Call Subsumption Mechanisms for Tabled Logic Programs

Flávio Cruz <flaviocruz@gmail.com>
Orientador: Ricardo Rocha <ricroc@dcc.fc.up.pt>

[1]Center for Research in Advanced Computing Systems
[2]Faculdade de Ciências da Universidade do Porto

26 de Junho de 2010

Plano

Prolog e o método SLD

- Na programação em lógica, o método de resolução SLD é um método inerentemente não-determinístico e do tipo *top-down*.
- Em Prolog usa-se o método SLD de forma determinística, avaliando as cláusulas de cima para baixo e da esquerda para a direita.
- Esta forma de avaliação pode ser aplicado de forma eficiente em máquinas virtuais baseadas em stack, tais como a *Warren's Abstract Machine* (WAM).

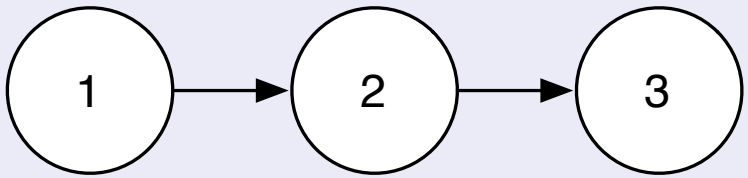
Prolog e o método SLD

- Na programação em lógica, o método de resolução SLD é um método inerentemente não-determinístico e do tipo *top-down*.
- Em Prolog usa-se o método SLD de forma determinística, avaliando as cláusulas de cima para baixo e da esquerda para a direita.
- Esta forma de avaliação pode ser aplicado de forma eficiente em máquinas virtuais baseadas em stack, tais como a *Warren's Abstract Machine* (WAM).
- No entanto, este método tem diversas limitações, tais como o tratamento de ciclos infinitos (positivos ou negativos) e computações redundantes.

Limitações do método SLD

Programa

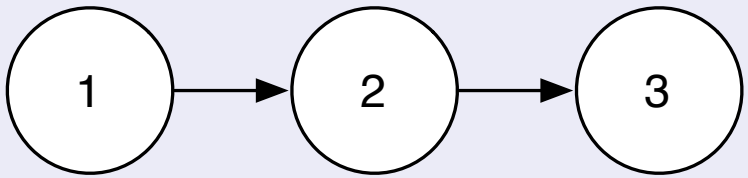
```
path(X, Z) :- path(X, Y),
              edge(Y, Z).
path(X, Z) :- edge(X, Z).

edge(1, 2).
edge(2, 3).
```



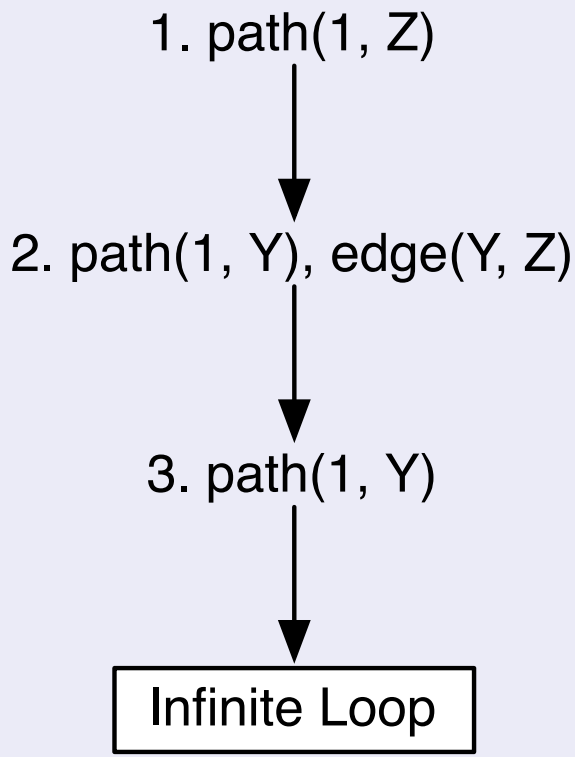
Limitações do método SLD

Programa

```
path(X, Z) :- path(X, Y),
              edge(Y, Z).
path(X, Z) :- edge(X, Z).

edge(1, 2).
edge(2, 3).
```



?- path(1, Z)



Plano

Tabulação

- A tabulação é um refinamento do método de resolução SLD.
- As primeiras chamadas a subgolos tabelados são avaliados normalmente através da execução do código do programa.
- As *chamadas similares* são avaliadas através do consumo das respostas guardadas na *tabela* e que foram geradas pelo subgolo similar correspondente.
- Permite que programas lógicos válidos sejam executáveis.

Similaridade entre Chamadas

Em geral, existem dois testes para verificar se um subgolo é similar a outro:

- *Tabulação por variantes:* o subgolo A é similar a B quando eles são iguais por renomeação das variáveis.

Similaridade entre Chamadas

Em geral, existem dois testes para verificar se um subgolo é similar a outro:

- *Tabulação por variantes:* o subgolo A é similar a B quando eles são iguais por renomeação das variáveis.

Exemplo

`p(X,1,Y)` e `p(Y,1,Z)` são variantes porque ambos podem ser transformados em `p(VAR0,1,VAR1)`

Similaridade entre Chamadas

Em geral, existem dois testes para verificar se um subgolo é similar a outro:

- *Tabulação por variantes:* o subgolo A é similar a B quando eles são iguais por renomeação das variáveis.

Exemplo

`p(X,1,Y)` e `p(Y,1,Z)` são variantes porque ambos podem ser transformados em `p(VAR0,1,VAR1)`

- A maioria dos motores de tabulação, incluindo o YapTab, apenas suportam este teste.

Similaridade entre Chamadas

- *Tabulação por subsumpção:* A é similar a B quando A é mais específico do que B (ou B é mais geral do que A).

Exemplo

`p(X,1,2)` é mais específico do que `p(Y,1,Z)` porque existe uma substituição $\{$`Y = X, Z = 2`$\}$ que torna `p(X,1,2)` uma *instância* de `p(Y,1,Z)`.

Similaridade entre Chamadas

- *Tabulação por subsumpção:* A é similar a B quando A é mais específico do que B (ou B é mais geral do que A).

Exemplo

`p(X,1,2)` é mais específico do que `p(Y,1,Z)` porque existe uma substituição $\{$`Y = X, Z = 2`$\}$ que torna `p(X,1,2)` uma *instância* de `p(Y,1,Z)`.

Teorema

Se A é mais específico do que B e S_A e S_B são os respectivos conjuntos de respostas, então $S_A \subseteq S_B$.

Similaridade entre Chamadas

- *Tabulação por subsumpção:* A é similar a B quando A é mais específico do que B (ou B é mais geral do que A).

Exemplo

`p(X,1,2)` é mais específico do que `p(Y,1,Z)` porque existe uma substituição $\{$`Y = X, Z = 2`$\}$ que torna `p(X,1,2)` uma *instância* de `p(Y,1,Z)`.

Teorema

Se A é mais específico do que B e S_A e S_B são os respectivos conjuntos de respostas, então $S_A \subseteq S_B$.

- Só o XSB Prolog implementa este tipo de tabulação, primeiro usando uma técnica chamada *Dynamic Threaded Sequential Automata* (DTSA) e mais recentemente usando a técnica de *Time Stamped Tries* (TST).

Exemplo por Subsumpção

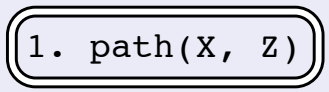
Respostas

- `path(X, Z)`:

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

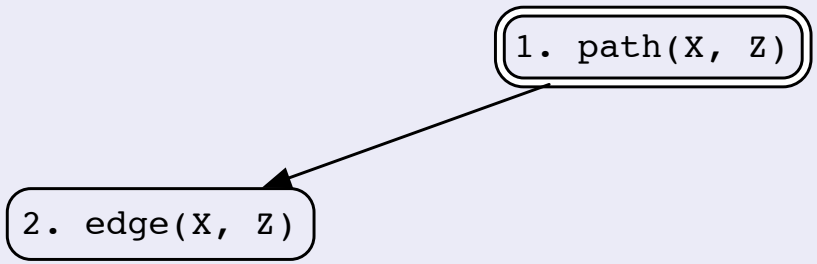
Respostas

- `path(X, Z)`:

