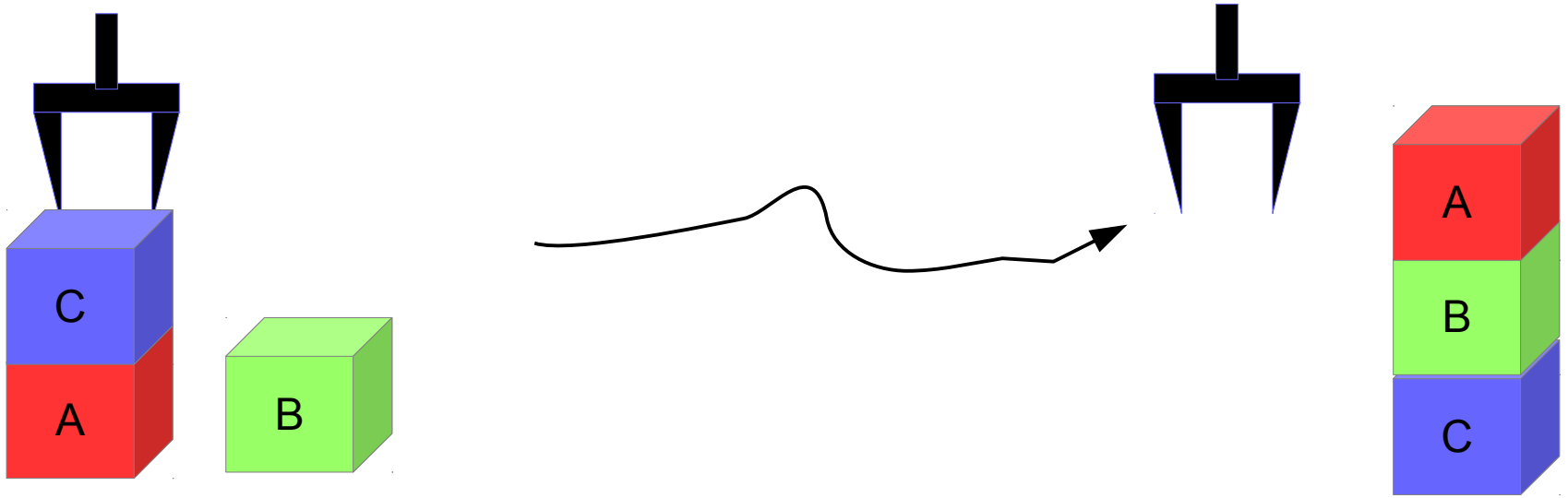


STRIPS planning
(Fikes&Nilsson'71)

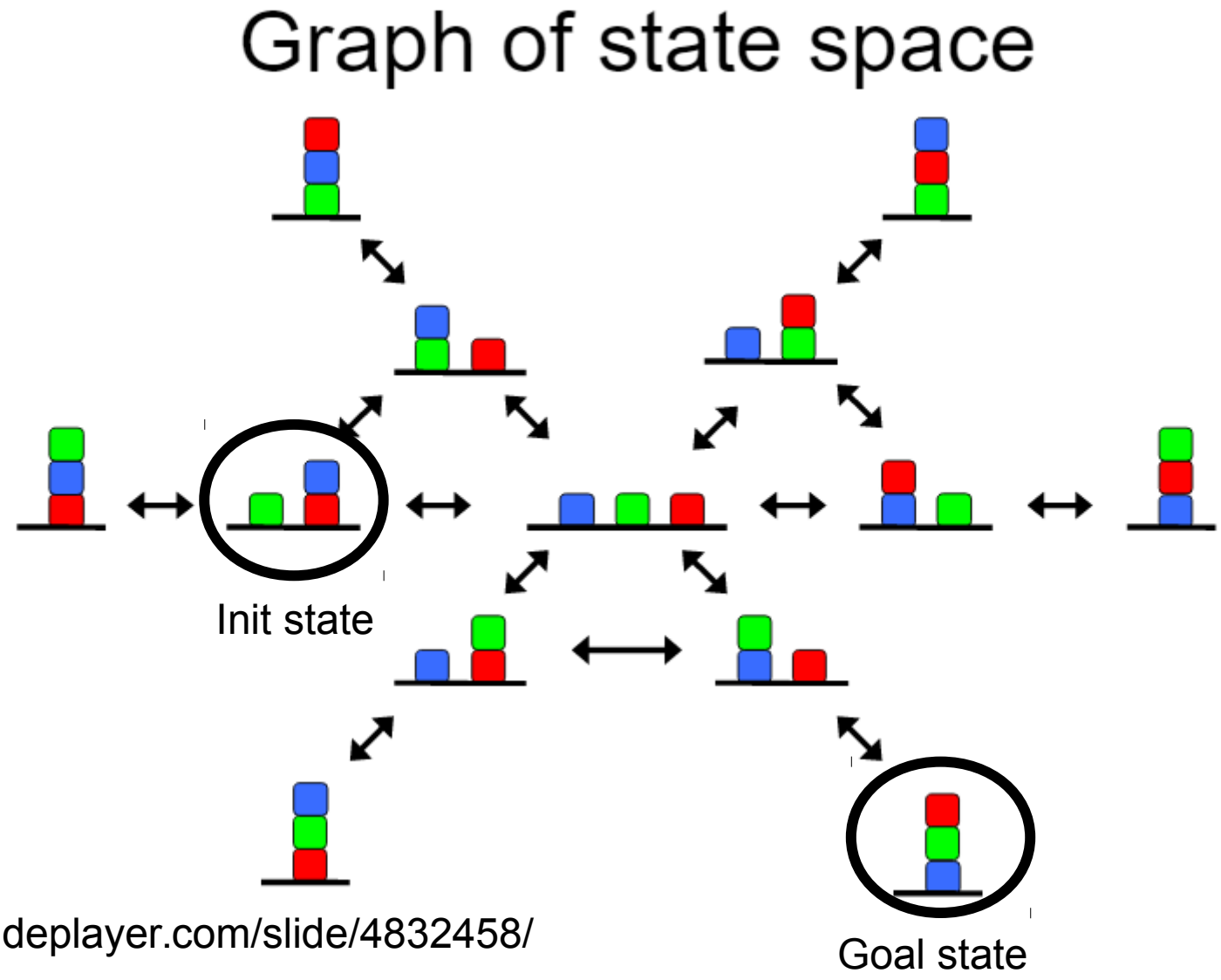
- ゴール状態集合 T はゴール条件 $g \subseteq AP$ がすべて真である状態の集合と定義される：

$$T = \{s; s \supseteq g\}$$

```
###Blocks-world-instance.pddl
Init: on(a,table),on(b,table),on(c,a),clear(b),
        clear(c)
Goal: on(a,b),on(c,b)
```



STRIPS planning



From http://slideplayer.com/slide/4832458/

→ グラフ探索アルゴリズムによって解くことが出来る！

Example input files

```
###gripper-domain.pddl
(define (domain gripper-strips)
   (:predicates (room ?r)
     (ball ?b)
     (gripper ?g)
     (at-robby ?r)
     (at ?b ?r)
     (free ?g)
     (carry ?o ?g))

   (:action move
       :parameters  (?from ?to)
       :precondition (and  (room ?from) (room ?to)
                           (at-robby ?from))
       :effect (and  (at-robby ?to)
          (not (at-robby ?from))))
```

Example input files

```
###gripper-instance.pddl
(define (problem strips-gripper-x-1)
   (:domain gripper-strips)
   (:objects rooma roomb ball4 ball3 ball2 ball1
left right)
   (:init (room rooma)
          (room roomb)
          (ball ball4)
          (ball ball3)
          (ball ball2)
          ...)
   (:goal (and (at ball4 roomb)
               (at ball3 roomb)
               (at ball2 roomb)
               (at ball1 roomb)
     )))
```

Example input files

```
###sokoban-domain.pddl
(define (domain sokoban-sequential)
  (:types thing location direction - object
          player stone - thing)
  (:predicates (clear ?l - location)
          (at ?t - thing ?l - location)
          (at-goal ?s - stone)
          (IS-GOAL ?l - location)
          (IS-NONGOAL ?l - location)
               (MOVE-DIR ?from ?to - location
                         ?dir - direction))
  (:functions (total-cost) - number)
  (:action move
   :parameters (?p - player ?from ?to - location
                ?dir - direction)
   :precondition (and (at ?p ?from)
                      (clear ?to)
```

Example input files

```
###sokoban-instance.pddl
(define (problem p012-microban-sequential)
  (:domain sokoban-sequential)
  (:objects
    dir-down - direction
    dir-left - direction
    dir-right - direction
    dir-up - direction
    player-01 - player
    ...)
  (:init
    (IS-GOAL pos-6-5)
    (IS-NONGOAL pos-1-1)
    ...)
  (:goal (and
    (at-goal stone-01)
    (at-goal stone-02)))
```

おはなし

- 古典的プランニング問題
- **A* 探索**
- BDD を用いたグラフ探索
- 並列 A* 探索 (Hash Distributed A*)
- Black-box Planning (ビデオゲーム AI)

古典的プランニング問題の解法

- グラフ探索アルゴリズムによるプランニング
  - A* 探索など
- モデルベースプランニング
  - SAT, IP, CSP に変換して解く

状態空間グラフ

- 状態空間グラフ $G=(V, E, s, T)$ はプランニング問題 $P=(S, A, s_0, T)$ に対して以下のように定義される

	プランニング問題 $P=(S, A, s_0, T)$	状態空間グラフ $G=(V, E, s, T)$
状態の集合	$S = \Omega^{AP}$	$V = S$
アクションの集合	$a \in A : S \rightarrow S$	$(u,v) \in E$ *iff* $a \in A$ *s.t.* $a(u) = v$
初期状態	$s_0 \in S$	$s = s_0$
ゴール状態集合	$T \subseteq S$	$T = T$

コスト付き状態空間グラフ

- コスト付き状態空間グラフ $G=(V, E, s, T, w)$ はコスト付きプランニング問題 $P=(S, A, s_0, T, w)$ に対して以下のように定義される

	プランニング問題 $P=(S, A, s_0, T, w)$	状態空間グラフ $G=(V, E, s, T, w)$
状態の集合	$S = \Omega^{AP}$	$V = S$
アクションの集合	$a \in A : S \rightarrow S$	$(u,v) \in E$ *iff* $a \in A$ *s.t.* $a(u) = v$
初期状態	$s_0 \in S$	$s = s_0$
ゴール状態集合	$T \subseteq S$	$T = T$
アクションコスト	$w: A \rightarrow \mathbb{R}$	$w: E \rightarrow \mathbb{R}$

非明示的状態空間グラフ探索
(Implicit state-space search)

- 初期状態 $s \in V$
- ゴール条件 Goal: $V \rightarrow$ {false, true}
- ノード展開関数 $Expand: V \rightarrow 2^V$ は入力ノードの子ノードを返す

- 多くの場合状態空間グラフ全てをメモリに乗せるのは不可能
- 探索開始時のエージェントは初期状態の情報しか保持していない
- エージェントが保持しているノードの集合をクローズドリスト (closed list) と呼ぶ
- クローズドリストにあるノードのうち未 Expand のノードの集合をオープンリスト (open list) と呼ぶ

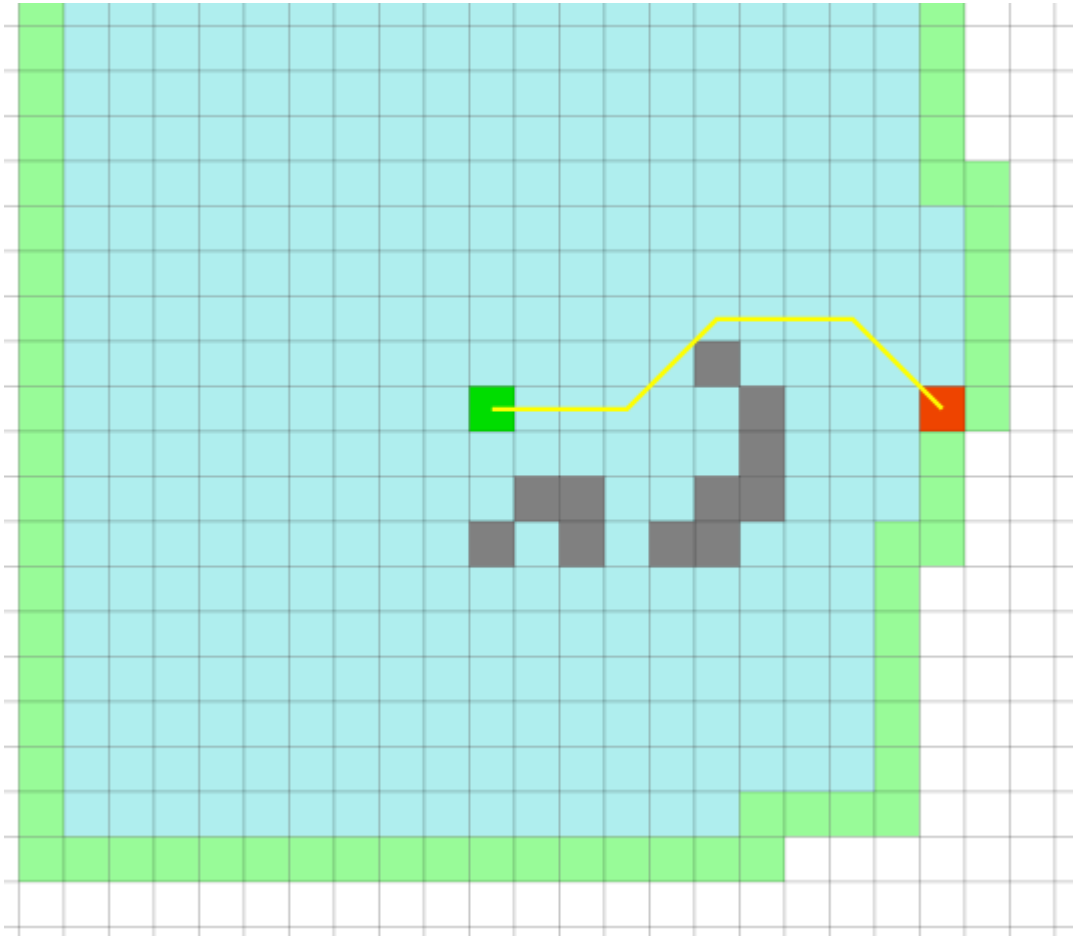
非明示的状態空間グラフ探索
(Implicit state-space search)

- 初期状態 $s \in V$
- ゴール条件 Goal: $V \rightarrow$ {false, true}
- ノード展開関数 $Expand: V \rightarrow 2^V$ は入力ノードの子ノードを返す

- オープンリストからノードを選び Expand を実行することによって新しいノードの情報を得ることが出来る
- **どのノードを Expand していくかが探索の効率を左右する**

例：幅優先探索

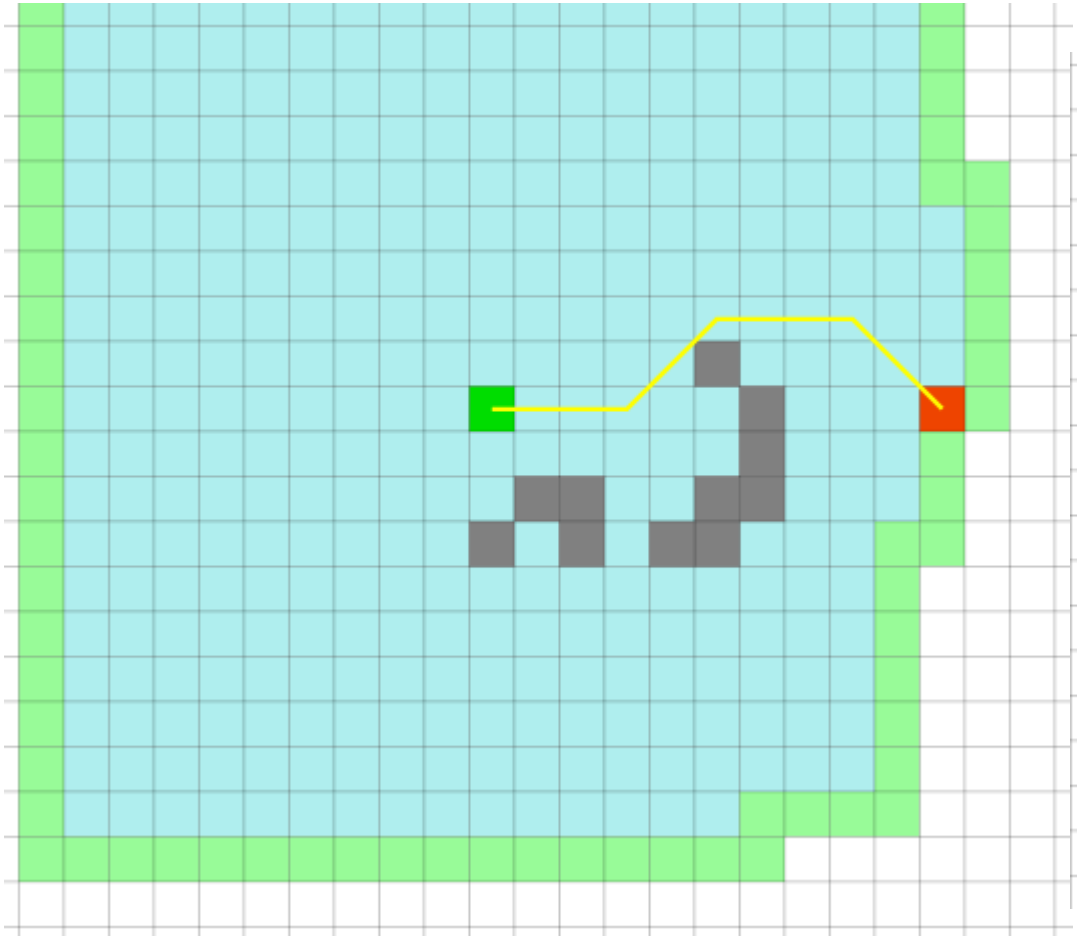
- 幅優先探索は最も初期状態からの距離が小さいノードを優先して Expand する (非明示的) 状態空間探索アルゴリズムである
- デモ : https://qiao.github.io/PathFinding.js/visual/



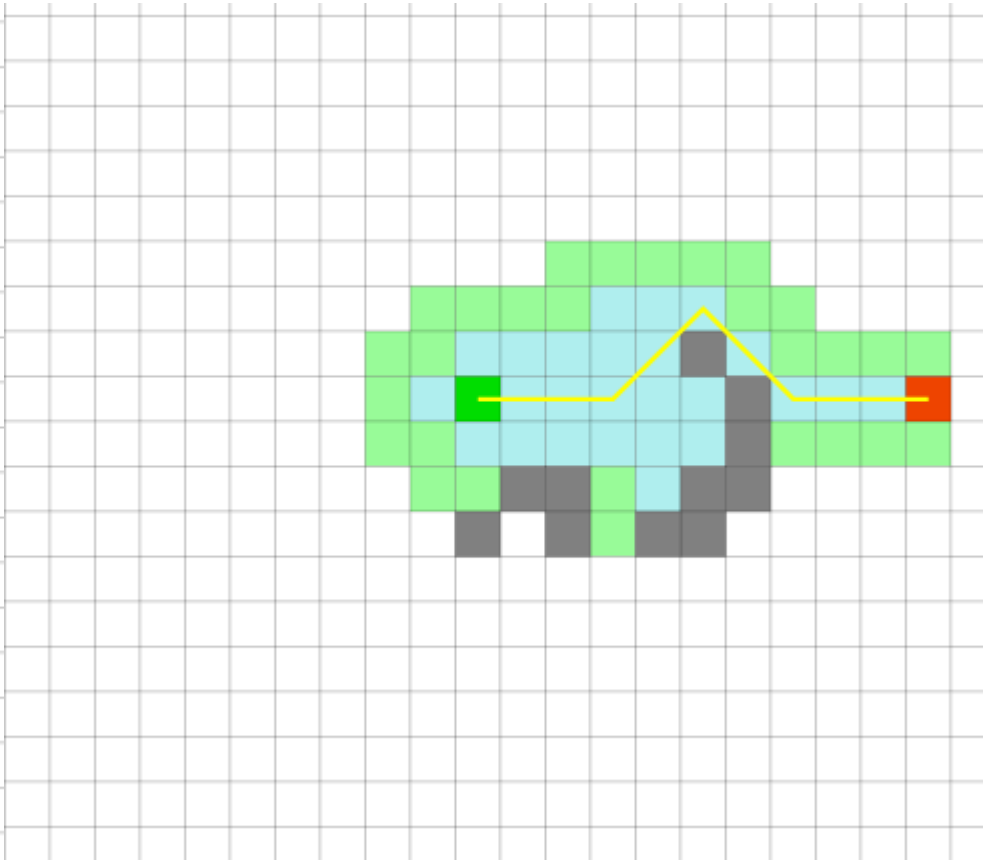
幅優先探索

例：幅優先探索

- 幅優先探索は最も初期状態からの距離が小さいノードを優先して Expand する (非明示的) 状態空間探索アルゴリズムである
- デモ : https://qiao.github.io/PathFinding.js/visual/



幅優先探索



A* 探索

A* 探索
(Hart et al.'68)

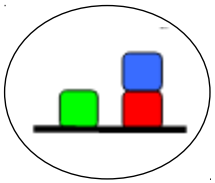
- ノード $n \in V$ の有望さ f 値が最小のノードを展開 (Expand) していく

$$f(n) = g(n) + h(n)$$

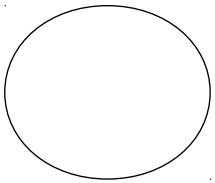
- **ヒューリスティック関数** $h: V \to \mathbb{R}$ はノードからゴールまでのコストを見積もる
- 初期状態からノードへの既知の最小コスト $g: V \to \mathbb{R}$
- f 値は初期状態からノード n を通ってゴールへ辿り着くためのコストの見積もり
- デモ : https://qiao.github.io/PathFinding.js/visual/

A* 探索
(Hart et al.'68)

- ゴール条件 : (on 赤 緑) (on 緑 青)
- ゴールカウントヒューリスティック :
  状態 n が満たしていないゴール条件の数を $h(n)$ とする



