

Call Subsumption Mechanisms for Tabled Logic Programs

Flávio Cruz <flaviocruz@gmail.com>

Orientador: Ricardo Rocha <ricroc@dcc.fc.up.pt>

¹Center for Research in Advanced Computing Systems

²Faculdade de Ciências da Universidade do Porto

24 de Junho de 2010

- 1 Prolog e o método SLD
 - Limitações
- 2 Tabulação
 - Similaridade de subgolos
 - Exemplo
- 3 Time Stamped tries
 - Implementação
- 4 Tabulação por Subsumção Retroactiva
 - Motivação
 - Tabulação por Subsumção Retroactiva
 - Corte da execução
 - Espaço das tabelas
 - Procura de subgolos específicos
- 5 Resultados
- 6 Conclusões

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*.

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.

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

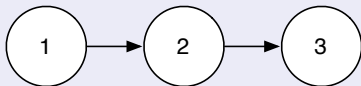
- Na programação em lógica, o método de resolução SLD é um método inerentemente não-determinístico e do tipo *top-down*.
- Em Prolog usa-se o método SLD de forma determinística, avaliando as cláusulas de cima para baixo e da esquerda para a direita.
- Esta forma de avaliação pode ser aplicado de forma 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).
```

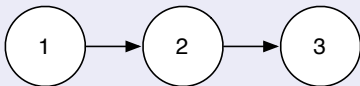


Limitações do método SLD

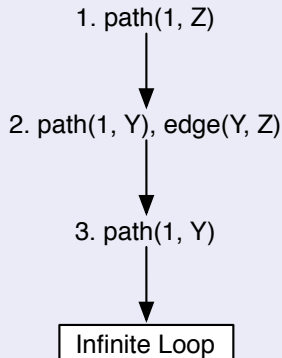
Programa

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

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



?- path(1, Z)



Tabulação

- A tabulação é um refinamento do método de resolução SLD.

Tabulação

- A tabulação é um refinamento do método de resolução SLD.
- As primeiras chamadas a subgoals tabelados são avaliados normalmente através da execução do código do programa.

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.

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 válidos em termos lógicos sejam executáveis.

Similaridade entre chamadas

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

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

Similiaridade entre chamadas

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

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

Example

$p(X, 1, Y)$ e $p(Y, 1, Z)$ são variantes porque ambas podem ser transformadas em $p(VAR_0, 1, VAR_1)$

Similiaridade entre chamadas

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

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

Example

$p(X, 1, Y)$ e $p(Y, 1, Z)$ são variantes porque ambas podem ser transformadas em $p(VAR_0, 1, VAR_1)$

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

Similaridade entre chamadas

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

Example

$p(X, 1, 2)$ 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 subsumção*: A é similar a B quando A é mais específico do que B (ou B é mais geral do que A).

Example

$p(X, 1, 2)$ 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 $p(Y, 1, Z)$.

Theorem

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 subsumção*: A é similar a B quando A é mais específico do que B (ou B é mais geral do que A).

Example

$p(X, 1, 2)$ 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 $p(Y, 1, Z)$.

Theorem

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).
```

Example

1. `path(X, Z)`

Exemplo por subsumção

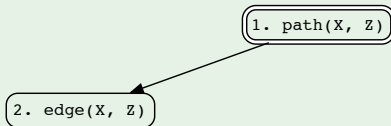
Respostas

• `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).
```

Example



Exemplo por subsumção

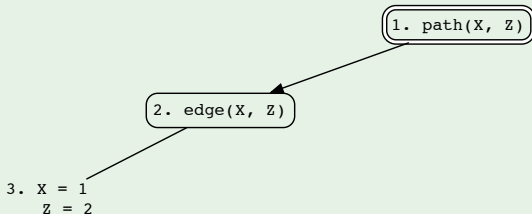
Respostas

• `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).
```

