

計画立案と行動

Chapter 12

概要

- 時間、スケジュール、資源
- 階層的タスク・ネットワーク計画立案
- 非決定論的領域
 - 条件付計画立案
 - モニタリングの実行と再計画立案
 - 連続した計画立案
- 多重エージェントの計画立案

時間、スケジュール、資源

- これまで:
 - どの動作をすべきか
- 現実世界:
 - + 動作は時間に合わせてあるときに行う
 - + 動作にははじめと終わりがある
 - + 動作には一定の時間かかる
- ジョブショップ・スケジューリング:
 - ジョブの集合を完了する。それぞれのジョブは動作の系列で構成される
 - 動作は与えられた時間を必要とし、資源を必要とすることも
 - 全てのジョブの完了に必要な時間を最小とするスケジュールを求めなさい(資源の制約を守りながら)

Car construction example

*Init(Chassis(C1) \wedge Chassis(C2) \wedge Engine(E1,C1,30) \wedge Engine(E1,C2,60) \wedge
Wheels(W1,C1,30) \wedge Wheels(W2,C2,15))*

Goal(Done(C1) \wedge Done(C2))

Action(AddEngine(e,c,m)

PRECOND: Engine(e,c,d) \wedge Chassis(c) \wedge \neg EngineIn(c)

EFFECT: EngineIn(c) \wedge Duration(d)

Action(AddWheels(w,c)

PRECOND: Wheels(w,c,d) \wedge Chassis(c)

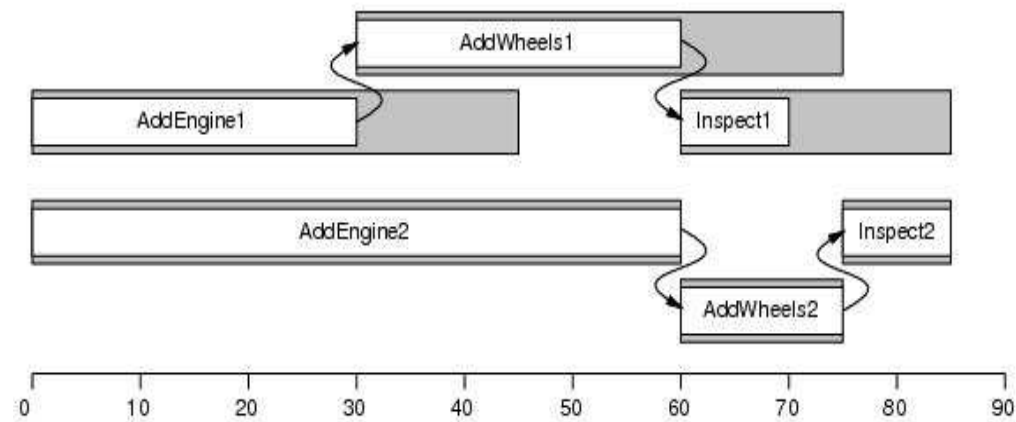
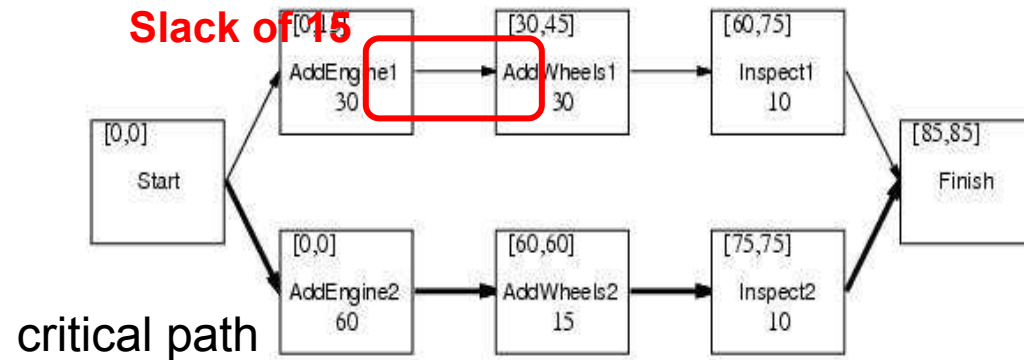
EFFECT: WheelsOn(c) \wedge Duration(d)

Action(Inspect(c)

PRECOND: EngineIn(c) \wedge WheelsOn(c) \wedge Chassis(c)

EFFECT: Done(c) \wedge Duration(10)

半順序計画立案による解



計画立案とスケジューリング

- スケジューリング問題と標準の計画立案問題との違いは
- いつ動作を開始し、いつ終了するのか
 - 動作にかかる時間とそれらの順番を考慮する
Duration(d)
- クリティカル・パスの方法が開始と終了の時間を決めるために使われる:
 - パス= 開始から終了までの線形な系列
 - クリティカル・パス = 最短の期間を持つパス
 - 全体計画の期間を決定する
 - クリティカル・パスは遅延なしに実行される

ES and LS

- 可能な最も早い (ES)、最も遅い (LS) 開始時間.
- $LS-ES =$ 動作の緩み
- 全ての動作に対して、問題全体でのスケジュールを決定する

$$ES(Start) = 0$$

$$ES(B) = \max_{A < B} ES(A) + Duration(A)$$

$$LS(Finish) = ES(Finish)$$

$$LS(A) = \min_{A < B} LS(B) - Duration(A)$$

最も早い時
間の最大

- 複雑さは $O(Nb)$ (半順序で計画は与えられている)

資源つきスケジューリング

- 資源の制約 = 仕事を達成するため材料やものを必要とする
 - 再利用可能な資源
 - 動作を行うときに占有されるが終了したときは利用可能となる資源
 - 動作の構文での必要な拡張:
Resource:R(k)
 - K個の資源を動作では必要とする
 - 動作を行うための前提条件である
 - 他の動作はこのk個を使うことができなくなる

Car example with resources

Init(*Chassis*(C1) \wedge *Chassis*(C2) \wedge *Engine*(E1, C1, 30) \wedge *Engine*(E1, C2, 60) \wedge
Wheels(W1, C1, 30) \wedge *Wheels*(W2, C2, 15) \wedge *EngineHoists*(1) \wedge *WheelStations*(1)
 \wedge *Inspectors*(2))

Goal(*Done*(C1) \wedge *Done*(C2))

Action(*AddEngine*(e, c, m)

PRECOND: *Engine*(e, c, d) \wedge *Chassis*(c) \wedge \neg *EngineIn*(c)

EFFECT: *EngineIn*(c) \wedge *Duration*(d),

RESOURCE: *EngineHoists*(1))

Action(*AddWheels*(w, c)

PRECOND: *Wheels*(w, c, d) \wedge *Chassis*(c)

EFFECT: *WheelsOn*(c) \wedge *Duration*(d)

RESOURCE: *WheelStations*(1))

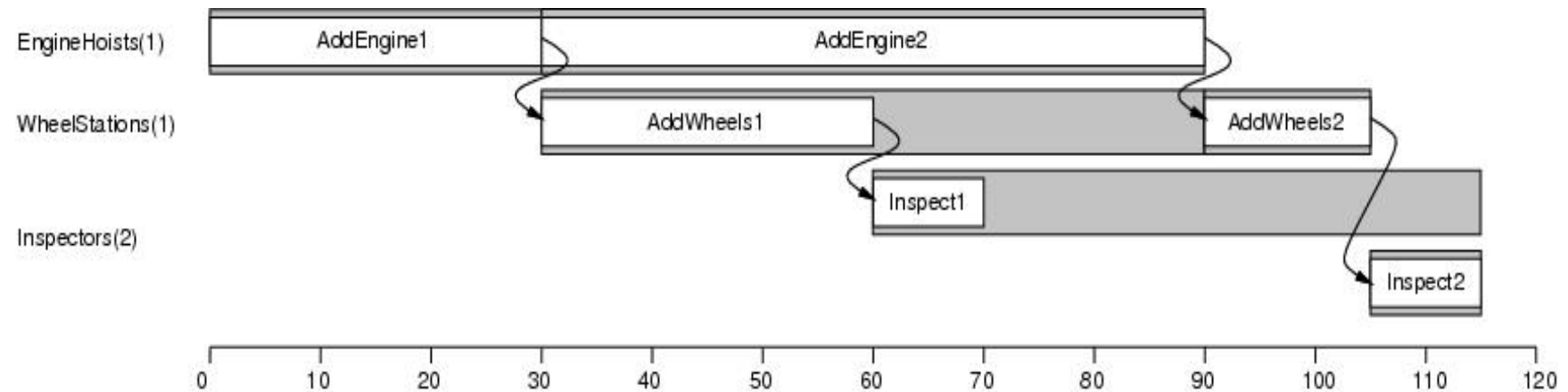
Action(*Inspect*(c)

PRECOND: *EngineIn*(c) \wedge *WheelsOn*(c) \wedge *Chassis*(c)

EFFECT: *Done*(c) \wedge *Duration*(10)

RESOURCE: *Inspectors*(1)) aggregation

Car example with resources



資源つきスケジューリング

- 集約 = 目的に関してものの間に区別がないとき
個々のものを量でくっつけたもの
 - 複雑性を軽減
- 資源制約はスケジューリング問題をより複雑にする
 - 動作の間での更なる相互作用
- 発見的手法: 最小緩みアルゴリズム
 - スケジュールし終えた先行の動作のあとに発生する(まだスケジュールされていない)動作の中から、最も早く開始できる、最小の緩みの動作を選択する