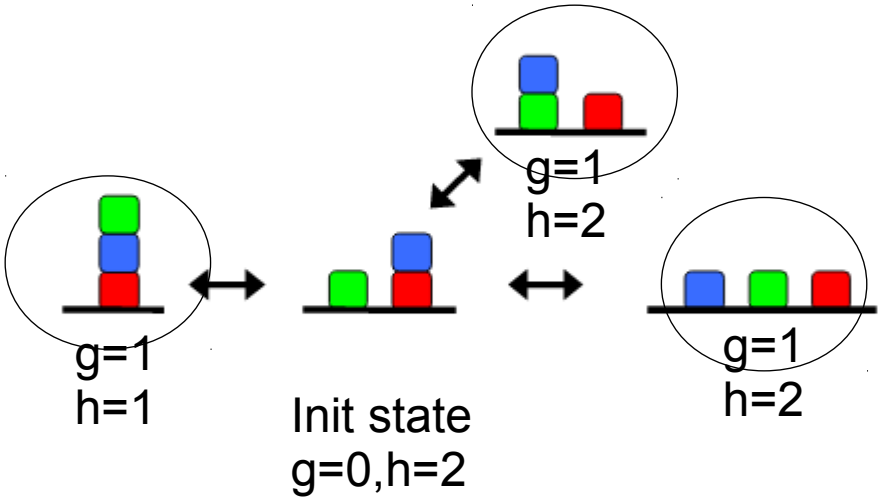
Init state
g=0,h=2

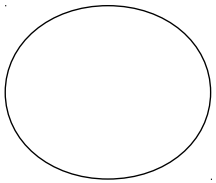


オープンリスト

A* 探索
(Hart et al.'68)

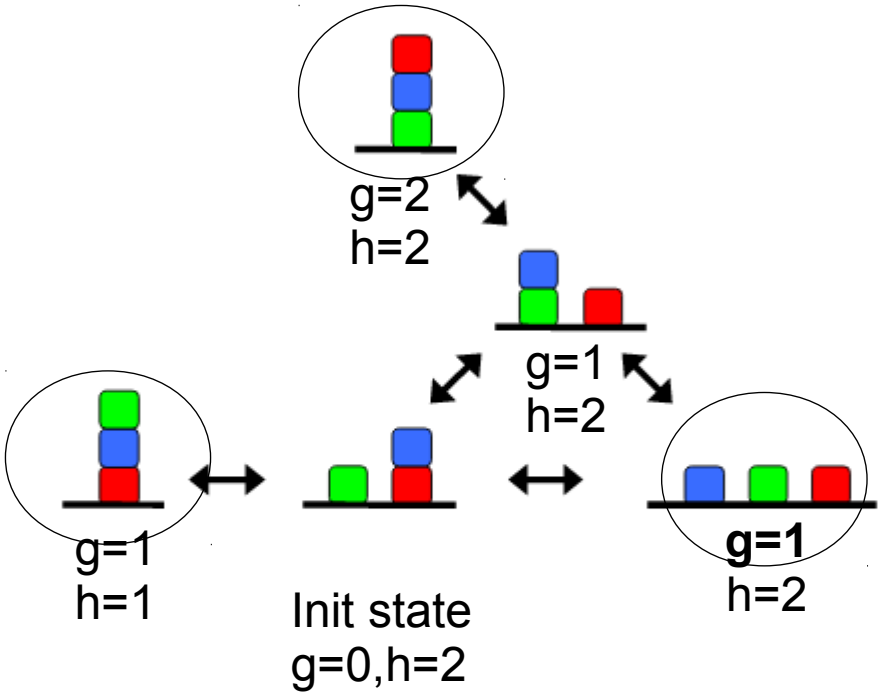
- ゴール条件 : (on 赤 緑) (on 緑 青)
- ゴールカウントヒューリスティック :
  状態 n が満たしていないゴール条件の数を $h(n)$ とする





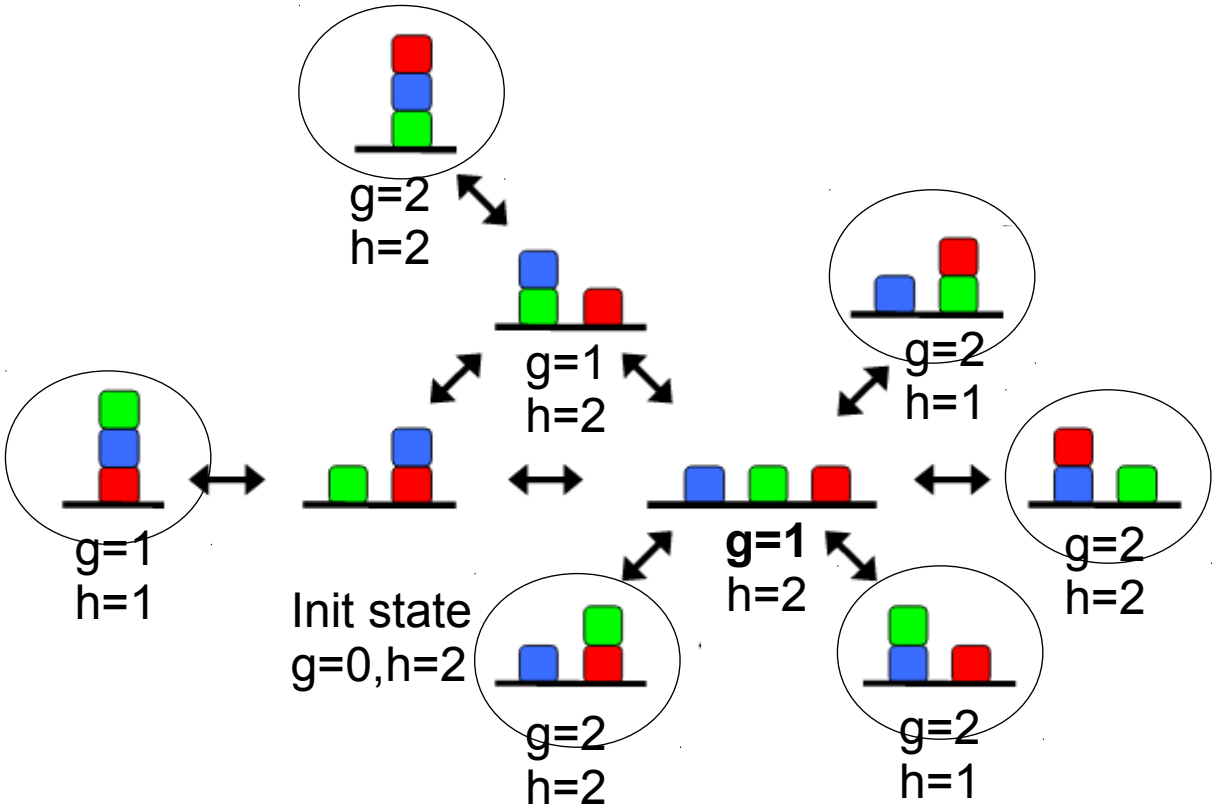
オープンリスト

A* 探索
(Hart et al.'68)



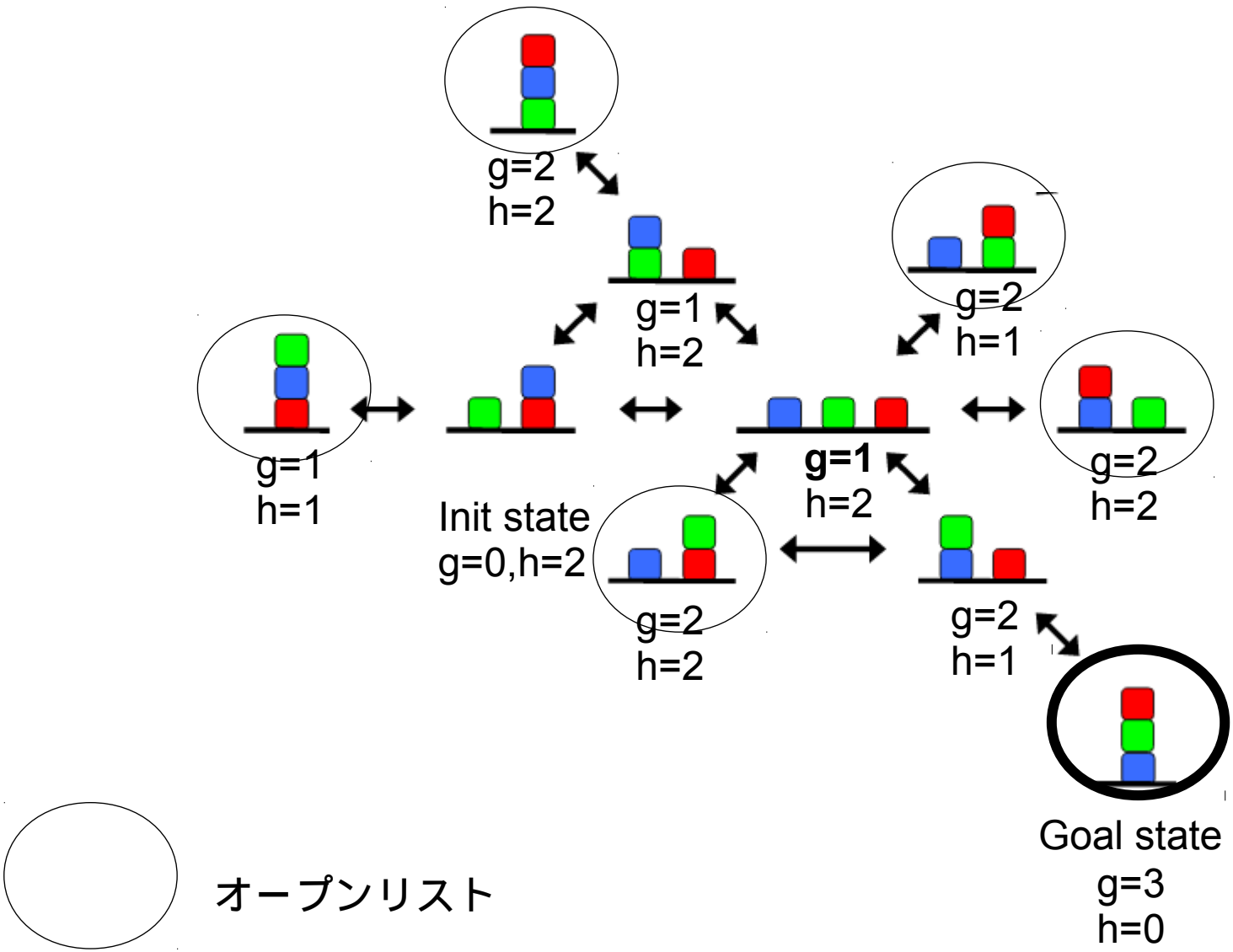
オープンリスト

A* 探索
(Hart et al.'68)



オープンリスト

A* 探索
(Hart et al.'68)



ヒューリスティック関数

- 正確であればあるほど効率的に (より少ないノードの展開回数で) ゴールを発見できる
- 許容的なヒューリスティック (admissible heuristic)
  - $h(n)$ がノード n からゴール状態までの最小コストの下界であるとき、許容的なヒューリスティックと呼ぶ
- 許容的なヒューリスティックを用いた A* 探索は最適解を返すことが知られている (Hart et al.'68)

→ 正確かつ許容的なヒューリスティック関数が理想的

ヒューリスティック関数

- ドメインが既知であれば人間のエンジニアがドメイン固有のヒューリスティック関数を実装出来る
- 古典的プランニングの場合はどんな問題か事前にはわからない
- 入力の STRIPS formulation から**自動的に**ヒューリスティック関数を生成しなければならない！

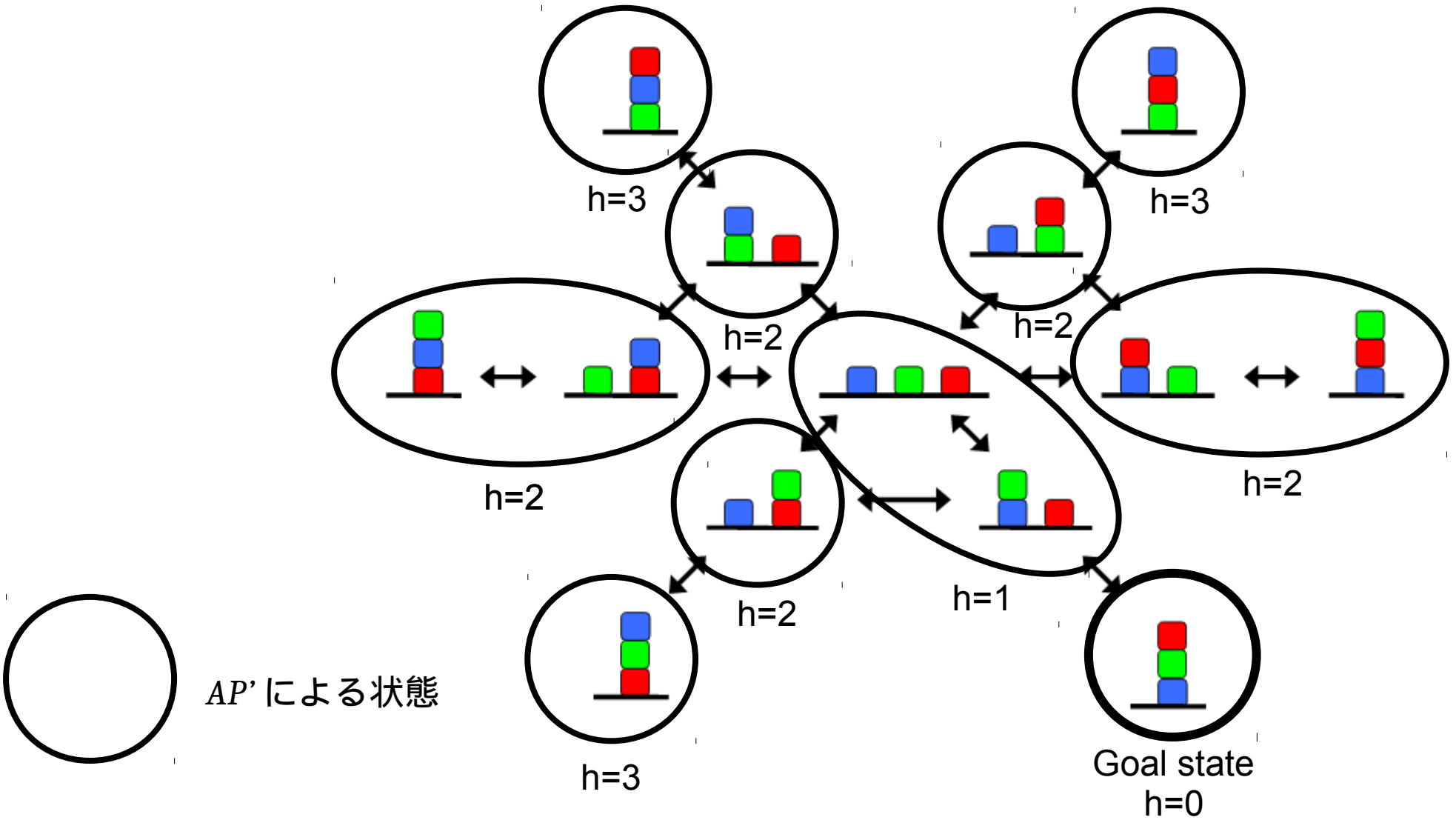
パターンデータベース
(Pattern Database) (Edelkamp'99)

- 状態 n から Abstract 状態 n' への mapping A: $n \rightarrow n'$ を生成する
- Abstract 状態空間は一般に元の状態空間よりも exponential に小さい
- 全ての $A(n)$ から $A(T)$ への最短コストをテーブルに保存する
  ($A(T)$ から逆向き幅優先探索で効率的に求められる)
- $h(n)$ は $A(n)$ から $A(T)$ への最短コストである

パターンデータベース
(Pattern Database) (Edelkamp'99)

$AP = \{(on\ 赤\ 青)(on\ 赤\ 緑)(on\ 青\ 赤)...\}$

$AP' = AP\ /\ \{(on\ 緑\ 赤)(on\ 緑\ 青)(on\ 緑\ table)\}$



ヒューリスティック関数の自動生成手法

- Merge&Shrink (Helmert et al.'14)
  - 複数の Pattern database を merge することでより正確な見積もりを求める
- Landmark-cut (Helmert&Domshlak'09)
  - すいませんよく分かりません
- etc....

おはなし

- 古典的プランニング問題
- A* 探索
- **BDD を用いたグラフ探索**
- 並列 A* 探索 (Hash Distributed A*)
- Black-box Planning (ビデオゲーム AI)

BDD を用いたプランニング

- 最近 BDD を用いたグラフ探索アルゴリズムが古典的プランニング業界で席巻している
  - 直近の International Planning Competition'04 では 1 位〜4 位までを BDD を用いた手法が独占！
- でもプランニングの人たちの多くは BDD のことはよく知りません　ブラックボックス的に用いています
- BDD の専門家と研究すればなんか出来るかも？

BDD を用いる利点

- A* 探索はオープンリストからノードを 1 つ取り出し、それぞれのノードに対していちいち Expand を実行する
- BDD を用いると複数のノードに対して一気に Expand が実行できる
- A* は全ての (圧縮されていない) ノードをクローズドリストに保存する
- BDD を用いるとオープン・クローズドリストを圧縮して表現できる
- パターンデータベースヒューリスティックは Abstract 問題の全ての状態の解コストを列挙しなければならないのでそのために有用

BDD を用いたグラフ探索

A* で用いられるデータ構造を特徴関数によって表す

- 特徴関数 $\varphi_S(x)$： 状態 x が集合 S に含まれる場合に真を返し、そうでない場合に偽を返す
  - e.g. $\varphi_T(x)$: x がゴール状態であるかをテストする関数
- $Open(x)$： 状態 x がオープンリストに含まれる場合に真
- $Closed(x)$： 状態 x がクローズドリストに含まれる場合に真
- $Trans_a(x, x') : x' = a(x)$ である場合に真
  （アクション a による状態遷移に対応する）
- $Trans(x, x')$: $\exists a \in A\ (\ x' = a(x)\ )$ である場合に真

BDD- 幅優先グラフ探索

(Edelkamp&Reffel'98)

$Closed(x) \leftarrow Open_0(x) \leftarrow \varphi_{\{s0\}}(x)$

for (i=1,2,...)

  $if\ (\ Open(x) = false)$

    $return\ \text{“}No\ path\ found\text{”}$

  $Succ(x') \leftarrow (\ \exists x\ (\ Open_{i-1}(x) \wedge Trans(x, x')\ )\ )$

  $Open_i(x) \leftarrow Succ(x) \wedge \neg Closed(x)$

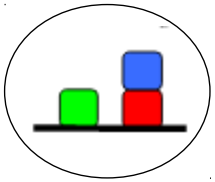
  $Closed(x) \leftarrow Closed(x) \vee Succ(x)$

  if $(\ Open_i(x) \wedge \varphi_T(x) \neq false)$

    return Construct$(\ Open_i(x) \wedge \varphi_T(x)\ ,\ i\ )$

BDD- 幅優先グラフ探索

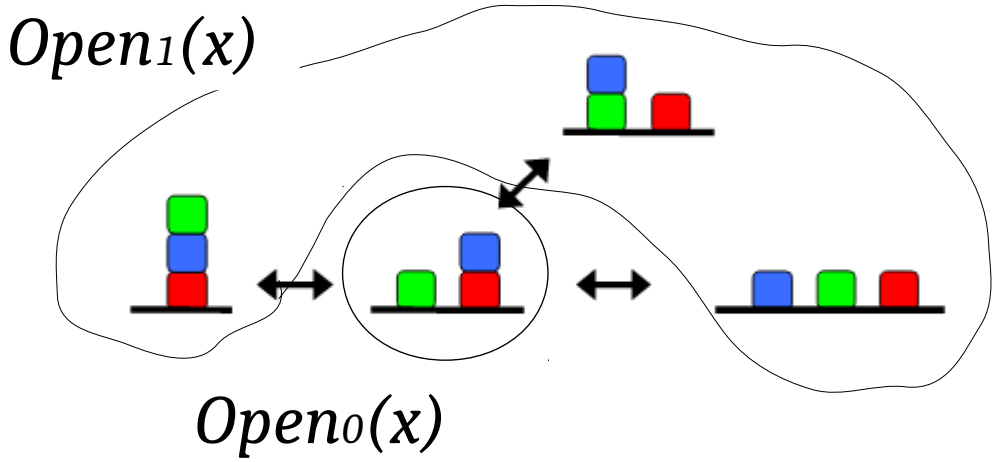
(Edelkamp&Reffel'98)



$Open_0(x)$

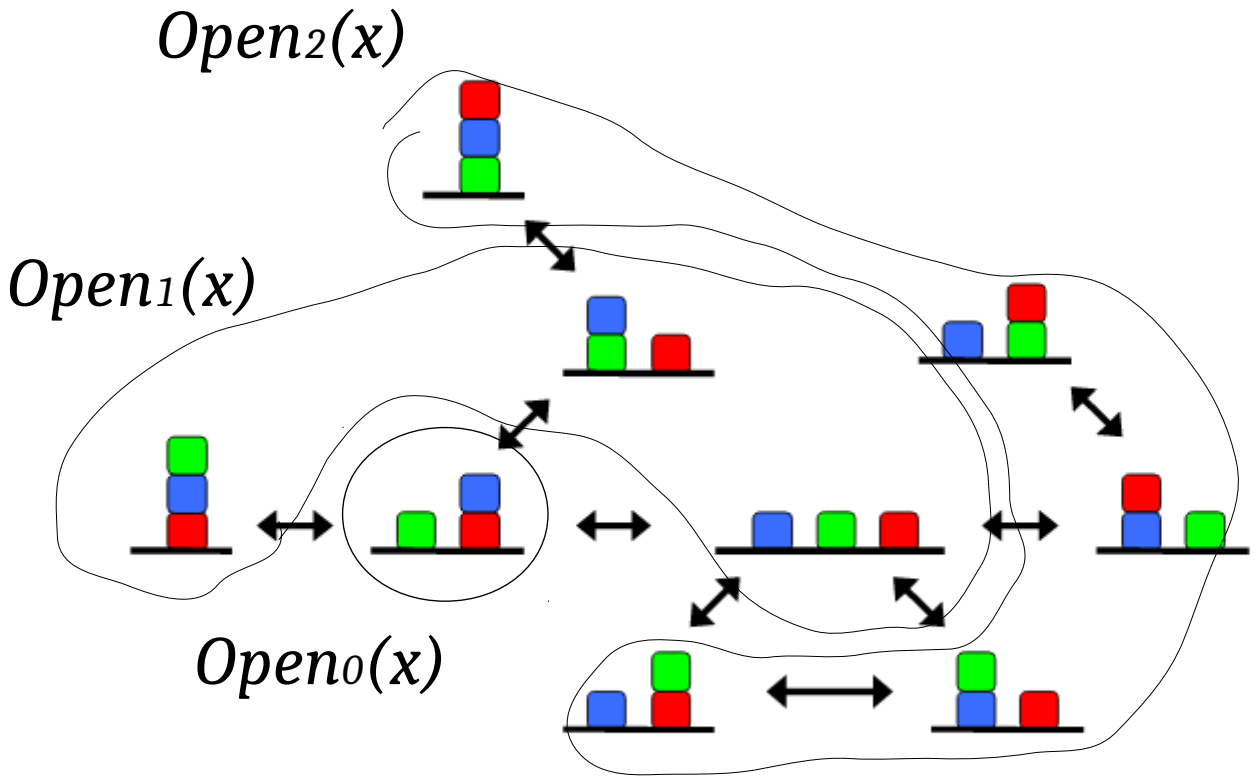
BDD- 幅優先グラフ探索

(Edelkamp&Reffel'98)