Example



Exemplo por subsumção

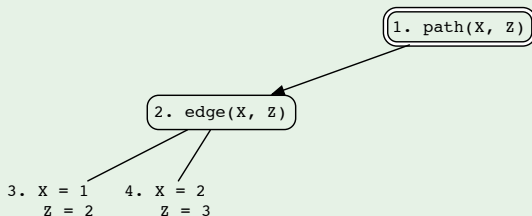
Respostas

- `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).
```

Example



Exemplo por subsumção

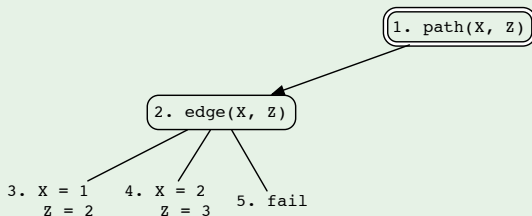
Respostas

- `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).
```

Example



Exemplo por subsumção

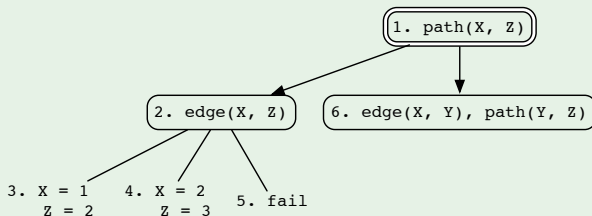
Respostas

- `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).
```

Example



Exemplo por subsumção

Respostas

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

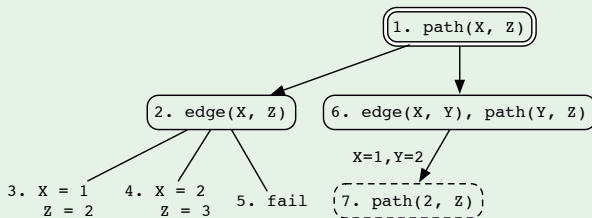
Programa

```

path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).
edge(1, 2). edge(2, 3).

```

Example



Exemplo por subsumção

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`

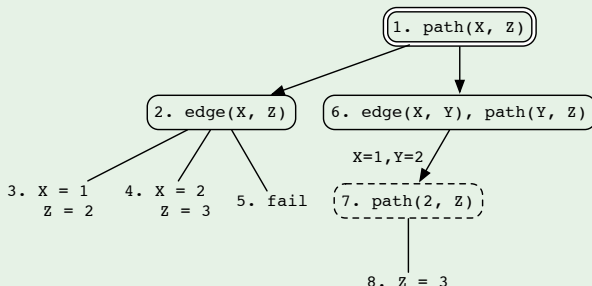
Programa

```

path(X, Z) :- edge(X, Z).
path(X, Z) :- edge(X, Y), path(Y, Z).
edge(1, 2). edge(2, 3).

```

Example



Exemplo por subsumção

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

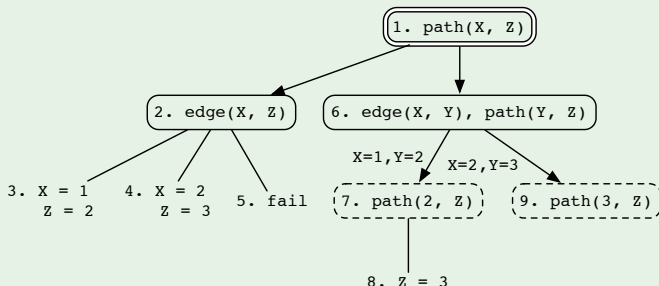
```

