Planejamento
em Inteligência Artificial

Capítulo 9
Heurísticas em Planejamento

Leliane Nunes de Barros

MAC5788
IME-USP 2005

Planejamento como Busca Não-determinística: uma visão conceitual

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)              ;;  refinement step
    B ← Branch(u)              ;;  branching step
    B' ← Prune(B)              ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

O nó u representa um conjunto de planos soluções Π_u : conjunto de todas as soluções alcançáveis de u.

Planejadores como instâncias de *Abstract-search* (I)

- **Planejamento no espaço de estados**: u é uma seqüência de ações. Toda solução alcançável apartir de u contém essa seqüência como prefixo ou sufixo.
- **Planejamento no espaço de planos**: u é um conjunto de ações com vínculos causais, restrições de ordenação e restrições de unificações. Toda solução alcançável apartir de u contém todas as ações e satisfazem as restrições
- **Algoritmo Graphplan**: u é um sub-grafo de um grafo de planejamento, isto é, uma seqüência de conjuntos de ações junto com restrições de pré-condições, efeitos e exclusões mútuas. Toda solução alcançável apartir de u contém as ações em u correspondente aos níveis já resolvidos (da busca no grafo) e pelo menos uma ação de cada nível qua ainda não foi resolvido em u.

Planejadores como instâncias de *Abstract-search* (II)

- **Planejamento basead em SAT:** *u* é um conjunto de literais valorados e o restante das cláusulas, cada uma sendo uma disjunção de literais que descrevem ações e estados. Toda solução alcançável apartir de *u* corresponde a uma atribuição de valores verdade ou falso aos literais não valorados tal que todas as cláusulas restantes sejam satisfeitas.
- **Planejamento baseado em CSP**: *u* é um conjunto de variáveis CSP e restrições com algumas variáveis com valores já atribuidos. Toda solução alcançável apartir de *u* inclue essas variáveis valoradas e atribuições para as outras variáveis CSP que satisfazerm as restrições.

Planejadores como instâncias de *Abstract-search* (III)

- **Planejamento no espaço de estados:** u é um plano parcial
- **Planejamento no espaço de planos:** u é um plano parcial

- **Algoritmo Graphplan**: nem todas as ações em u farão parte do plano solução
- **Planejamento basead em SAT:** nem todas as ações em u farão parte do plano solução
- **Planejamento baseado em CSP:** nem todas as ações em u farão parte do plano solução

Planejadores como instâncias de *Abstract-search* (III)

- **Planejamento no espaço de estados:** *u* é um plano parcial
- **Planejamento no espaço de planos:** *u* é um plano parcial

Disjunctive refinement approaches:

- **Algoritmo Graphplan**: nem todas as ações em *u* farão parte do plano solução
- **Planejamento basead em SAT:** nem todas as ações em *u* farão parte do plano solução
- **Planejamento baseado em CSP:** nem todas as ações em *u* farão parte do plano solução

Planejadores como instâncias de *Abstract-search* (IV)

- ***Refine***: modifica a coleção de ações e/ou restrições associadas com um nó u
- ***Branch***: gera um ou mais filhos de u que serão os nós candidatos para ser o próximo nó visitado. Cada filho v representa um sub-conjunto de soluções $\Pi_v \subseteq \Pi_u$
- ***Prune***: remove do conjunto de nós candidatos $\{u_1, u_2, \ldots, u_k\}$ alguns nós que parecem ser não promissores para a busca (por exemplo, um nó já visitado que não levou a solução)

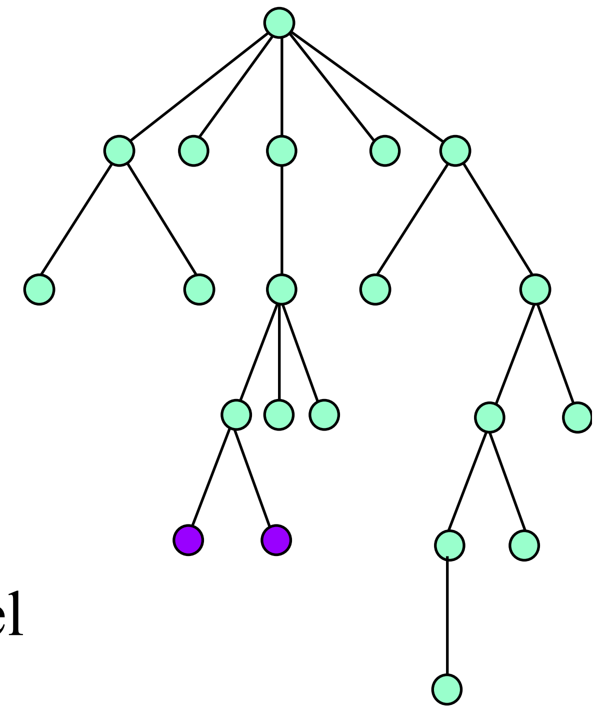
- Planejadores podem:
  - » Variar a ordem em que esses 3 procedimentos são executados
  - » Usar diferentes mecanismos de controle, como IDS ou A*
- Como os planejadores que estudamos realizam esses procedimentos (páginas 195 à 197)

Tornando *Abstract-search* Determinística

```
Depth-first-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)              ;;  refinement step
    B ← Branch(u)              ;;  branching step
    C ← Prune(B)               ;;  pruning step
    while C ≠ ∅ do
        v ← Select(C)          ;;  node-selection step
        C ← C − {v}
        π ← Depth-first-search(v)
        if π ≠ failure then return(π)
    return(failure)
end
```