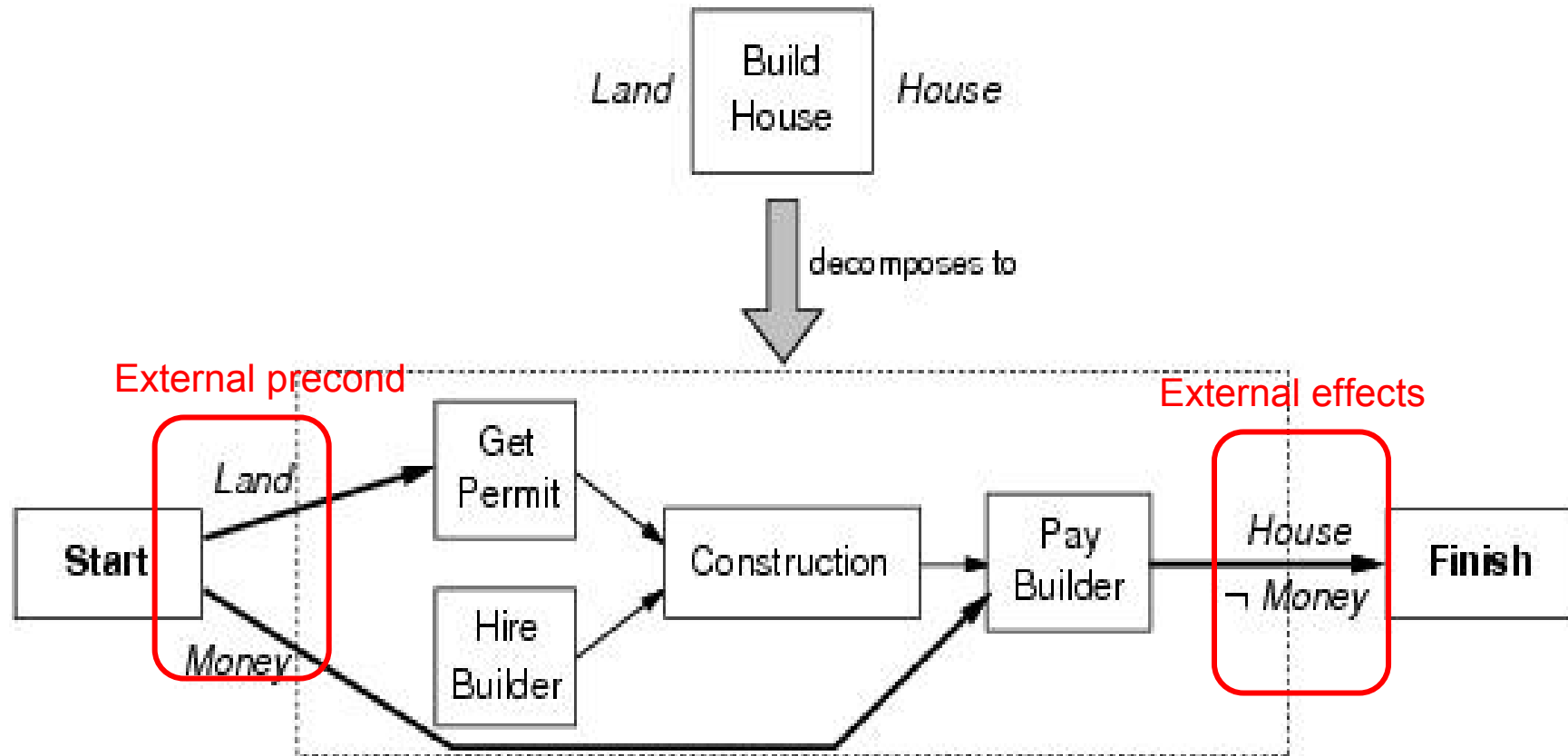
階層的タスクネットワーク計画立案

- 複雑性を減少 ⇒ 階層的分解
 - 階層のそれぞれのレベルで、計算機の仕事を次の低い階層での少数の活動に分解する
 - これらの活動に分解する計算機のコストは小さい
- 階層的ネットワーク(HTN)計画立案は分解を通して動作を洗練する
 - 例: building a house = getting a permit + hiring a contractor + doing the construction + paying the contractor.
 - 原子的な動作だけになるまで続ける
- 純粹なあるいはハイブリッドな階層的ネットワーク計画立案.

表現の分解

- 一般的な記述を計画ライブラリーに蓄える
 - それぞれの方法 = Decompos(a,d); a= 動作 d= 半順序計画
- Build Houseの例を参照
- 開始動作は他の動作によっては供給されない動作の前提条件を全て与える。
 - =外部の前提条件
- 終了動作は他の動作では示されない動作の全ての効果を示す
 - =外部効果
 - 1次効果 (ゴール達成のため)と2次効果

Build House example



Build House example

Action(Buyland, PRECOND: Money, EFFECT: Land \wedge \neg Money)

Action(GetLoan, PRECOND: Goodcredit, EFFECT: Money \wedge Mortgage)

Action(BuildHouse, PRECOND: Land, EFFECT: House)

Action(GetPermit, PRECOND: LAnd, EFFECT: Permit)

Action(HireBuilder, EFFECT: Contract)

Action(Construction, PRECOND: Permit \wedge Contract, EFFECT: HouseBuilt \wedge \neg Permit),

Action(PayBuilder, PRECOND: Money \wedge HouseBuilt, EFFECT: \neg Money \wedge House \wedge \neg \neg Contract),

Decompose(BuildHouse,

Plan ::STEPS{ S1: GetPermit, S2:HireBuilder, S3:Construction, S4 PayBuilder}

ORDERINGS: {Start < S1 < S3< S4<Finish, Start<S2<S3},

LINKS $\left\{ \begin{array}{l} \text{Start} \xrightarrow{\text{Land}} \text{S1}, \text{Start} \xrightarrow{\text{Money}} \text{S4}, \text{S1} \xrightarrow{\text{Permit}} \text{S3}, \text{S2} \xrightarrow{\text{Contract}} \text{S3}, \\ \text{S3} \xrightarrow{\text{HouseBuilt}} \text{S4}, \text{S4} \xrightarrow{\text{house}} \text{Finish}, \text{S4} \xrightarrow{\neg\text{Money}} \text{Finish} \end{array} \right\}$

分解の性質

- 分割は動作の正しい実装でなければならない
 - 動作 a の前提条件が与えられ a の効果達成する問題に対して、計画 d が完備で一貫性のある半順序計画であれば、計画 d は動作 a を正しく実装している
- 分解は一意的である必要はない
- 情報隠蔽を行う:
 - より高位での動作に対するSTRIPSの動作記述は前提条件と効果を隠蔽する
 - 分解での内部的な効果を全て無視する
 - 前提条件と効果が使われていない間は、活動内部での間隔を明確にしない
- 情報隠蔽は階層的ネットワーク計画立案では本質的である

半順序計画立案の要約 (1)

- 命題論理での計画立案問題を仮定する:
 - 初期計画は $Start$ と $Finish$ を有し、順序付けの制約は $Start < Finish$ で、因果リンクはなく、 $Finish$ でのすべての前提条件は未解決である。
 - 継続関数:
 - 動作 B についての未解決の前提条件 p を一つ取り出す
 - p を成し遂げる動作 A を一貫性が保たれるように選んで、継続の計画を生成する
 - ゴールのテストをする

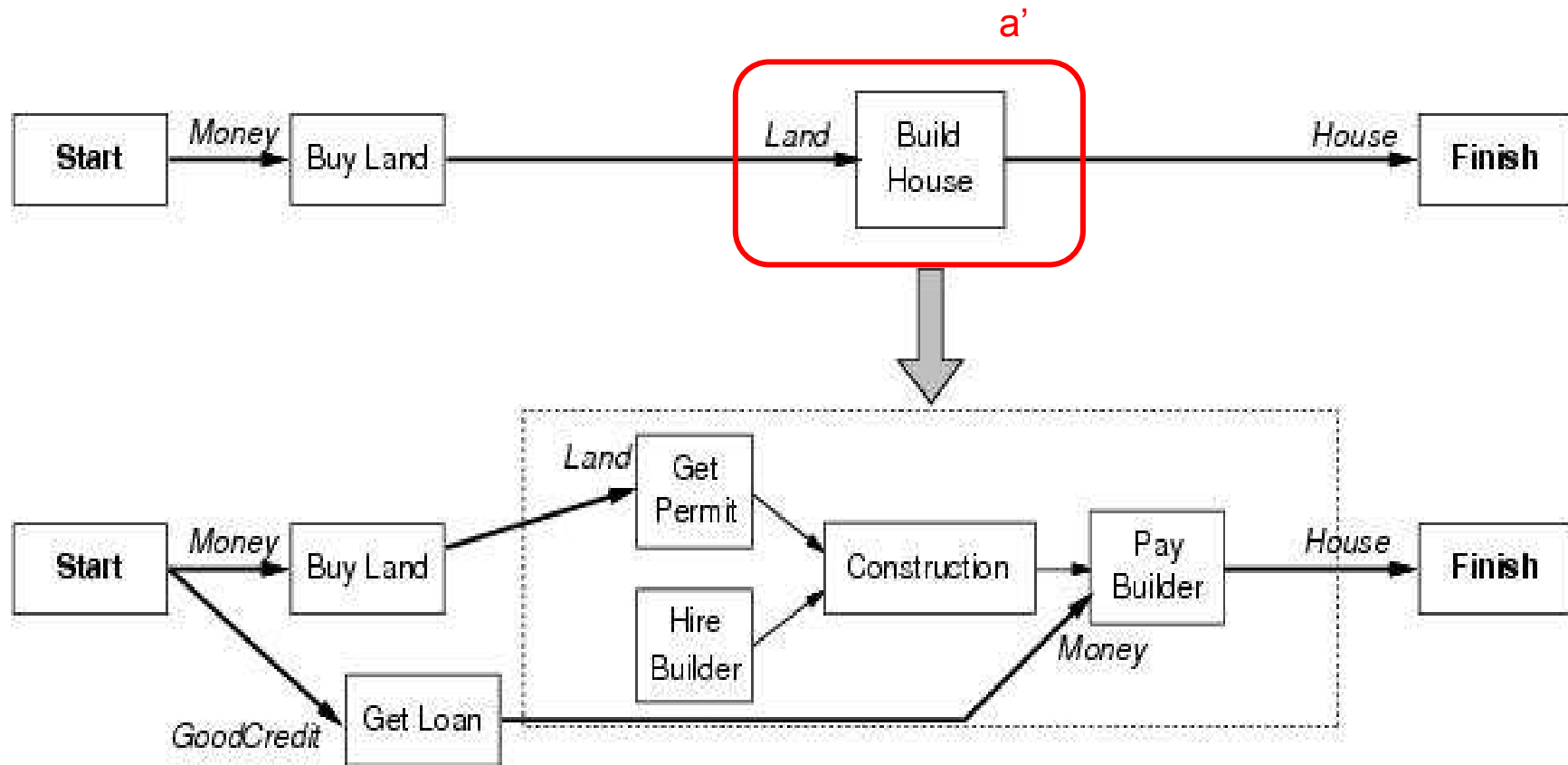
半順序計画立案の要約(2)