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

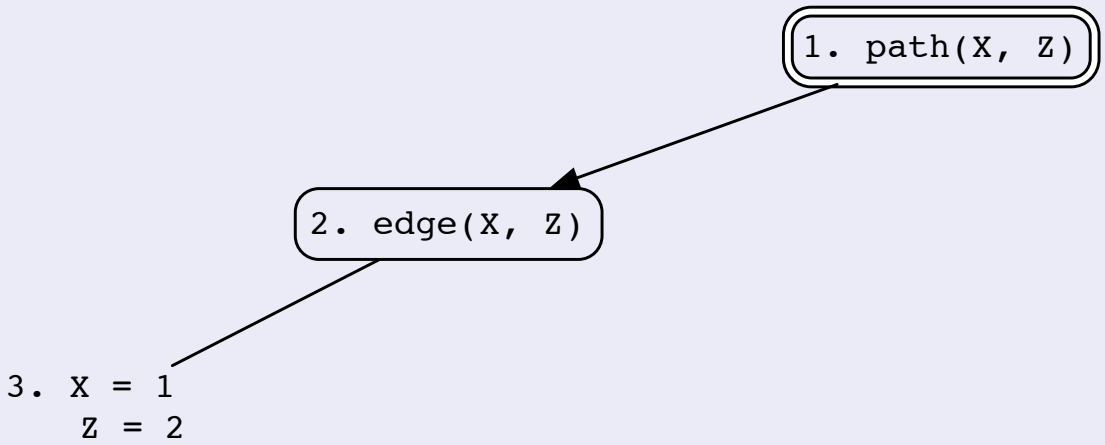
Respostas

- path(X, Z): (3) X=1 Z=2

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

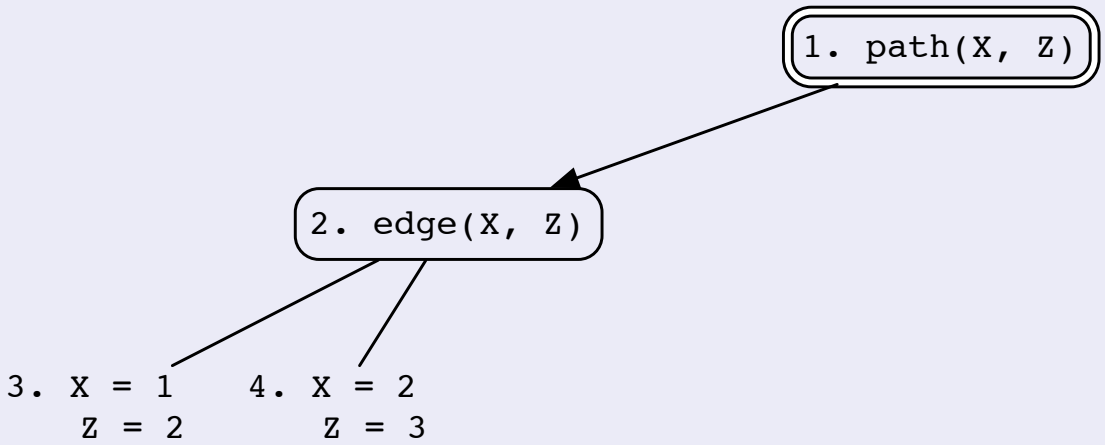
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3`

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

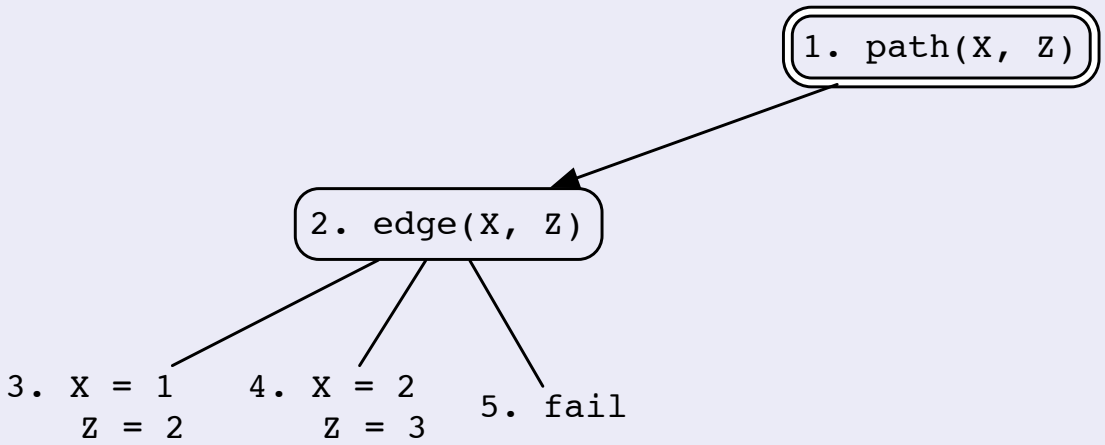
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3`

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

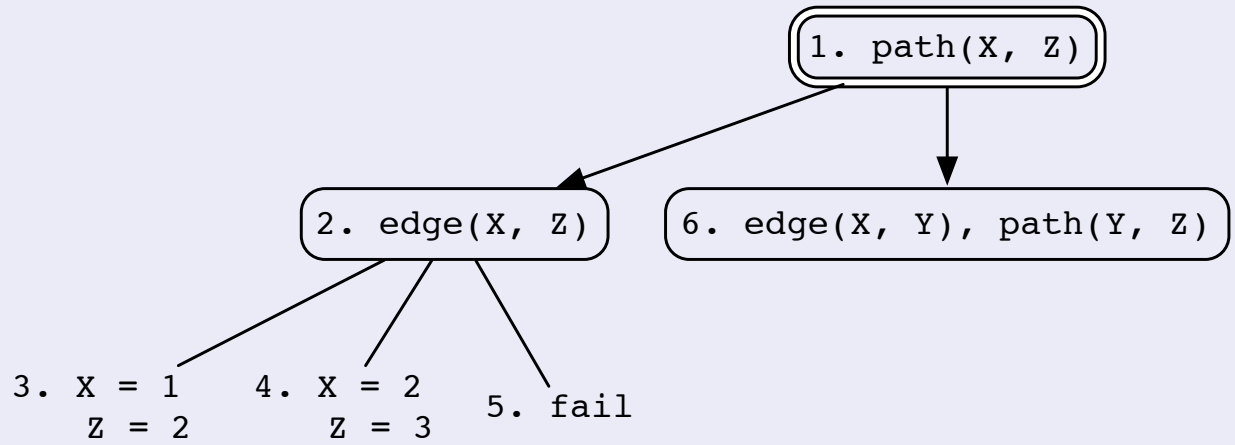
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3`

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

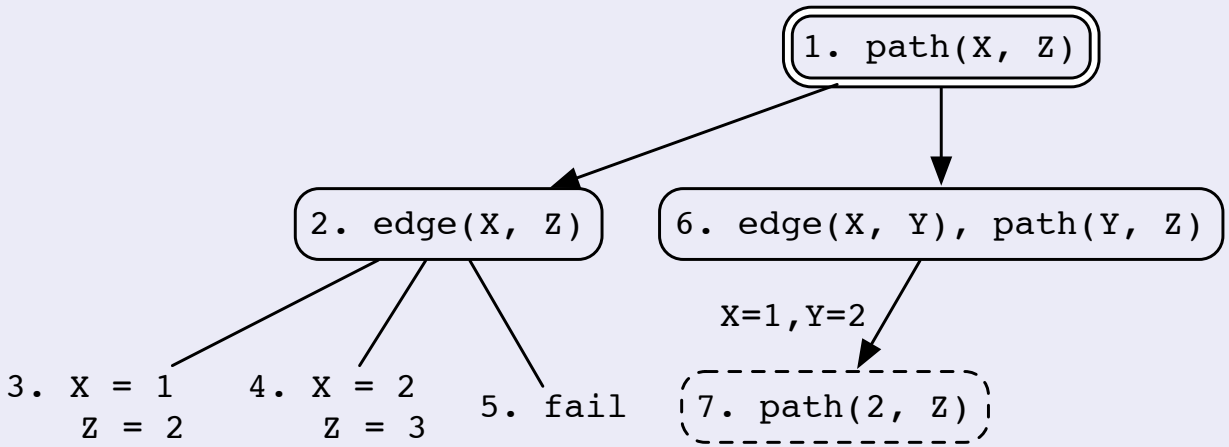
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3`
- `path(2, Z)`:

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

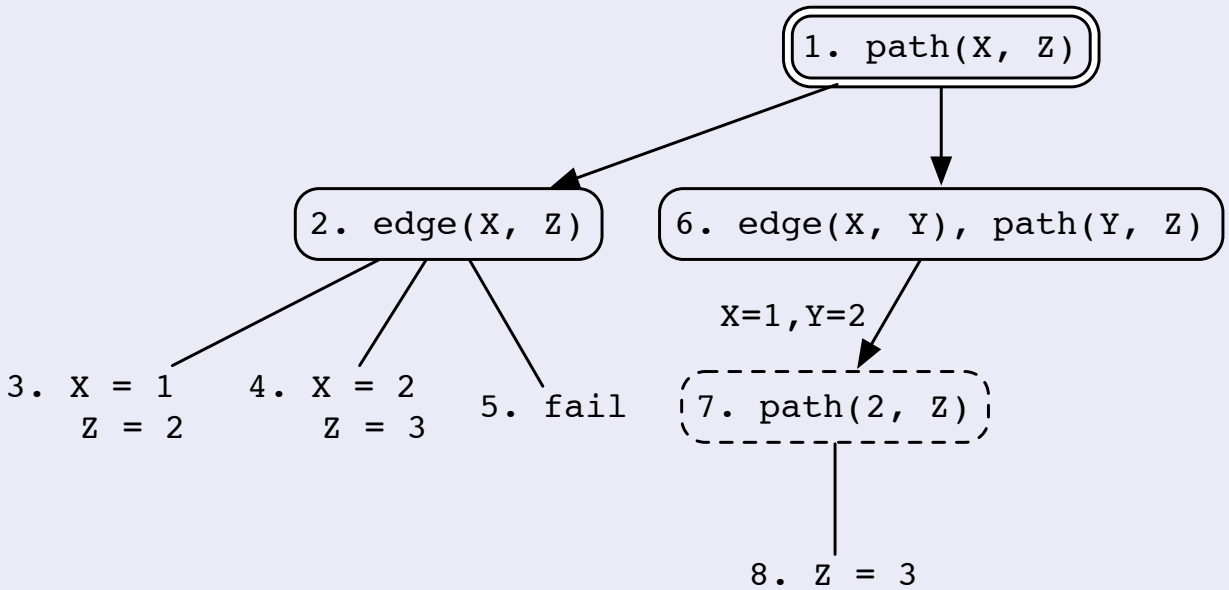
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3` (8) `X=1 Z=3`
- `path(2, Z)`: (8) `Z=3`