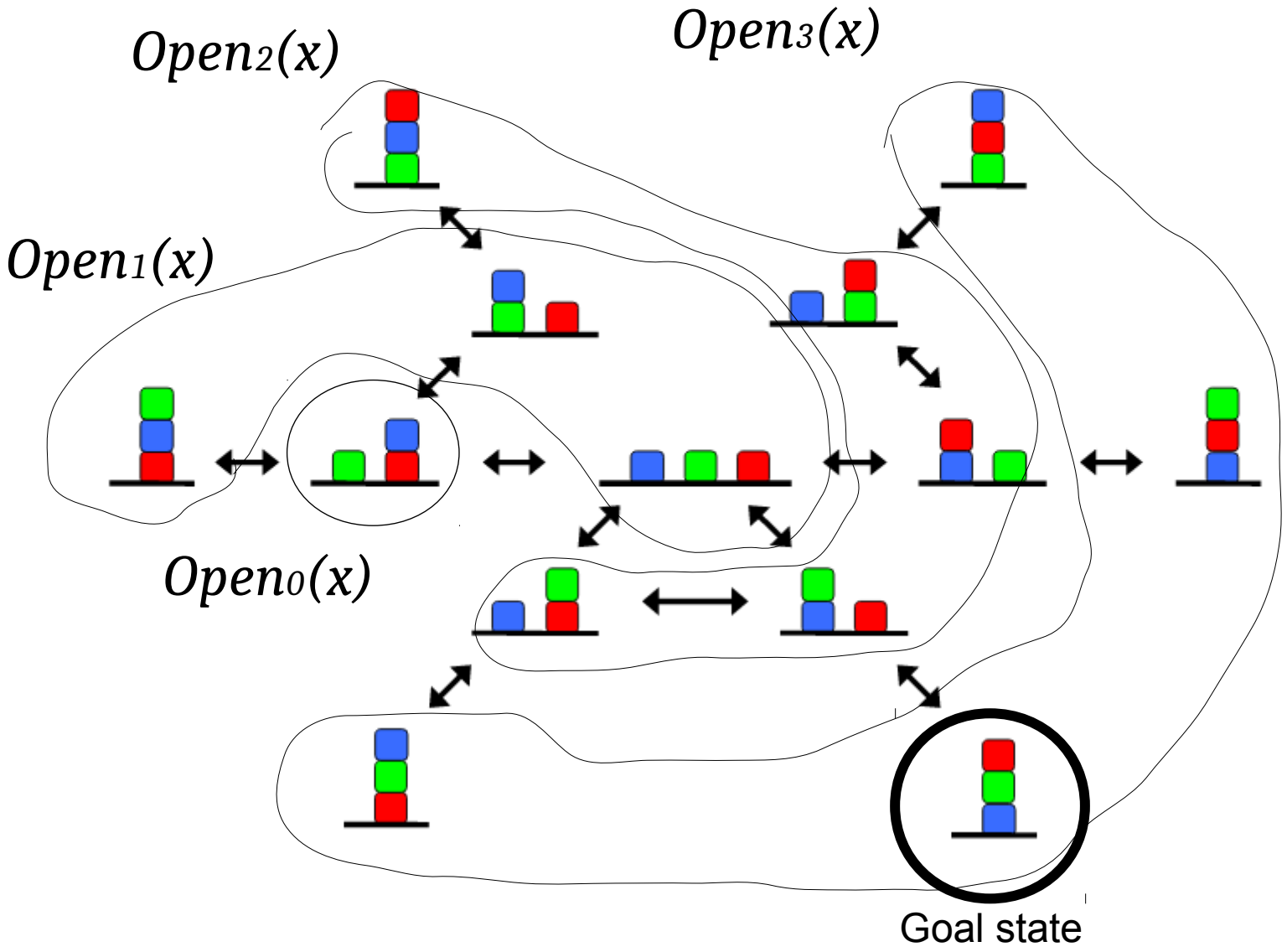
BDD- 幅優先グラフ探索

(Edelkamp&Reffel'98)



BDD- 幅優先グラフ探索

(Edelkamp&Reffel'98)



BDD-A*

(Edelkamp&Reffel'98)

- BDD を用いた A* 探索もあるが幅優先探索の方が広く使われている

$Open(f, x) \leftarrow Heur(f, x) \wedge \varphi_{\{s0\}}(x)$

for (i=1,2,...)

$fmin \leftarrow min\{f \mid \exists f'. f = f' \wedge Open(f', x) \neq false\}$

$Min(x) \leftarrow \exists f(Open(f, x) \wedge f = fmin)$

if $(Min(x) \wedge \varphi_T(x) \neq false)$

$return\ Construct(Min(x) \wedge \varphi_T(x))$

$Rest(f,x) \leftarrow Open(x) \wedge \neg Min(x)$

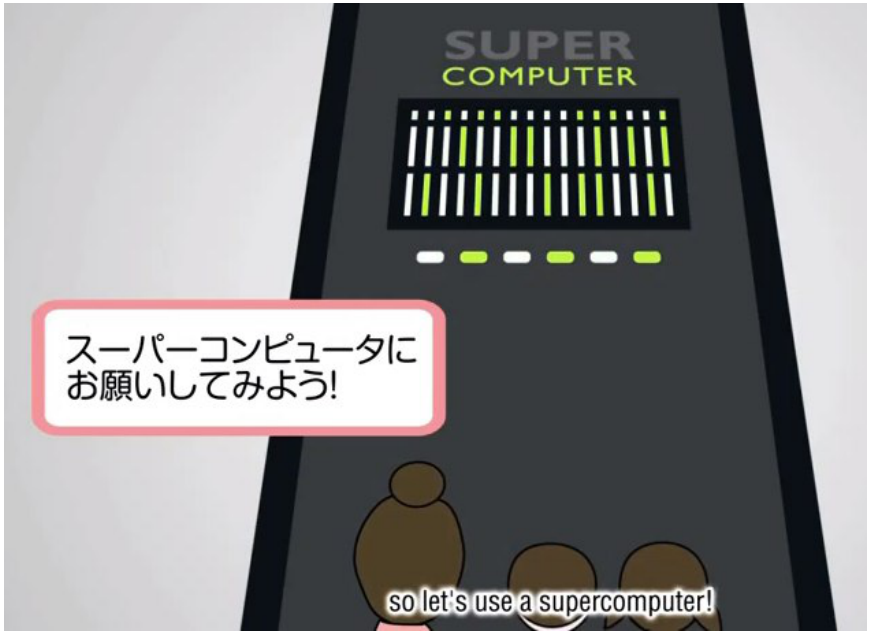
$Succ(f, x) \leftarrow \exists w,x,h,h',f'\ Min(x) \wedge Trans(w, x, x') \wedge Heur(h, x) \wedge Heur(x') \wedge Fvalue(h, h', w, f', f) \wedge f' = fmin\ [x'/x]$

$Open(f, x) \leftarrow Rest(f, x) \vee Succ(f, x)$

$whereas\ Fvalue(h, h', w, f', f) := 1 \quad iff\ f' = f + w - h + h'$

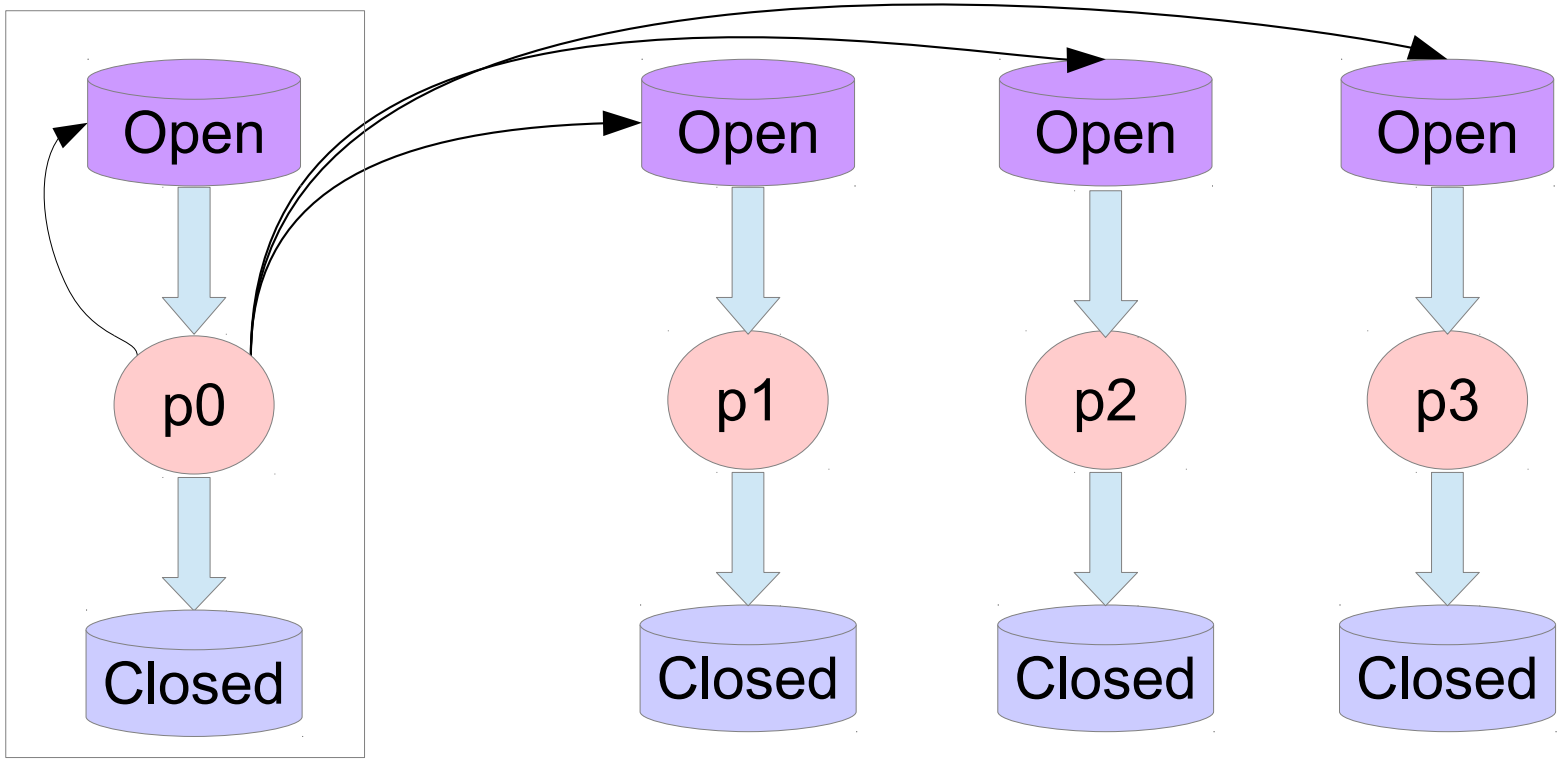
おはなし

- 古典的プランニング問題
- A* 探索
- BDD を用いたグラフ探索
- **並列 A* 探索　(Hash Distributed A*)**
- Black-box Planning (ビデオゲーム AI)



Hash Distributed A* (HDA*)

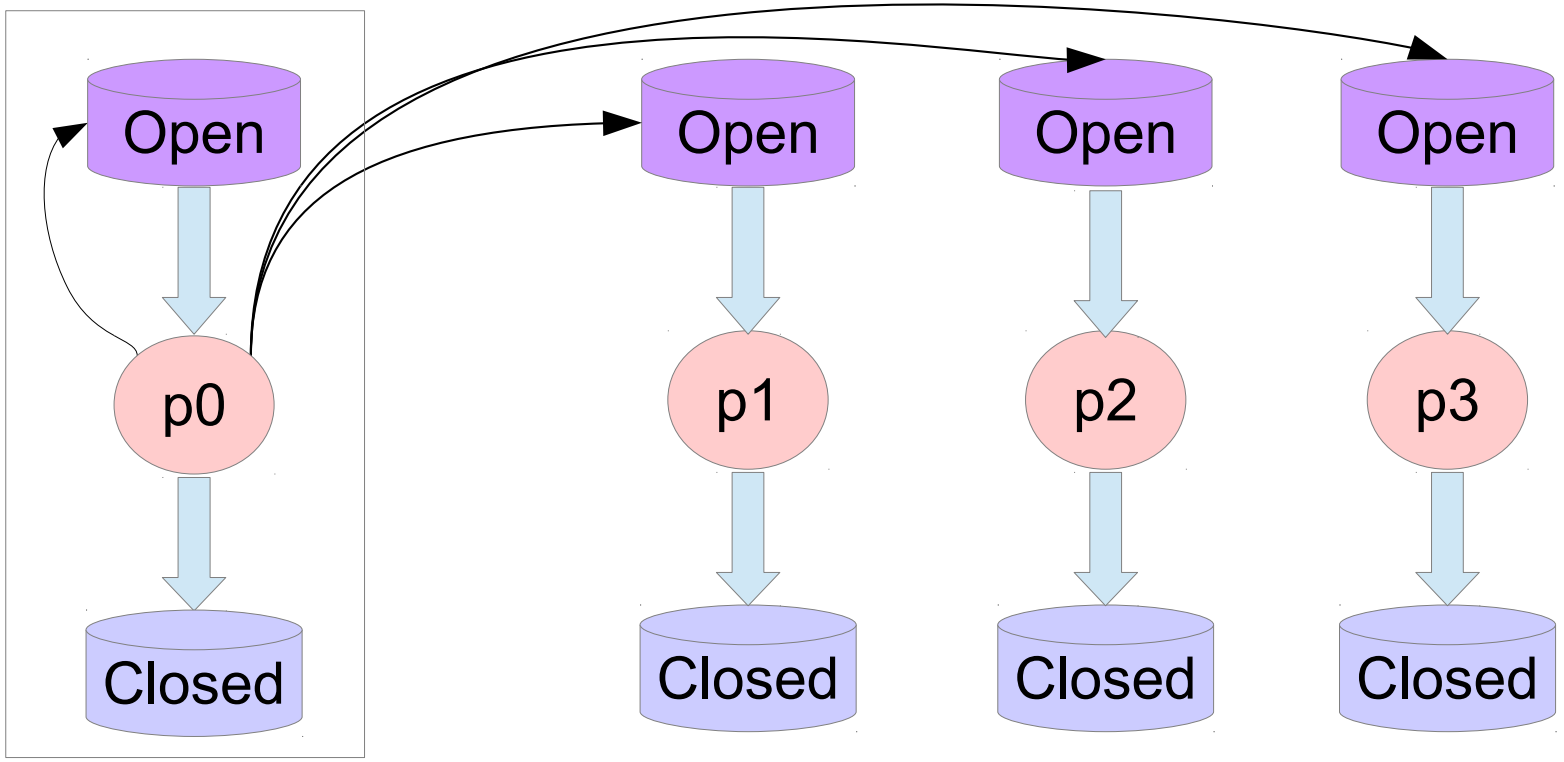
Kishimoto, Fukunaga, & Botea (2009)



- 各プロセスはそれぞれローカルにオープンリスト、クローズドリストを持ち、おおよそ A* 同様の処理を行う
- 生成されたノードはハッシュ値によって一意に定まる担当のプロセスに送信される
- 送受信は非同期に行われる

Hash Distributed A* (HDA*)

Kishimoto, Fukunaga, & Botea (2009)



- 生成されたノードはハッシュ値によって一意に定まる担当のプロセスに送信される

→ ハッシュ関数の選択によって HDA* の性能は大きく変わる！

仕事分配手法研究の概要

Zobrist hashing (ZHDA*)

(Zobrist 1970; Kishimoto et al. 2013)

+ ロードバランスに優れる
- 通信オーバーヘッドが大きい

State abstraction (AHDA*)

(Burns et al. 2010)

- ロードバランスが悪い
+ 通信オーバーヘッドが小さい

仕事分配手法研究の概要

Zobrist hashing (ZHDA*)
(Zobrist 1970; Kishimoto et al. 2013)

+ ロードバランスに優れる
- 通信オーバーヘッドが大きい

State abstraction (AHDA*)
(Burns et al. 2010)

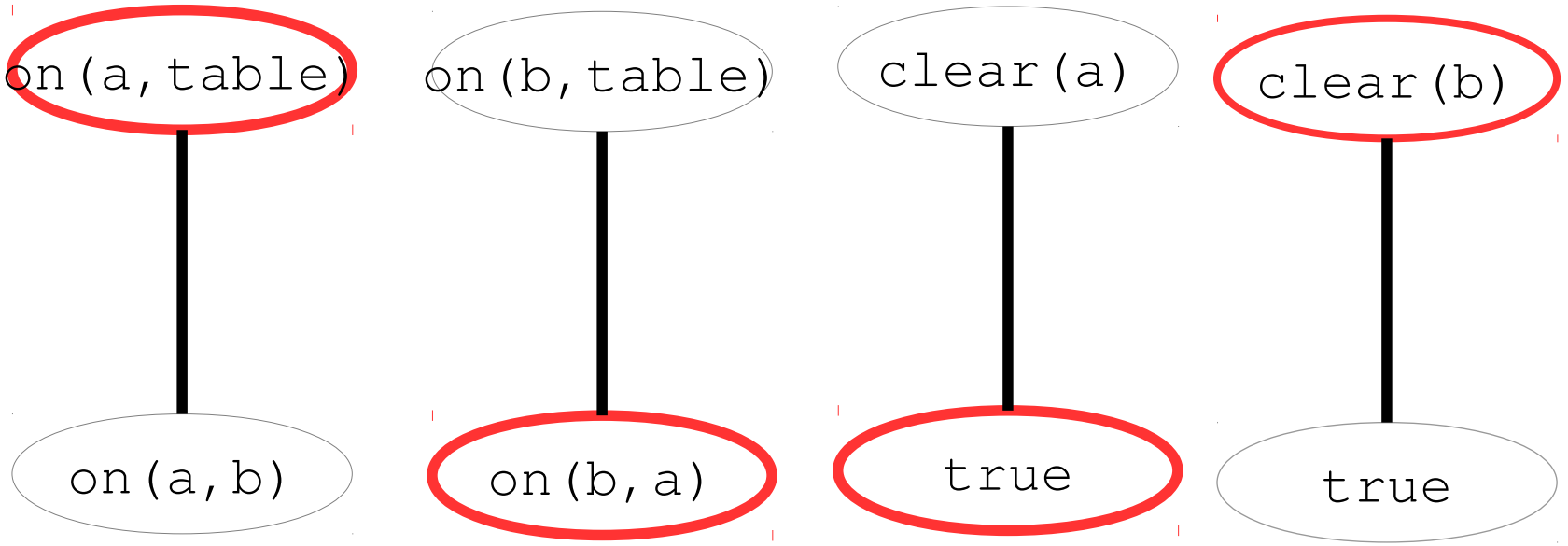
- ロードバランスが悪い
+ 通信オーバーヘッドが小さい

Abstract Zobrist Hashing (AZHDA*)
(Jinnai&Fukunaga 2016)

+ ロードバランスに優れる
+ 通信オーバーヘッドが小さい
* パラメータとして **feature abstraction** を設定する必要がある

状態表現

- 状態を表す proposition の集合から multi-valued variable (SAS+ variable) の列への変換が可能である (Backstrom et al. 1995)



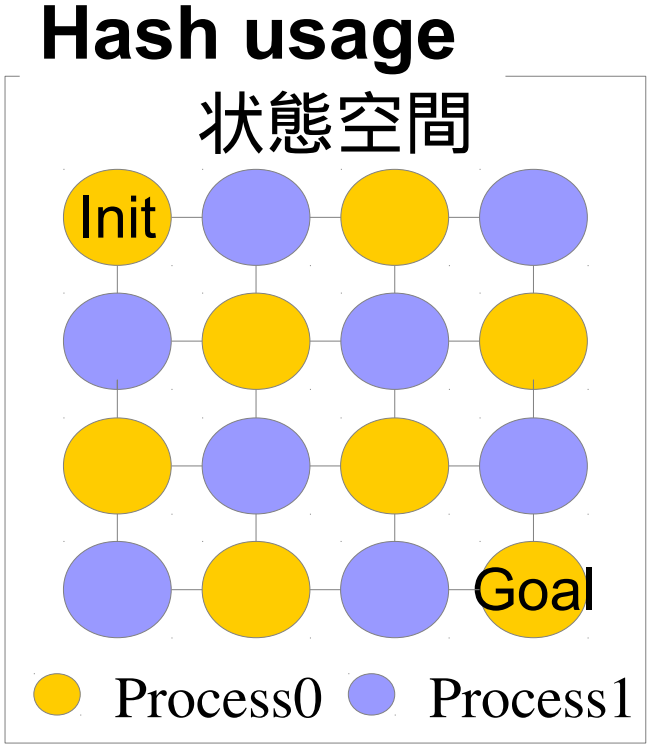
例 : blocks world

HDA* の為のハッシュ関数

- 状態 s は feature (multivalued variable) x_i の列で表せられる：

  state s = (x_1, x_2,...,x_n)