- 継続計画を生成するとき:
 - 因果リンク $A \rightarrow B$ と順序付けられた制約 $A < B$ を計画に加える
 - A が新しければ $start < A$ と $A < B$ を計画に加える
 - 新しい因果リンクとすでに存在する動作との間の衝突を解決する
 - 動作 A (新しければ)とすでに存在する因果リンクの間に横たわる衝突を解決する

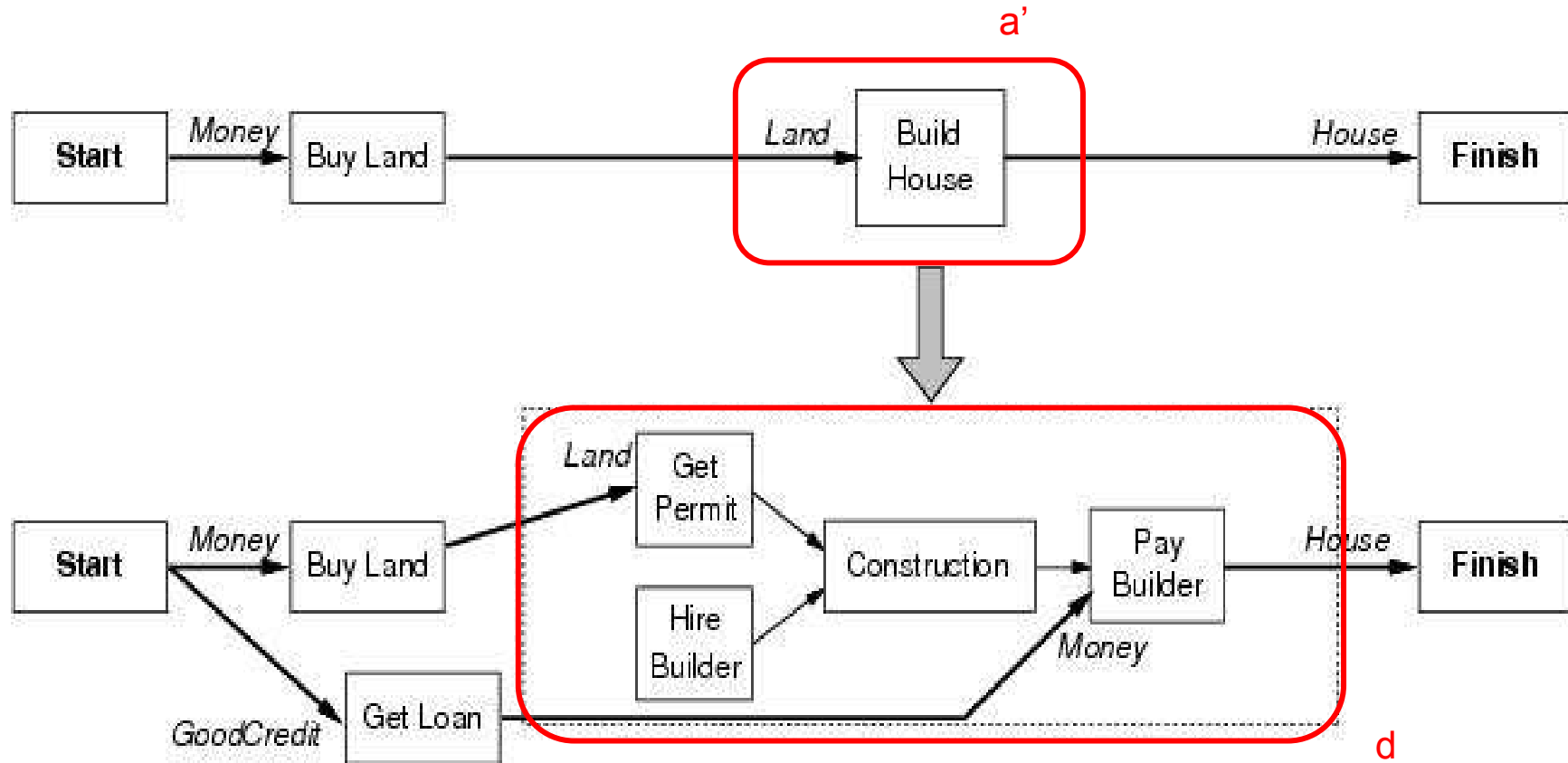
階層的ネットワーク計画立案に半順序計画立案を採用する

- 半順序計画立案に
 - 継続関数を修正する: 現在の計画に分解を適用する
- 新しい継続関数:
 - P から原始的でない動作 a' を選ぶ
 - θ で a と a' を単一化できるライブラリーでの $Decompose(a', d')$ の方法に対して、
 - a' を d' で置換する = $subst(\theta, d)$

POP+HTN example



POP+HTN example



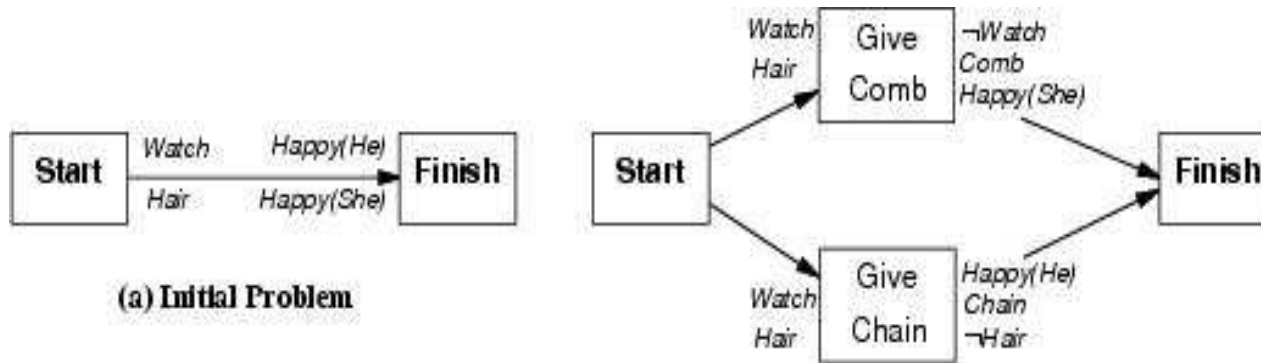
a' の中で d をいかに取り付けるか

- 動作 a' を P から除き、 $d\theta$ で置き換える
 - d' のそれぞれのステップ s に対して、 s (新しい s か P の既存の s' と単一化した s)の役割をする動作を選ぶ
 - サブタスクの共有も可能
- a' の順序付けされたステップを d' のステップに接続する
 - 全ての制約はそのままにする。従って、形式 $B < a'$ はそのまま維持される
 - あまりに厳格な順序付けには注意をする
- 因果リンクを接続する
 - $B -p-> a'$ が P での因果リンクならば、開始ステップで供給された前提条件 p を維持しながら、それを B から d' の全てのステップへの因果リンクの集合によって置き換える
 - 同様に $a' -p-> C$ に対しても

階層的ネットワークはどうか

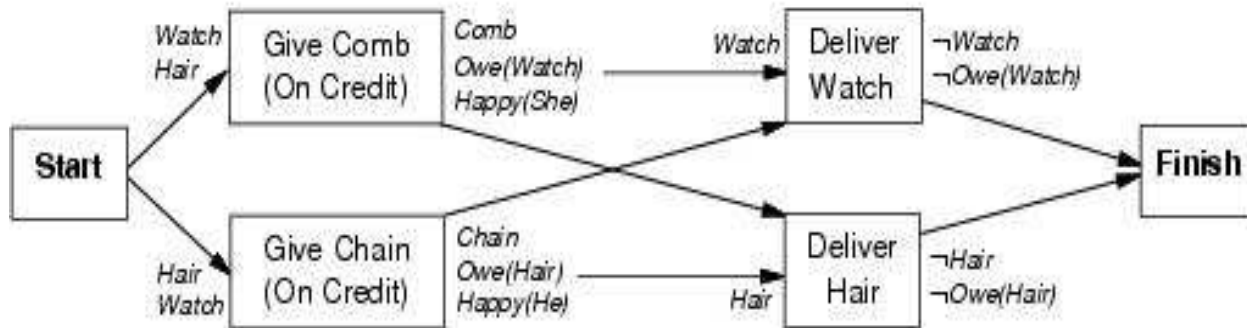
- 半順序計画立案への更なる修正が必要
- 悪いニュース: 純粋な階層的ネットワーク計画立案は再帰的分解動作のため非決定的である。
 - Walk=make one step and walk
- 次のように解決する
 - 再起を禁止にする
 - 関連する解の長さを制限する
 - 半順序計画立案でのハイブリッド化された階層的ネットワーク
- それでも階層的ネットワークは効率的である(see motivations in book)