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

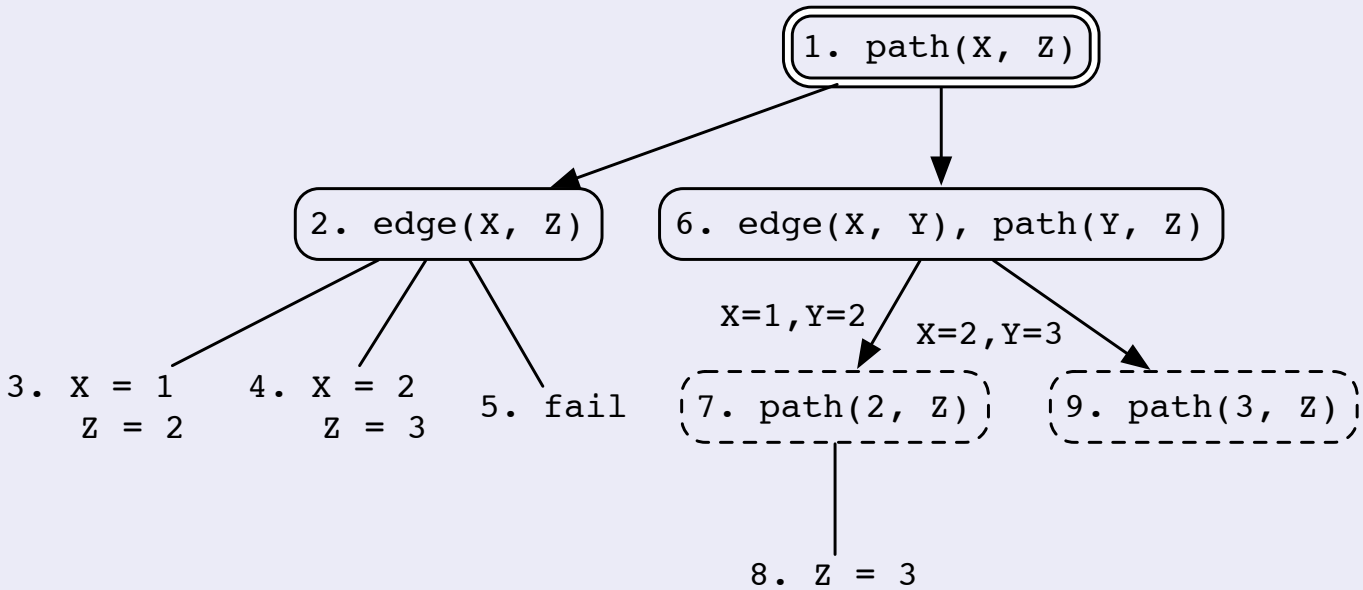
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3` (8) `X=1 Z=3`
- `path(2, Z)`: (8) `Z=3`
- `path(3, Z)`:

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

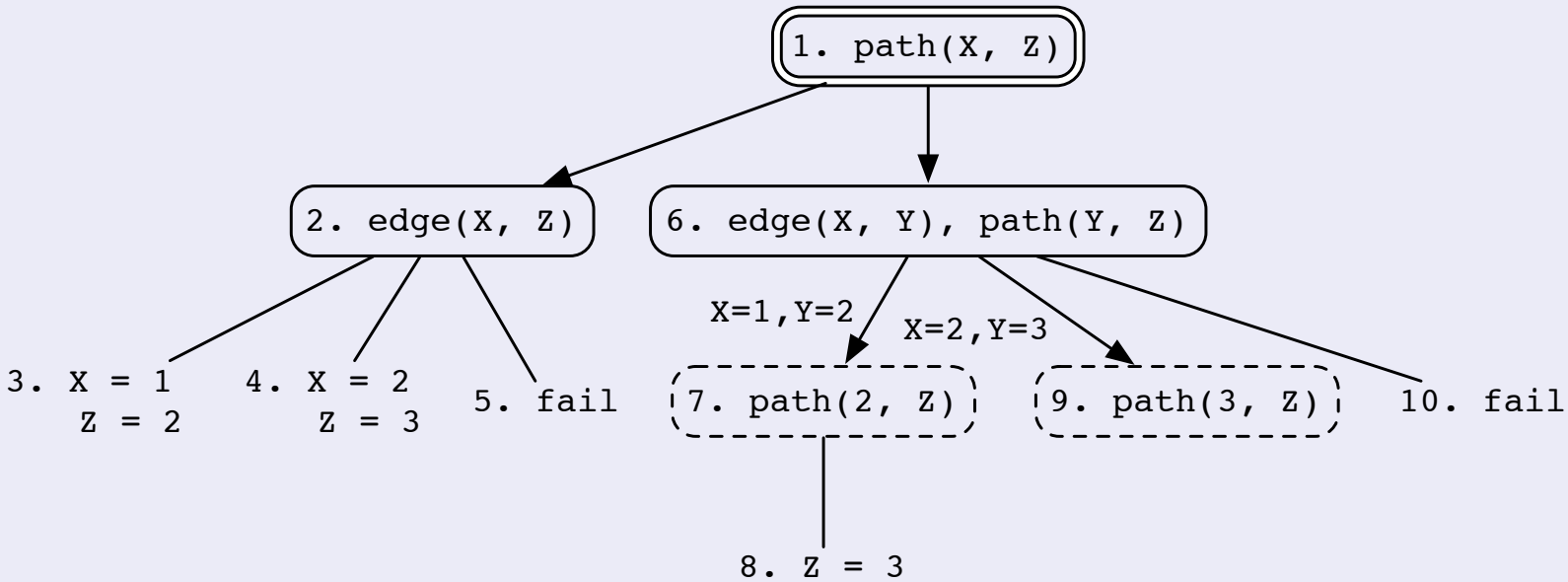
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3` (8) `X=1 Z=3`
- `path(2, Z)`: (8) `Z=3`
- `path(3, Z)`:

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Exemplo por Subsumpção

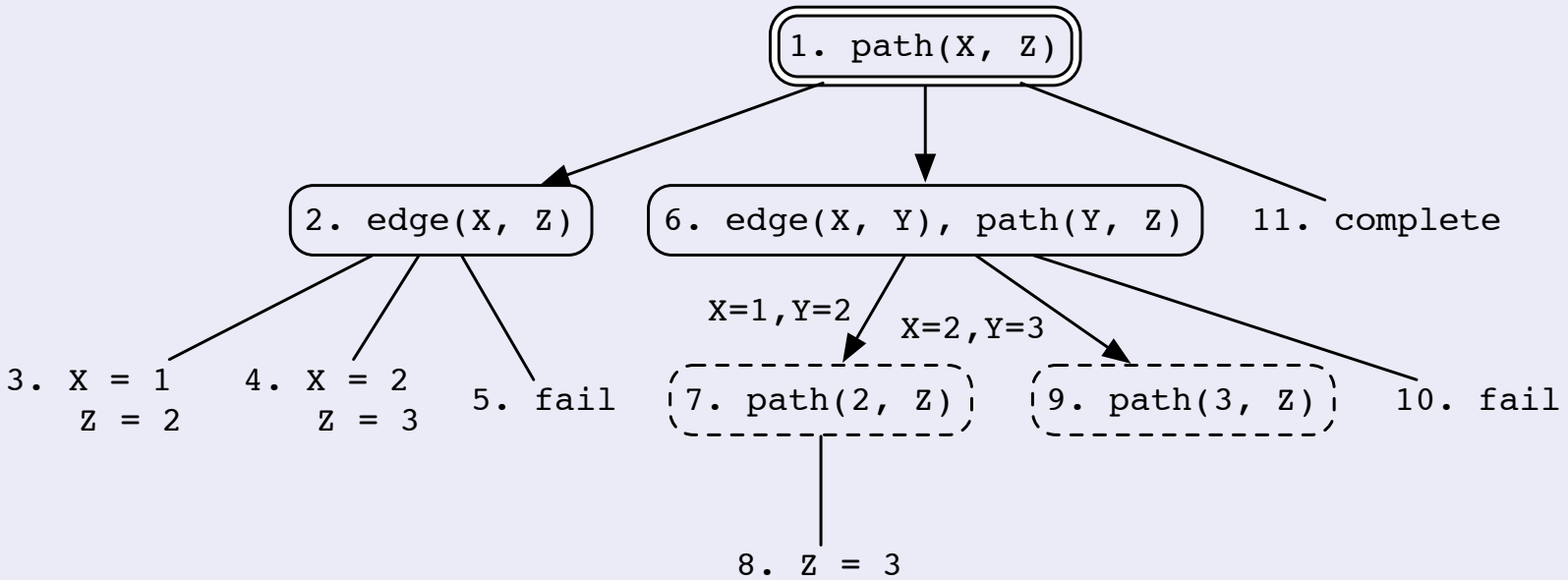
Respostas

- `path(X, Z)`: (3) `X=1 Z=2` (4) `X=2 Z=3` (8) `X=1 Z=3`
- `path(2, Z)`: (8) `Z=3`
- `path(3, Z)`: ∅

Programa

```
path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).

edge(1, 2). edge(2, 3).
```

Exemplo



Espaço das Tabelas

- Tries: estruturas em árvore onde os prefixos comuns dos termos são representados apenas uma vez.