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

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

```

Example



Exemplo por subsumção

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

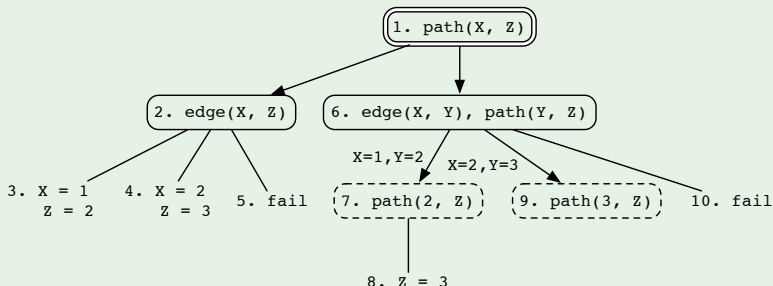
```

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

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

```

Example



Exemplo por subsumção

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

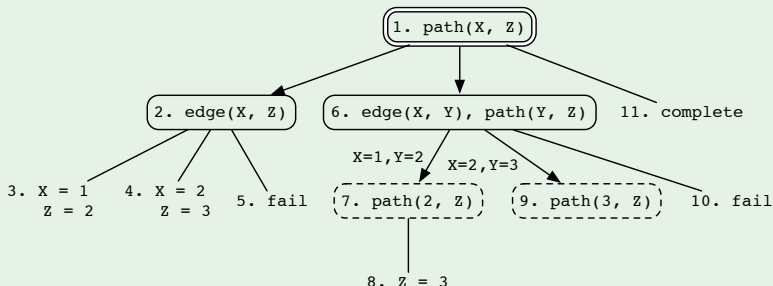
```