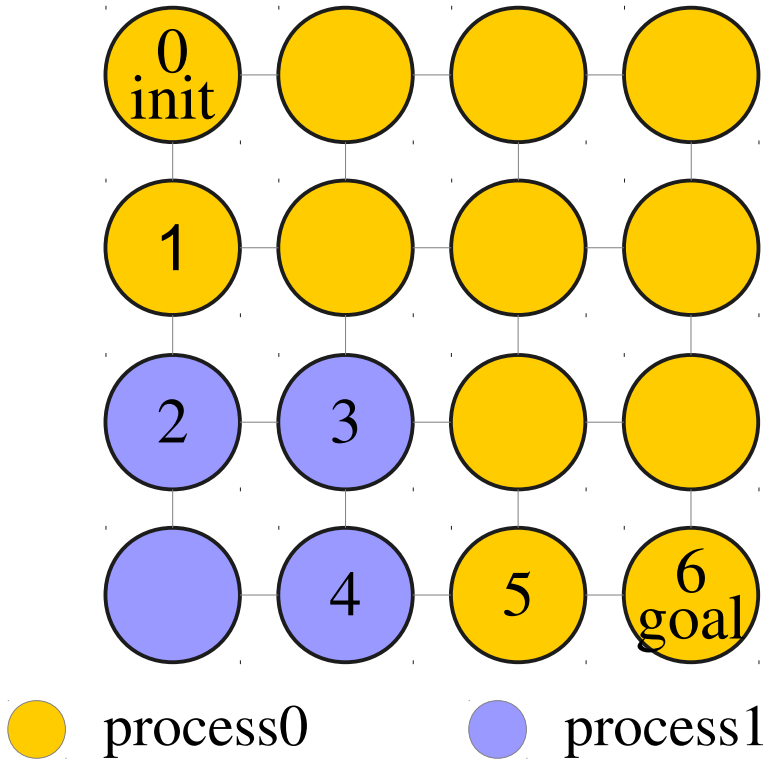
- 状態 s に対してハッシュ関数 $H(s)$ は s を担当するプロセスのプロセス ID を返す



ハッシュ関数に求められる要件

- $H(s)$ はなるべく各値を
  均等な割合で返す
  → ロードバランス

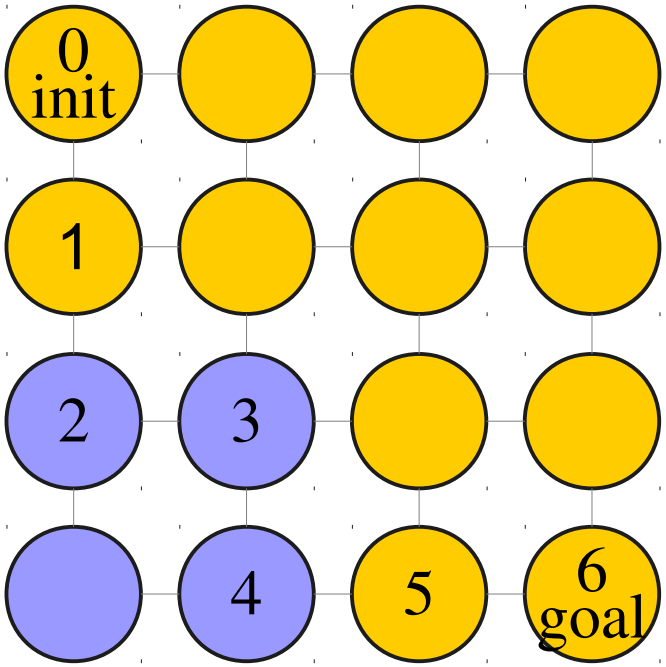
悪い例



ハッシュ関数に求められる要件

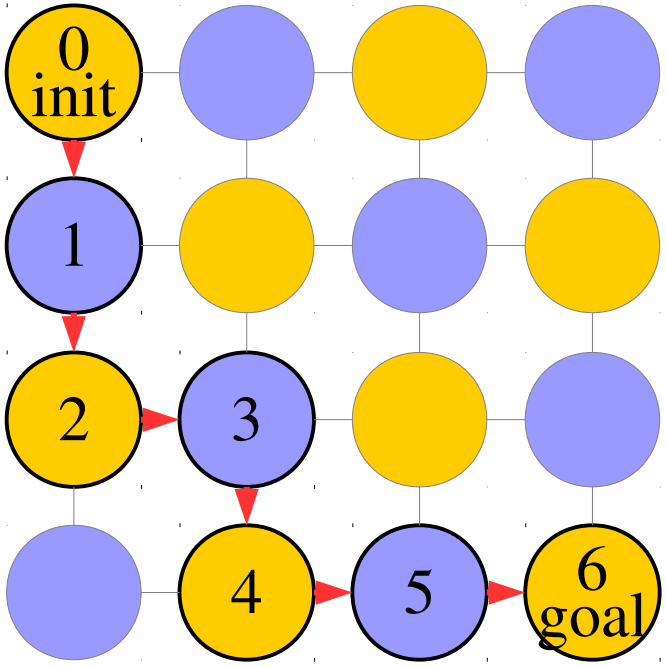
- $H(s)$ はなるべく各値を均等な割合で返す
  → ロードバランス
- $H(s)$ はなるべく同じ値が連続して欲しい
  → 通信オーバーヘッド

悪い例



process0　process1

悪い例



process0　process1

Zobrist Hashing (ZHDA*)

Zobrist (1970); Kishimoto et al. (2009)

- 目的：ロードバランスを最適化
- 初期化時に状態の各 feature (x_i) に対してランダムテーブル Ri を生成する
- ハッシュ値は以下のように計算される：

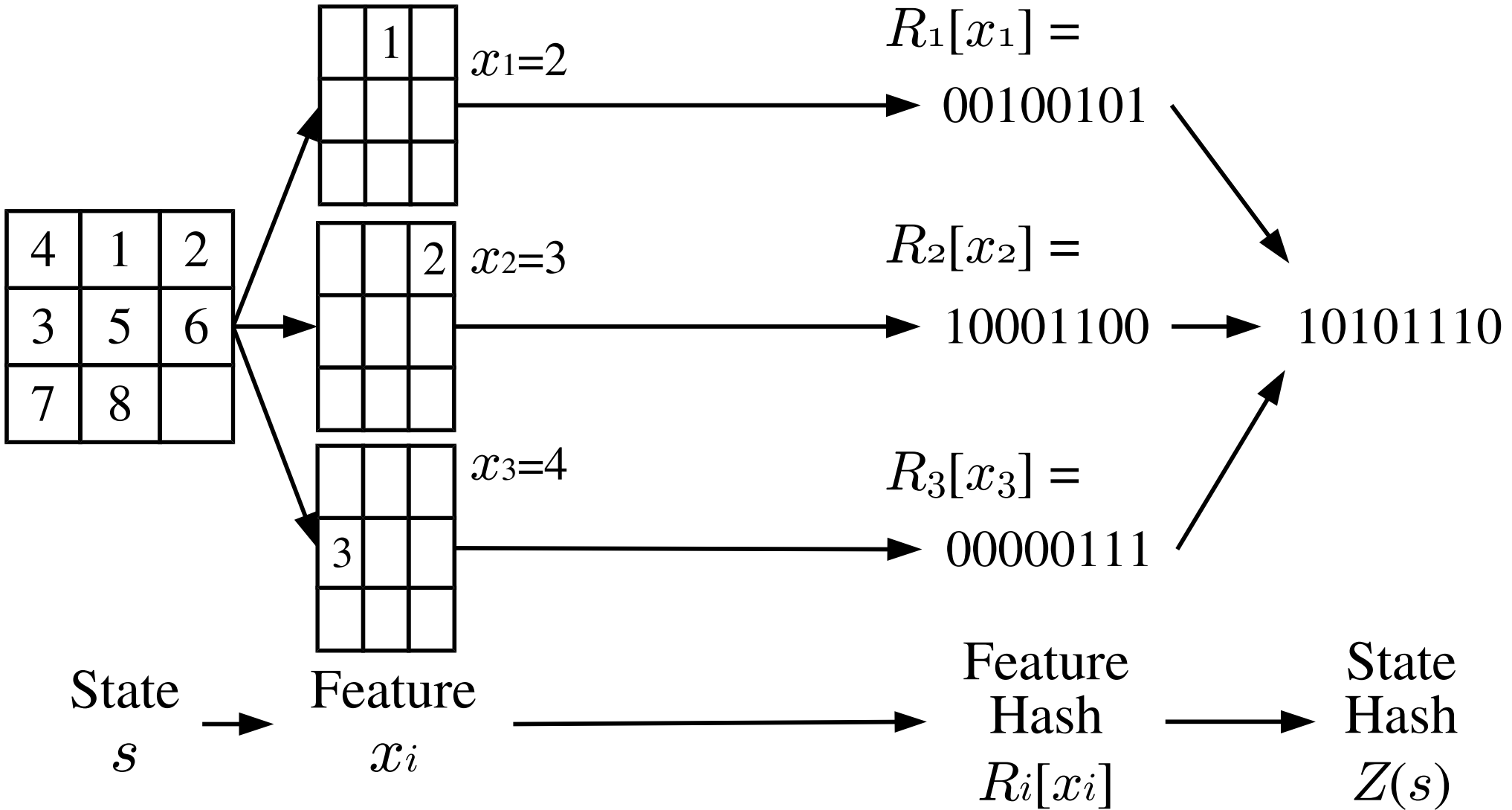
$$Z(s) = R_1[x_1] \text{ xor } R_2[x_2] \text{ xor } \ldots \text{ xor } R_n[x_n]$$

Zobrist Hashing (ZHDA*)

Zobrist (1970); Kishimoto et al. (2009)

$$Z(s) = R_1[x_1] \text{ xor } R_2[x_2] \text{ xor } \ldots \text{ xor } R_n[x_n]$$

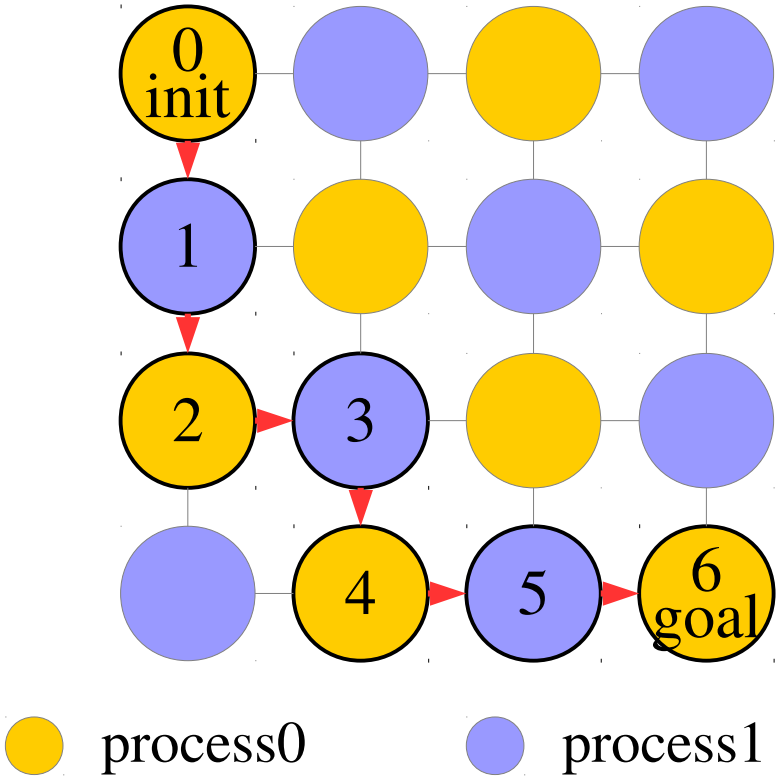
feature x_i が i 番タイルの位置を表すとすると、

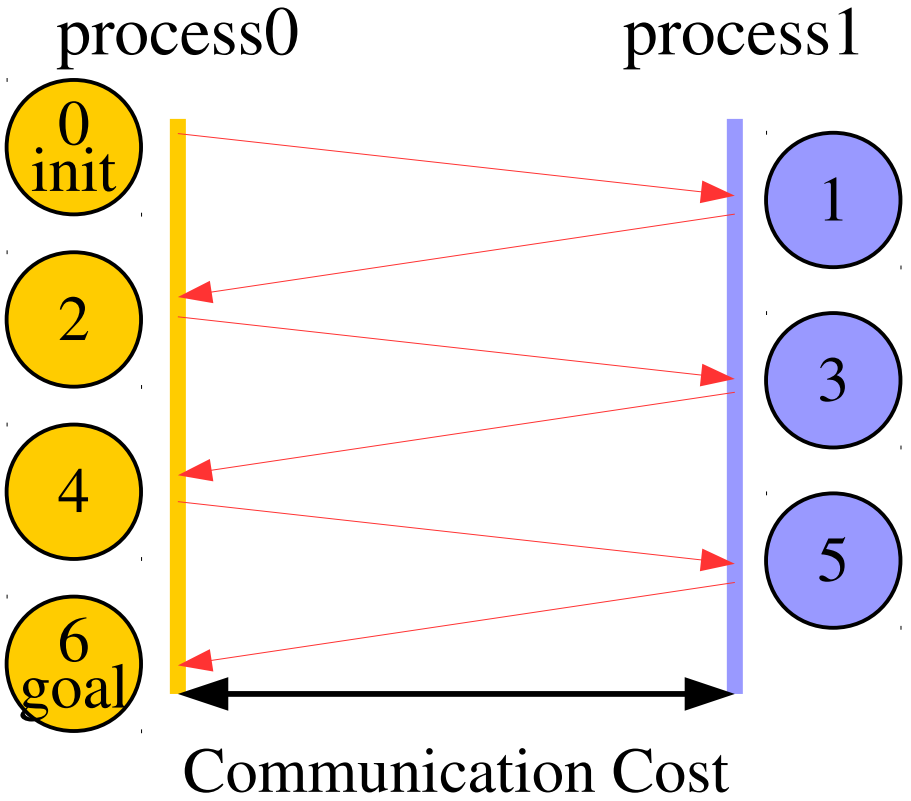


Zobrist Hashing (ZHDA*)

Zobrist (1970); Kishimoto et al. (2009)

- 利点： ロードバランスに優れる
- 問題点： 通信オーバーヘッドが大きい





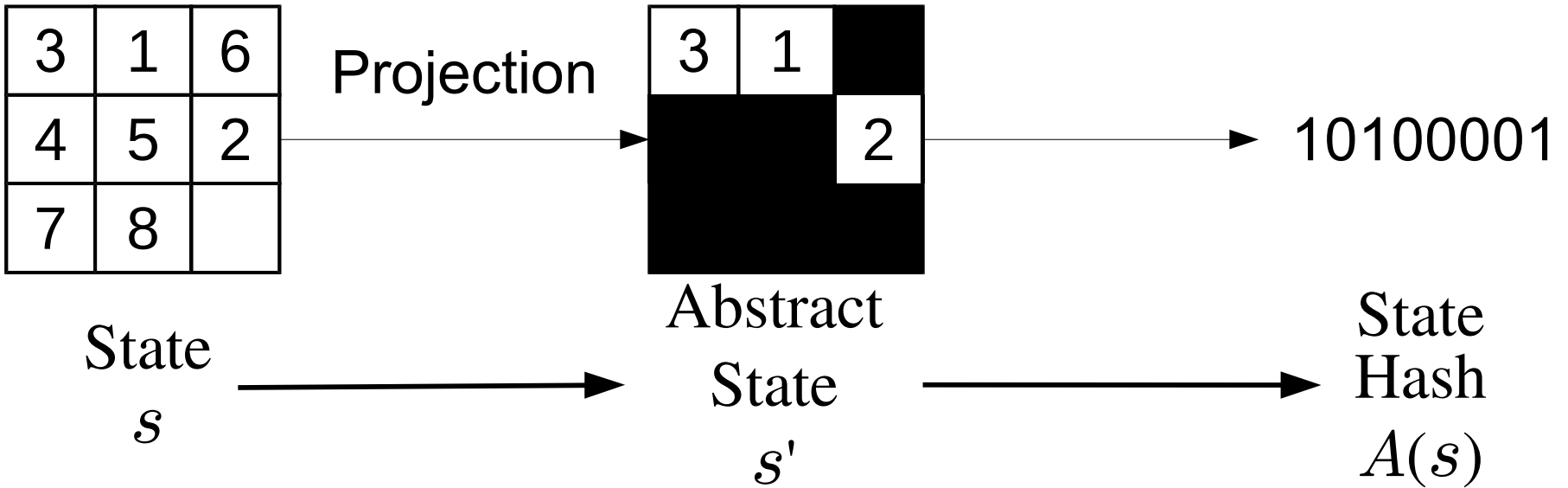
State abstraction (AHDA*)

Burns et al. (2010)

- 目的 : 隣り合う state がなるべく同じプロセスに割り当てられる
- State s を abstract state s' に mapping し、各 abstract state に対してハッシュ値を与える

$$A(s) = R[s']$$

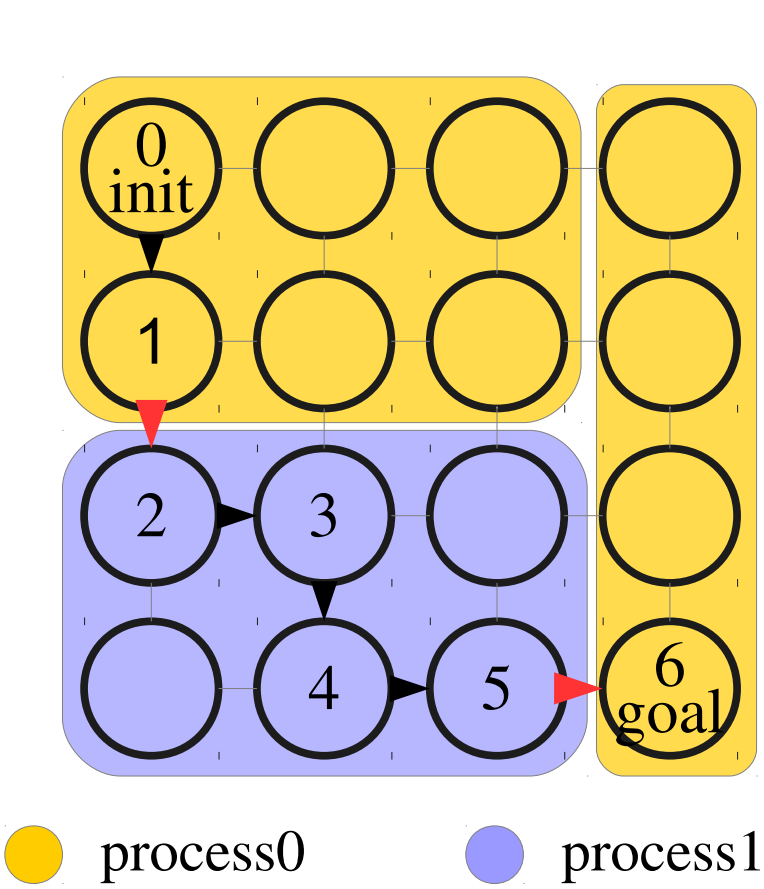
1,2,3 番タイルの位置だけに依存する abstract state を考えると、

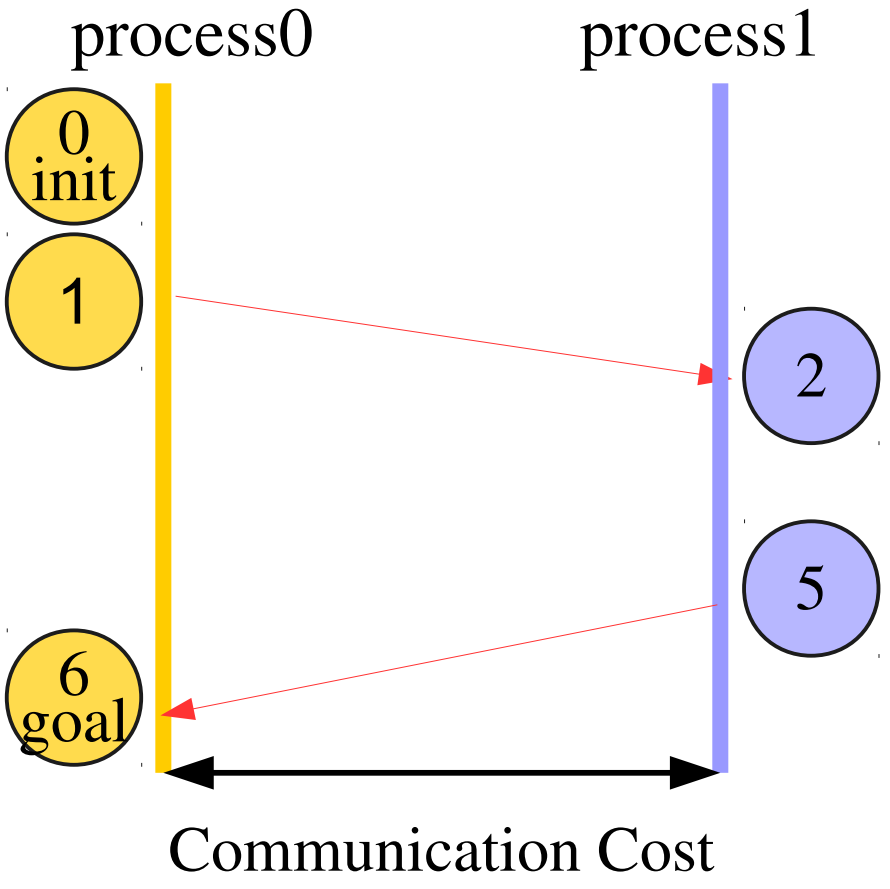


State abstraction (AHDA*)

Burns et al. (2010)

- 利点：通信オーバーヘッドが小さい
- 問題点：ロードバランスが悪い





研究の概要

Zobrist hashing (ZHDA*)

(Zobrist 1970; Kishimoto et al. 2013)

+ ロードバランスに優れる
- 通信オーバーヘッドが大きい

State abstraction (AHDA*)

(Burns et al. 2010)

- ロードバランスが悪い
+ 通信オーバーヘッドが小さい

仕事分配手法研究の概要

Zobrist hashing (ZHDA*)
(Zobrist 1970; Kishimoto et al. 2013)

+ ロードバランスに優れる
- 通信オーバーヘッドが大きい

State abstraction (AHDA*)
(Burns et al. 2010)