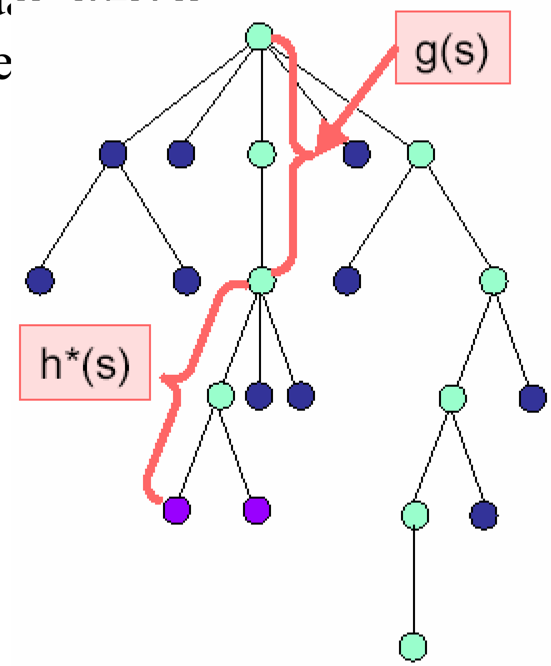
Heurística de seleção de nós: resolve o problema da escolha não-determinística

- Suponha que para cada estado *s*, temos uma estimativa *h(s)* da distância de *s* para uma solução
  - *h(s)* é usada para decidir qual será o próximo nó a ser expandido
  - quanto mais informativa for *h(s)* menor será o número de escolhas erradas
  - *h(s)* deve ainda ser facilmente computável
    - » Heurísticas são geralmente baseadas mo princípio de relaxação de problemas: para avaliar o quão desejável é um nó consideramos um problema mais simples obtido apartir do original fazendo suposições de simplicação e relaxando restrições

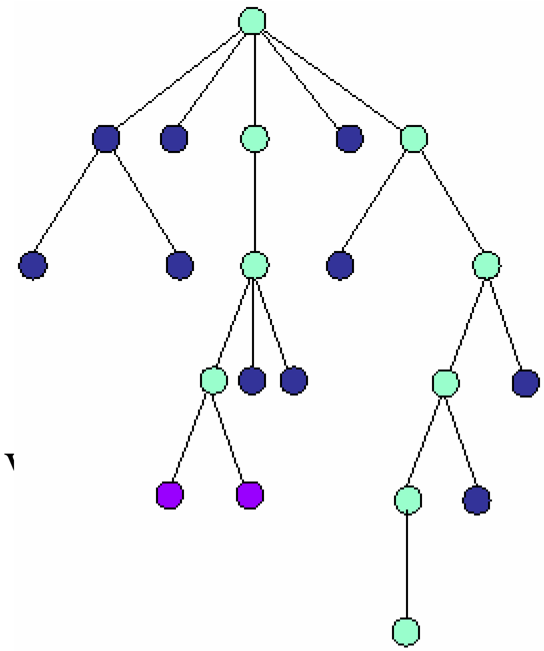


Heurística de seleção de nós

- Suponha que estamos fazendo uma busca em uma árvore na qual cada aresta (s,s') tem um custo $c(s,s')$
  - Se p é um caminho, seja $c(p)$ = soma dos custos das arestas
  - No planejamento clássico isso é o comprimento de
- Para cada estado s, seja
  - $g(s)$ = custo do caminho de s_0 à s
  - $h^*(s)$ = o menor custo de todos os caminhos de s aos nós meta
  - $f^*(s) = g(s) + h^*(s)$ = o menor custo de todos os caminhos
  - de $s0$ aos nós metaque passam por s
- Seja $h(s)$ a função que estimate $h^*(s)$
  - Let $f(s) = g(s) + h(s)$
    - » $f(s)$ is uma estimativa de $f^*(s)$
  - h é *admissível* se para todo estado state s, $0 \leq h(s) \leq h^*(s)$
  - Se h é admissível então f é um limitante inferior de f^*



O algorítmo A*



- A* em árvores:

  laço

  Escolha um nó folha s tal que $f(s)$ seja o menor

  se s é uma solução então *return* e *exit*

  Expanda s (isto é, gere os filhos de s)

- Em grafos, A* é mais complicado: é preciso mecanismos adicionais para lidar com múltiplos caminhos para chegar num mesmo nós
- Se existir uma solução (e algumas outras condições forem satisfeitas), then:
  - Se $h(s)$ é admissível, então A* garante encontrar uma solução ótima
  - Quanto mais "informativa" for a heurística (isto é, o quão próxima ela for de h^*), menor será o número de nós expandidos por A*
  - Se $h(s)$ for admissível a menos de c, então A* garante encontrar uma solução ótima a menos da constante c

Leliane Nunes de Barro. Slides extraidos de Lecture slides for *Automated Planning by Dana Nau.*

Família de Funções Herísticas para Planejamento

- $\Delta^*(s,p)$: distância mínima do estado s ao estado que contém p
- $\Delta^*(s,s')$: distância mínima do estado s a todas as prosições de em s'
- $\Delta_i(s,p)$ e $\Delta_i(s,s')$, onde i = 0, 1, 2, …
  - estimativa de $\Delta^*(s,p)$ e $\Delta^*(s,s')$

Δ_0 ignora os efeitos negativos das ações

$$
\begin{aligned}
&\Delta_0(s,p) = 0 && \text{if } p \in s, \quad \text{e } p \notin s,\\
&\Delta_0(s,p) = \infty && \text{if } \forall a \in A, p \notin \text{effects}^+(a),\\
&\Delta_0(s,g) = 0 && \text{if } g \subseteq s,\\
&\text{otherwise:}\\
&\Delta_0(s,p) = \min_a\{1 + \Delta_0(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\}\\
&\Delta_0(s,g) = \textstyle\sum_{p\in g} \Delta_0(s,p)
\end{aligned}
\tag{9.1}
$$

- $h(s) = \Delta_0(s,g)$, onde g é a meta dada por um conjunto de proposições

Heurística da relaxação de independência (polinomial no número de proposições e ações)