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

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

```

Example



Espaço das tabelas

Como são implementadas as tabelas?

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

Espaço das tabelas

Como são implementadas as 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.

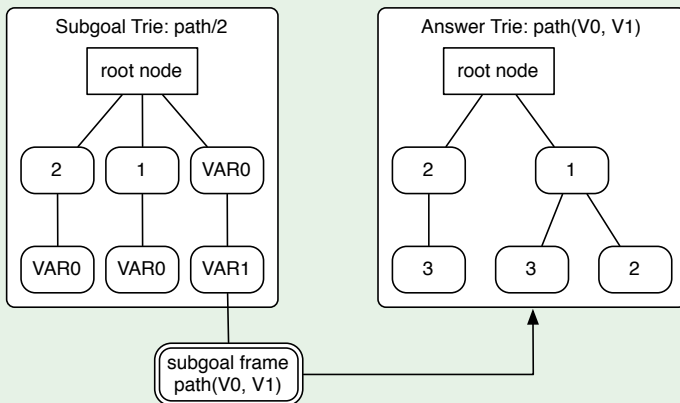
Espaço das tabelas

Como são implementadas as 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.
- Num nó folha da subgoal trie existe uma estrutura chamada *subgoal frame* que contém informação sobre o subgolo respectivo.

Espaço das tabelas

Example



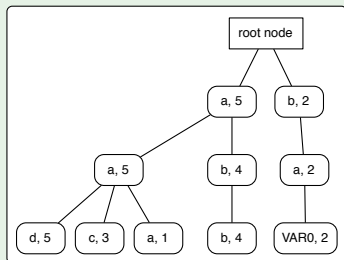
Time Stamped Tries

- Estende-se a answer trie com informação temporal: *timestamps*.
- Quando uma resposta é inserida, incrementa-se o timestamp da resposta.
- O objectivo é permitir uma pesquisa incremental de respostas para os subgolos mais específicos.
- O subgolo específico guarda o timestamp da última procura para evitar respostas repetidas no futuro.
- O algoritmo de pesquisa faz corte dos ramos pelo timestamp e através de operações de unificação ao longo da trie.

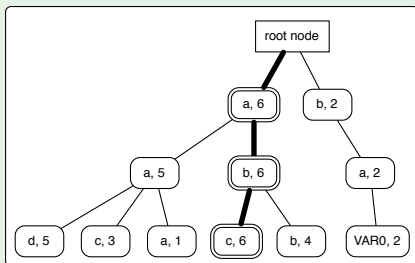
Time Stamped Tries

- Time stamped trie do subgolo $p(X, Y, Z)$:

Example (antes)



Example (inserir $p(a, b, c)$)



Implementação no YapTab

- 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

- 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

- 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

- 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

- 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

- 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.
- Todas as instruções da trie tiveram que ser alteradas para usar unificação por forma a serem usadas em subgolos específicos quando completas.
- O sistema permite usar uma mistura de predicados por variantes e por subsumção.

Problemas na tabulação por subsumção

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

Example

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.

- Assim, para existir partilha de respostas entre subgolos subsumptivos é estritamente necessário que o golo mais específico apareça depois do subgolo mais geral.

Como solucionar este problema?

- Tabulação por Subsumção Retroactiva (TSR).

Como solucionar este problema?

- Tabulação por Subsumção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.
- Assim, G' passa a usar as soluções de G e deixa de gerar as suas próprias soluções.
- O corte de ramos de execução potencia ganhos de tempo de execução e a partilha de respostas ganhos em termos de utilização de memória.

Como solucionar este problema?

- Tabulação por Subsumção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.
- Assim, G' passa a usar as soluções de G e deixa de gerar as suas próprias soluções.
- O corte de ramos de execução potencia ganhos de tempo de execução e a partilha de respostas ganhos em termos de utilização de memória.
- Desafios:
 - ▶ Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G' .

Como solucionar este problema?

- Tabulação por Subsumção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.
- Assim, G' passa a usar as soluções de G e deixa de gerar as suas próprias soluções.
- O corte de ramos de execução potencia ganhos de tempo de execução e a partilha de respostas ganhos em termos de utilização de memória.
- Desafios:
 - ▶ Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G' .
 - ★ Construindo uma árvore das dependências dos subgolos.

Como solucionar este problema?

- Tabulação por Subsumção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.
- Assim, G' passa a usar as soluções de G e deixa de gerar as suas próprias soluções.
- O corte de ramos de execução potencia ganhos de tempo de execução e a partilha de respostas ganhos em termos de utilização de memória.
- Desafios:
 - ▶ Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G' .
 - ★ Construindo uma árvore das dependências dos subgolos.
 - ▶ Manter a execução consistente devido aos cortes.

Como solucionar este problema?

- Tabulação por Subsumção Retroactiva (TSR).
- Quando um subgolo G é chamado, cortam-se os ramos de execução do subgolo mais específico G' para transformar G' num consumidor.
- Assim, G' passa a usar as soluções de G e deixa de gerar as suas próprias soluções.
- O corte de ramos de execução potencia ganhos de tempo de execução e a partilha de respostas ganhos em termos de utilização de memória.
- Desafios:
 - ▶ Determinar que subgolos geradores e consumidores pertencem à execução do subgolo G' .
 - ★ Construindo uma árvore das dependências dos subgolos.
 - ▶ Manter a execução consistente devido aos cortes.
 - ★ Considerando intervalos de pontos de escolha.

Exemplo de TSR

Programa

```
:- use_retroactive_tabling p/2.
```

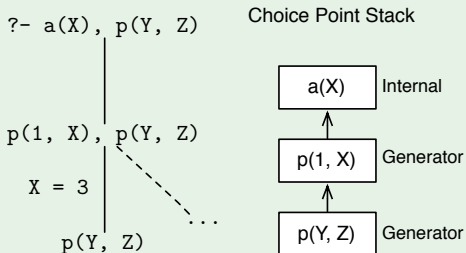
```
a(X) :- p(1, X).
```

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

TSR

Subgolo $p(X, Y)$ é mais geral que $p(1, X)$

Example



Exemplo de TSR

Programa