- ロードバランスが悪い
+ 通信オーバーヘッドが小さい

Abstract Zobrist Hashing (AZHDA*)
(Jinnai&Fukunaga 2016)

+ ロードバランスに優れる
+ 通信オーバーヘッドが小さい
* パラメータとして **feature abstraction** を設定する必要がある

Abstract Zobrist Hashing (AZH)

- Zobrist hashing と Abstraction を組み合わせた手法
- 各 feature に対して **feature abstraction** を施し abstract feature を生成し、 abstract feature を用いて Zobrist hashing を計算する

$$AZ(s) = R_1[A_1(x_1)] \text{ xor } R_2[A_2(x_2)] \text{ xor } \ldots \text{ xor } R_n[A_n(x_n)]$$

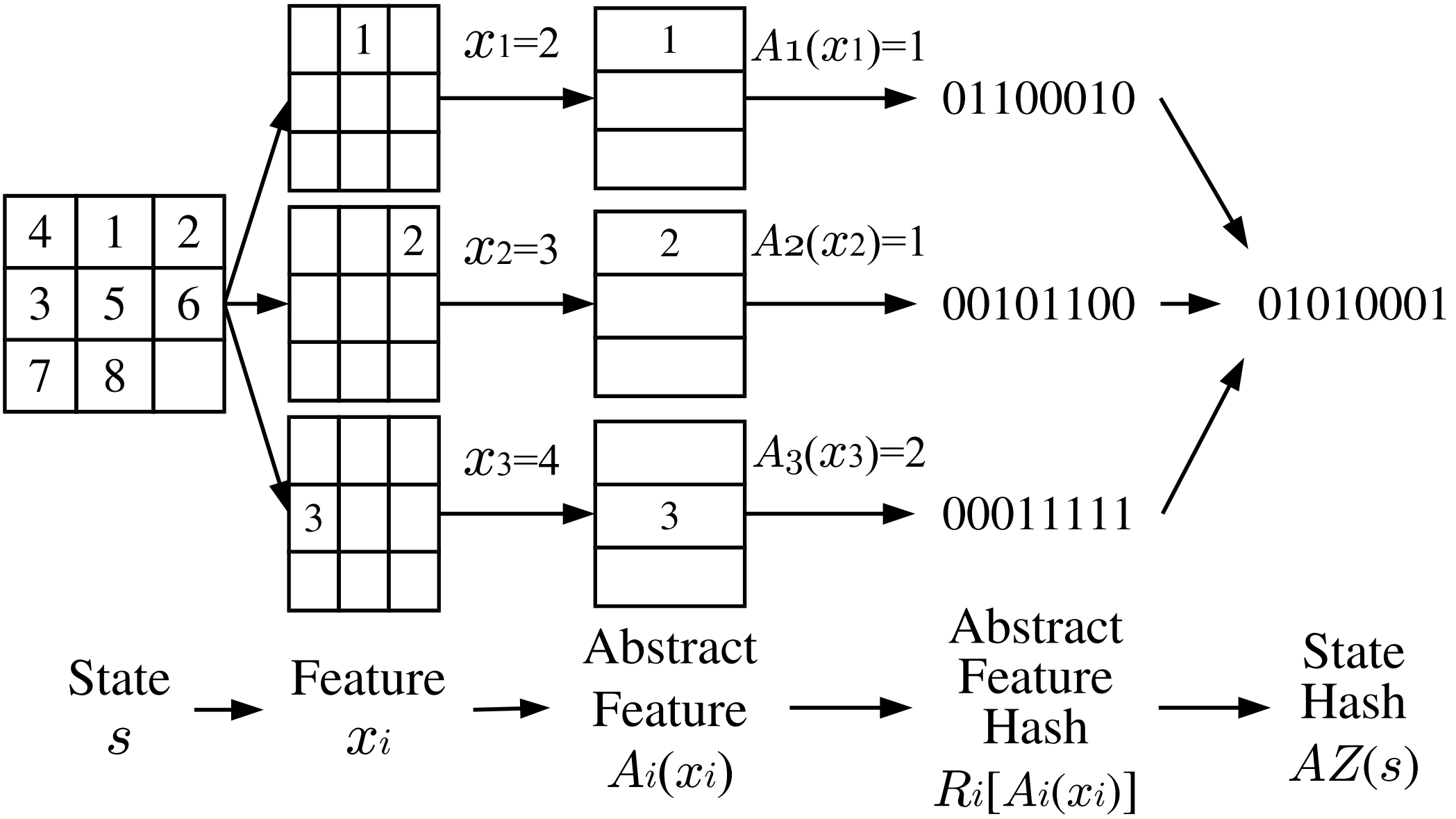
もしくは

$$AZ(s) = Z(s'), \text{ where } s' = (A_1(x_1), A_2(x_2), \ldots, A_n(x_n))$$

Abstract Zobrist Hashing (AZH)

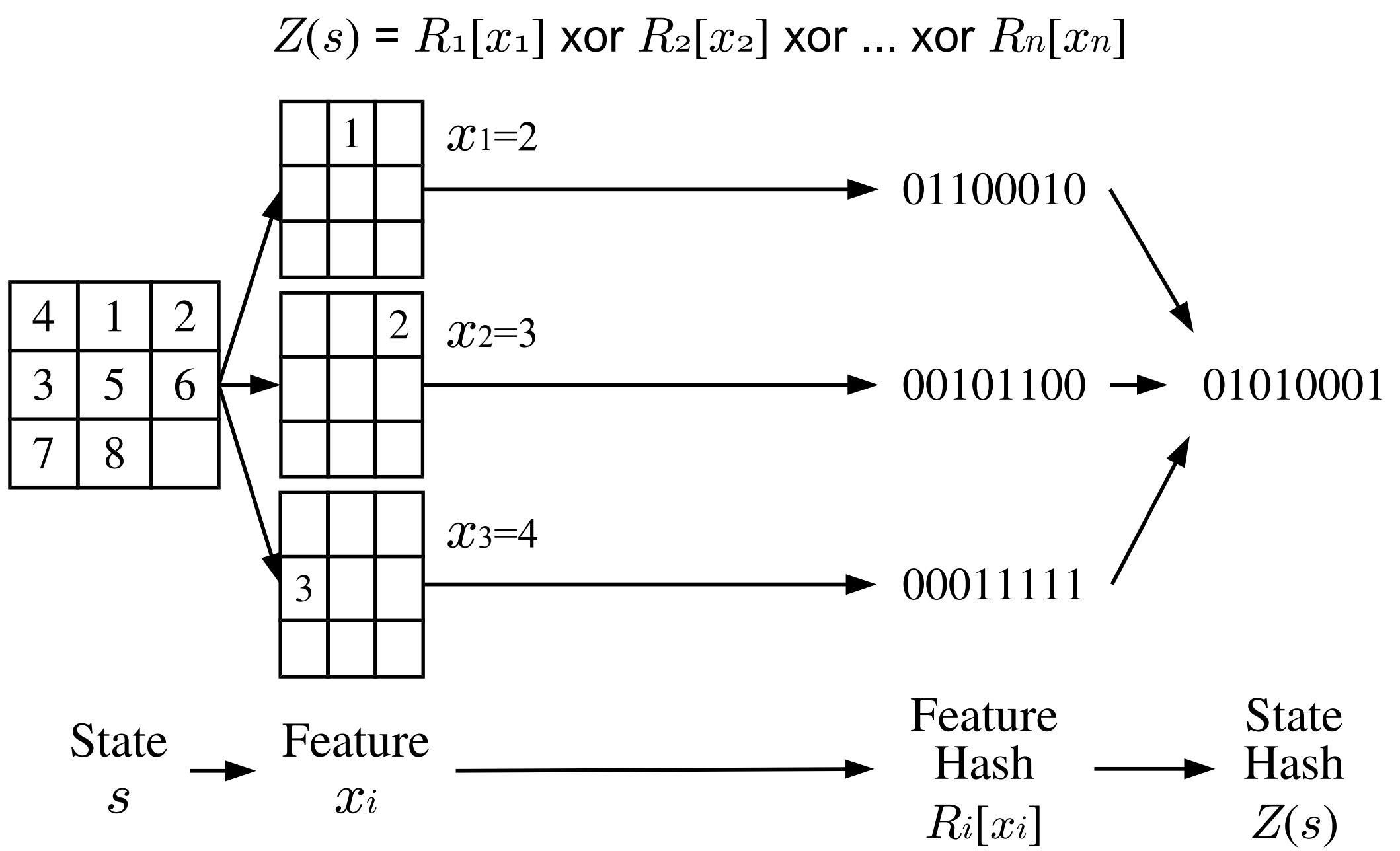
Jinnai&Fukunaga (2016)

$$AZ(s) = R_1[A_1(x_1)] \text{ xor } R_2[A_2(x_2)] \text{ xor } \ldots \text{ xor } R_n[A_n(x_n)]$$



Zobrist Hashing

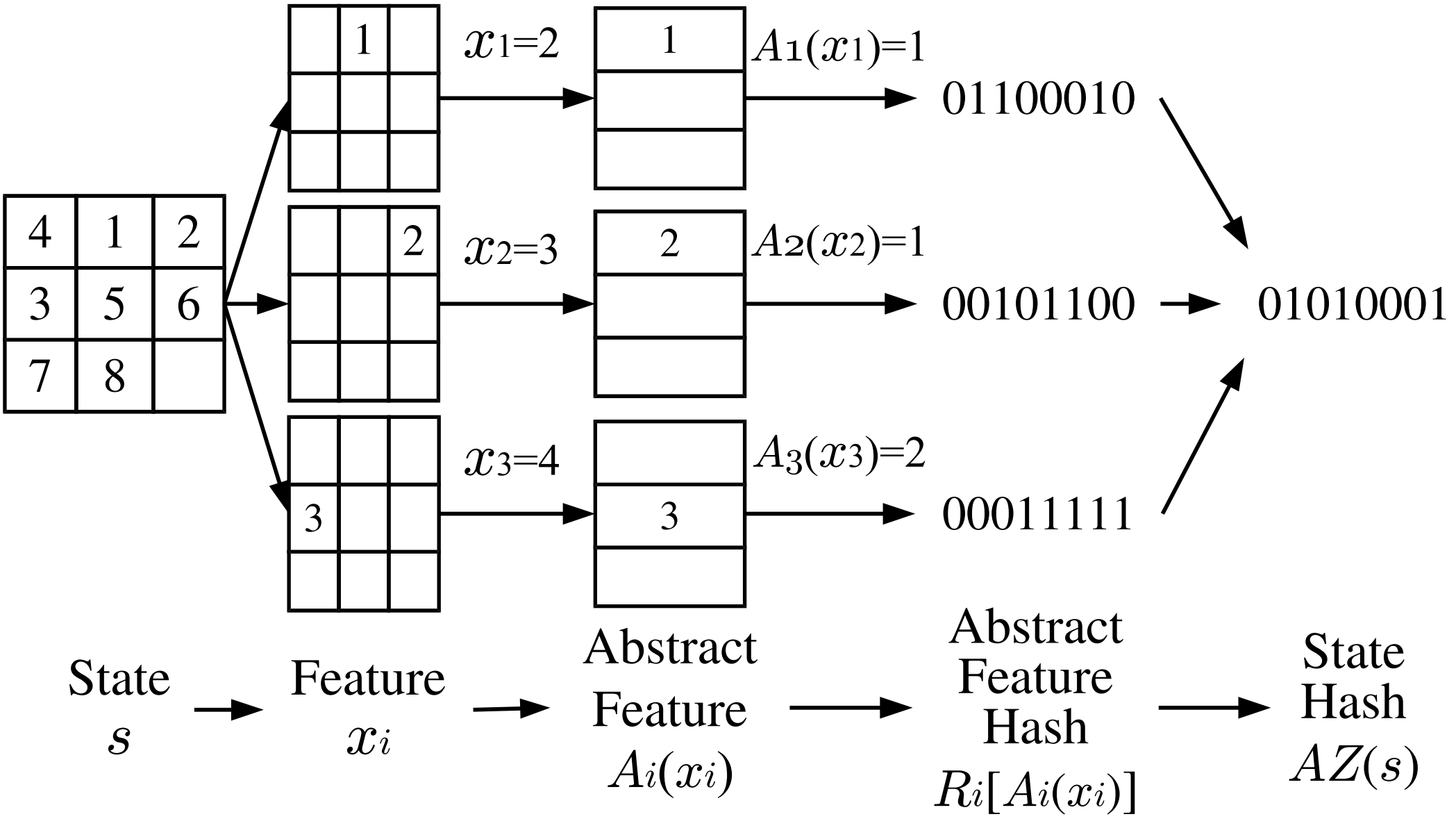
Zobrist (1970); Kishimoto et al. (2009)



Abstract Zobrist Hashing (AZH)

Jinnai&Fukunaga (2016)

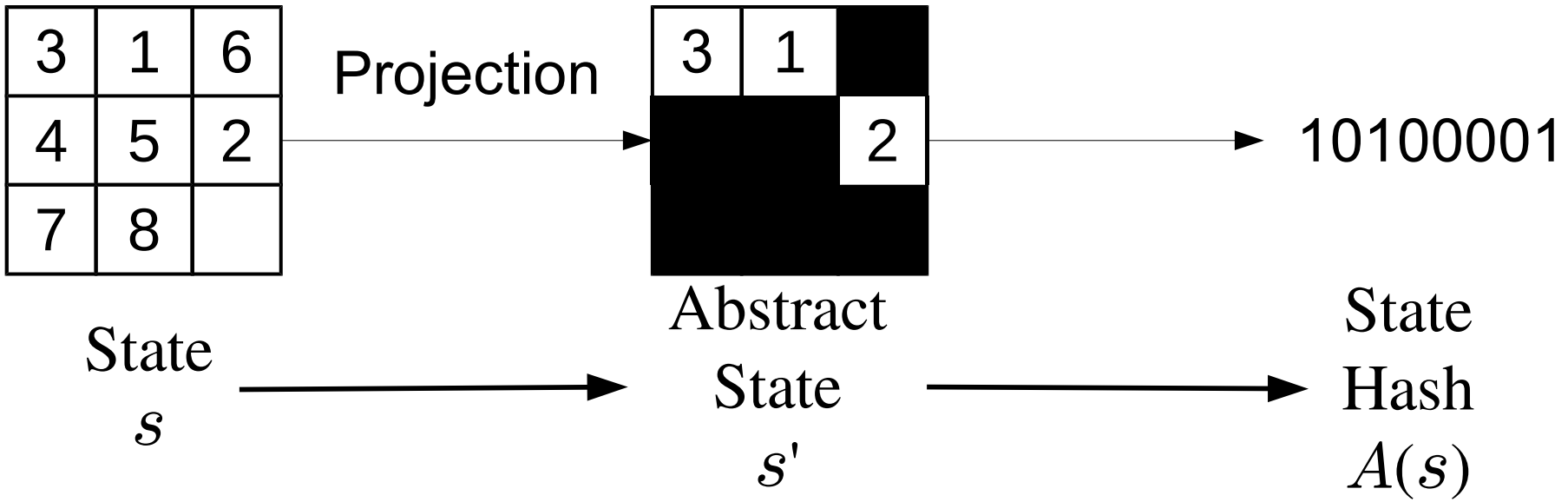
$$AZ(s) = R_1[A_1(x_1)] \text{ xor } R_2[A_2(x_2)] \text{ xor } \ldots \text{ xor } R_n[A_n(x_n)]$$



State abstraction

Burns et al. (2010)

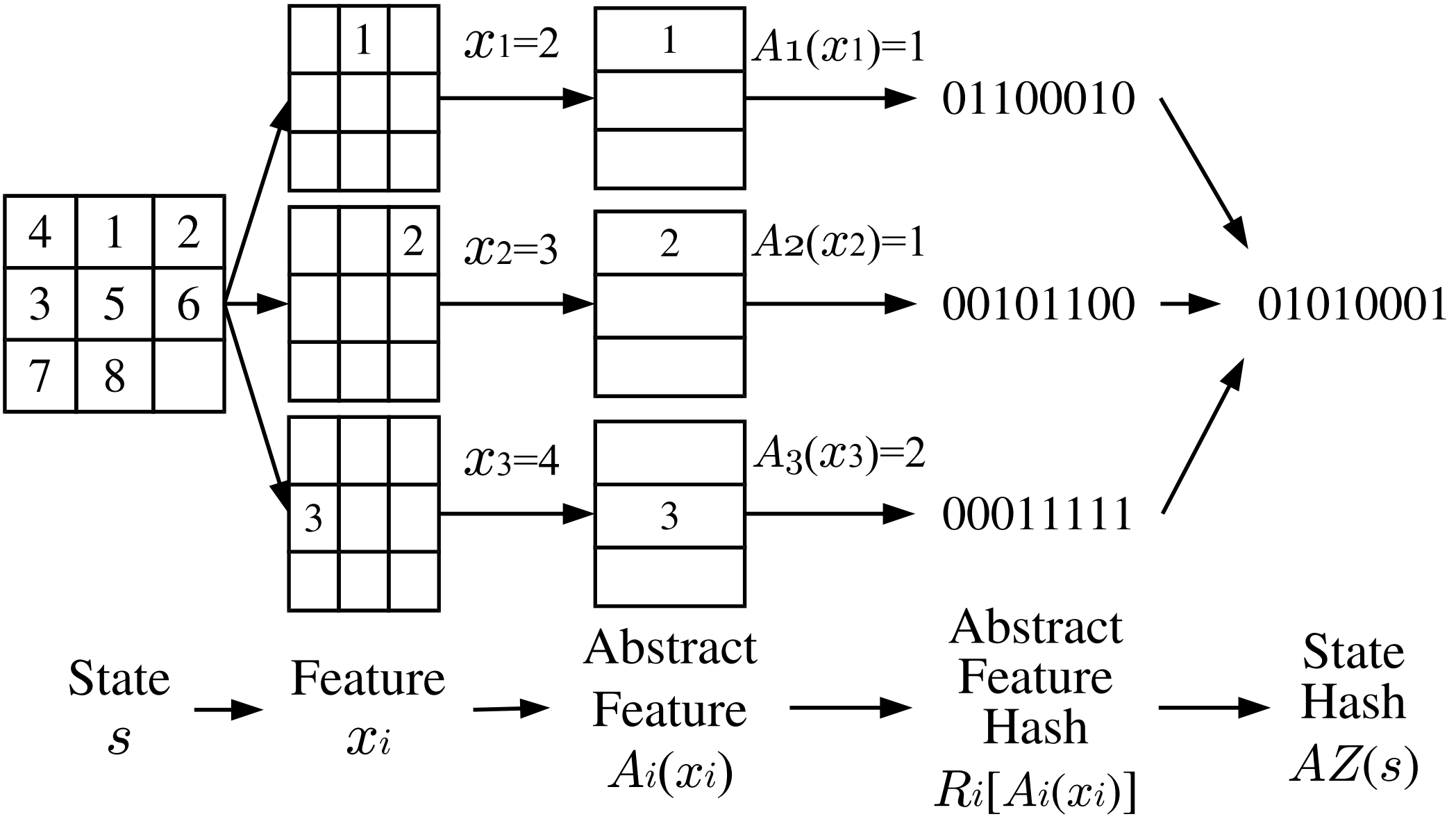
$$A(s) = R[s']$$



Abstract Zobrist Hashing (AZH)

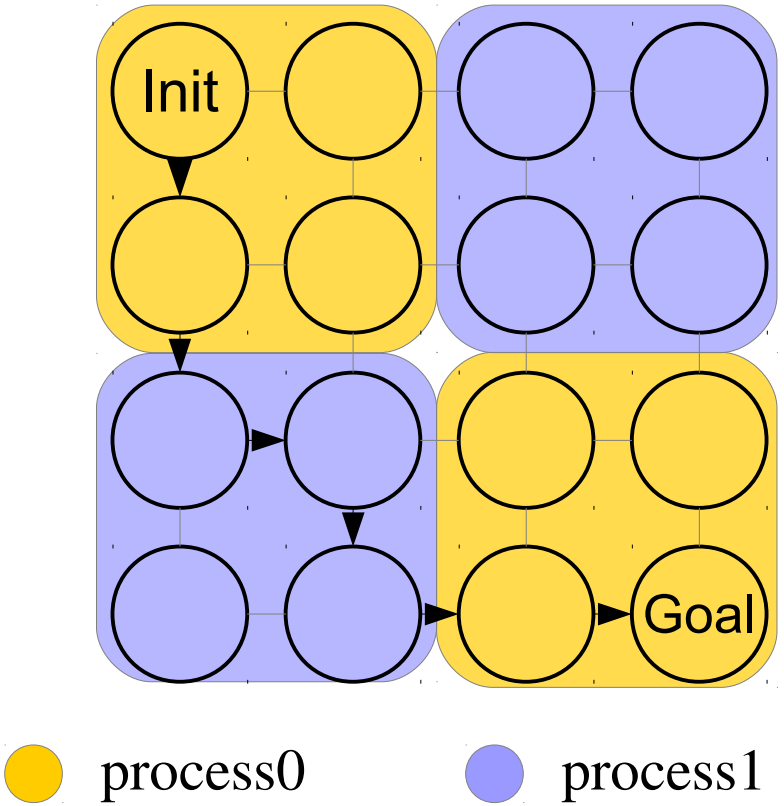
Jinnai&Fukunaga (2016)