- Dado s, pode computar $\Delta_0(s,p)$, para cada proposição p:

```
Delta(s)
    for each p do: if p ∈ s then Δ0(s,p) ← 0, else Δ0(s,p) ← ∞
    U ← {s}
    iterate
        for each a such that ∃u ∈ U, precond(a) ⊆ u do
            U ← {u} ∪ effects+(a)
            for each p ∈ effects+(a) do
                Δ0(s,p) ← min{Δ0(s,p) , 1 + Σ_{q∈precond(a)} Δ0(s,q)}
    until no change occurs in the above updates
end
```

- A partir disso, computa $h(s) = \Delta_0(s,g) = \sum_{p \in g} \Delta_0(s,p)$

Heuristic Forward Search

```
Heuristic-forward-search(π, s, g, A)
    if s satisfies g then return π
    options ← {a ∈ A | a applicable to s}
    for each a ∈ options do Delta(γ(s, a))
    while options ≠ ∅ do
        a ← argmin{Δ0(γ(s, a), g) | a ∈ options}
        options ← options − {a}
        π' ← Heuristic-forward-search(π.a, γ(s, a), g, A)
        if π' ≠ failure then return(π')
    return(failure)
end
```

- Nota: essa é uma busca em profundidade e portanto admissibilidade é irrelevante

Heuristic Backward Search

Backward-search(π, s_0, g, A)
    if s_0 satisfies g then return(π)
    $options \leftarrow \{a \in A \mid a$ relevant for $g\}$
    while $options \neq \emptyset$ do
        $a \leftarrow \text{argmin}\{\Delta_0(s_0, \gamma^{-1}(g, a)) \mid a \in options\}$
        $options \leftarrow options - \{a\}$
        $\pi' \leftarrow$ Backward-search($a.\pi, s_0, \gamma^{-1}(g, a), A$)
        if $\pi' \neq$ failure then return(π')
    return failure
end

Uma herística simples

$$
\begin{aligned}
&\Delta_0(s,p) = 0 && \text{if } p \in s, \\
&\Delta_0(s,p) = \infty && \text{if } \forall a \in A, p \notin \text{effects}^+(a), \text{ and } p \notin s, \\
&\Delta_0(s,g) = 0 && \text{if } g \subseteq s, \\
&\text{otherwise:} \\
&\Delta_0(s,p) = \min_a\{1 + \Delta_0(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
&\Delta_0(s,g) = \textstyle\sum_{p \in g} \Delta_0(s,p)
\end{aligned}
\tag{9.1}
$$

- $h(s) = \Delta_0(s,g)$ é uma heurística não-admissível
- Δ_1: igual a Δ_0 exceto que $\Delta_1(s,g) = \max_{p \in g} \Delta_0(s,p)$
  - Essa heurística é admissível; portanto poderia ser usada com A*
  - Não é muito informativa

Uma Heurística mais informativa

- Ao invés de computar a distância máxima para cada p em g, compute a distância máxima de cada par {p,q} em g:

$$
\begin{aligned}
\Delta_2(s,p) &= \min_a\{1+\Delta_2(s,\text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
\Delta_2(s,\{p,q\}) &= \min\{ \\
&\quad \min_a\{1+\Delta_2(s,\text{precond}(a)) \mid \{p,q\} \subseteq \text{effects}^+(a)\} \\
&\quad \min_a\{1+\Delta_2(s,\{q\}\cup\text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
&\quad \min_a\{1+\Delta_2(s,\{p\}\cup\text{precond}(a)) \mid q \in \text{effects}^+(a)\}\} \\
\Delta_2(s,g) &= \max_{p,q}\{\Delta_2(s,\{p,q\}) \mid \{p,q\} \subseteq g\}
\end{aligned} \tag{9.3}
$$

Generalização

Recall that $\Delta^\star(s,g)$ is the true minimal distance from a state s to a goal g. $\Delta^\star$ can be computed (albeit at great computational cost) according to the following equations:

$$\Delta^\star(s,g) = \begin{cases} 0 & \text{if } g \subseteq s, \\ \infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \text{ and} \\ \min_a\{1 + \Delta^\star(s, \gamma^{-1}(g,a)) \mid a \text{ relevant for } g\} \\ & \text{otherwise.} \end{cases} \tag{9.4}$$

From $\Delta^\star$, let us define the following family Δ_k, for $k \geq 1$, of heuristic estimates:

$$\Delta_k(s,g) = \begin{cases} 0 & \text{if } g \subseteq s, \\ \infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \\ \min_a\{1 + \Delta^\star(s, \gamma^{-1}(g,a)) \mid a \text{ relevant for } g\} \\ & \text{if } |g| \leq k, \\ \max_{g'}\{\Delta_k(s,g') \mid g' \subseteq g \text{ and } |g'| = k\} \\ & \text{otherwise.} \end{cases} \tag{9.5}$$

Complexidade no cálculo da heurística

- Consome tempo $\Omega(n^k)$.
- Se $k = \max(|g|, \max\{|precond(a)| : a$ é uma ação$\})$ então computar $\Delta(s,g)$ é tão difícil quanto resolver o problema inteiro de planejamento

Extração de heurísticas a partir do grafo de planejamento

- O grafo de planejamento, além de fornecer uma estimativa da distância a partir de s_0 para atingir cada proposição alcançável *p* , também fornece informações de *mutex* (μP_i)
  - O procedimento *Solution extraction* seleciona um conjunto de proposições *g* em uma camada somente se nenhum par de elementos em *g* for um *mutex*. Teorema: se um par de elementos não estiver em *mutex* em uma camada, então eles continuam sendo *nonmutex* nas camadas seguintes ➔ isso é muito parecido com to Δ_2(s,g)