```
:- use_retroactive_tabling p/2.
```

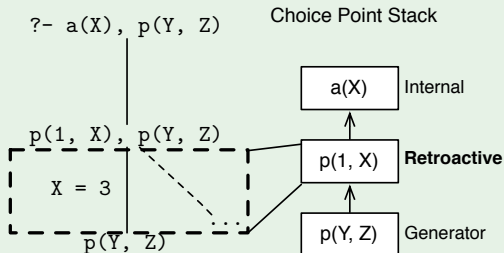
```
a(X) :- p(1, X).
```

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

TSR

Subgolo $p(X, Y)$ torna-se num nó retroactivo

Example



Exemplo de TSR

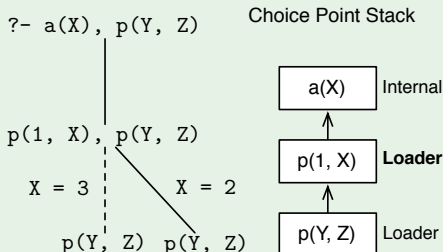
Programa

```
:- use_retroactive_tabling p/2.
a(X) :- p(1, X).
p(1, 3). p(2, 3). p(1, 2).
```

TSR

Dado que o subgolo mais geral completou, o nó retroactivo transforma-se num nó *loader* e carrega as soluções relevantes

Example



Corte da execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
 - ▶ Corte externo: se aparece fora.
 - ▶ 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.
 - ▶ 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*
 - ▶ *Lost consumers*
 - ▶ *Pseudo-Completion*
 - ▶ *Leader Re-Computation*

Corte da execução

- Existem dois tipos de corte dependendo onde o subgolo mais geral aparece relativamente ao subgolo específico:
 - ▶ Corte externo: se aparece fora.
 - ▶ 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*
 - ▶ *Lost consumers*
 - ▶ *Pseudo-Completion*
 - ▶ *Leader Re-Computation*
- Após o corte, o ponto de escolha do subgolo específico é transformado num nó retroactivo, para que possa haver *resolução retroactiva*.
 - ▶ A instrução que implementa resolução retroactiva é vital para a TSR, pois permite que os nós de execução sejam transformados em tipos de nós correctos e desta forma, permitir que a computação termine.

Corte Externo

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)

Example

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

Choice Point Stack

Subgoal Frame Stack

Dependency Space

a(X,Y)

Generator

a(V0,V1)

top_gen

Corte Externo

Programa

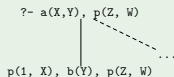
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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)

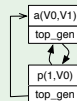
Example



Choice Point Stack



Subgoal Frame Stack



Dependency Space

Corte Externo

Programa

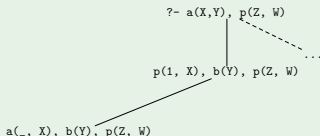
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

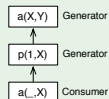