The Gift of magi



(a) Initial Problem

(b) Abstract Inconsistent Plan



(c) Decomposition of (b) into a Consistent Solution

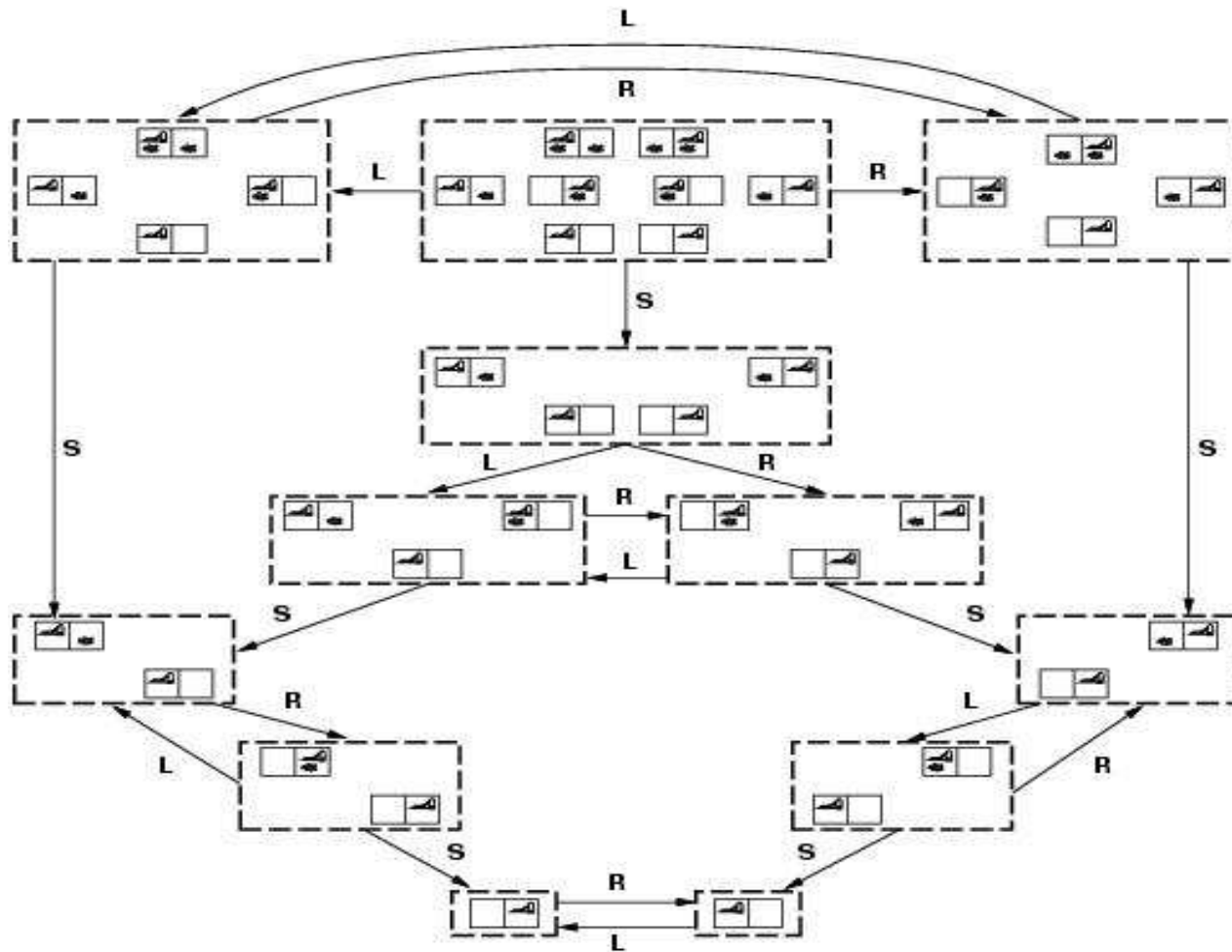
非決定論的領域

- ここまでは観察ができて、静的で決定的な領域
 - エージェントは最初に計画し、目を閉じて計画を実行できる
- 不確実な環境: 不完全(部分的に観察可能で非決定的)かつ不確か(世界とモデルの間の差)な情報
 - 知覚を使う
 - 必要なときに計画を採用する
- 不確実性によって決まる不確実さの程度
 - 有界: 動作は予測できない効果を持つが、動作の記述公理でそれをリストアップできる
 - 有界でない: 前提条件も効果も未知か数えられないほど大きい

不確定性を扱うには

- センサーレス計画立案 (整合計画立案)
 - 可能な状況の中でゴールを達成する計画を探す (初期状態と動作の効果によらず)
- 条件付計画立案(偶発計画立案)
 - 起こりうる偶発的な出来事に対して異なる分岐を有する条件付計画を構成する
- モニタリングの実行と再計画立案
 - 計画を立てているときに、計画が修正を必要とするかを判断する
- 連続した計画立案
 - 存続期間の間計画立案を生きたままに: 変化した環境に適応させるとともに、必要があればゴールを修正する

Sensorless planning



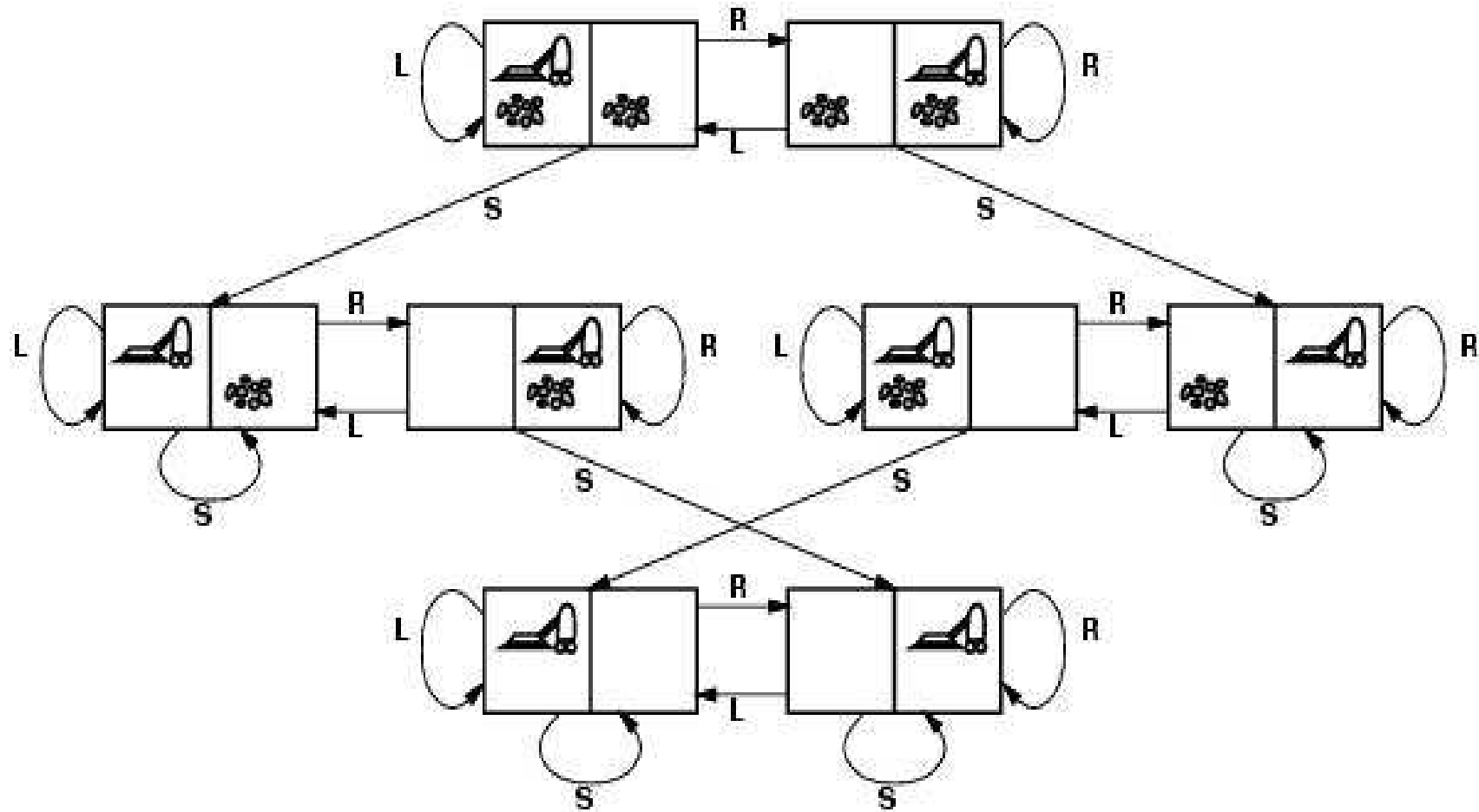
抽象的な例

- 初期状態 = <chair, table, cans of paint, unknown colors>, goal state = <color(table) = color(chair)>
- センサーレス計画立案 (整合計画立案)
 - 任意のペイントの缶を開けて椅子と机にそれを塗る
- 条件付計画立案(偶発計画立案)
 - 椅子と机の色を見て、同じであれば終了とし、そうでなければペイントのラベルを見て、color(label) = color(Furniture)であれば、そうでないほうにその色を塗り、そうでなければ両方に適応する
- モニタリングの実行と再計画立案
 - 条件付と同じでかつ間違いを修正できる (missed spots)
- 連続した計画立案
 - 机と椅子にペイントを塗る前に先に食事を取りたいのであれば、ゴールを修正できる