Extração de heurísticas a partir do grafo de planejamento

- Lembremos como GraphPlan trabalha:

loop

Graph expansion:

this takes polynomial time

extend a “plan[illegible] from the initial state
until we have achieved a necessary (but insufficient) condition for plan existence

this takes exponential time

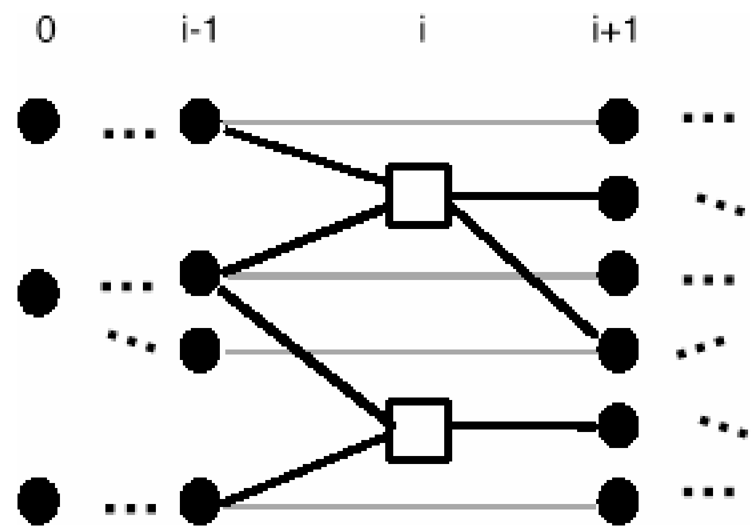
Solution extraction:

search backward from the goal, looking for a correct plan

if we find one, then return it

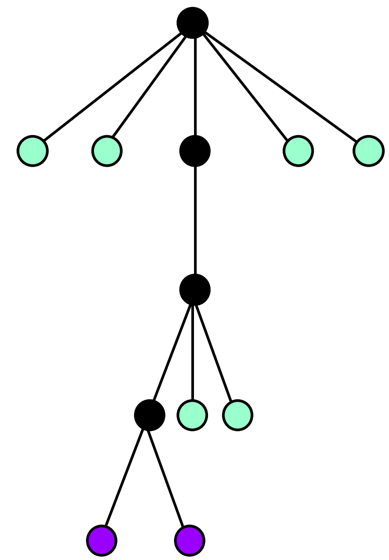
repeat

Usando o Grafo de Planejamento para calcular *h(s)*



- No grafo, há camadas alternantes de literais instanciados e ações
- O número de camadas de ações é um *lower bound* sobre o número de ações no plano
- Construa um grafo de planejamento, começando em *s*
- $\Delta^g(s,p)$ = é o nível da primeira camada que "possivelmente atinge" p
- $\Delta^g(s,g)$ é muito parecida com $\Delta_2(s,g)$
  - $\Delta_2(s,g)$ conta cada ação individualmente
  - $\Delta^g(s,g)$ agrupa todas as ações independentes
  - em uma camada

The FastForward Planner (FF)



- Usa uma heurística similar a $h(s) = \Delta^g(s,g)$
- Busca A* é ruim: consome muita memória
- Melhor usar busca gulosa:

  until we have a solution, do

  expand the current state *s*

  s := the child of *s* for which *h(s)* is smallest

  (i.e., the child we think is closest to a solution)

- Existem várias maneiras na literatura de melhorias para o FF
- FF não pode garantir quanto tempo leva para encontrar um a solução ou quão ótima ela será (busca local)
  - No entanto, FF apresenta um bom desempenho em muitos domínios e problemas

FF na competição de planejamento AIPS-2000

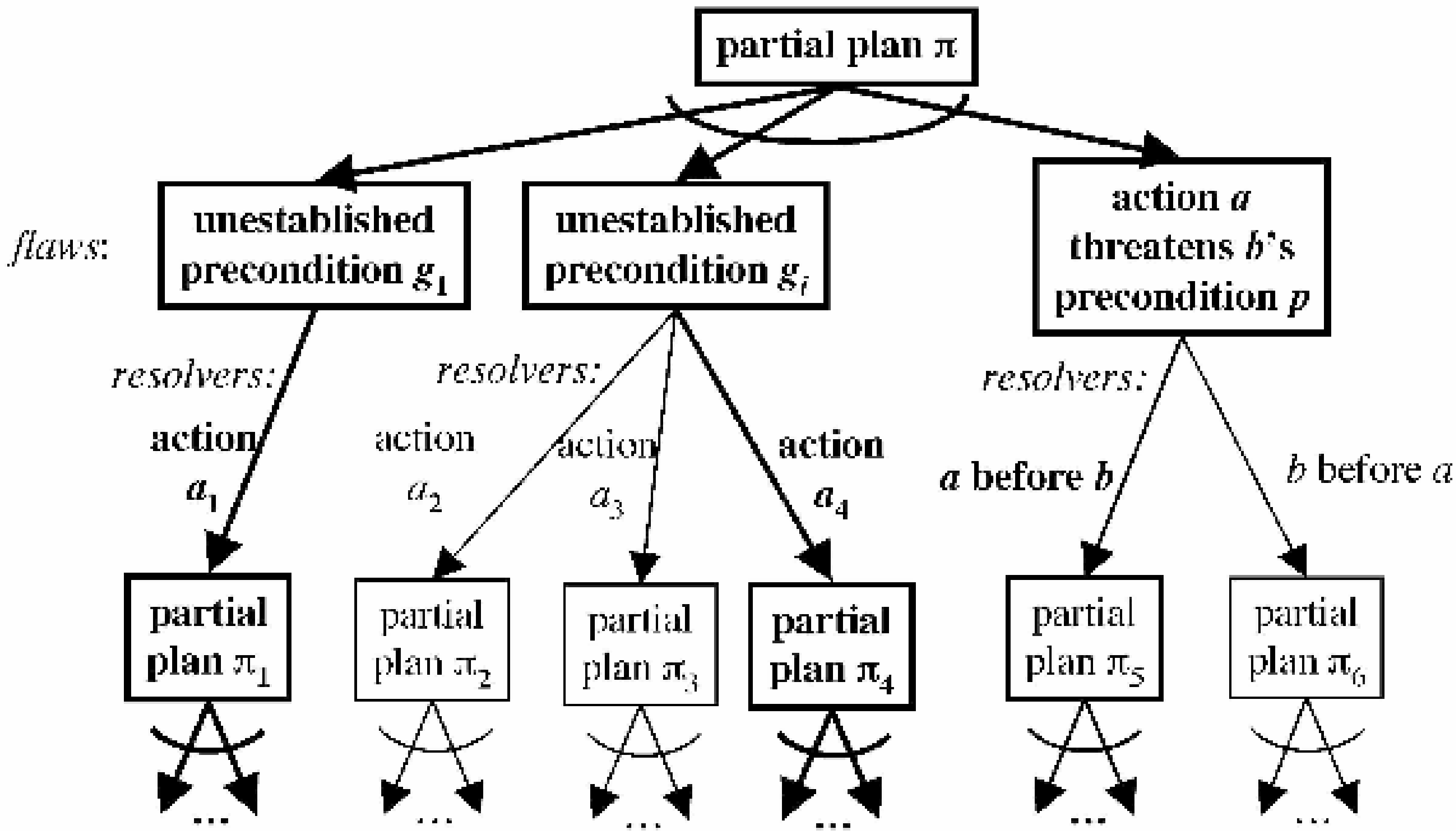
- FastForward foi um dos melhores
- Nessa competição todos os problemas de planejamento eram clássicos
- Duas *tracks*:
  - *Fully automated planners*
    - » FF ganhou um prêmio de "outstanding performance"
    - » FF apresentou uma grande variação na qualidade dos planos encontrados
    - » No entanto, FF encontrou planos num tempo muito curto quando comparado com outros planejadores clássicos