TSR

Novo consumidor a(., X)

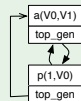
Example



Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

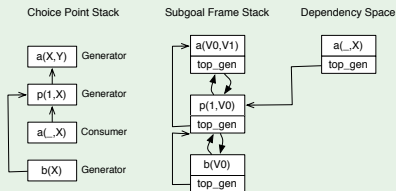
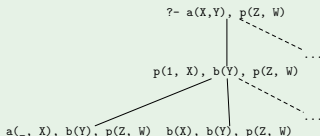
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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)

Example



Corte Externo

Programa

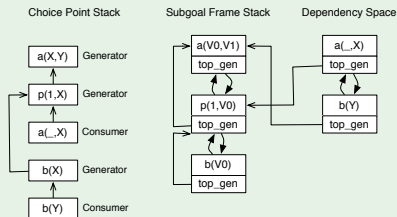
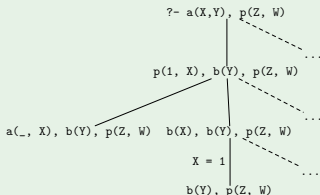
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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)

Example



Corte Externo

Programa

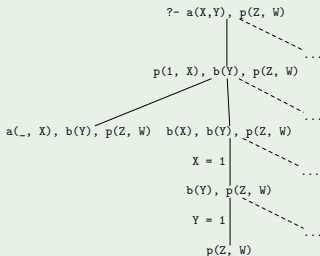
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

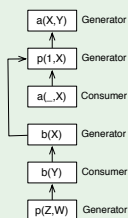
TSR

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

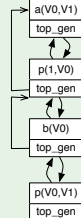
Example



Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

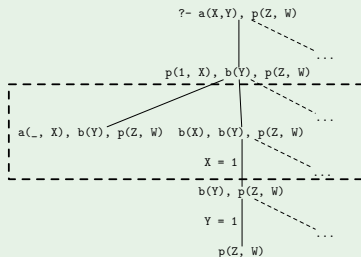
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

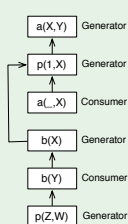
TSR

Determinar ramos a cortar

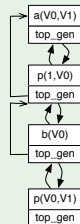
Example



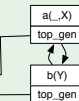
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

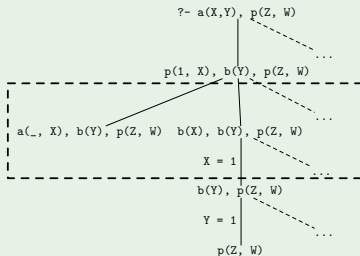
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

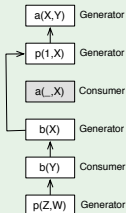
TSR

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

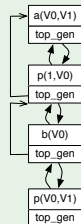
Example



Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

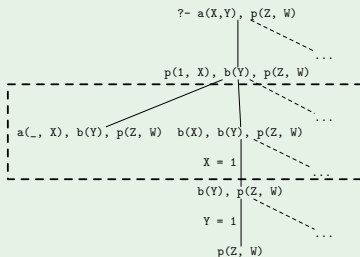
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

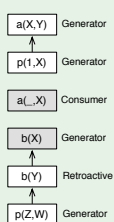
TSR

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

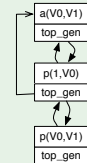
Example



Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

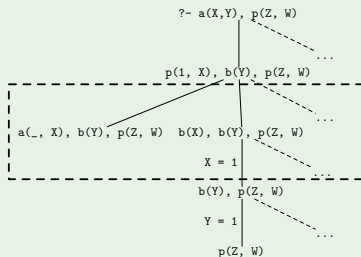
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

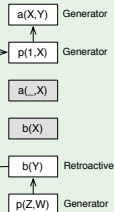
TSR

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

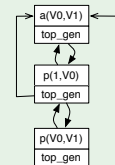
Example



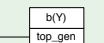
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