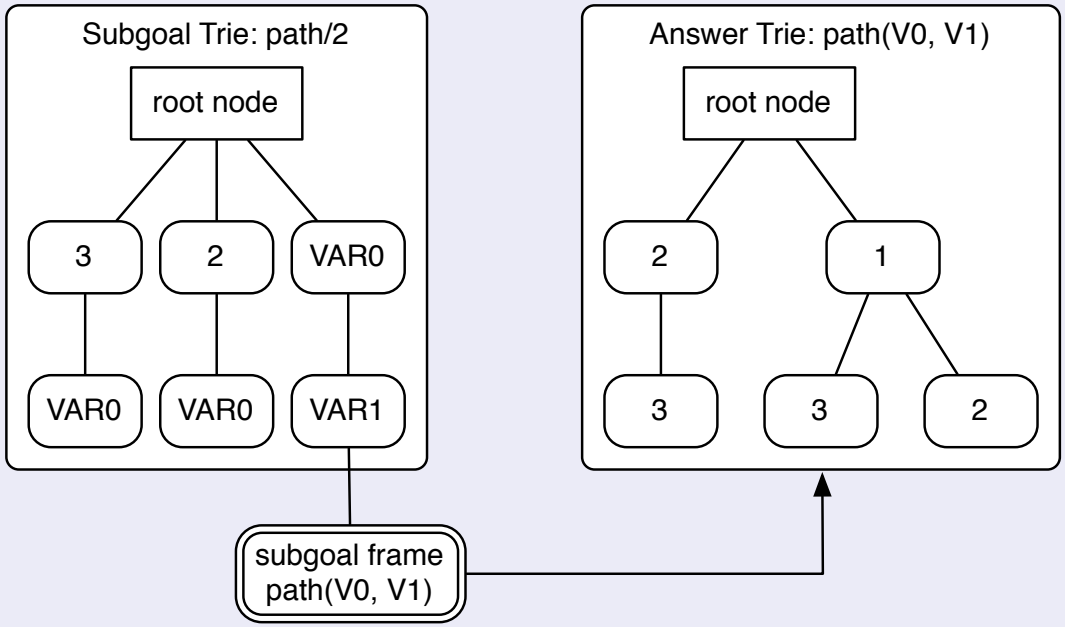
Espaço das Tabelas

- Tries: estruturas em árvore onde os prefixos comuns dos termos são representados apenas uma vez.
- Normalmente, existem dois níveis de tries:
  - *Subgoal trie*: guarda os subgolos para um certo predicado (por exemplo `path/2`).
  - *Answer trie*: guardas as respostas.

Espaço das Tabelas

- Tries: estruturas em árvore onde os prefixos comuns dos termos são representados apenas uma vez.
- Normalmente, existem dois níveis de tries:
  - *Subgoal trie*: guarda os subgolos para um certo predicado (por exemplo `path/2`).
  - *Answer trie*: guardas as respostas.

Exemplo

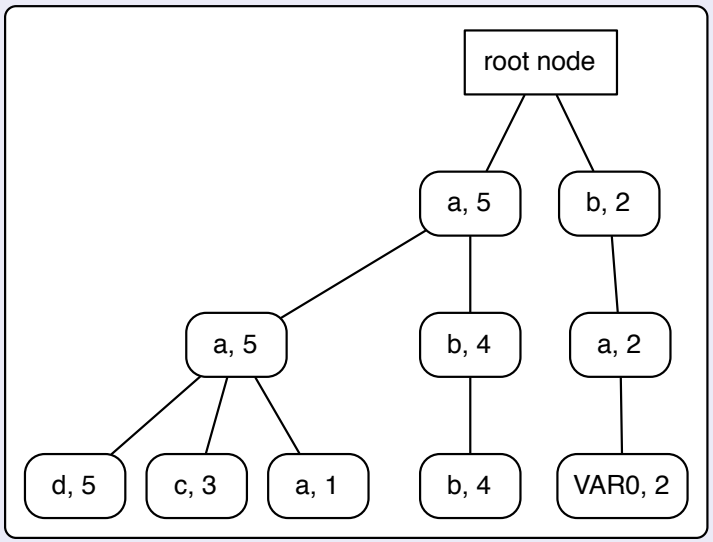


Plano

Time Stamped Tries

- Estende-se a answer trie com informação temporal: *timestamps*.

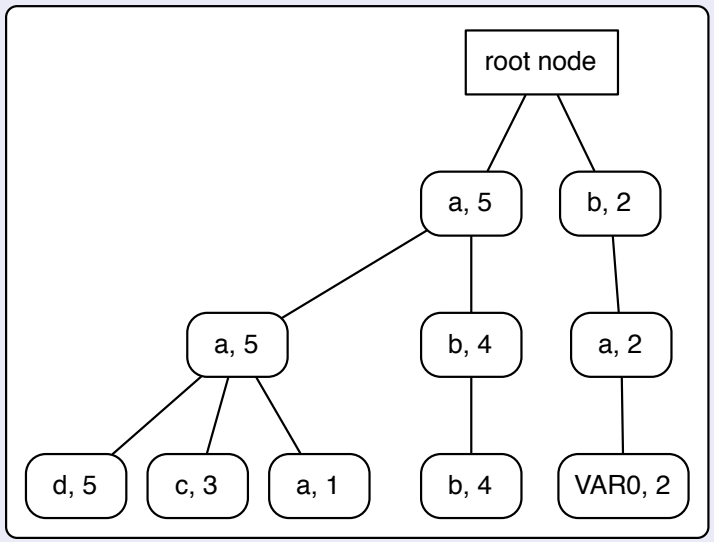
Time Stamped Trie



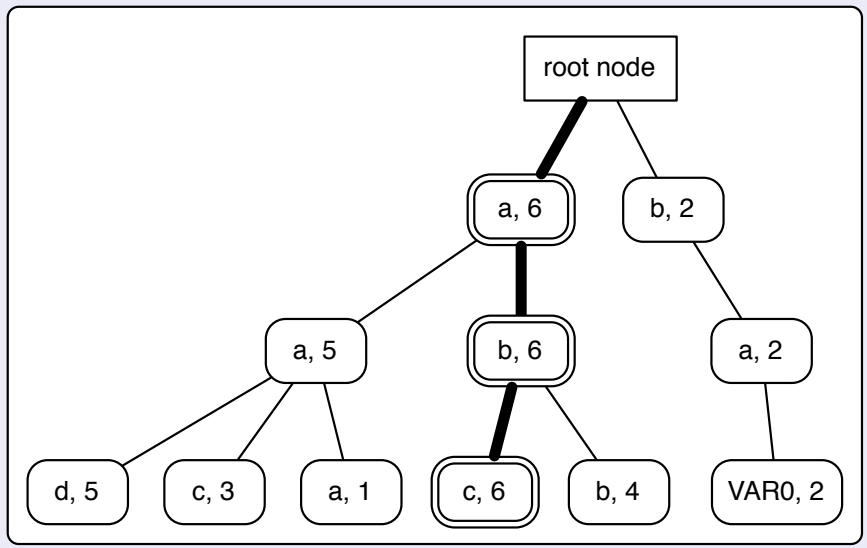
Time Stamped Tries

- Estende-se a answer trie com informação temporal: *timestamps*.
- Quando uma resposta é inserida, incrementa-se o timestamp da resposta.

Time Stamped Trie



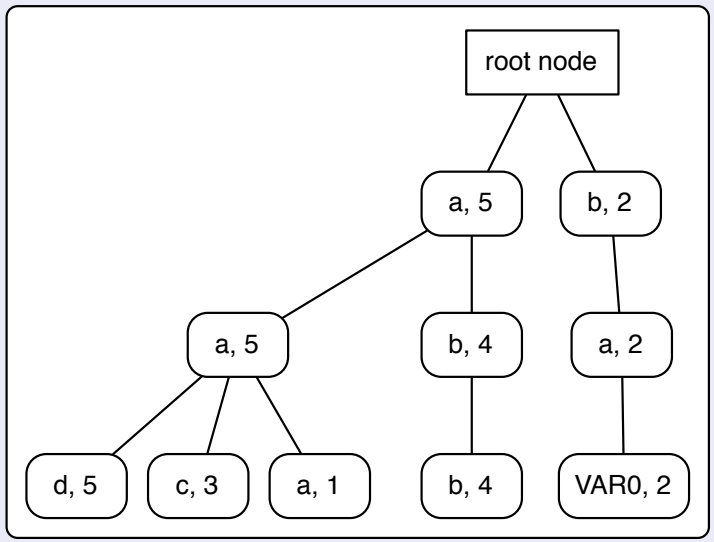
Inserir **p(a, b, c)**



Time Stamped Tries

- O subgolo específico guarda o timestamp da última procura para permitir uma pesquisa incremental.
- O algoritmo de pesquisa faz corte dos ramos pelo timestamp e através de operações de unificação.

Time Stamped Trie



Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.

Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.
  - Cálculo do líder.

Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.
  - Cálculo do líder.
  - Nova chamada.

Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.
  - Cálculo do líder.
  - Nova chamada.
  - Nova resposta.

Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.
  - Cálculo do líder.
  - Nova chamada.
  - Nova resposta.
  - Calcular a próxima resposta a consumir.

Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.
  - Cálculo do líder.
  - Nova chamada.
  - Nova resposta.
  - Calcular a próxima resposta a consumir.
  - Instruções da trie completa.

Implementação no YapTab

- Para implementar tabulação por subsumpção usou-se a técnica das TSTs.
- Reutilizou-se o código de forma quase integral do XSB Prolog: foram usados macros para permitir que ambos os sistemas Prolog usassem o mesmo código.
- As alterações nas operações principais de tabulação foram mínimas.
  - Cálculo do líder.
  - Nova chamada.
  - Nova resposta.
  - Calcular a próxima resposta a consumir.
  - Instruções da trie completa.
- **O sistema permite usar uma mistura de predicados por variantes e por subsumpção.**

Plano

Problemas na Tabulação por Subsumpção

- Apesar da tabulação por subsumpção atingir bons resultados, sofre de um problema: a ordem na qual os subgolos são chamados pode afectar a performance do sistema.

Exemplo

Se `p(1,X)` for chamado antes de `p(X,Y)`, `p(1,X)` não usará as respostas de `p(X,Y)`, mas irá executar o código para gerar as suas próprias respostas.