FF na competição de planejamento AIPS-2002

- FF ficou na média entre todos os planejadores
- LPG (graphplan + local search) se saiu muito melhor e ganhou um prêmio de "distinguished performance of the first order"
- Uma das coisas que causaram dificuldades para o FastForward
  - Os problemas na competição de AIPS-2002 foram além de planejamento clássico e envolveram:
    - » Variáveis numéricas, otimização, durações de tempo
- Exemplo:
  - Um domínio inspirado no telescópio espacial Hubble (muito mais simples do que um domínio real!)
    - » Um satélite precisa fazer observações de estrelas
    - » Coletar a maior quantidade possível de dados antes de consumir todo o seu combustível
  - Qualquer quantidade de dados coletado é uma solução
    - » Para esse domínio FF sempre devolvia um plano vazio

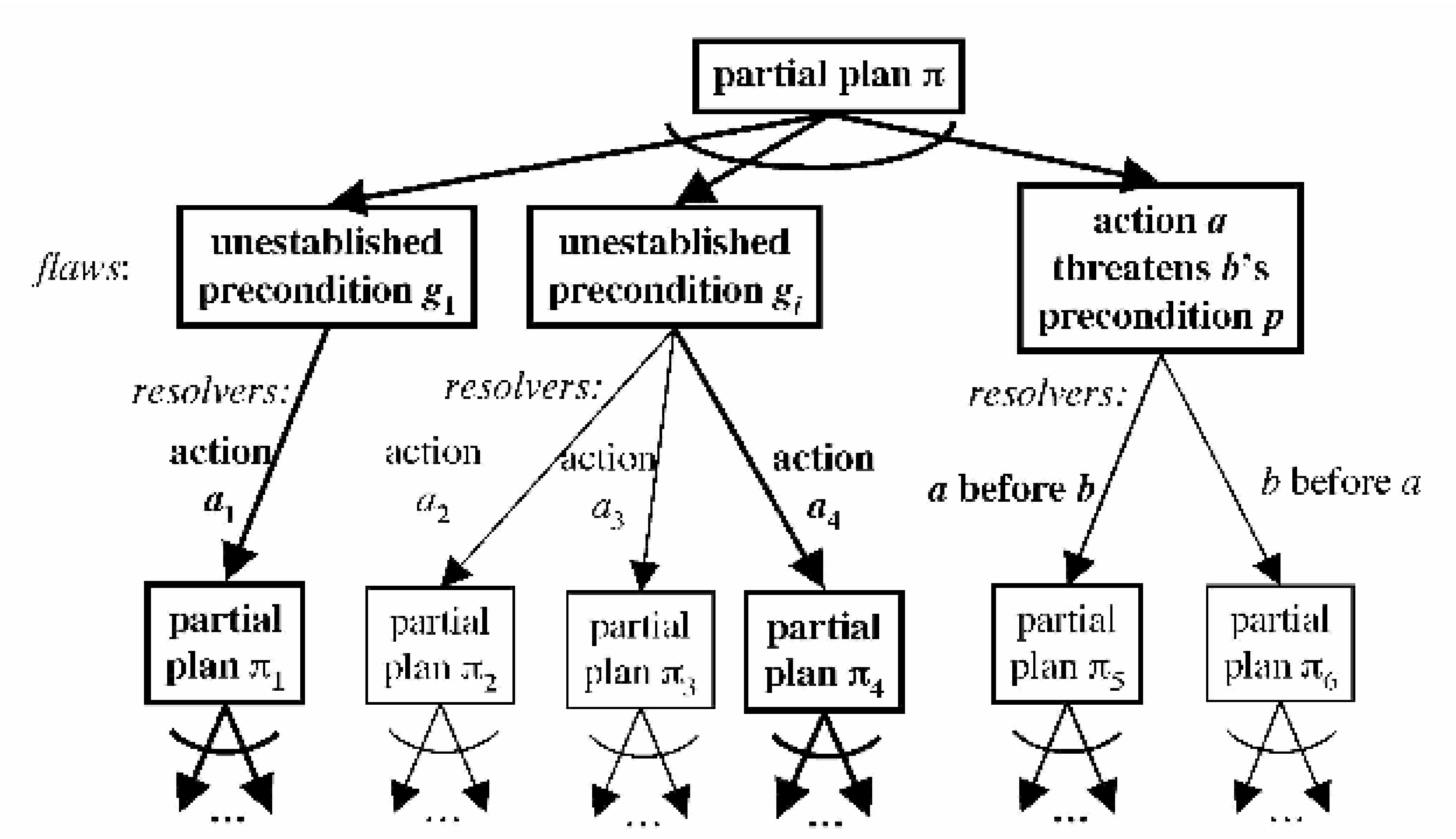
Heurísticas para Planejamento no espaço de planos

- Como selecionar a próxima *flaw* ?