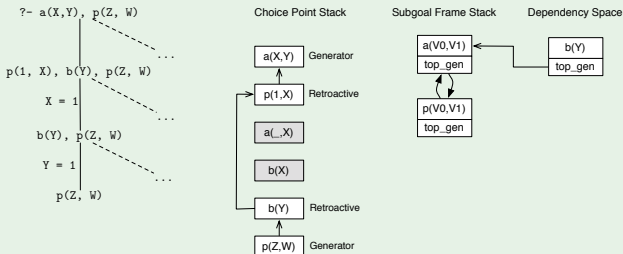
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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 retroativo e remover o *subgoal frame* da pilha respectiva

Example



Corte Externo

Programa

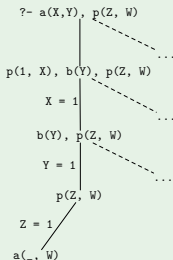
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

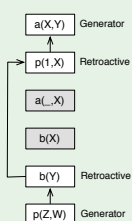
TSR

Novo consumidor $a(_, W)$

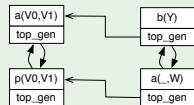
Example



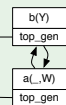
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

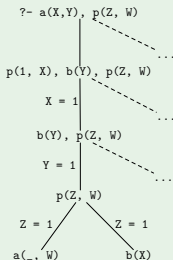
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

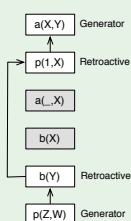
TSR

Gerador $b(V0)$ é reactivado e completa

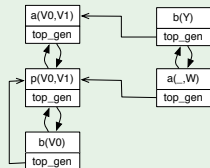
Example



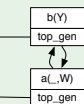
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

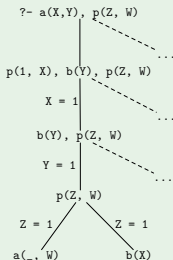
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

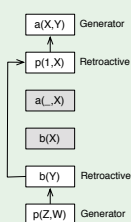
TSR

Subgolo $p(Z, W)$ tenta completar mas não é líder

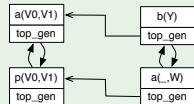
Example



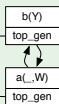
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

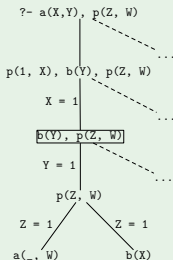
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

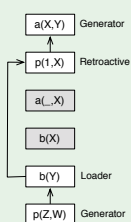
TSR

Após backtracking, o nó retroativo $b(Y)$ executa a instrução de resolução retroativa e transforma-se num nó carregador (*loader*)

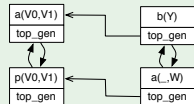
Example



Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

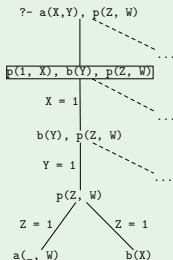
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

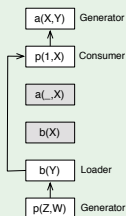
TSR

Ao chegar-mos ao nó $p(1, X)$, este é transformado num consumidor, pois $p(Z, W)$ ainda não completou

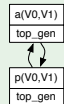
Example



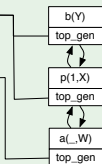
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Externo

Programa

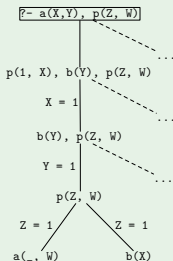
```
:- use_variant_tabling [a/2, b/1].
:- use_retroactive_tabling p/2.
```