条件付計画立案

- 実際に何が起きているかを検査して不確かさを扱う
- 完全に観察できて非決定的な環境を使うことで:
 - 動作の結果は不明
 - 条件付ステップで環境の状態を検査する
 - どのように条件付計画を構成するか

Example, the vacuum-world



条件付計画立案

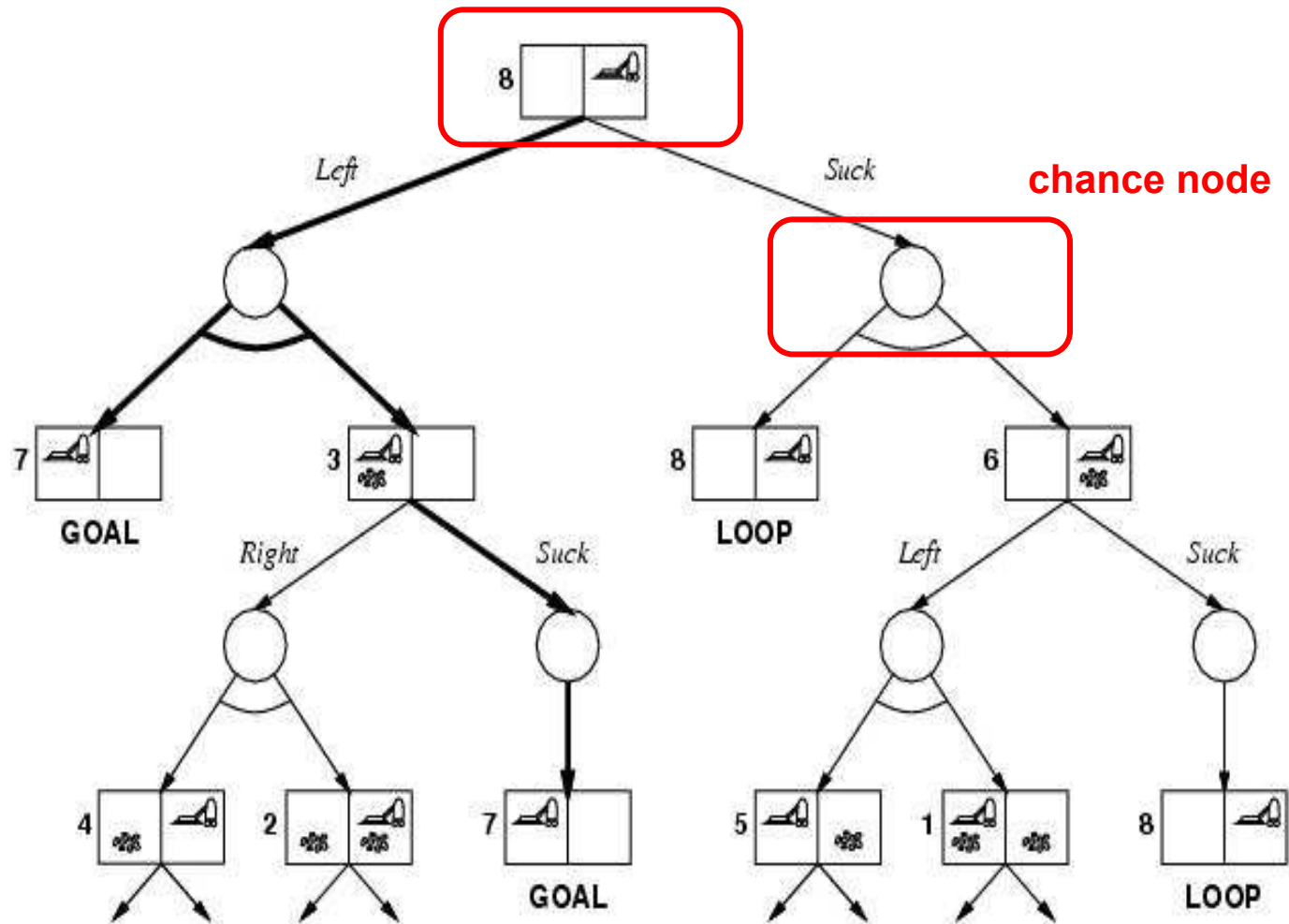
- 動作: left, right, suck
- 状態を決める命題: AtL, AtR, CleanL, CleanR
- 非決定性をどのように含めるか
 - 動作は一つ以上の効果を持つ
 - 例: 左への移動はしばしば失敗する
Action(Left, PRECOND: AtR, EFFECT: AtL)
Becomes : Action(Left, PRECOND: AtR, EFFECT: AtL \vee AtR)
 - 動作は条件付の結果をもたらす
Action(Left, PRECOND:AtR, EFFECT: AtL \vee (AtL \wedge **when**
cleanL: \neg cleanL)
両方とも選言的で条件付き

条件付計画立案

- 条件付計画は条件付のステップを必要とする:
 - If *<test>* then *plan_A* else *plan_B*
if $AtL \wedge CleanL$ then *Right* else *Suck*
 - 計画は真となる
- 自然に対するゲーム:
 - どちらの結果になったとしても、それとは無関係に機能する条件付計画を見つける
 - バキュームの世界を仮定する
初期状態 = $AtR \wedge CleanL \wedge CleanR$
二重のマーヒュー: 他の四角に行ったとき埃を落とす可能性があるし、動作が吸い取るであっても埃を落とす可能性がある

Game tree

State node



自然に対するゲームの解

- 解は部分木で
 - 全ての葉はゴールのノードを持つ
 - 状態の各ノードは動作を表す
 - 機会の各ノードは結果の分岐を持っている
- 前の例で:
 - [*Left*, if *AtL* \wedge *CleanL* \wedge *CleanR* then [] else *Suck*]
- 解を求めるには: 次の2点を修正したminimax アルゴリズムを用いる:
 - MaxとMinのノードはORとANDのノードに
 - アルゴリズムは単一に決まる移動ではなく条件付計画を返す

And-Or-search algorithm

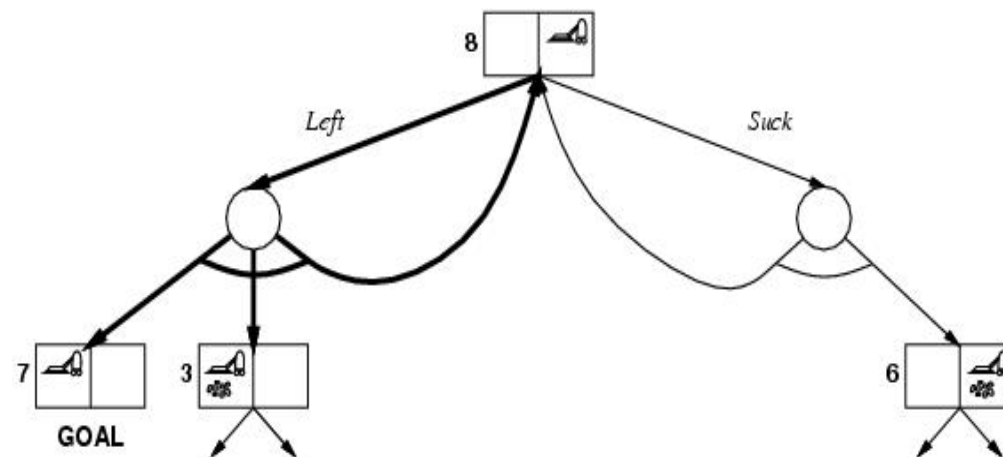
function AND-OR-GRAPH-SEARCH(*problem*) **returns** *a conditional plan or failure*
return OR-SEARCH(INITIAL-STATE[*problem*], *problem*, [])

function OR-SEARCH(*state*, *problem*, *path*) **returns** *a conditional plan or failure*
if GOAL-TEST[*problem*](*state*) **then return** the empty plan
if *state* is on *path* **then return** failure
for *action*, *state_set* in SUCCESSORS[*problem*](*state*) **do**
 plan ← AND-SEARCH(*state_set*, *problem*, [*state* | *plan*])
 if *plan* ≠ failure **then return** [*action* | *plan*]
return failure

function AND-SEARCH(*state_set*, *problem*, *path*) **returns** *a conditional plan or failure*
for each s_i in *state_set* **do**
 plan_i ← OR-SEARCH(s_i , *problem*, *path*)
 if *plan* = failure **then return** failure
return [**if** s_1 **then** *plan₁* **else if** s_2 **then** *plan₂* **else ... if** s_{n-1} **then** *plan_{n-1}* **else** *plan_n*]

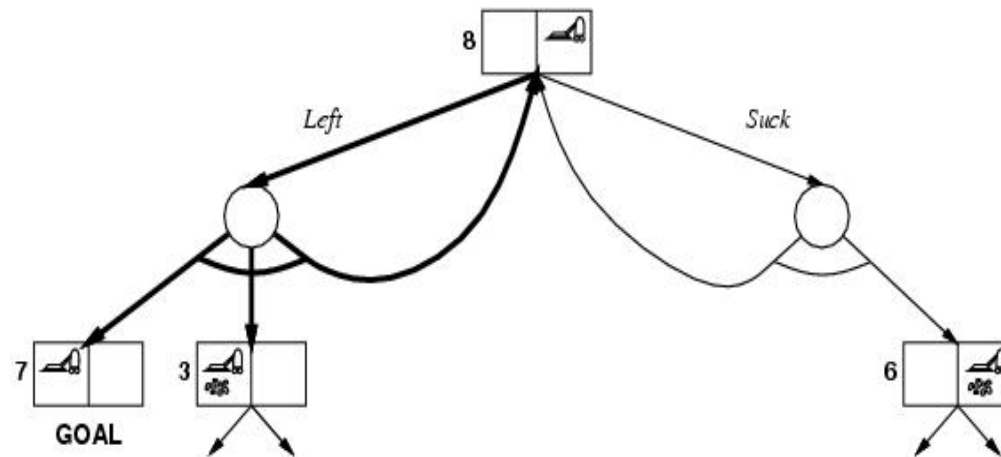
And-Or探索アルゴリズム

- サイクルはどのように扱うか
 - パス上に既に存在する状態が現れたときは、偽を返す
 - サイクルとなる解の禁止
 - アルゴリズムの終了を保証(サイクルにならないので)
 - アルゴリズムはルートからの別のパス上にその状態があるかどうかを調べない