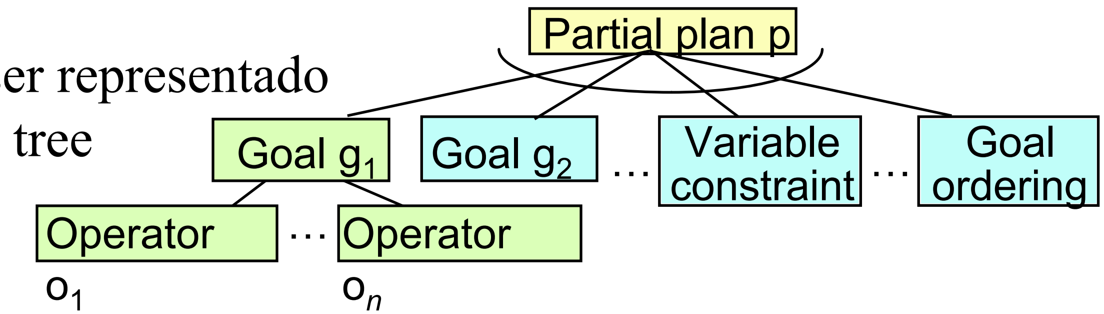
Uma Heurística Possível

- *Fewest Alternatives First* (FAF)

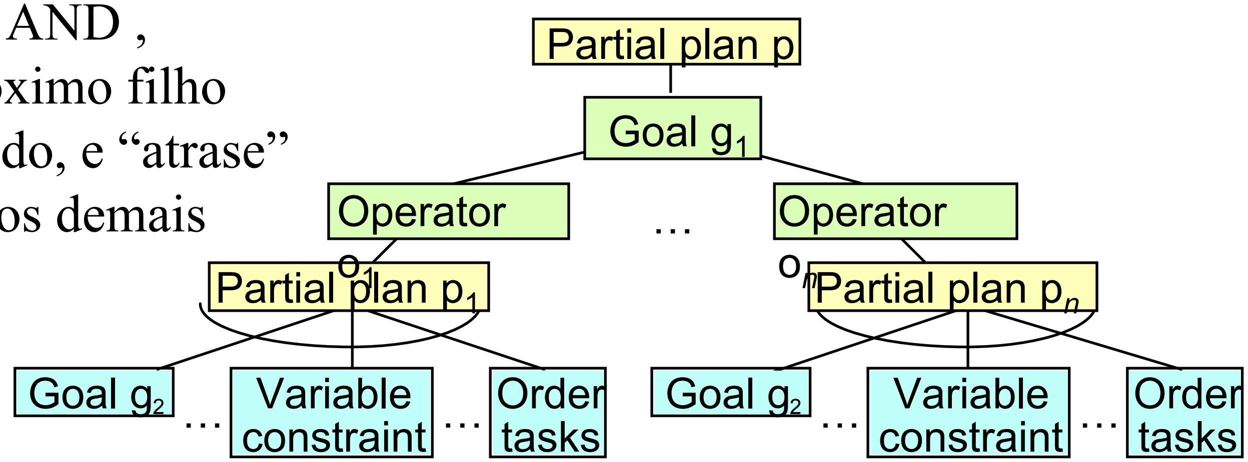


Serialização e árvore AND/OR

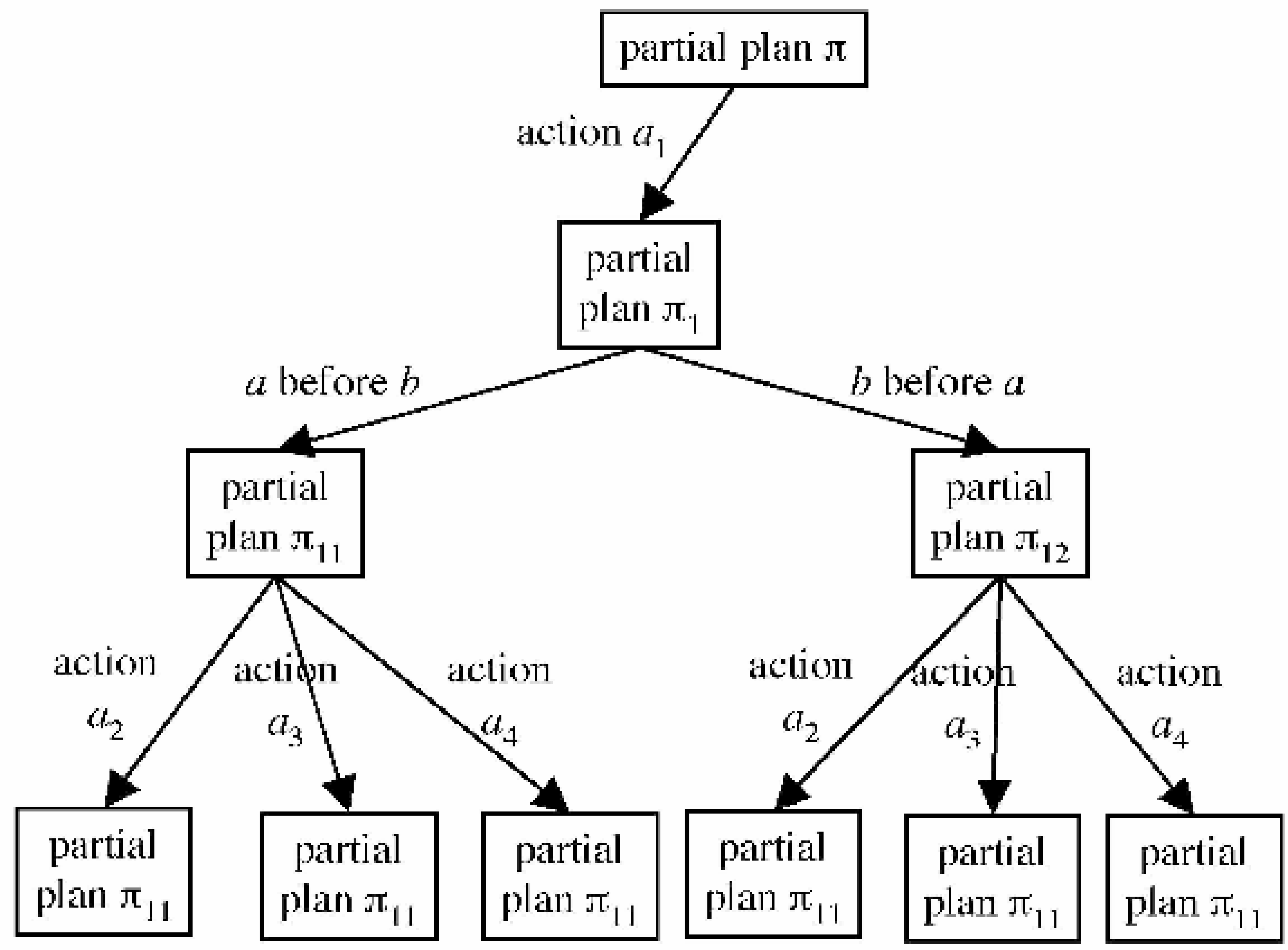
- O espaço de busca pode ser representado por uma árvore AND/OR tree