Como solucionar este problema?

- Tabulação por Subsumpção Retroactiva (TSR).

Como solucionar este problema?

- Tabulação por Subsumpção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.

Como solucionar este problema?

- Tabulação por Subsumpção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.
- Ganhar tempo de execução e aumentar partilha de respostas.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.
- Manter a execução consistente devido aos cortes.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.
- Manter a execução consistente devido aos cortes.
  - Através do desenho de estratégias de controlo de execução para a resolução retroactiva.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.
- Manter a execução consistente devido aos cortes.
  - Através do desenho de estratégias de controlo de execução para a resolução retroactiva.
- Evitar que o novo consumidor consuma respostas já geradas.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.
- Manter a execução consistente devido aos cortes.
  - Através do desenho de estratégias de controlo de execução para a resolução retroactiva.
- Evitar que o novo consumidor consuma respostas já geradas.
  - Nova organização do espaço de tabelas.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.
- Manter a execução consistente devido aos cortes.
  - Através do desenho de estratégias de controlo de execução para a resolução retroactiva.
- Evitar que o novo consumidor consuma respostas já geradas.
  - Nova organização do espaço de tabelas.
- Saber de forma eficiente que subgolos geradores mais específicos estão a executar.

Desafios

- Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G'.
  - Construindo uma árvore das dependências dos subgolos.
  - Considerando intervalos de pontos de escolha da máquina virtual.
- Manter a execução consistente devido aos cortes.
  - Através do desenho de estratégias de controlo de execução para a resolução retroactiva.
- Evitar que o novo consumidor consuma respostas já geradas.
  - Nova organização do espaço de tabelas.
- Saber de forma eficiente que subgolos geradores mais específicos estão a executar.
  - Novo algoritmo.

Exemplo de Execução

Programa

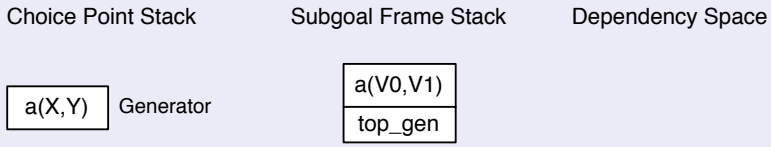
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

TSR

Novo gerador `a(X,Y)`

Exemplo

```
?- a(X,Y), p(Z, W)
```



Exemplo de Execução

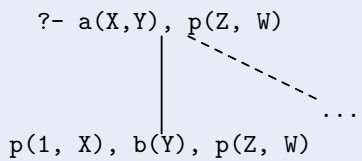
Programa

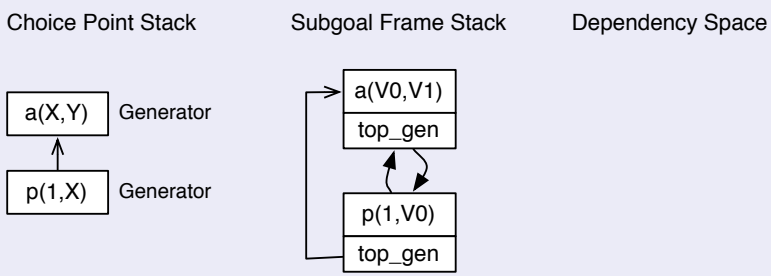
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

TSR

Novo gerador `p(1,X)`

Exemplo





Exemplo de Execução

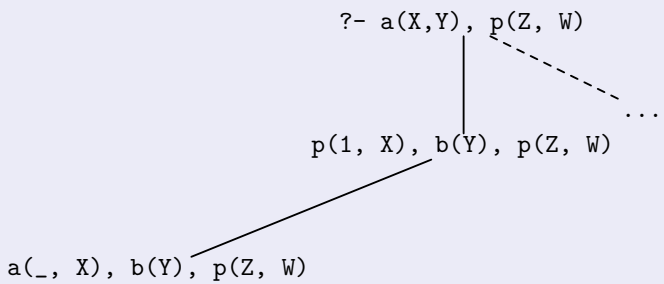
Programa

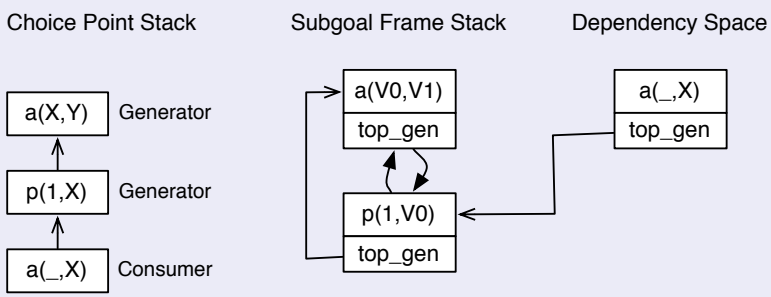
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

TSR

Novo consumidor `a(_,X)`

Exemplo





Exemplo de Execução

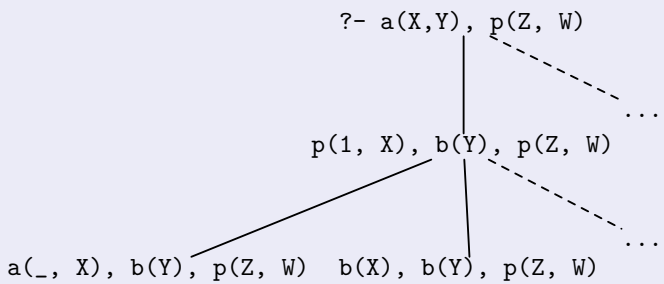
Programa

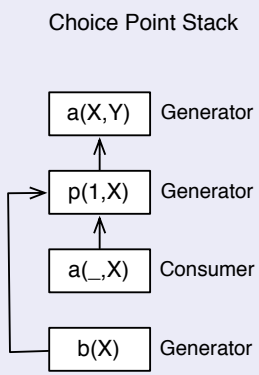
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

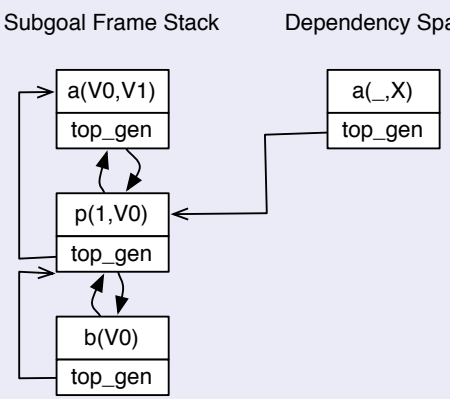
TSR

Novo gerador `b(X)`

Exemplo







Exemplo de Execução

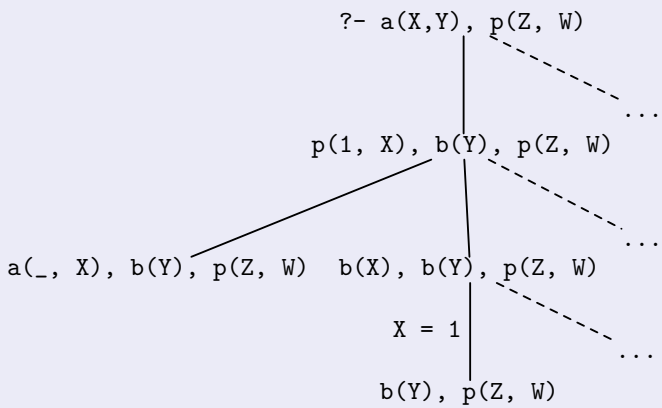
Programa

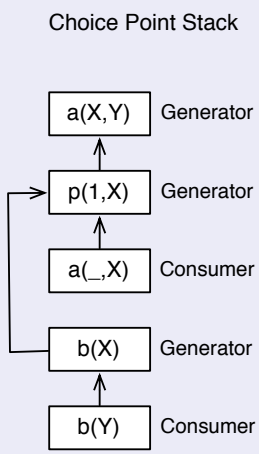
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