And-Or探索アルゴリズム

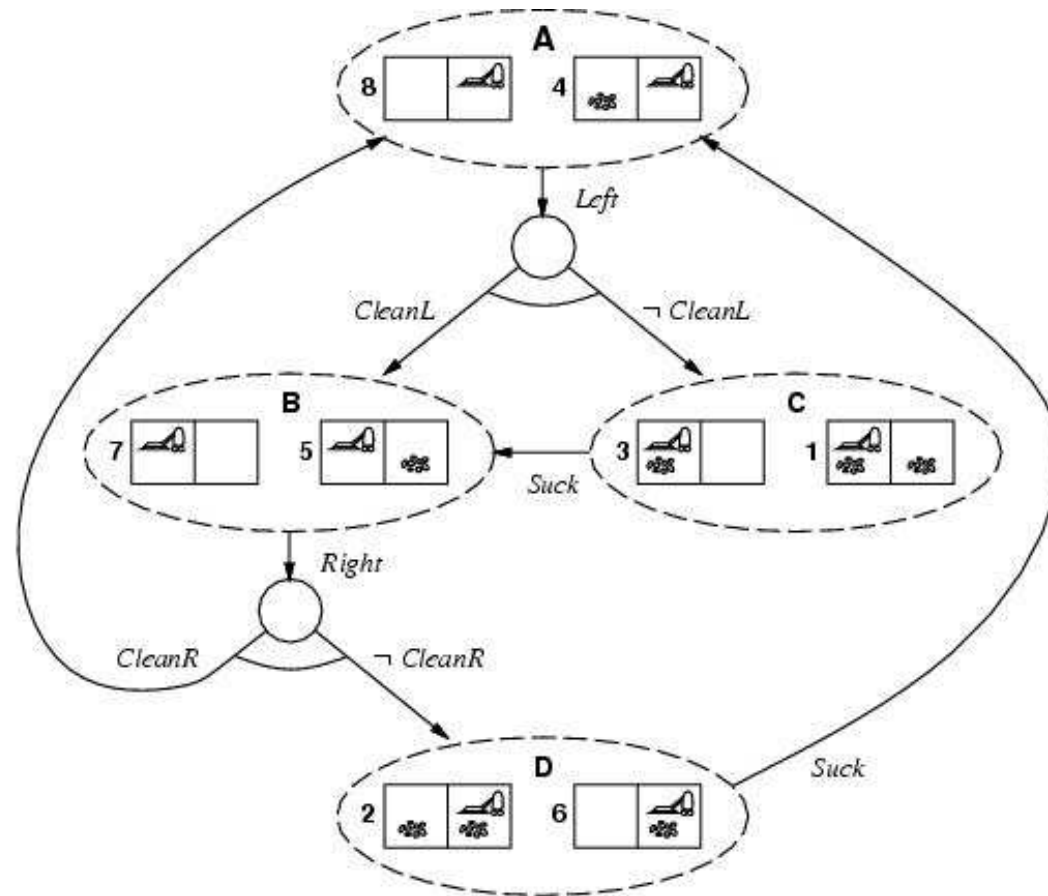
- 時々、サイクルとなる解だけが存在する
 - 三重のマーヒュー: 時々、移動は達成されない
[Left, if CleanL then [] else Suck] is not a solution
 - 計画の繰り返しにラベルを使う
[L1: Left, if AtR then L1 else if CleanL then [] else Suck]



条件付計画と部分的に観測可能な空間

- 完全に観測可能: 条件付テストはいかなる質問もでき、答えを得ることもできる
- 部分的に観測可能
 - エージェントは環境に関して制限された情報を有する
 - 次のようにモデル化する: 状態の集合 = 信念の状態の集まり
 - 例: バキューム・エージェントは一方の四角にいるときはそこに埃があることを知ることができるが、他方に埃があるかどうかは分からない
 - + 別のマーフィー: 他の四角に移ったとき埃を残すことがある
 - 完全に観察可能な世界として解を求める: 左右に動き続け、両方の四角がきれいになるまで埃を吸い続け、そして、左の四角にとどまる

PO: alternate double murphy



信念の状態

- 表現は？

- 完全な状態の記述

$$\{(AtR \wedge CleanR \wedge CleanL) \vee (AtR \wedge CleanR \wedge \neg CleanL)\}$$

- 信念の状態の中で可能な世界の集合を捕らえる論理的な文章 (開いた世界の仮定)

$$AtR \wedge CleanR$$

- エージェントの知識を記述した知識の命題論理 (閉じた世界の仮定)

$$K(AtR) \wedge K(CleanR)$$

信念の状態

- 選択の2と3は同じ (3で続ける)
- シンボルは3つの方法で表される: 肯定、否定、未知: n の命題シンボルに対して 3^n 個の可能な状態
 - それでも信念の状態は物理的な状態のパワー集合である。物理的な状態は 3^n を超える
 - 選択3は表現のために制限されている
 - 全ての可能な信念の状態を表現しようとするスキームでは、最悪の場合 $O(2^n)$ ビットを必要とする
 - 現在のスキームは $O(n)$ だけを要求する

条件付計画立案での検出

- どのように機能するか
 - 自動検出
 - 全てのステップでエージェントは利用可能な知覚を得る
 - 積極的な検出
 - 知覚は検出の動作を実行することで得られる
 - checkDirt* and *checkLocation*
- 表現と検出が与えられて、動作の記述を定式化できる

モニタリングと再計画立案

- モニタリングの実行: 全てが計画されたように実行されているかを調べる
 - 有界でない非決定性: ある予知されない環境が生じる
 - 現実の世界での必要性
- モニタリングの種類:
 - 動作のモニタリング: 次の動作が機能するかを見る
 - 計画のモニタリング: 残されている計画の全てをモニタリングする

モニタリングと再計画立案

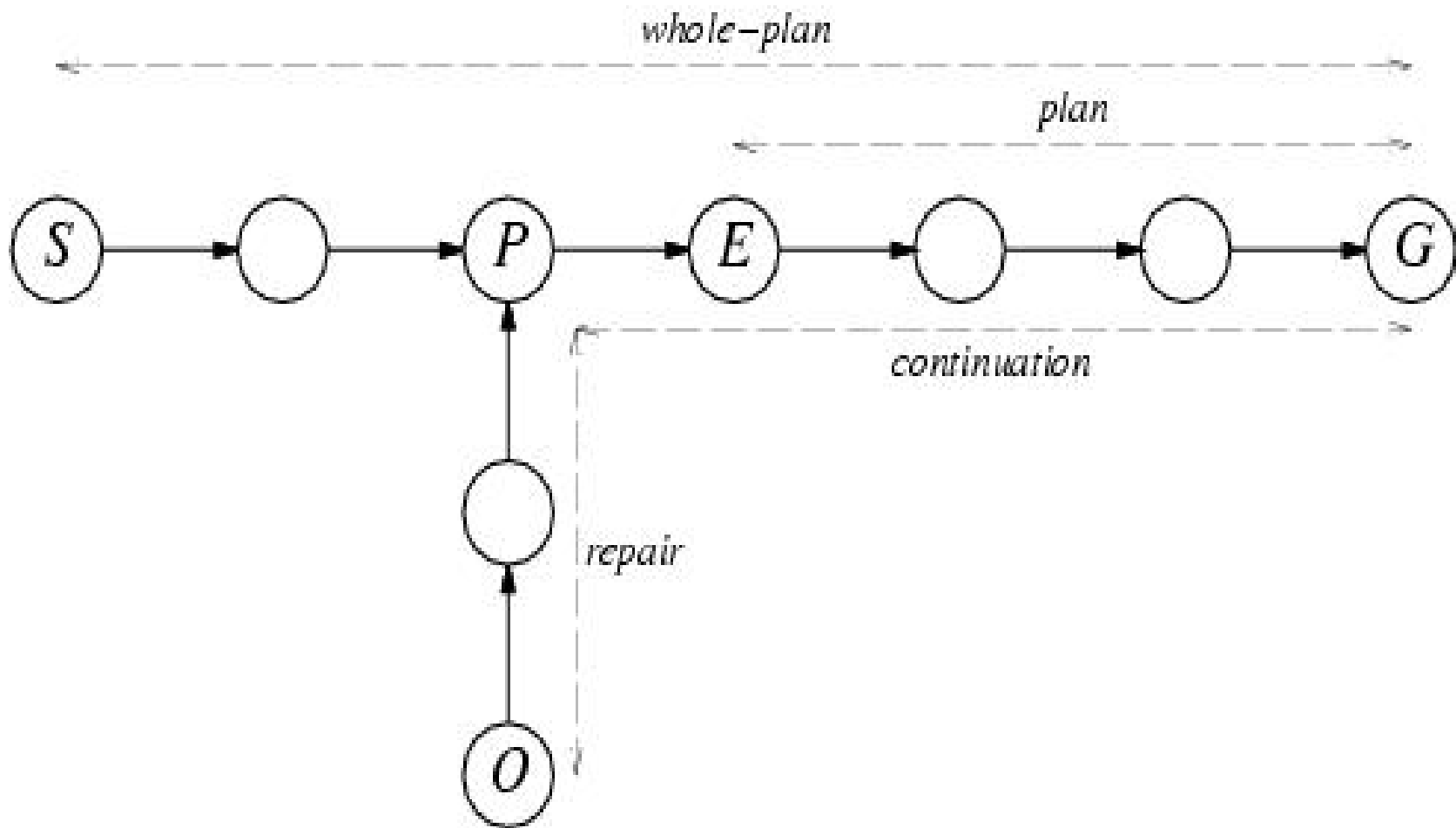
- 予期しないことが生じたとき: 再計画
 - 計画立案にたくさんの時間がとられることを避けるため、古い計画の修復を試みる。
- 完全にあるいは部分的に観察可能な環境に
そして、多様な計画立案の表現に、利用できる。