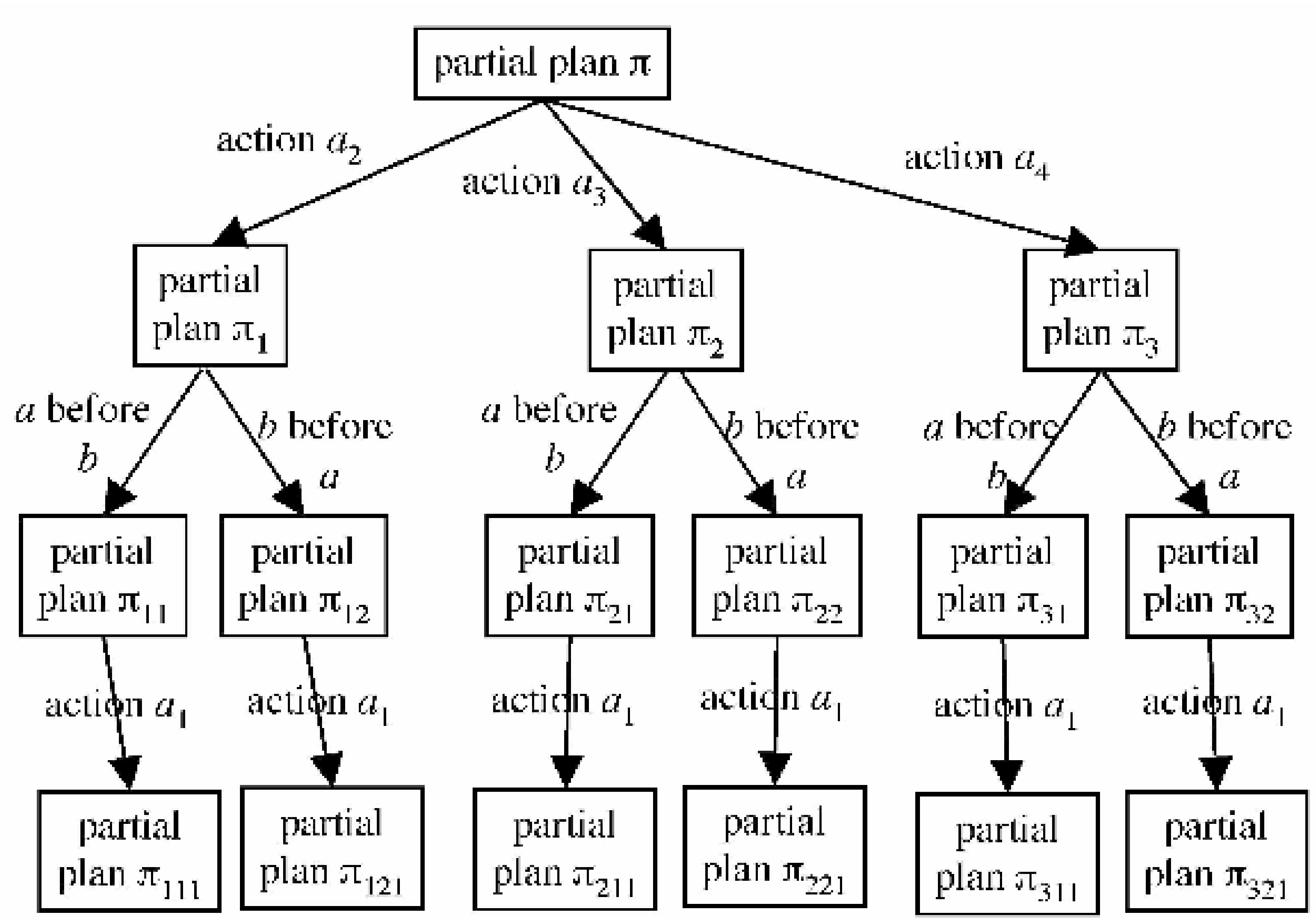
- Decidir qual será a próxima *flaw* = **serialização da árvore** (isto é, transformar a árvore AND/OR em uma árvore de espaço de estados
    - A cada ramo AND , escolha o próximo filho a ser expandido, e "atrase" a expansão dos demais filhos