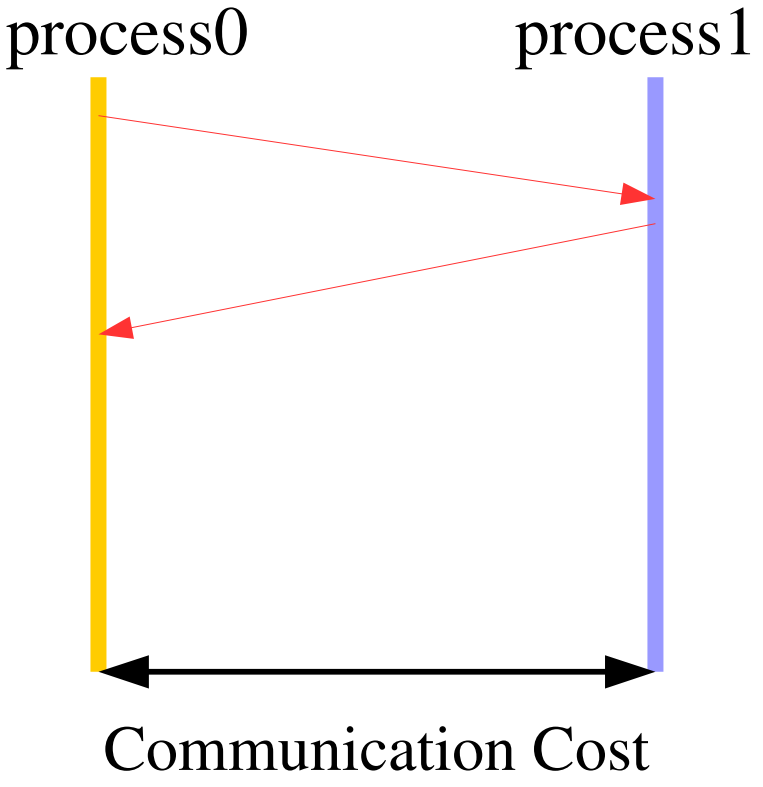
$$AZ(s) = R_1[A_1(x_1)] \text{ xor } R_2[A_2(x_2)] \text{ xor } \ldots \text{ xor } R_n[A_n(x_n)]$$



Abstract Zobrist Hashing の利点

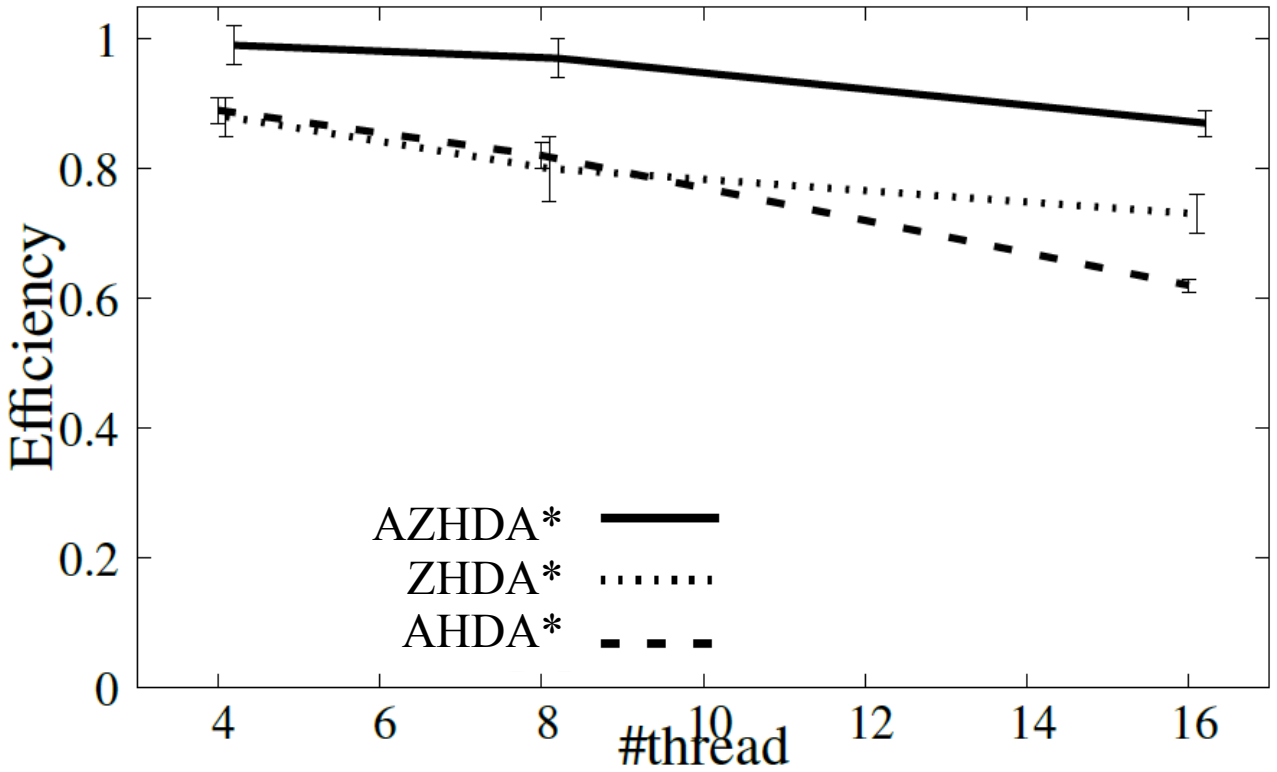
- Zobrist hashing を利用することで偏りなくノードを分配することが出来る
- (Feature) abstraction を利用することによって通信オーバーヘッドを抑えることが出来る





実験比較

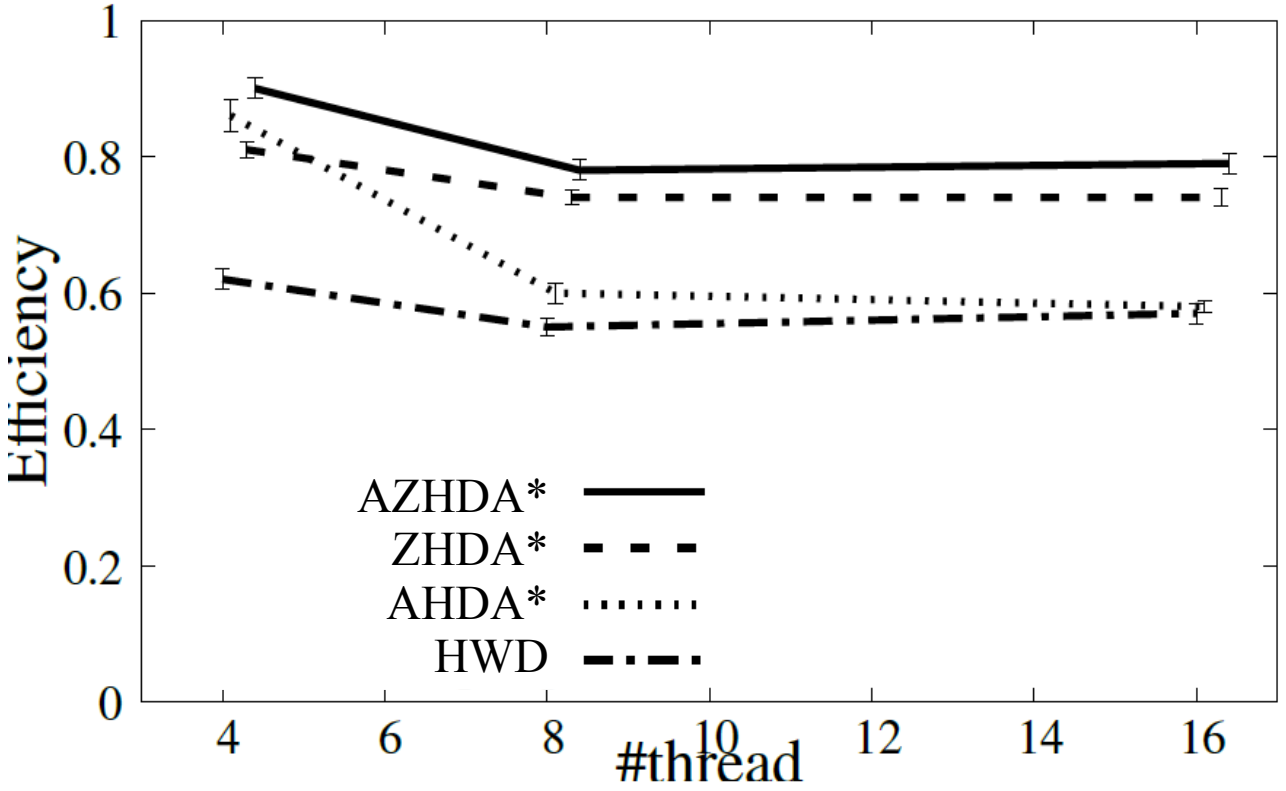
- 15-puzzle, 24-puzzle, grid-pathfinding, multiple seuqence alignment において**手動生成された abstract feature** を用いて既存手法 (Zobrist hashing, Abstraction) よりも高速であることが示した



24-puzzle における性能評価
(Jinnai&Fukunaga 2016)

実験比較

- 15-puzzle, 24-puzzle, grid-pathfinding, multiple seuqence alignment において**手動生成された abstract feature** を用いて既存手法 (Zobrist hashing, Abstraction) よりも高速であることが示した



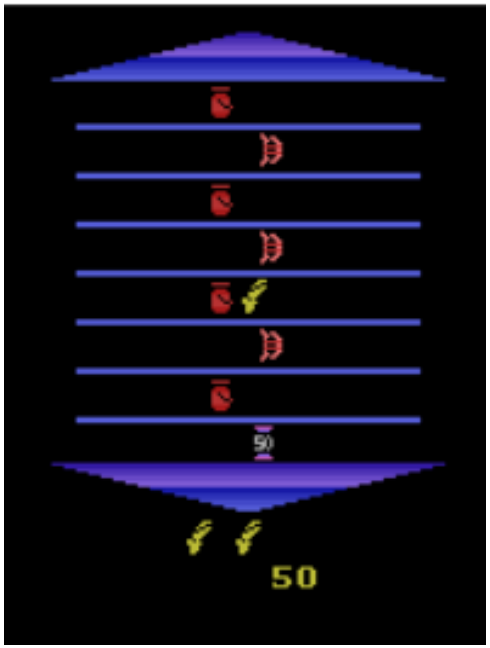
Multiple sequence alignment における性能評価
(Jinnai&Fukunaga 2016)

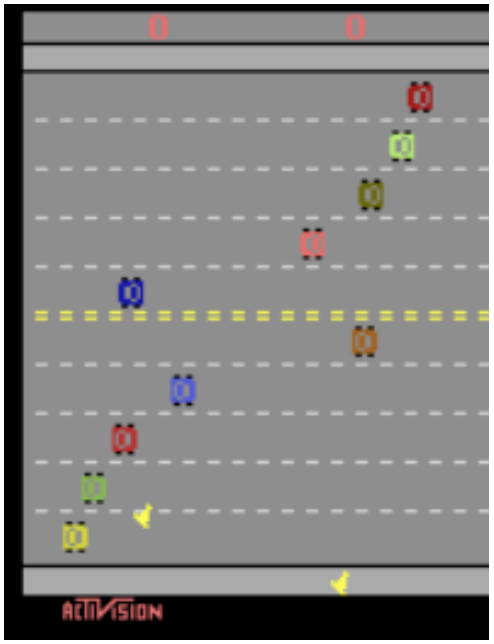
おはなし

- 古典的プランニング問題
- A* 探索
- BDD を用いたグラフ探索
- 並列 A* 探索 (Hash Distributed A*)
- **Black-box Planning (ビデオゲーム AI)**

Black-box Planning in Arcade Learning Environment

- What a human sees



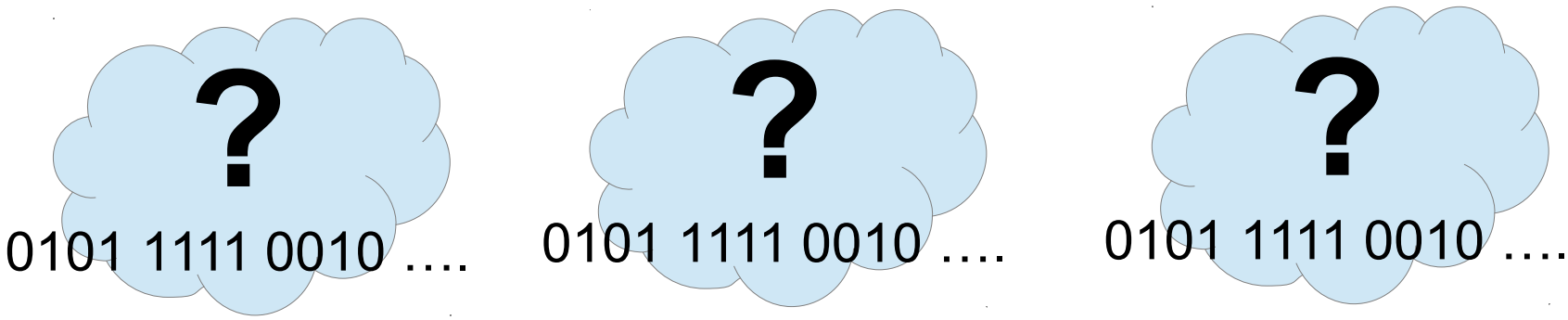


Arcade Learning Environment

(Bellemare et al. 2013)

Black-box Planning in Arcade Learning Environment

- What the computer sees



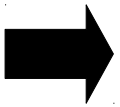
Arcade Learning Environment
(Bellemare et al. 2013)

General-purpose agents have many irrelevant actions

- 各ゲームで使われるアクションは ALE の可能なアクション (18 個) の一部

Neutral	Neutral + fire
Up	Up + fire
Up-left	Up-left + fire
Left	Left + fire
Down-left	Down-left + fire
Down	Down + fire
Down-right	Down-right + fire
Right	Right + fire
Up-right	Up-right + fire

ALE の可能なアクション (18 個)

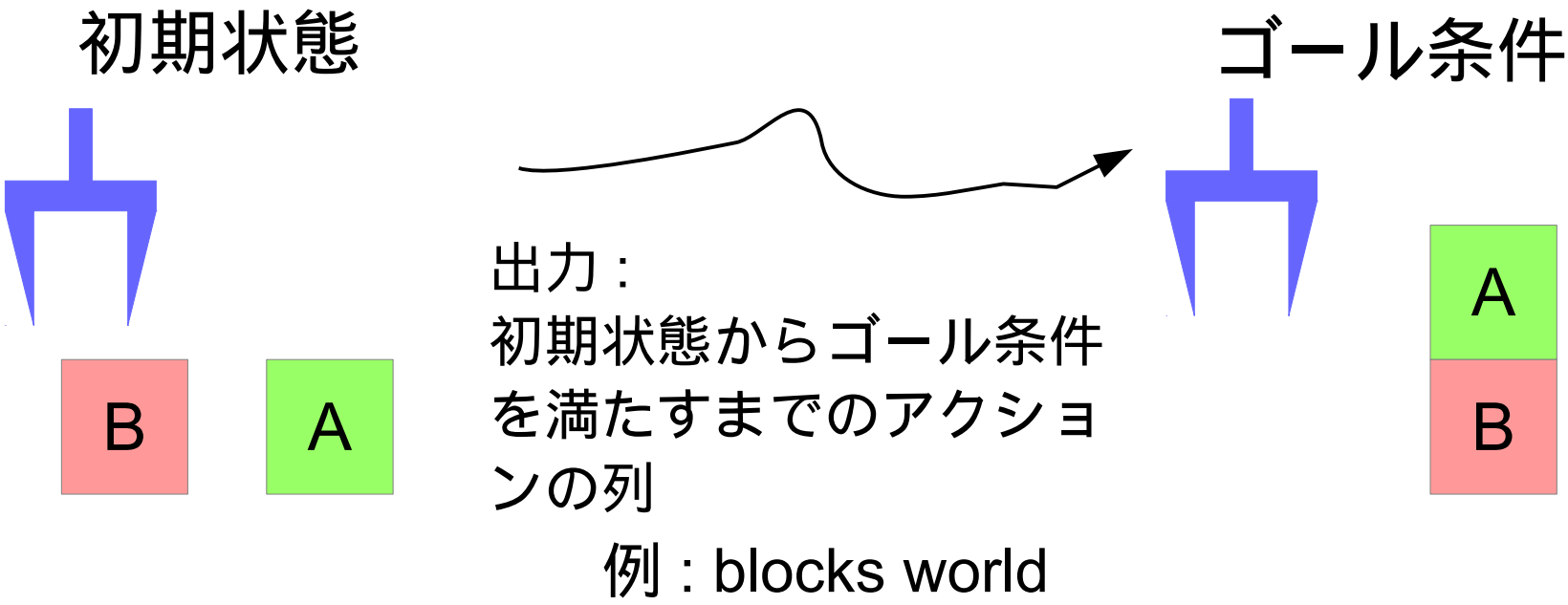


Neutral
Up
Left
Down
Right

PACMAN で使われる
アクション

プランニング問題

- プランニングは初期状態とゴール条件、可能なアクション集合を入力とし、初期状態からゴール条件を満たすためのアクションの列を返す問題である
- ドメインの記述方法
  - Transparent model domain (e.g. PDDL)
  - Black-box domain

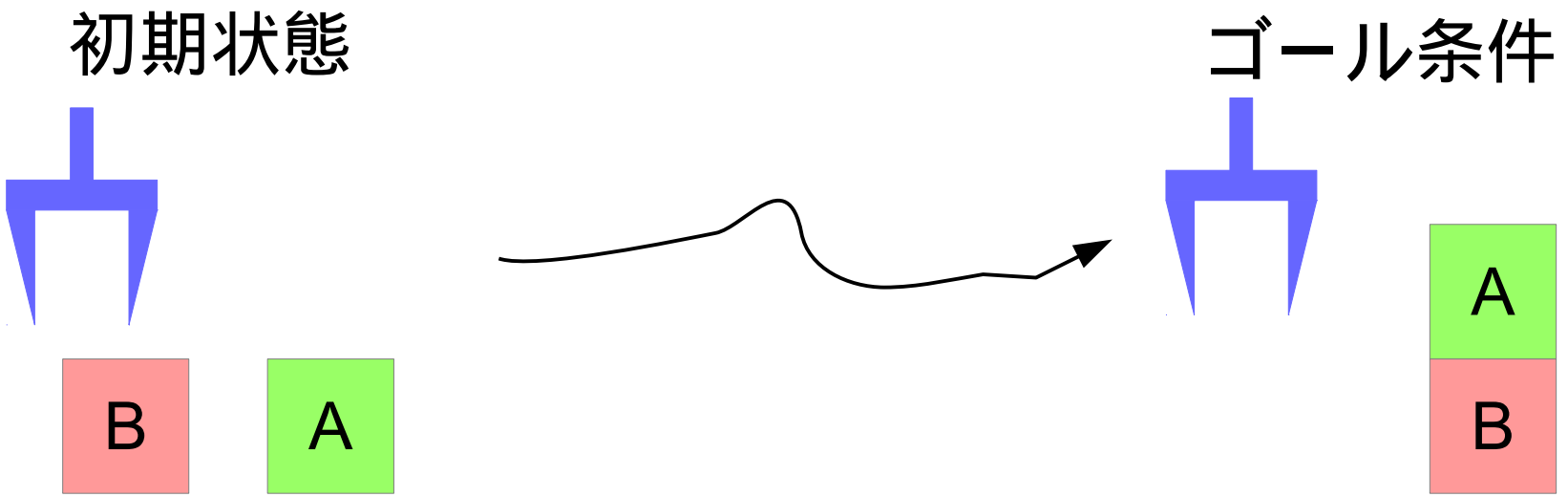


例：blocks world

Transparent Model Domain

入力：初期状態・ゴール条件・アクションが PDDL という言語で詳細に記述される

```
Init: ontable(a),ontable(b),clear(a),clear(b)
Goal: on(a,b)
Action:
   Move(b,x,y)
      Precond: on(b,x),clear(x),clear(y)
      Effect:  on(b,y),clear(x),¬on(b,x),¬clear(y)
```

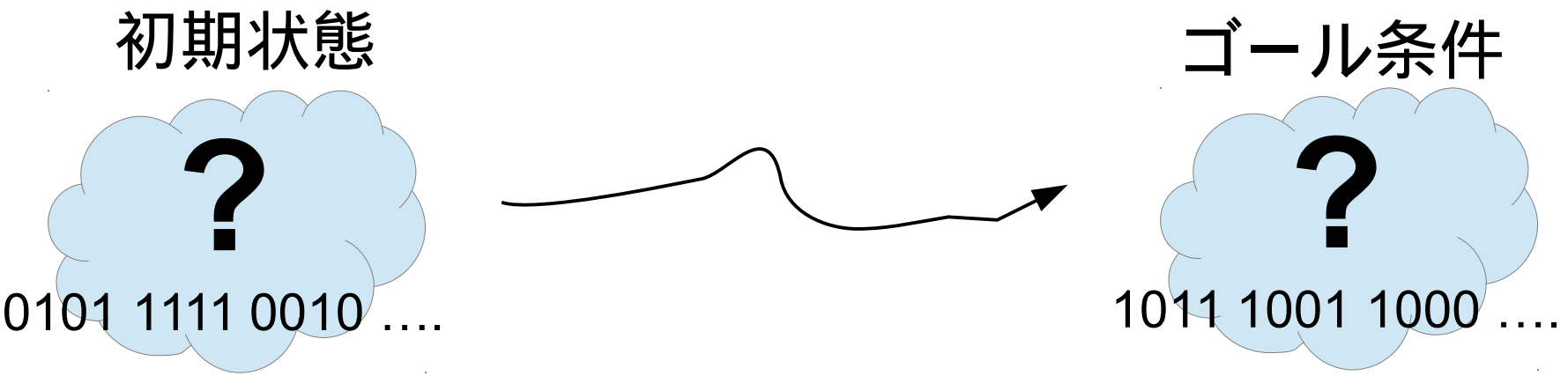


例：blocks world

Black-box Domain

- 未知の環境に対してはドメインモデルが存在しない
- Black-box ドメインにおけるドメインの記述 :
  - s_0:　　初期状態 (bit vector)
  - *suc*(*s*, *a*): 状態とアクションの組に対して次の状態を返す Successor generator 関数 (ブラックボックス)
  - *r*(*s*, *a*):　reward 関数 / ゴール条件 (ブラックボックス)

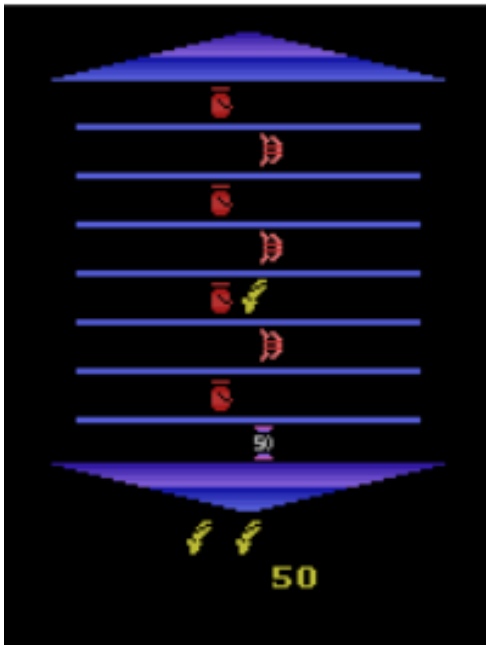
  → ドメインモデルを利用したあらゆる工夫が使えない！



Arcade Learning Environment (ALE)