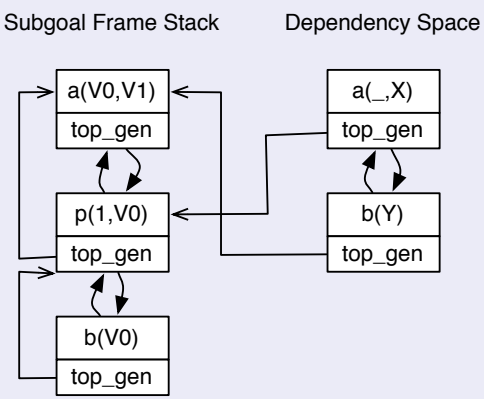
TSR

Novo consumidor `b(Y)`

Exemplo







Exemplo de Execução

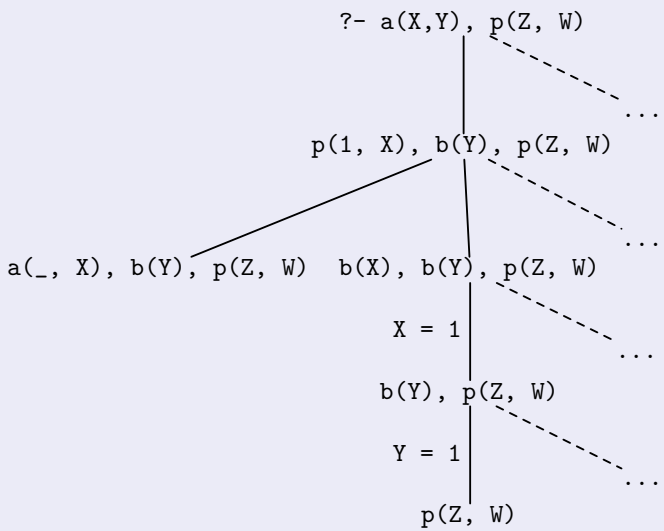
Programa

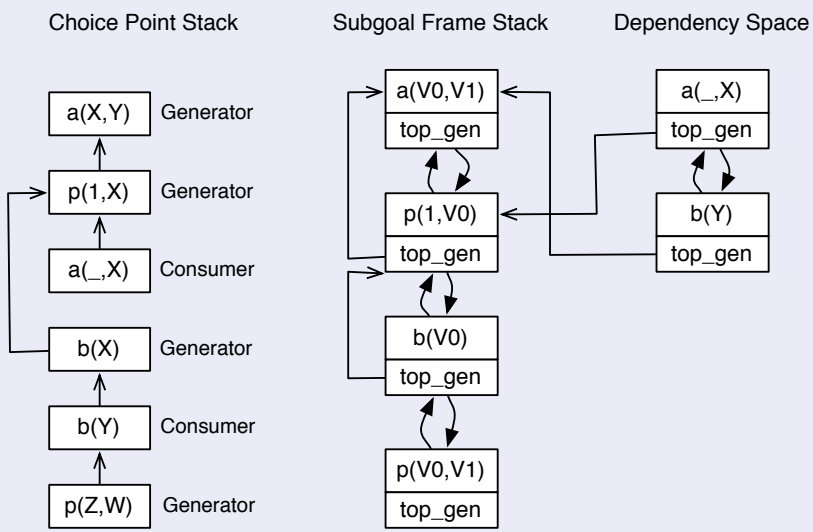
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

TSR

Novo gerador mais geral `p(Z,W)`

Exemplo





Exemplo de Execução

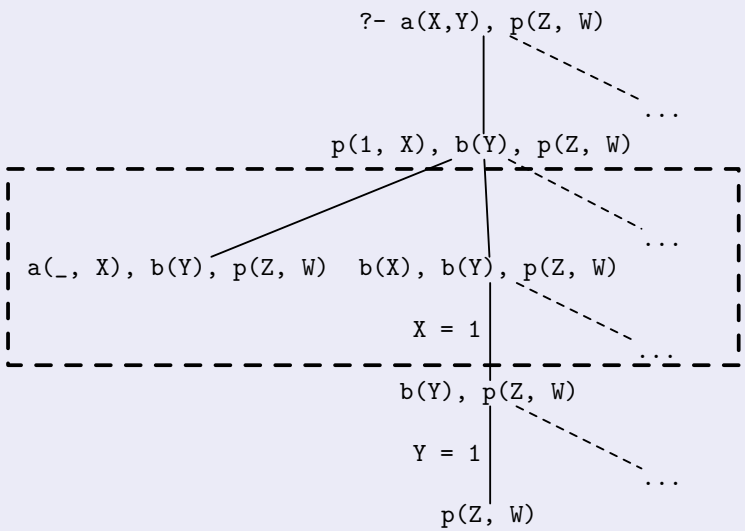
Programa

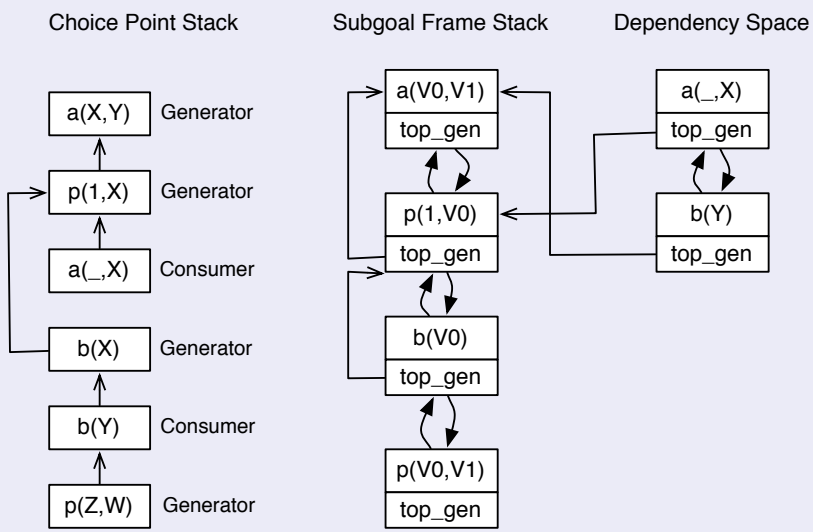
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

TSR

Determinar ramos a cortar

Exemplo





Exemplo de Execução

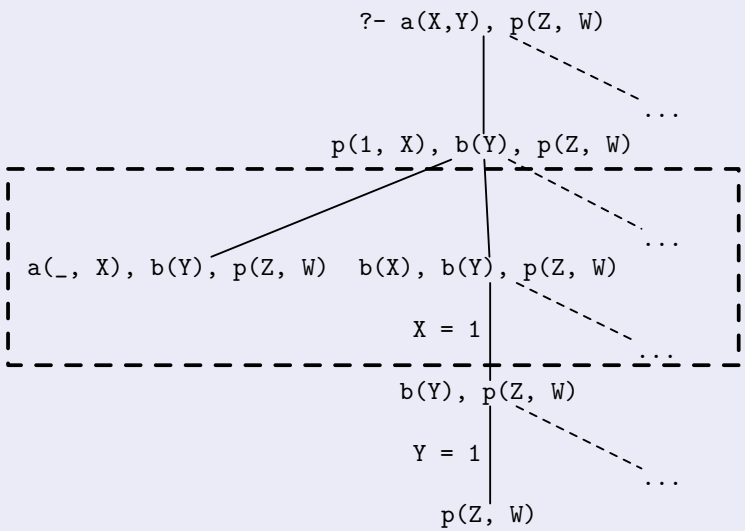
Programa

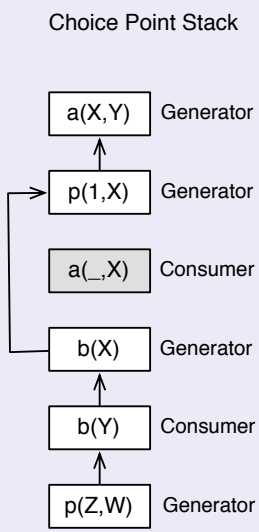
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

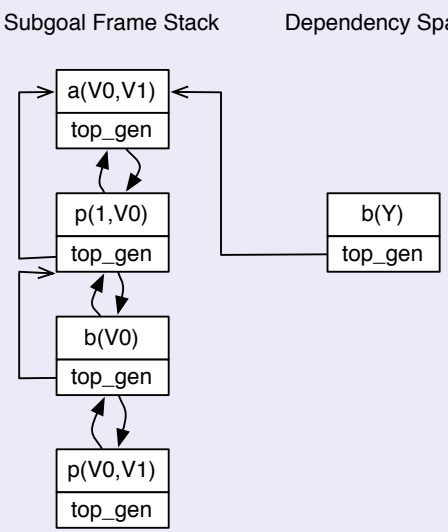
TSR

Consumidores como o `a(_,X)` são simplesmente removidos do *dependency space*

Exemplo







Exemplo de Execução

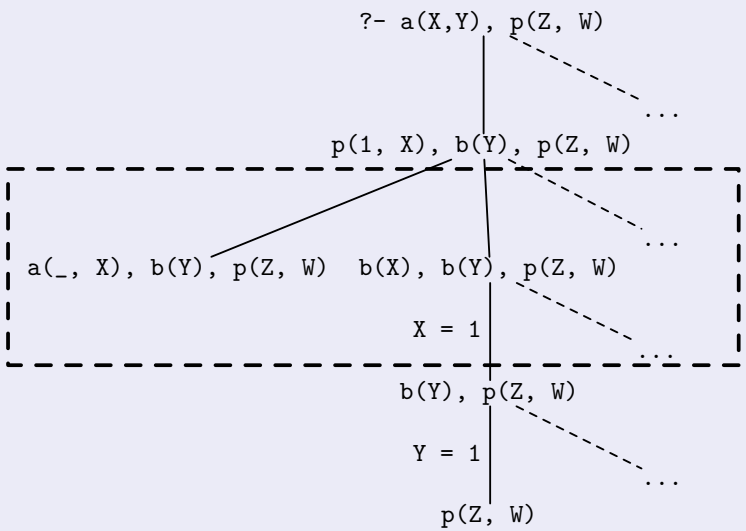
Programa

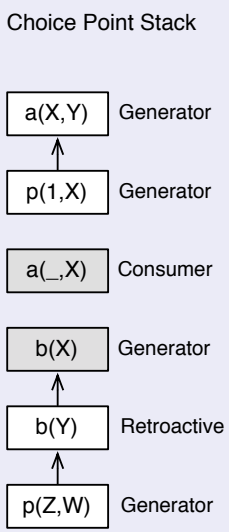
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

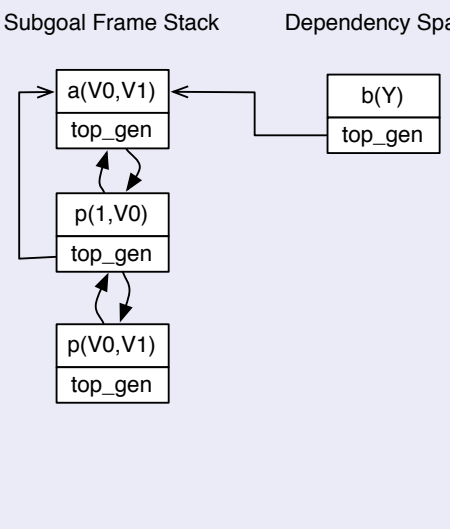
TSR

`b(X)` é um subgolo gerador interno, mudar o seu estado para *pruned*. Transformar *consumidores externos* (*orphaned consumers*) em nós retroactivos