Um exemplo de serialização



Outro exemplo

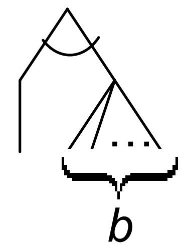


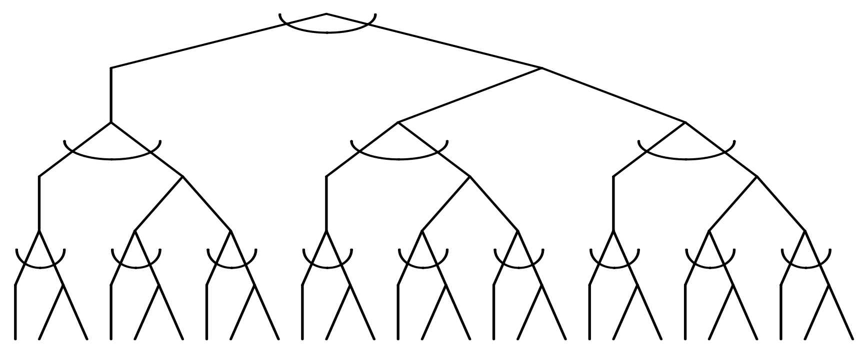
Como serializações diferemntes afetam o desempenho do planejador?

- Estratégias diferentes de refinamento de planos produzem diferentes serializações
  - Os espaços de busca possui diferentes quantidades de nós
- No pior caso, o planejador buscará o espaço de busca serializado inteiro
- Quanto menor for a serialização mais eficiente o planejador
- Uma boa heurística: *fewest alternatives first* (***FAF heuristic***) => escolha uma *flaw* com o menor número de *resolvers*. Isso irá limitar o custo de eventuais *backtracks*. Gasta tempo O(n), sendo n é o número de *flaws* no plano parcial.

Qual é a diferença que uma estratégia de refinamento faz?

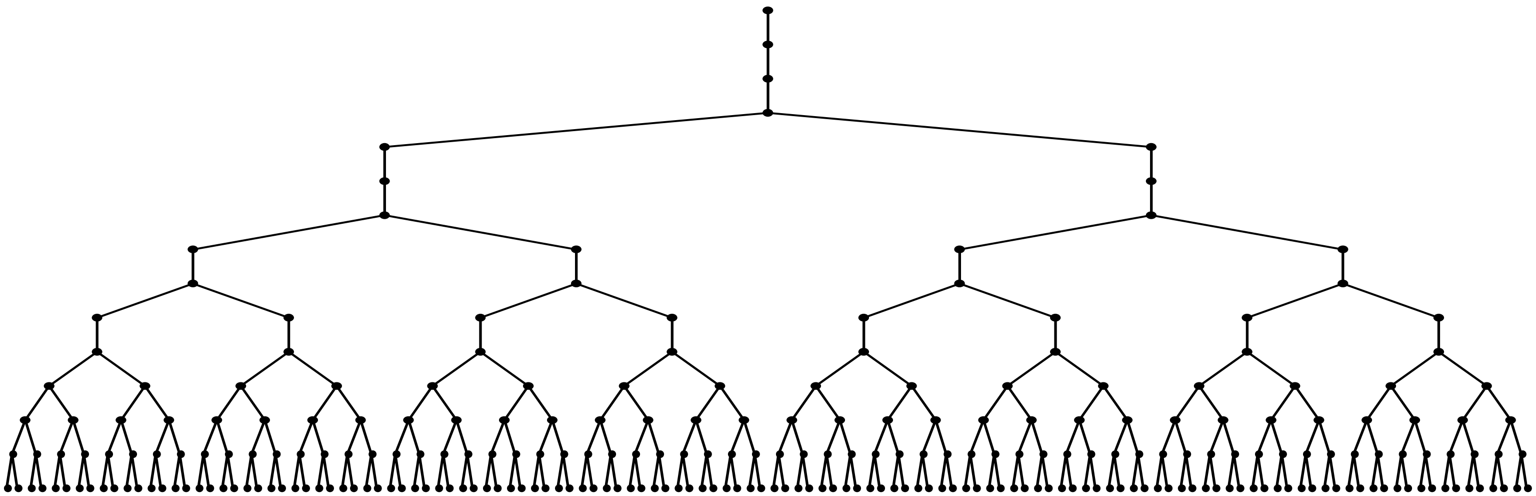
- Caso de estudo: construir um grafo AND/OR a partir de ocorrências repetidas do seguinte padrão :





- Exemplo:
  - Número de níveis $k = 3$
  - Fator de ramificação $b = 2$
- Análise:
  - O número total de nós no grafo AND/OR é $n = \Theta(b^k)$
  - Quantos nós existem na melhor e na pior serialização?

Caso de estudo (continuação)



- A melhor serialização contém $\Theta(b^{2^k})$ nós
- A pior serialização contém $\Theta(2^k b^{2^k})$ nós
  - Os tamanhos diferem de um fator exponencial, mas a melhor serialização ainda é exponencialmente maior
  - Para melhor isso, é preciso boas estratégias de seleção de nós, ramificação e poda

Heurística de seleção de *resolvers*

- Seja Θ o conjunto de todos os planos geradoscom todos os possíveis *resolvers*. Selecione um plano π que possui o menor conjunto de sub-metas sem *causal links* (não é uma heurística muito informativa).
- Uma heurística mais informatia: construa um grafo AND/OR através de passos da busca regressiva definida por γ^{-1} até algum nível fixo k
- Seleciobe o plano π que minimize a soma ponderada de :
  1. o número de ações no grafo que não estão em π e
  2. o número de sub-metas que restam em suas folhas que não estão no estado inicial