(Bellemare et al. 2013)

- Arcade Learning Environment におけるドメインの記述 :
  - 状態 : RAM の状態 (1024 bits の bit vector)
  - Successor generator: 完全な emulator (+ random seed)
  - reward 関数 : 完全な emulator (+ random seed)

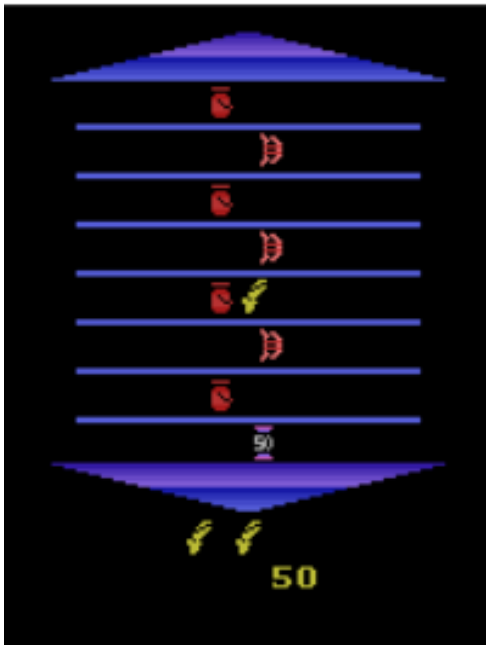


Arcade Learning Environment

Arcade Learning Environment (ALE)

(Bellemare et al. 2013)

- Arcade Learning Environment におけるドメインの記述 :
  - 可能なアクションは 18 個
    (各ゲームは必ずしも 18 個全てを必要としない)
  - ゲームは無限に長くなりうる →探索で解き切るのは不可能



Arcade Learning Environment

ALE における Agent の問題設定

(Bellemare et al. 2013)

- **On-line planning setting**

  Agent はゲームの最中に k (=5) フレーム毎にプランニングを繰り返し行い、次の 5 フレーム間継続して行う一つのアクションを選択する

  (agent は 5 フレーム間同じアクションを続けて行う)

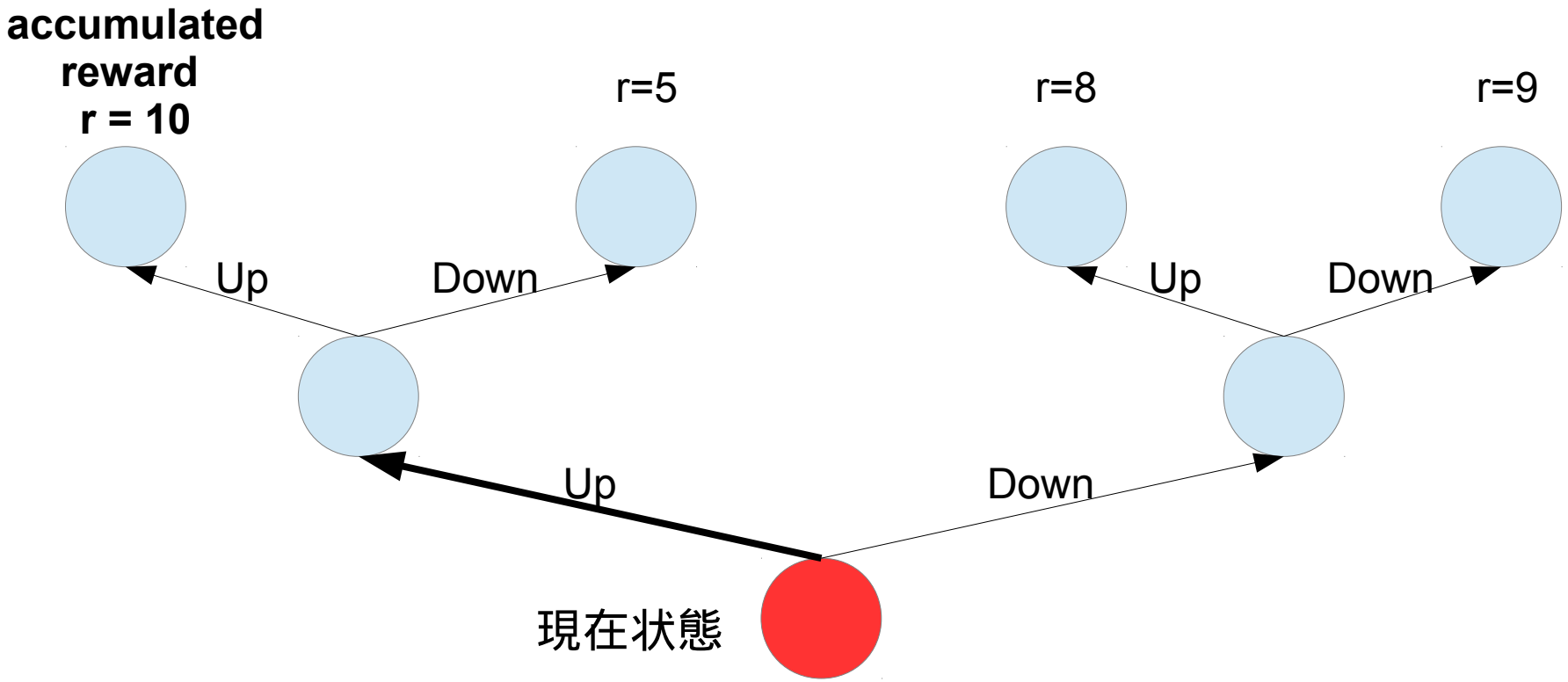
- Learning setting

  Agent はゲーム開始前に学習を行い、すべての状態から次に選ぶアクションのマップを生成する。ゲーム実行時はそのマップに従ってアクションを選択する (e.g. Google DQN)

Online planning setting on ALE

(Bellemare et al 2013)

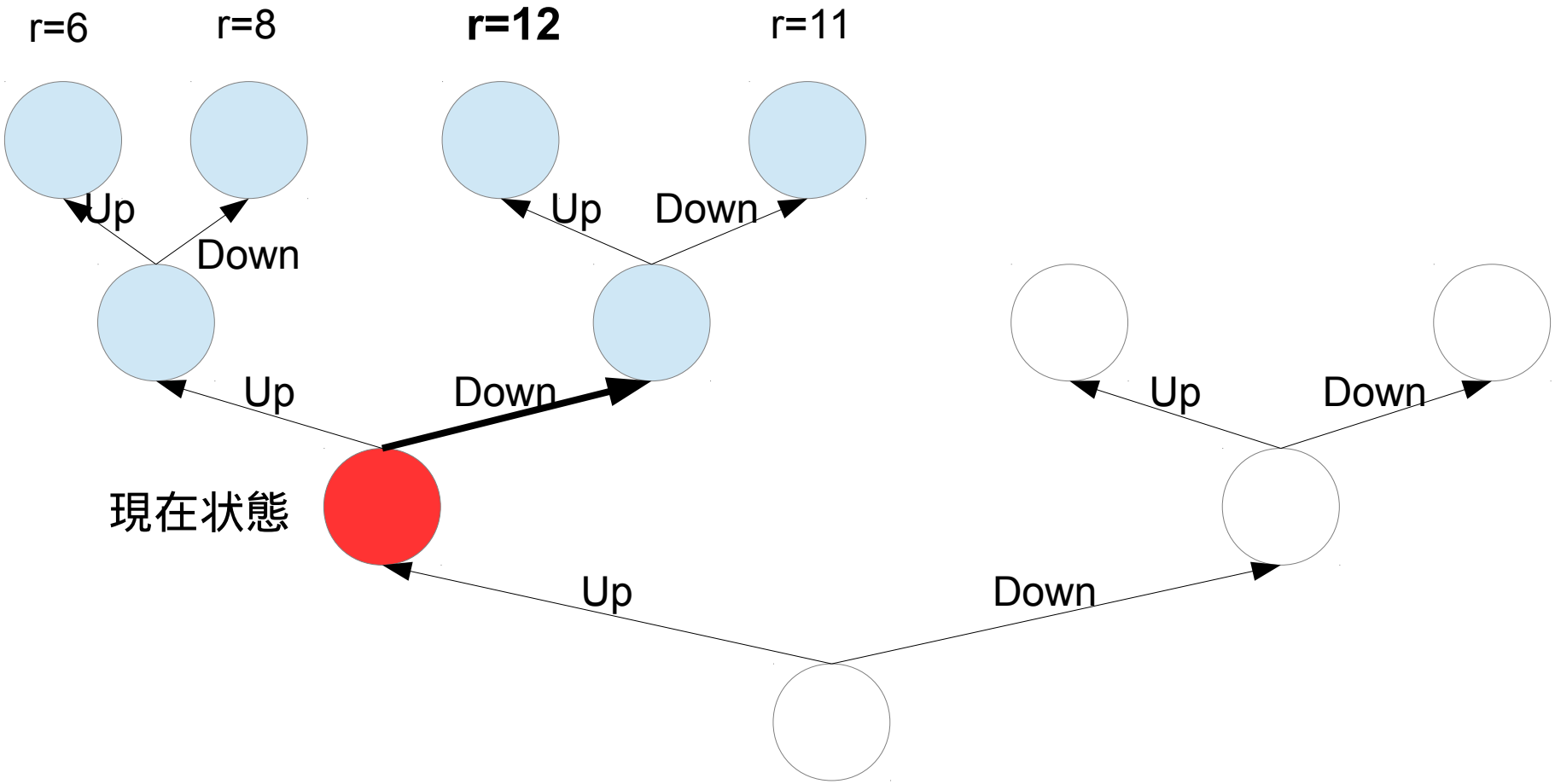
- 幅優先探索
  現在状態から幅優先探索を行い accumulated reward を最大化する経路につながるアクションを選択する



Online planning setting on ALE

(Bellemare et al 2013)

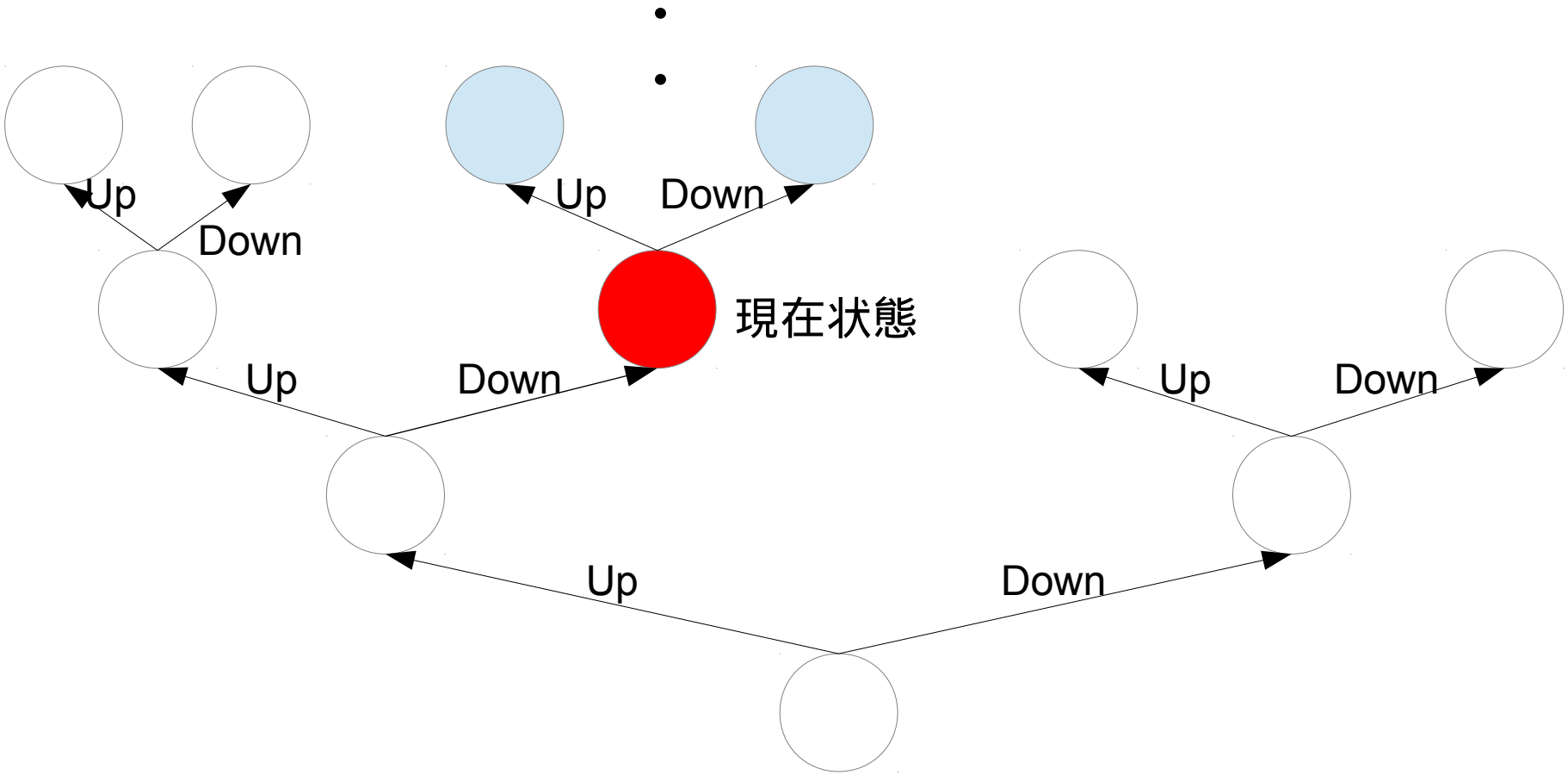
- 幅優先探索
  現在状態から幅優先探索を行い accumulated reward を最大化する経路につながるアクションを選択する



Online planning setting on ALE

(Bellemare et al 2013)

- 幅優先探索
  現在状態から幅優先探索を行い accumulated reward を最大化する経路につながるアクションを選択する



General-purpose agents have many irrelevant actions

- 各ゲームで使われるアクションは ALE の可能なアクション (18 個) の一部

Neutral	Neutral + fire
Up	Up + fire
Up-left	Up-left + fire
Left	Left + fire
Down-left	Down-left + fire
Down	Down + fire
Down-right	Down-right + fire
Right	Right + fire
Up-right	Up-right + fire

ALE の可能なアクション (18 個)

Neutral
Up
Left
Down
Right

PACMAN で使われる
アクション

General-purpose agents have many irrelevant actions

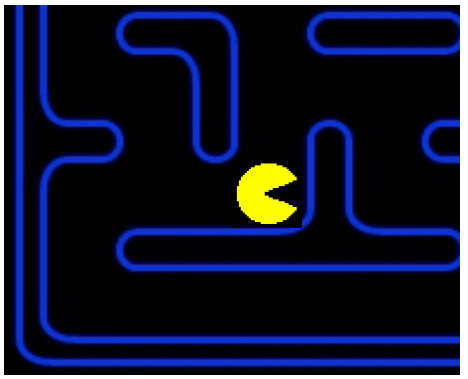
- 各ゲームで使われるアクションは ALE の可能なアクション (18 個) の一部
- ゲームの各状態で意味のあるアクションはさらにその一部

Neutral	Neutral + fire
Up	Up + fire
Up-left	Up-left + fire
Left	Left + fire
Down-left	Down-left + fire
Down	Down + fire
Down-right	Down-right + fire
Right	Right + fire
Up-right	Up-right + fire

ALE の可能なアクション (18 個)

Neutral
Up
Left
Down
Right

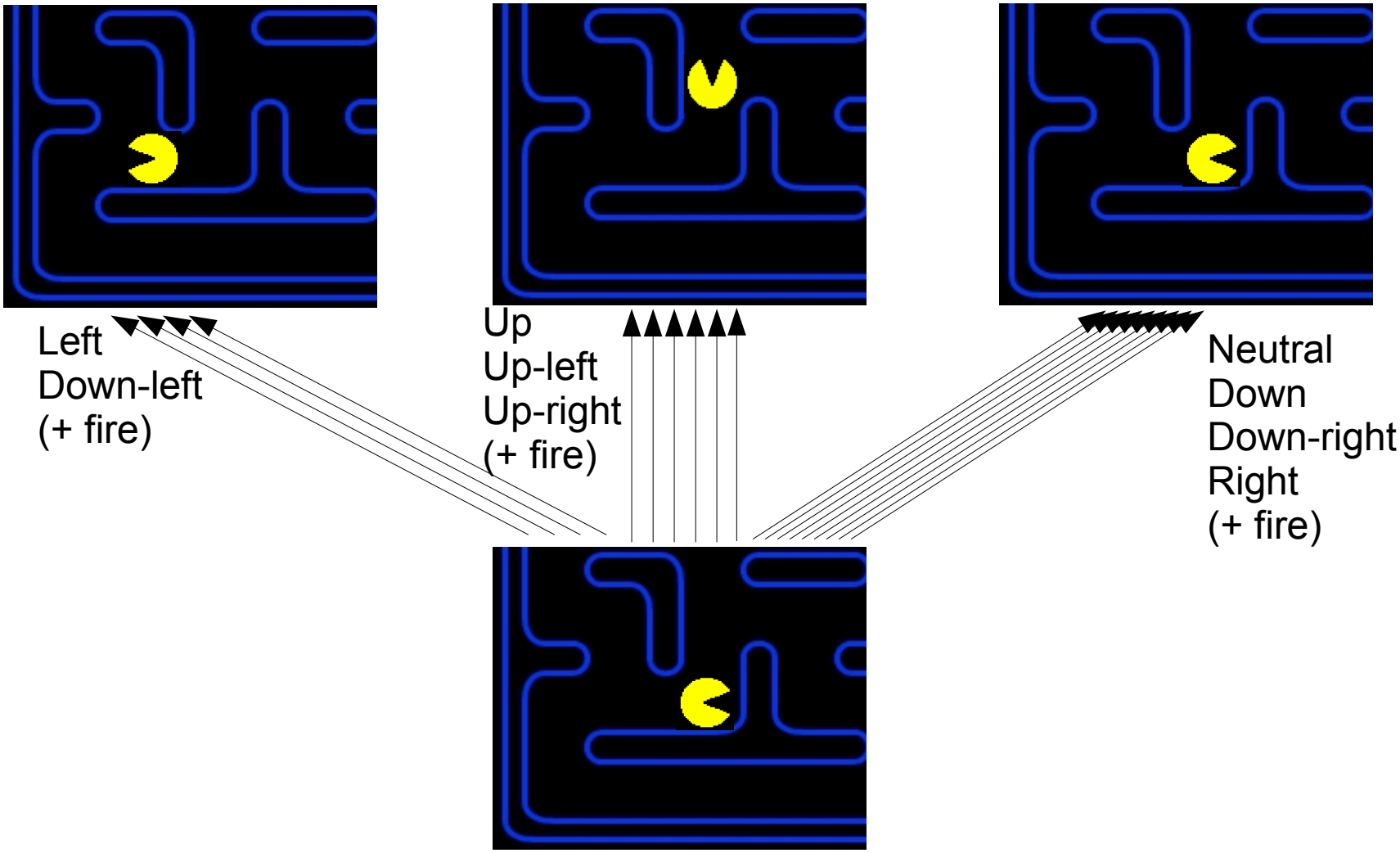
PACMAN で使われるアクション



Neutral
Up
Left

画像の状態で意味のあるアクション

General-purpose agents have many irrelevant actions



最も時間のかかるノードの生成を 6 倍も行っている！

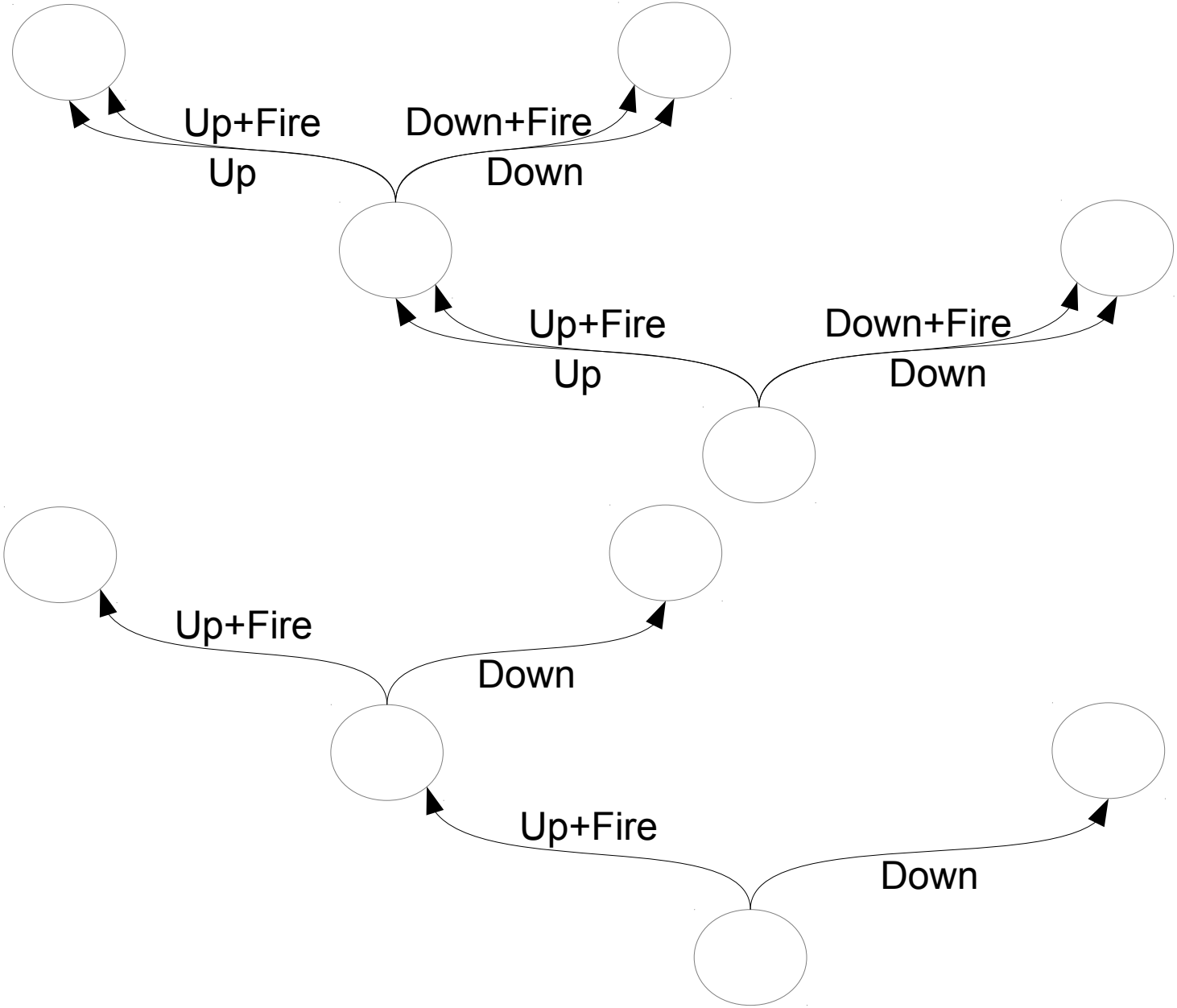
→ ムダなアクションを検出することでより有意義に
計算資源を使えるはず！

提案手法：Dominated action sequence pruning (DASP) Jinnai&Fukunaga (2017)