Exemplo







Exemplo de Execução

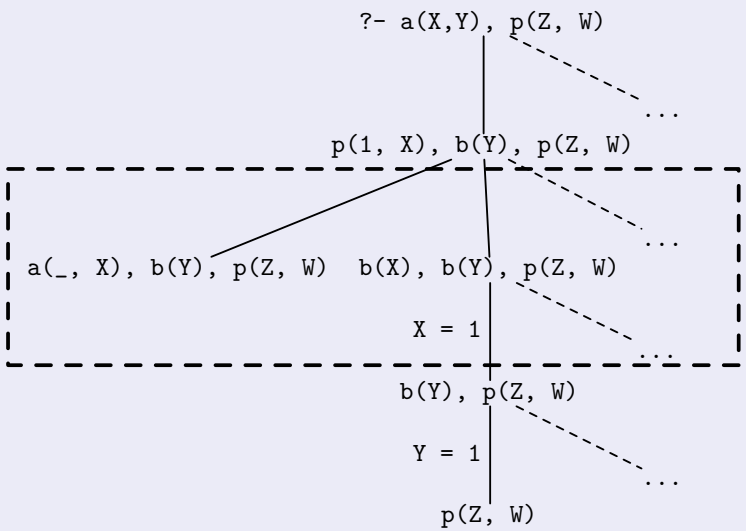
Programa

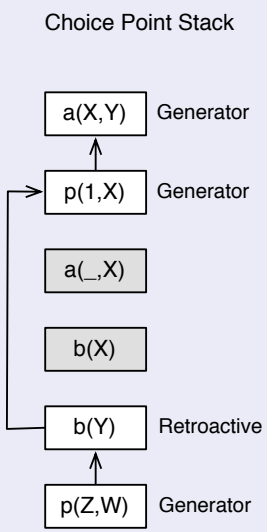
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

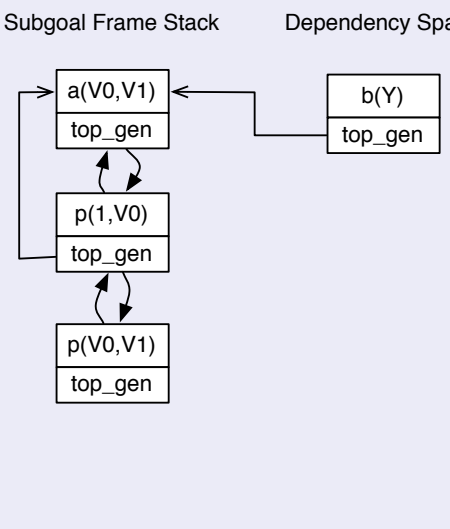
TSR

O nó `b(Y)` é um *nó fronteira*. É necessário ligá-lo ao nó `p(1,X)` para evitar que a execução salte para ramos mortos

Exemplo







Exemplo de Execução

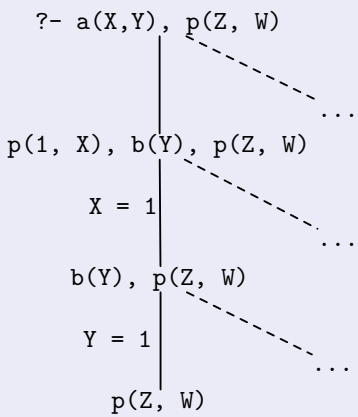
Programa

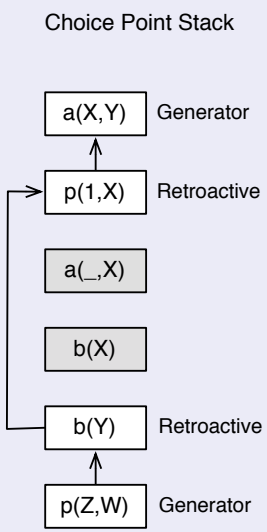
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.