Replanning-agent

```
function REPLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (+ action descriptions)
           plan, a plan initially []
           whole_plan, a plan initially []
           goal, a goal
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
  current ← STATE-DESCRIPTION(KB,t)
  if plan = [] then return the empty plan
           whole_plan ← plan ← PLANNER(current, goal, KB)
  if PRECONDITIONS(FIRST(plan)) not currently true in KB then
           candidates ← SORT(whole_plan, ordered by distance to current)

           find state s in candidates such that
                   failure ≠ repair ← PLANNER(current, s, KB)
           continuation ← the tail of whole_plan starting at s
           whole_plan ← plan ← APPEND(repair, continuation)
  return POP(plan)
```

Repair example



Repair example: painting

Init(*Color*(*Chair*, *Blue*) \wedge *Color*(*Table*, *Green*) \wedge
 ContainsColor(*BC*, *Blue*) \wedge *PaintCan*(*BC*) \wedge
 ContainsColor(*RC*, *Red*) \wedge *PaintCan*(*RC*))

Goal(*Color*(*Chair*, *x*) \wedge *Color*(*Table*, *x*))

Action(*Paint*(*object*, *color*))

 PRECOND: *HavePaint*(*color*)

 EFFECT: *Color*(*object*, *color*)

Action(*Open*(*can*))

 PRECOND: *PaintCan*(*can*) \wedge *ContainsColor*(*can*, *color*)

 EFFECT: *HavePaint*(*color*)

[*Start*; *Open*(*BC*); *Paint*(*Table*, *Blue*), *Finish*]

修理の例: ペインティング

- 机と椅子の色が異なっていると、エージェントが知覚したと仮定する
 - 全計画の中で目標とする点を明確にする
現在の状態はペイント前の前提条件と同じである
 - そこにいたる動作の系列を修正する
Repair = [] and plan=[*Paint*, *Finish*]
 - 新しい計画を実行し続ける
机と椅子が同じ色であると知覚される前にループになりうる
- 動作のモニタリングはインテリジェントとはいえない行動をとることがある
 - 赤が選ばれ、机と椅子の両方を塗るのに十分でないとする
 - 動作のモニタリングでは片方を塗り終わるまで見つけられない
 - しかし、計画のモニタリングではこれは改良される

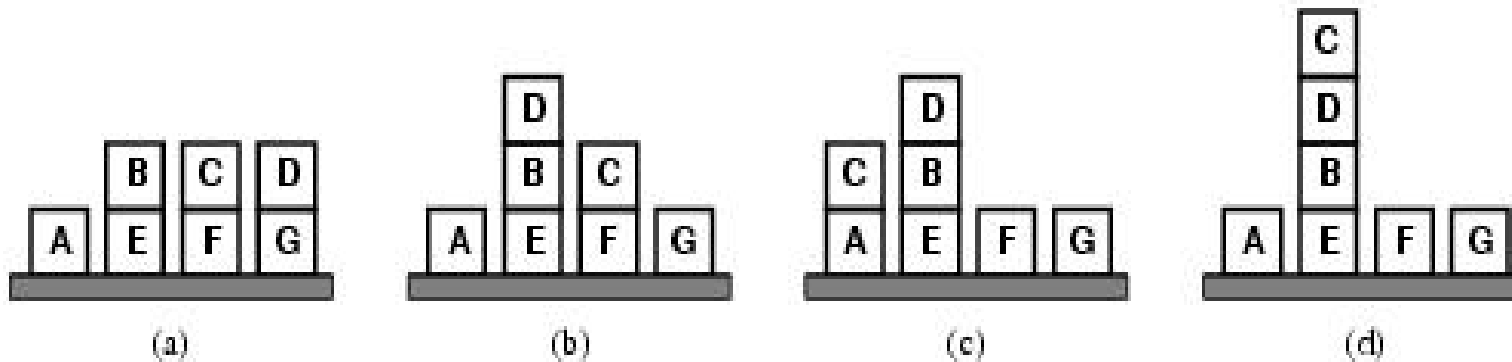
計画のモニタリング

- 全部の計画の成功のために前提条件を調べる
 - 計画の他のステップで達成されたものを除いて
 - 役立たなくなった計画の実行はできるだけ早く取りやめる
- 再計画立案エージェントの限界:
 - 新しいゴールを定式化できなかったり、現在のゴールに新しいゴールを加えられなかったりする

連続した計画立案

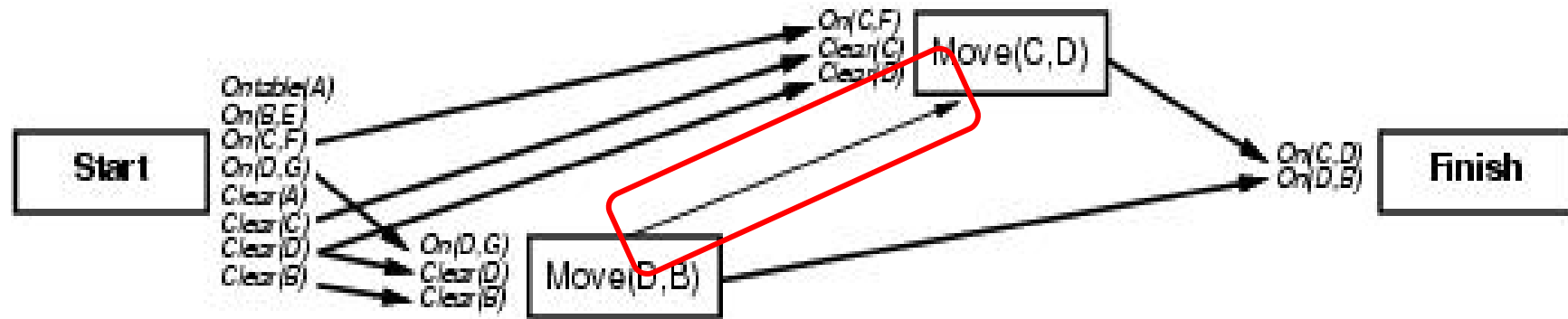
- エージェントは環境の中で無限にやり通す
 - ゴールの定式化、計画立案、動作のフェーズを
- モニタリングと計画の立案を連続するプロセスとして
- 例: ブロックの世界
 - 完全に観察可能な環境と仮定する
 - 半順序計画と仮定する

Block world example



- Initial state (a)
- $Action(Move(x, y))$,
PRECOND: $Clear(x) \wedge Clear(y) \wedge On(x, z)$
EFFECT: $On(x, y) \wedge Clear(z) \wedge \neg On(x, z) \wedge \neg Clear(y)$
- The agent first need to formulate a goal: $On(C, D) \wedge On(D, B)$
- Plan is created incrementally, return $NoOp$ and check percepts

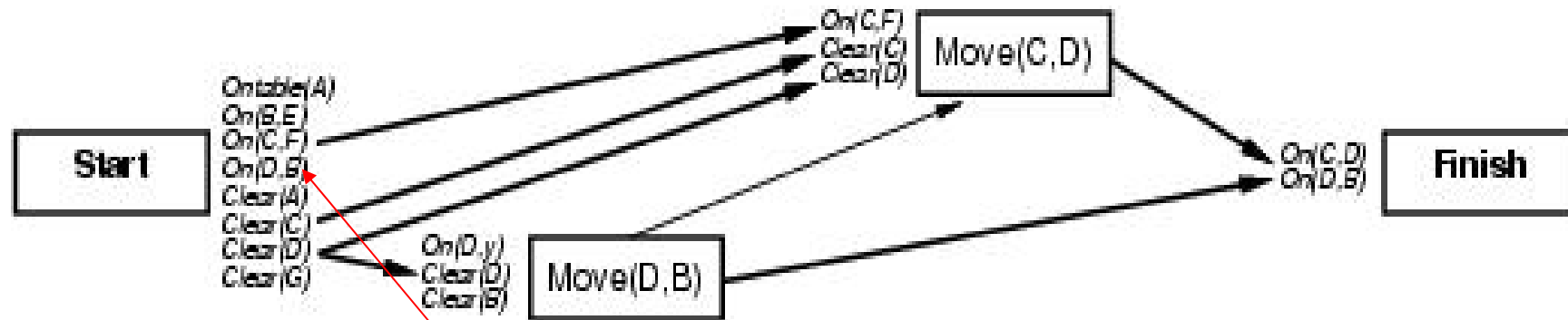
Block world example



- Assume that percepts don't change and this plan is constructed
- Ordering constraint between $Move(D,B)$ and $Move(C,D)$
- Start is label of current state during planning.
- Before the agent can execute the plan, nature intervenes:
D is moved onto B