・過去の探索木を再現できる最小のアクションの集合を求める

アクション集合
{Up,
Down,
Up+Fire,
Down+Fire}

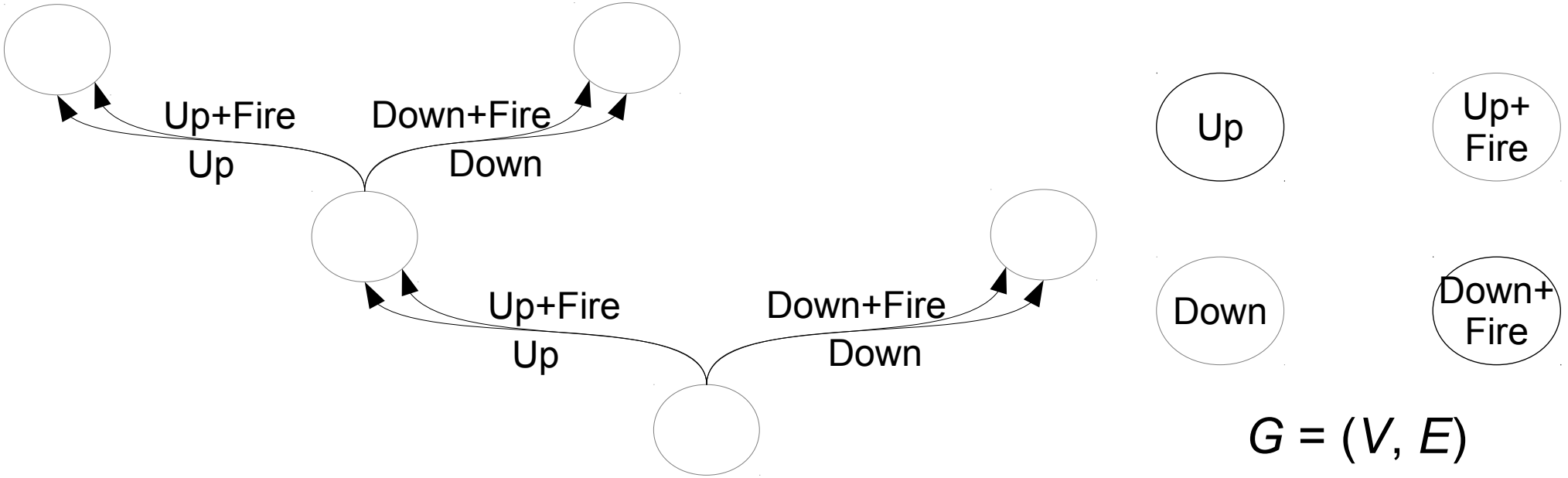
アクション集合
{Down,
Up+Fire}



提案手法：Dominated action sequence pruning (DASP)Jinnai&Fukunaga (2017)

- Algorithm: 過去の探索木を再現できる最小のアクションの集合 A^L を求める

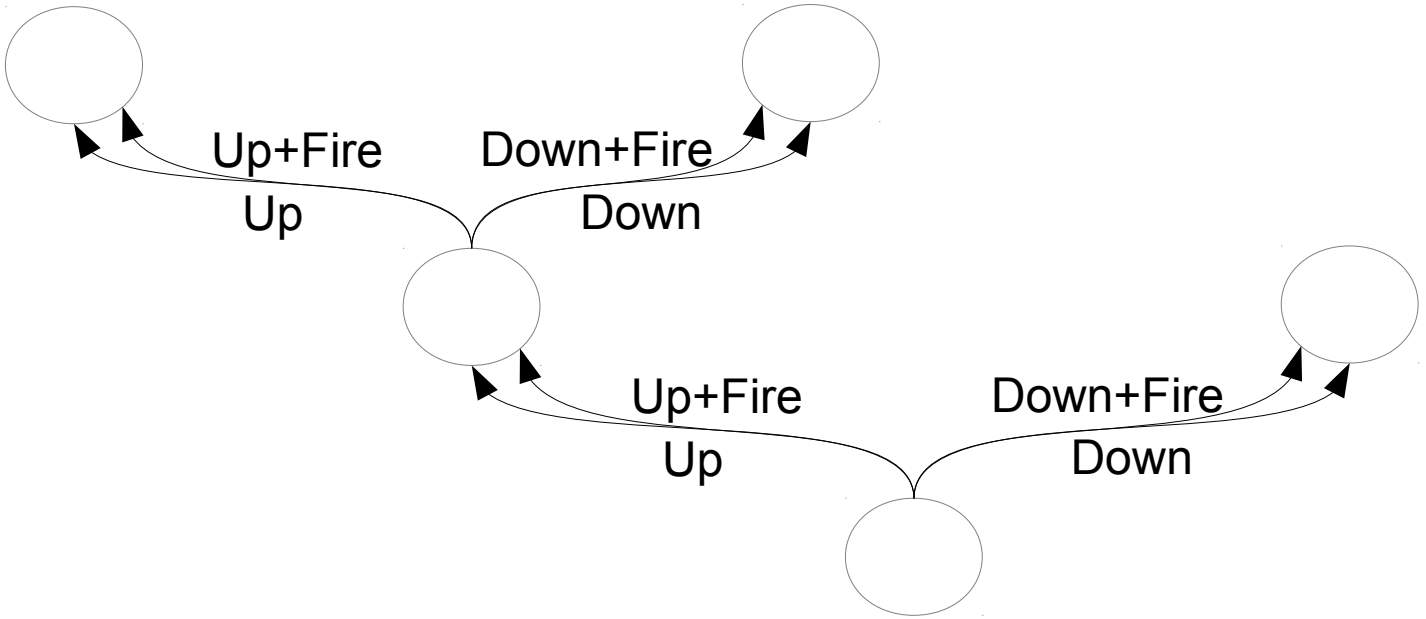
1. ハイパーグラフ $G = (V, E)$ を考える。 $v_i \in V$ はアクション i に対応する。

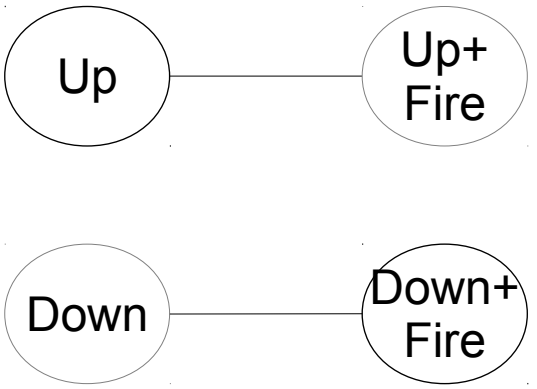


提案手法：Dominated action sequence pruning (DASP)Jinnai&Fukunaga (2017)

- Algorithm: 過去の探索木を再現できる最小のアクションの集合 A^L を求める

1. ハイパーグラフ $G = (V, E)$ を考える。 $v_i \in V$ はアクション i に対応する。アクション $v_0, v_1, \ldots, v_n$ によって重複して生成されるノードが存在する場合に $e(v_0, v_1, \ldots, v_n) \in E$ が存在する



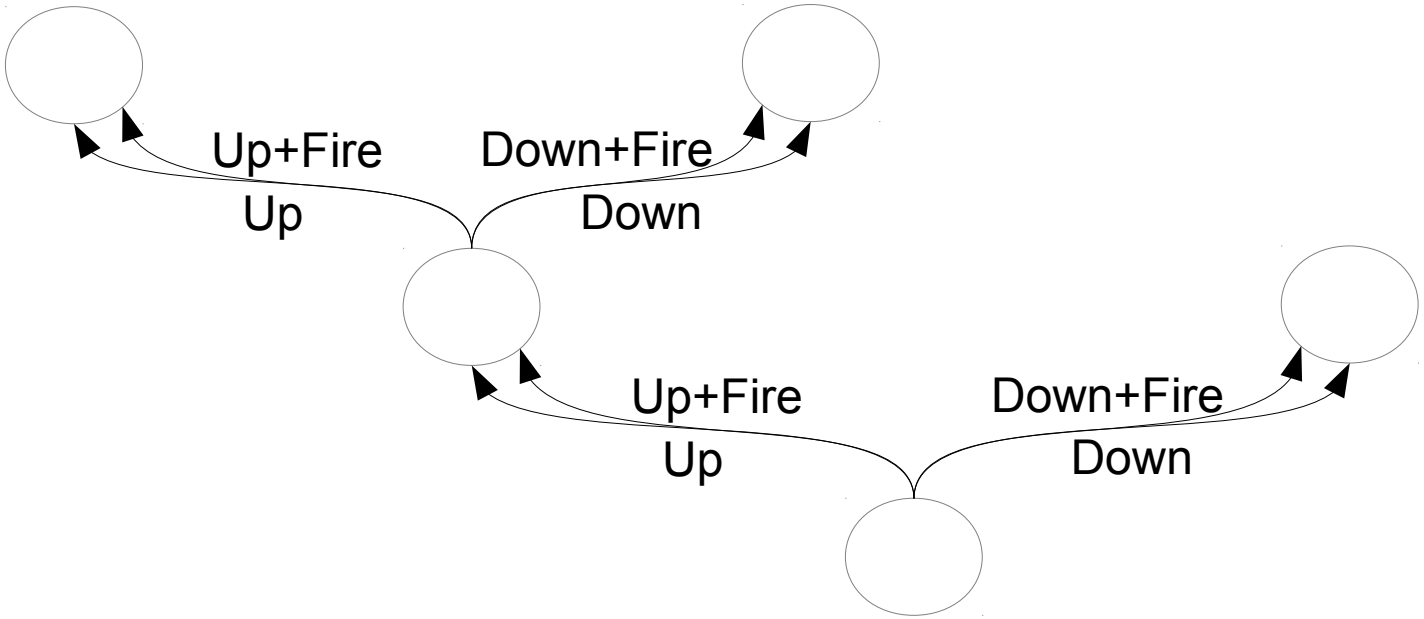


$G = (V, E)$

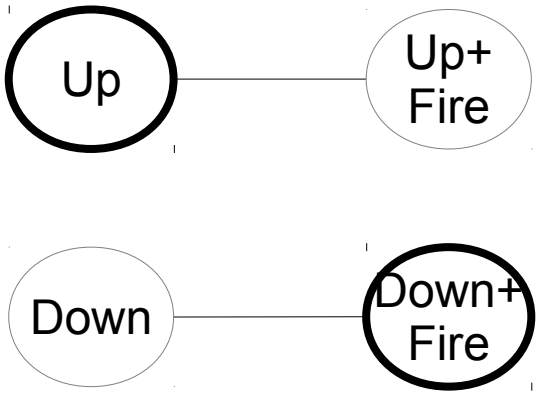
提案手法：Dominated action sequence pruning (DASP)Jinnai&Fukunaga (2017)

- Algorithm: 過去の探索木を再現できる最小のアクションの集合 A^L を求める

1. ハイパーグラフ $G = (V, E)$ を考える。 $v_i \in V$ はアクション i に対応する。アクション $v_0, v_1, \ldots, v_n$ によって重複して生成されるノードが存在する場合に $e(v_0, v_1, \ldots, v_n) \in E$ が存在する
2. G の minimal vertex cover を A^L に加える



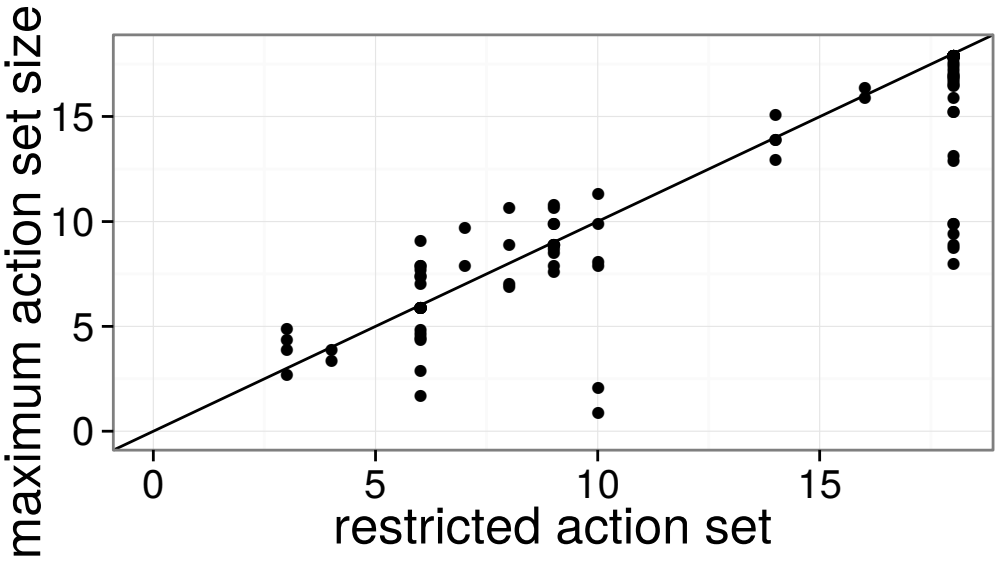
A^L = {Up,Down+Fire}



$G = (V, E)$

実験結果：枝刈りされたアクションの数

- 多くのアクションの枝刈りに成功している
- Restricted action set: 各ゲームに対して人間が作った必要最小限のアクションセットのアクション数



DASP1
(jittered)

DASP の問題点

- DASP は枝刈りをするかしないか二者択一である
- 殆どのアクションは**ある条件下で**効果があるものである
  (conditionally effective)
- 例えば FIRE アクションは Agent が SWORD や BOMB を獲得してはじめて意味があるものかもしれない (必要なアクションなのに DASP では枝刈りされてしまうかもしれない)
  → 過去の探索木の再現が出来ても未来のものは分からない
- 例えば Agent の左手に壁がある場合 LEFT アクションは無意味かもしれない (DASP では上手く枝刈りされない)
  → 現在状態の context に応じて枝刈りをするべき！

提案手法：Dominated action sequence avoidance (DASA)Jinnai&Fukunaga (2017)

- t 回目のプランニングにおいてアクション a が新しいノードを生成した割合を p(a, t) と置く。
- p を利用して、 t+1 回目のプランニングにおいてアクション a が新しいノードを生成する確率 p* を推定する

$$p^*(a,0)=1$$

$$p^*(a,t+1)=\frac{p(a,t)+\alpha p^*(a,t)}{1+\alpha}$$

- t 回目のプランニングでは各ノードの展開時に P(a, t) の確率でアクション a を適用する

$$P(a,t)=(1-\epsilon)s(p^*(a,t))+\epsilon$$

s は適当な smoothing 関数 (e.g. sigmoid)
ε はアクション a を適用する最小限の確率

実験結果：Prune されたアクションの数

- ALE で提供される 53 個のゲームでスコアを比較
- p-IW(1) (Shleyfman et al. 2016), IW(1), 幅優先探索に DASA, DASP を適用
- 各プランニングステップにおける生成ノード数の上限を 2000 に設定 (≒ 実行時間の制限)
- ただし前プランニングで生成済のノードはカウントせず

DASA2: DASA を長さ 2 のアクション列に対して適用

DASA1: DASA を長さ 1 のアクション列に対して適用

DASP1: DASP を長さ 1 のアクション列に対して適用

default: ALE で可能な 18 個のアクションをすべて適用

restricted: 各ゲームに対して人間が作った必要最小限のアクションセット

実験結果：スコアの比較

- 全ての探索アルゴリズムにおいて DASA2 の coverage が最も高かった

- 400gend: 生成ノード数を 400 までさらに制限した場合でも DASA2 が最も coverage が高かった
- extend: 冗長な押ボタンを 2 つ加えたアクションセットを用いて比較した場合、 DASA2 と default の差はさらに広がった

	DASA2	DASA1	DASP1	default	restricted
p-IW(1)	**22**	10	4	6	10
p-IW(1) (400gend)	**24**	14	6	5	7
IW(1)	**22**	9	7	7	8
幅優先探索	**18**	11	11	6	11
p-IW(1) (extend)	**39**	22	19	16	-

各列 e)

実験結果：展開ノード数・探索の深さ

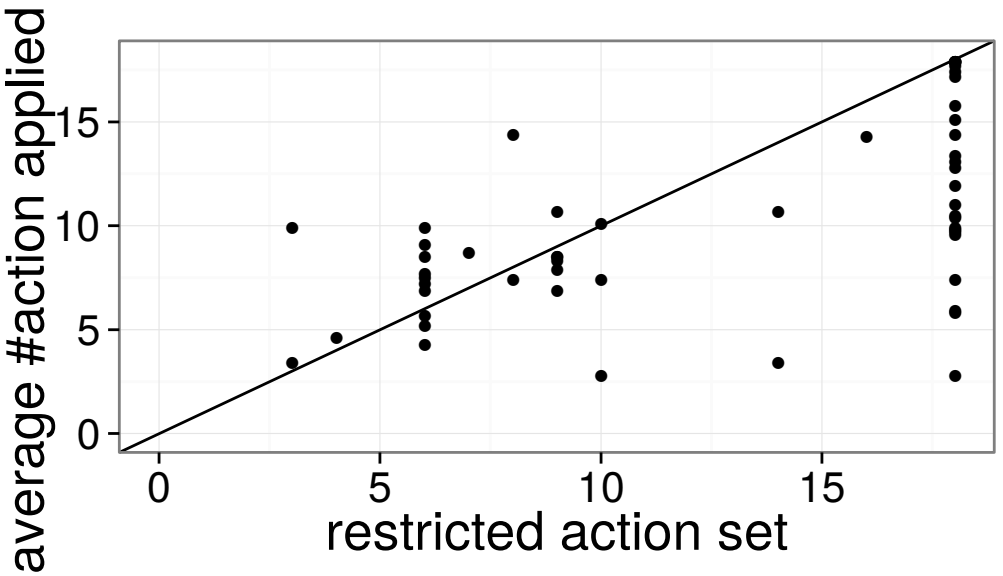
- p-IW(1) における各プランニングの展開ノード数・深さの比較
  (展開ノード数 ＜ 生成ノード数 ≦ 1000)
- DASA2 が最も展開ノード数が多く最も深く探索が出来ている

	DASA 2	DASA 1	DASP 1	default	restricted
expanded	**254.9**	191.1	119.9	119.6	234.0
depth	**82.8**	59.5	34.6	34.1	40.8

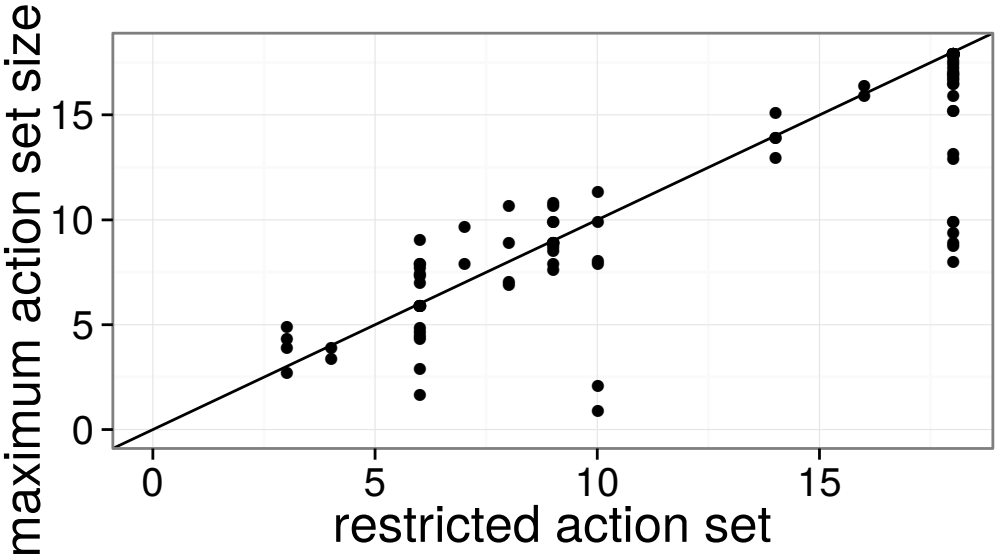
平均生成ノード数・探索の深さ

実験結果：枝刈りされたアクションの数

- 多くのアクションの枝刈りに成功している
- Restricted action set: 各ゲームに対して人間が作った必要最小限のアクションセットのアクション数



DASA2



DASP1
(jittered)

まとめ

- Black-box ドメインにおけるプランニングの効率化のための手法として DASP, DASA を提案
- DASP/DASA によって冗長なアクションを枝刈りすることによって展開ノード数が増加しより高いスコアが得られた

Future Work

- Reinforcement Learning では対称な 2 つのアクションを検出するフレームワークは存在しない (?)
- Dominated Action Sequences を検出することで RL の学習に必要な時間が大きく減らせると考えられる

これから

- 現状の BDD ベースのプランナーは BDD をブラックボックスとして扱っている
- BDD の中身まで触ればより効率的なプランナーが出来るかも？

- FastDownward というプランナーが state-of-the-art でありこれを元に研究開発が行われている (http://fast-downward.org/)
- プランニング・ヒューリスティック探索の入門書を書き途中 (https://github.com/jinnaiyuu/search-ja)