a(X, Y) :- p(1, X), b(Y).
a(3, 4).
b(1). b(2).
p(1, X) :- a(_, X).
p(1, X) :- b(X).
```

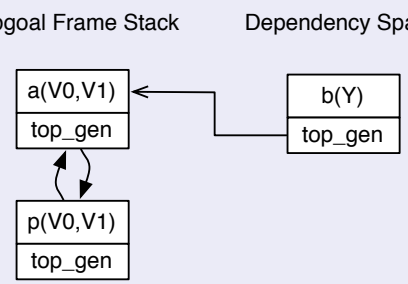
TSR

Transformar o nó `p(1,X)` em nó retroactivo e remover o *subgoal frame* da pilha respectiva

Exemplo







Corte da Execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
  - Corte externo: se aparece fora (visto anteriormente).

Corte da Execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
  - Corte externo: se aparece fora (visto anteriormente).
  - Corte interno: se aparece dentro.

Corte da Execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
  - Corte externo: se aparece fora (visto anteriormente).
  - Corte interno: se aparece dentro.
- Independente do tipo de corte, existe um conjunto de problemas que advém do corte da execução de geradores ou consumidores internos ao subgolo específico:
  - *Orphaned Consumers* (visto anteriormente)

Corte da Execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
  - Corte externo: se aparece fora (visto anteriormente).
  - Corte interno: se aparece dentro.
- Independente do tipo de corte, existe um conjunto de problemas que advém do corte da execução de geradores ou consumidores internos ao subgolo específico:
  - *Orphaned Consumers* (visto anteriormente)
  - *Lost consumers*

Corte da Execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
  - Corte externo: se aparece fora (visto anteriormente).
  - Corte interno: se aparece dentro.
- Independente do tipo de corte, existe um conjunto de problemas que advém do corte da execução de geradores ou consumidores internos ao subgolo específico:
  - *Orphaned Consumers* (visto anteriormente)
  - *Lost consumers*
  - *Pseudo-Completion*

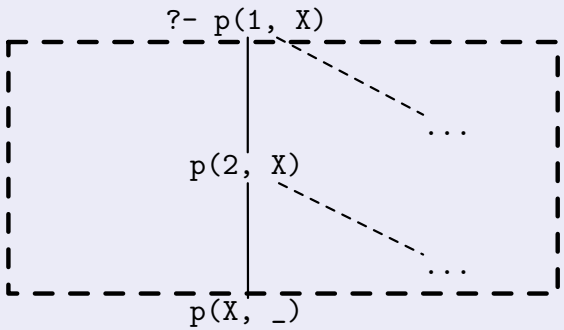
Corte da Execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
  - Corte externo: se aparece fora (visto anteriormente).
  - Corte interno: se aparece dentro.
- Independente do tipo de corte, existe um conjunto de problemas que advém do corte da execução de geradores ou consumidores internos ao subgolo específico:
  - *Orphaned Consumers* (visto anteriormente)
  - *Lost consumers*
  - *Pseudo-Completion*
  - *Leader Re-Computation*

Corte Interno

- Nesta situação cortam-se os ramos referentes a G', excepto a parte que irá computar as soluções do subgolo G.
  - G executa normalmente mas não devolve as soluções para o ambiente externo[1].

Antes

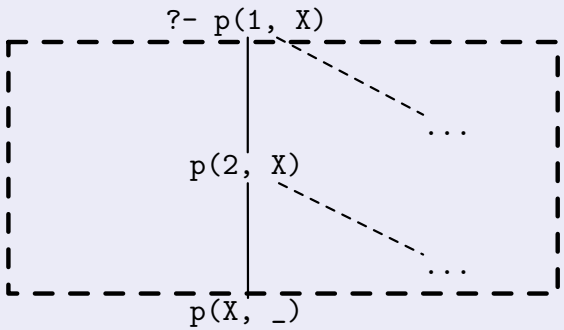


[1] O subgolo executa usando *local scheduling*

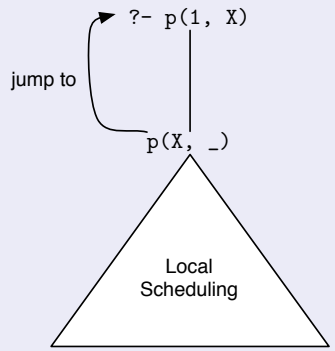
Corte Interno

- Nesta situação cortam-se os ramos referentes a G', excepto a parte que irá computar as soluções do subgolo G.
  - G executa normalmente mas não devolve as soluções para o ambiente externo[1].
- Quando G ou completar ou não conseguir completar por ser o líder, salta-se para o ponto de escolha do subgolo G'.

Antes



Depois

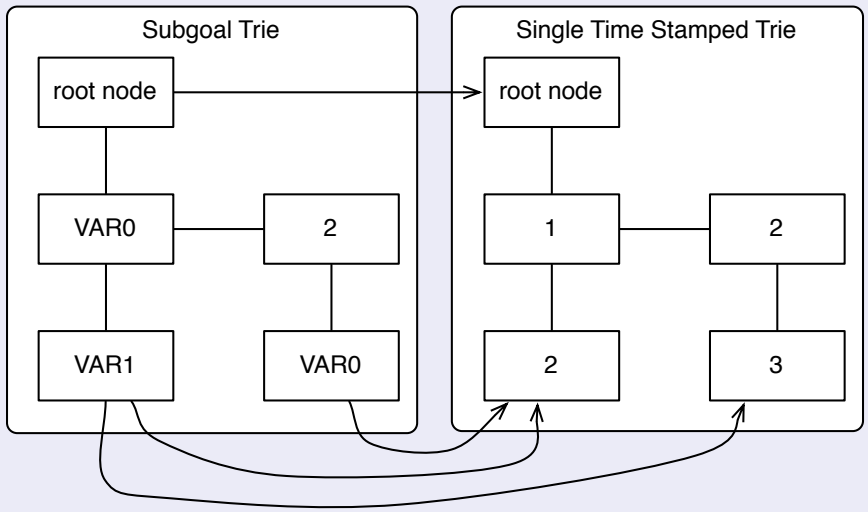


[1] O subgolo executa usando *local scheduling*

Espaço das Tabelas

- *Single Time Stamped Trie*: uma *answer trie* única por predicado.
  - As respostas são representadas apenas uma vez e referenciadas pelos subgolos que as usam.
  - Usa-se um timestamp por cada subgolo de forma a facilitar a transformação de gerador para consumidor.

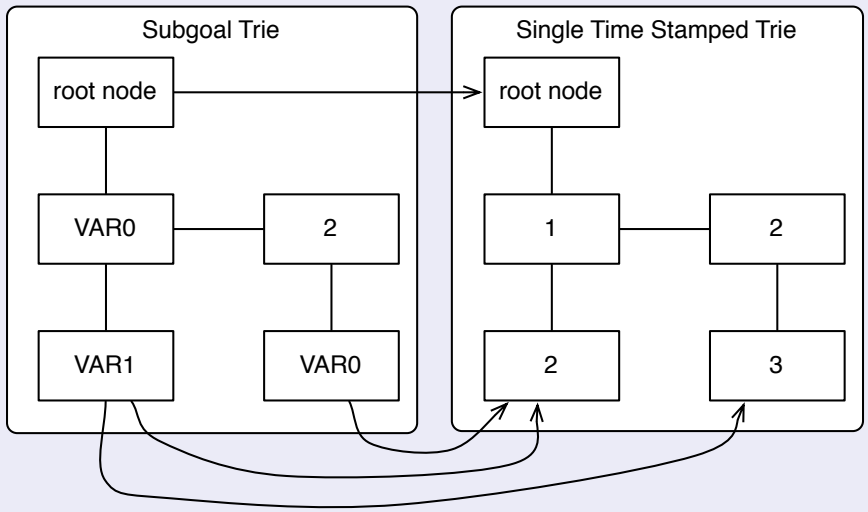
Single Time Stamped Trie



Espaço das Tabelas

- *Single Time Stamped Trie*: uma *answer trie* única por predicado.
  - As respostas são representadas apenas uma vez e referenciadas pelos subgolos que as usam.
  - Usa-se um timestamp por cada subgolo de forma a facilitar a transformação de gerador para consumidor.
- Permite que se possam reutilizar respostas relevantes a um novo subgolo gerador antes de executar o código.

Single Time Stamped Trie



Procura de Subgolos Específicos

- Percorrer a *subgoal trie* para encontrar subgolos mais específicos.

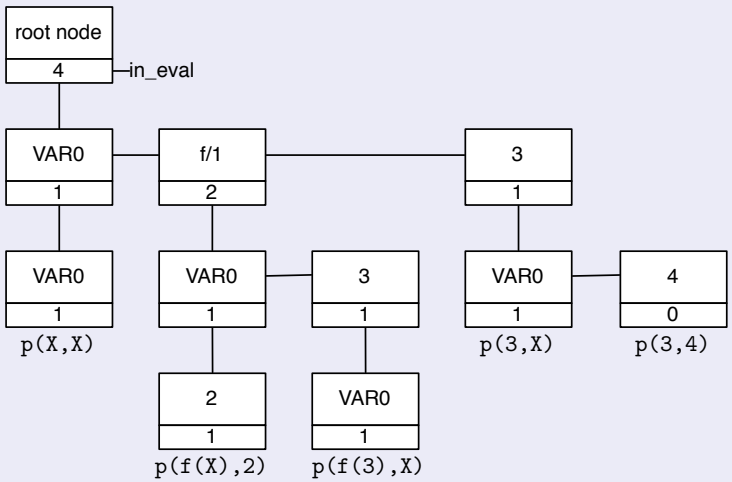
Procura de Subgolos Específicos

- Percorrer a *subgoal trie* para encontrar subgolos mais específicos.
- O problema resume-se a encontrar atribuições para as variáveis do subgolo mais geral.
- Usa-se a heap e a trilha da máquina virtual para maior eficiência.

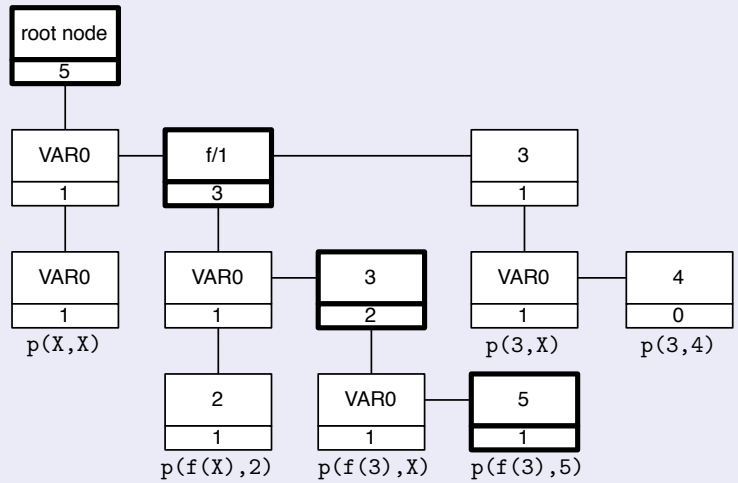
Procura de Subgolos Específicos

- Percorrer a *subgoal trie* para encontrar subgolos mais específicos.
- O problema resume-se a encontrar atribuições para as variáveis do subgolo mais geral.
- Usa-se a heap e a trilha da máquina virtual para maior eficiência.
- Para melhorar a eficiência, estendeu-se cada nó da *subgoal trie* com o número de subgolos sobre aquele ramo da trie que estão a executar.

Subgoal trie



Novo gerador



Plano

Tabulação por Subsumpção

- Avaliou-se o desempenho do motor de tabulação por subsumpção e comparou-se com o motor de tabulação do XSB Prolog.
- Sendo que ambos usam os mesmos algoritmos e estruturas de dados, o desempenho é muito parecido.

Programa	**XSB Prolog** *Speedup médio*	**Yap Prolog** *Speedup médio*
left_first	0.78	**1.02**
left_last	0.77	**0.96**
right_first	**1.01**	**1.01**
right_last	0.94	**1.07**
double_first	1.37	**1.48**
double_last	1.31	**1.40**
samegen	**339.76**	1.03
genome	559.54	**648.51**
reach_first	**0.96**	0.94
reach_last	**0.97**	0.90

Custo da TSR

- Foi medido o desempenho da TSR para programas que não tiram vantagens de usar os novos mecanismos.
- Comparou-se o desempenho com tabulação por subsumpção tradicional e tabulação por variantes.

Programa	**Yap Prolog**	
	Retro / Var	*Retro / Sub*
left_first	1.06	1.01
left_last	1.07	1.03
right_first	**0.97**	**0.95**
right_last	1.25	**0.94**
double_first	1.01	1.16
double_last	1.04	1.16
samegen	1.19	1.14
reach_first	1.11	1.04
reach_last	1.17	1.04
Média Total	1.10	1.05

Ganhos da TSR

- Por outro lado, comparou-se o desempenho para programas onde usar TSR é vantajoso.

Programa	**Yap Prolog**	
	Var / Retro	*Sub / Retro*
left_first[2]	0.89	0.95
left_last	0.88	0.90
double_first	1.07	**1.09**
double_last	1.05	**1.10**
genome	450.33	0.74
reach_first	2.54	**1.76**
reach_last	3.22	**1.87**
flora	3.17	**1.17**
fib	1.95	**2.02**
big	13.26	**13.66**

[2] Para os programas do tipo `path/2` usou-se o golo `path(X,1)`.

Plano

Conclusões

- Contribuições:
  - YapTab suporta tabulação por subsumpção.
  - Mecanismos e algoritmos que controlam a execução retroactiva.
  - Algoritmo de pesquisa de subgolos específicos.
  - Espaço de tabelas inovador que permite uma maior reutilização de respostas.
  - Suporte para uma mistura de métodos de avaliação: retroactivo, variantes e subsumpção.

Conclusões

- Contribuições:
  - YapTab suporta tabulação por subsumpção.
  - Mecanismos e algoritmos que controlam a execução retroactiva.
  - Algoritmo de pesquisa de subgolos específicos.
  - Espaço de tabelas inovador que permite uma maior reutilização de respostas.
  - Suporte para uma mistura de métodos de avaliação: retroactivo, variantes e subsumpção.
- Trabalho futuro:
  - Integrar o trabalho na distribuição oficial do Yap Prolog.
  - Melhoramento dos algoritmos do espaço das tabelas.
  - Maior experimentação com aplicações reais.
  - Explorar outros métodos de avaliação, como o *Call Abstraction*.

Artigos Publicados

- Retroactive Subsumption-Based Tabled Evaluation of Logic Programs, Flávio Cruz and Ricardo Rocha. 12th European Conference on Logics in Artificial Intelligence (JELIA 2010), Springer-Verlag. Helsinki, Finland, September 2010.
- Submetidos:
  - Efficient Instance Retrieval of Executing Subgoals for Tabled Evaluation, Flávio and Ricardo Rocha. 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 17), Yogyakarta, Indonesia, October 2010.
  - Efficient Retrieval of Subsumed Subgoals in Tabled Logic Programs, Flávio Cruz and Ricardo Rocha. Compilers, Programming Languages, Related Technologies and Applications (CORTA 2010), Braga, Portugal, September 2010.