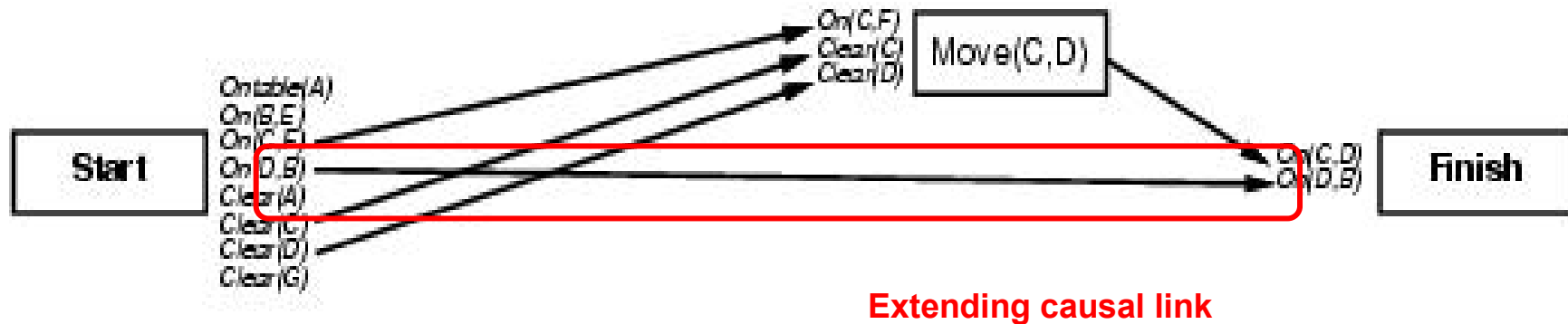
DをBの上に移した
とすると、Startは次
のように変わる

Block world example



- Start contains now $On(D,B)$
- Agent perceives: $Clear(B)$ and $On(D,G)$ are no longer true
 - Update model of current state (start)
- Causal links from *Start* to *Move(D,B)* ($Clear(B)$ and $On(D,G)$) no longer valid.
- Remove causal relations and two PRECOND of *Move(D,B)* are open
- Replace action and causal links to *Finish* by connecting *Start* to *Finish*.

Block world example



- *Extending*: whenever a causal link can be supplied by a previous step
- All redundant steps (*Move(D,B)* and its *causal links*) are removed from the plan
- Execute new plan, perform action *Move(C,D)*
 - *This removes the step from the plan*

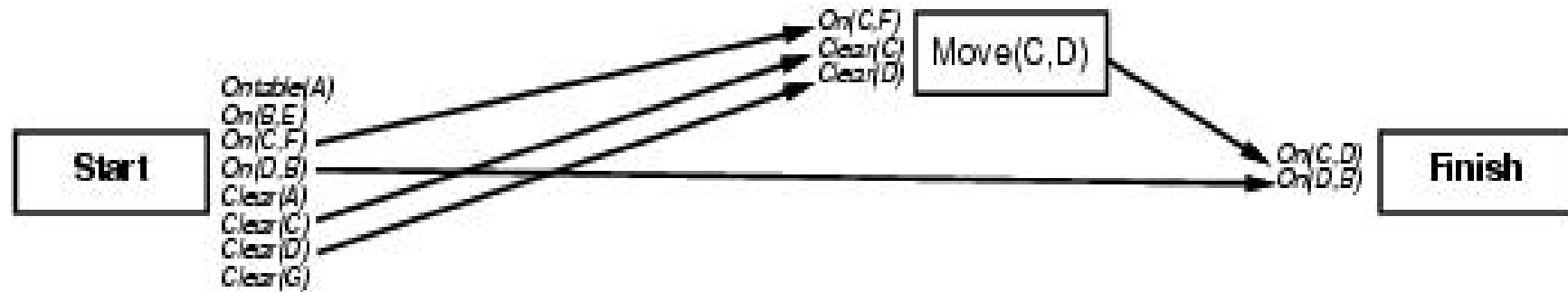
さらにCをDの上に移したとすると、Startは次のように変わる

Block world example



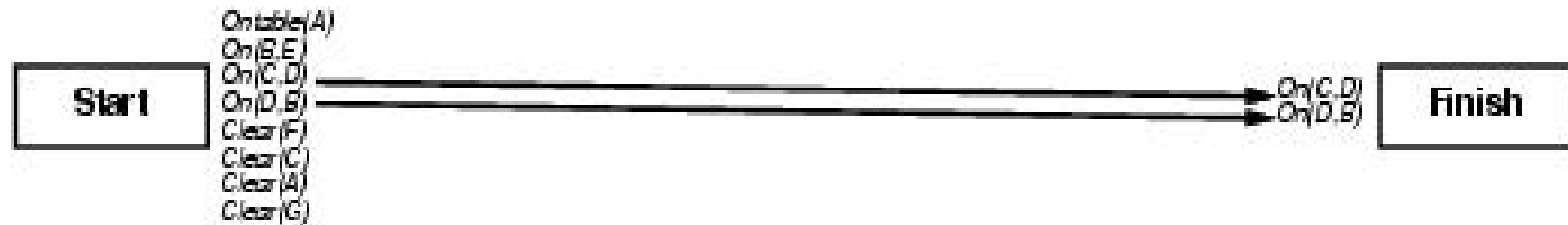
- Execute new plan, perform action *Move(C,D)*
 - Assume agent is clumsy and drops C on A
- No plan but still an open PRECOND
- Determine new plan for open condition

Block world example



- Again *Move(C,D)*

Block world example



- Similar to POP
- On each iteration find plan-flaw and fix it
- Possible flaws: Missing goal, Open precondition, Causal conflict, Unsupported link, Redundant action, Unexecuted action, unnecessary historical goal

多重エージェントの計画立案

- これまででは一人のエージェントの環境について論じてきた
- ほかにエージェントも世界のモデルに簡単に付け加えることができる
 - エージェントが他のエージェントの意図に無関心であったりするため性能が出ない
- 多重エージェントの環境には二つのタイプ:
 - 協力的
 - 競争的

協力的: 結合したゴールと計画

- 多重計画立案の問題: テニスのダブルスでエージェントがボールを返そうとする場面

Agents(A,B)

Init(At(A,[Left,Baseline]) \wedge At(B,[Right, Net]) \wedge Approaching(Ball,[Right, Baseline]) \wedge Partner(A,B) \wedge Partner(B,A))

Goal(Returned(Ball) \wedge At(agent,[x,Net]))

Action(Hit(agent, Ball))

PRECOND: Approaching(Ball,[x,y]) \wedge At(agent,[x,y]) \wedge Partner(agent, partner) \wedge \neg At(partner,[x,y])

EFFECT: Returned(Ball)

Action(Go(agent,[x,y]))

PRECOND: At(agent,[a,b])

EFFECT: At(agent,[x,y]) \wedge \neg At(agent,[a,b])

協力的: 結合したゴールと計画

- 解は両方のエージェントがとる動作が結合した計画となる
- Example:
 - A: [Go(A,[Right, Baseline]), Hit(A,Ball)]
 - B: [NoOp(B), NoOp(B)]Or
 - A: [Go(A,[Left, net), NoOp(A)]
 - B: [Go(B,[Right, Baseline]), Hit(B, Ball)]
- 同じ結合した計画に辿りつくためには *協調*が必要になる

多体計画立案

- 一つのエージェントのための計画立案だが、このエージェントはいくつかの物理的実体に動作を指示する。
- 真の意味では多重エージェントではない
- 重要性: 動作の同期化
 - 簡単化するために、全ての動作は1タイムステップ必要とし、結合した計画のそれぞれの場所で動作は同時に行われる
 - [<Go(A,[Left,Net]), Go(B,[Right,Baseline])>;
<NoOp(A), Hit(B, Ball)>]
 - 計画立案は半順序計画で行われ、それは可能な結合した動作の集合に対して適応される
 - この集合の大きさは

多体計画立案

- 結合した動作の集合に対する代替: 動作の記述に同時の記述を加える
 - Concurrent action
 - Action(Hit(A, Ball)*
CONCURRENT: $\neg Hit(B, Ball)$
PRECOND: $Approaching(Ball, [x, y]) \wedge At(A, [x, y])$
EFFECT: *Returned(Ball)*
 - Required actions (carrying object by two agents)
 - Action(Carry(A, cooler, here, there)*
CONCURRENT: *Carry(B, cooler, here there)*
PRECOND: ...)
- 計画立案者は、半順序計画を用いることができるが順序付けの関係で少し修正が必要

協調の機構

- 結合計画での意見の一致を確保するために: *因習*を用いる
 - 因習 = 結合した計画を選ぶ上での制約 (エージェントがそれを採用したら、結合した計画は機能しなければならないという制約を超えたものである).
e.g. stick to your court or one player stays at the net.
- ある因習は広く受け入れられている = 交通ルールや人間が使う言語
- 領域に従属かそれとも独立である
- 進化的プロセスを通して生じる (群れの行動)

群れの例

- 三つの規則:
 - 分離:
あまりにも近づき過ぎたときは、近所から離れるように舵を取る
 - 団結
近所との平均的距離となるように舵を取る
 - 整列
近所の平均的な進行方向になるように舵を取る
- 群れは擬似的な固まった塊としての創発的な飛行行動を見せる

協調の機構

- 因習がないとき: コミュニケーション
e.g. Mine! Or Yours! in tennis example
- 成功に導く結合した計画に辿りつくためにかかる負担は
 - エージェントの設計者(エージェントの計画が今いる環境の中で機能するならば、エージェントはうまく対応できる。そして、他のエージェントの明確なモデルを持つ必要はない)
 - エージェント(エージェントは審議型で他のエージェントの推論は配慮して、自身の行動が有効であることを示す必要がある)

競争環境

- エージェント同士は衝突する用役を持つ
 - e.g. zero-sum games like chess
- エージェントは:
 - 他のエージェントがいることを認識する必要がある
 - 他のエージェントの計画を計算しなければならない
 - 他のエージェントが自分の計画とどのようにかかわるか計算しなければならない
 - この相互作用を配慮した上で最善の動作を決定しなければならない
- 他のエージェントのモデルが必要である
- しかし、結合した動作の計画には付託しない