```
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).
```

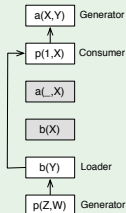
TSR

O subgolo $a(X, Y)$ como líder poderá depois completar a computação em segurança

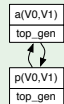
Example



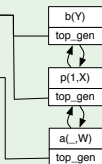
Choice Point Stack



Subgoal Frame Stack



Dependency Space



Corte Interno

- O corte interno acontece quando o subgolo mais geral G aparece dentro da execução do subgolo específico G' .
- Nesta situação cortam-se os ramos referentes a G' , excepto a parte que irá computar as soluções do subgolo G .

¹O subgolo executa usando *local scheduling*

Corte Interno

- O corte interno acontece quando o subgolo mais geral G aparece dentro da execução do subgolo específico G' .
- 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 externo¹.
- Quando G ou completar ou não conseguir completar por ser o líder, salta-se para o ponto de escolha do subgolo G' , onde este poderá carregar as suas soluções relevantes ou transformar-se em consumidor.

¹O subgolo executa usando *local scheduling*

Corte Interno

- O corte interno acontece quando o subgolo mais geral G aparece dentro da execução do subgolo específico G' .
- 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 externo¹.
- Quando G ou completar ou não conseguir completar por ser o líder, salta-se para o ponto de escolha do subgolo G' , onde este poderá carregar as suas soluções relevantes ou transformar-se em consumidor.
- O nosso sistema é capaz de detectar situações de cortes internos múltiplos.

¹O subgolo executa usando *local scheduling*

Espaço das tabelas

- *Single Time Stamped Trie*
- Uma *answer trie* única por predicado.

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.
 - ▶ Situações em que diferentes subgolos estão a inserir respostas na trie requerem cuidado.

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.
 - ▶ Situações em que diferentes subgolos estão a inserir respostas na trie requerem cuidado.
- Permite que se possam reutilizar respostas relevantes a um novo subgolo gerador antes de executar o código.
 - ▶ Esta é uma forma elegante de reutilizar respostas e resolver o problema das tabelas incompletas.

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.
 - ▶ Situações em que diferentes subgolos estão a inserir respostas na trie requerem cuidado.
- Permite que se possam reutilizar respostas relevantes a um novo subgolo gerador antes de executar o código.
 - ▶ Esta é uma forma elegante de reutilizar respostas e resolver o problema das tabelas incompletas.
- Tem como desvantagem a necessidade de representar todos os argumentos de um dado subgolo e não apenas o valor das variáveis.

Procura de subgolos específicos

- Uma componente importante da TSR é o algoritmo que percorre a *subgoal trie* para encontrar subgolos mais específicos.

Procura de subgolos específicos

- Uma componente importante da TSR é o algoritmo que percorre a *subgoal trie* para encontrar subgolos mais específicos.
- O problema resume-se a encontrar atribuições para as variáveis do subgolo mais geral.

Procura de subgolos específicos

- Uma componente importante da TSR é o algoritmo que percorre a *subgoal trie* para encontrar subgolos mais específicos.
- O problema resume-se a encontrar atribuições para as variáveis do subgolo mais geral.
- A pesquisa é feita navegando pela trie e usando backtracking sempre que a pesquisa falhar.
- Usa-se uma pilha de nós alternativos de pesquisa.

Procura de subbolos específicos

- Dado que é necessário construir termos Prolog e registar atribuições de variáveis durante a pesquisa, usa-se estruturas da máquina virtual, tais como a heap e a trilha.

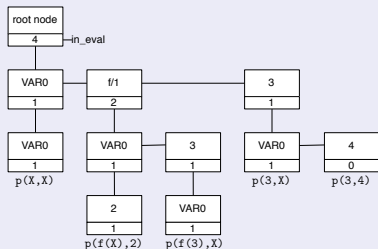
Procura de subgolos específicos

- Dado que é necessário construir termos Prolog e registar atribuições de variáveis durante a pesquisa, usa-se estruturas da máquina virtual, tais como a heap e a trilha.
- Para melhorar a eficiência, estendeu-se cada nó da *subgoal trie* com um campo chamado `in_eval` que regista o número de subgolos sobre aquele ramo da trie que estão a executar.

Procura de subgolos específicos

- Dado que é necessário construir termos Prolog e registar atribuições de variáveis durante a pesquisa, usa-se estruturas da máquina virtual, tais como a heap e a trilha.
- Para melhorar a eficiência, estendeu-se cada nó da *subgoal trie* com um campo chamado *in_eval* que regista o número de subgolos sobre aquele ramo da trie que estão a executar.

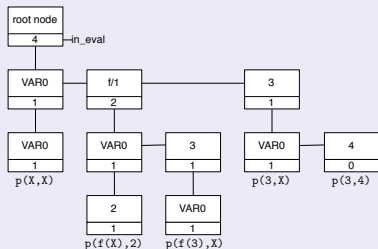
Subgoal trie



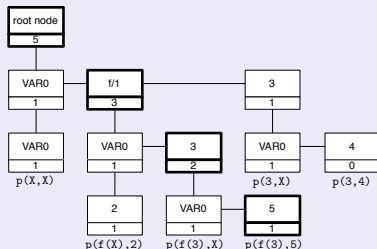
Procura de subgolos específicos

- Dado que é necessário construir termos Prolog e registar atribuições de variáveis durante a pesquisa, usa-se estruturas da máquina virtual, tais como a heap e a trilha.
- Para melhorar a eficiência, estendeu-se cada nó da *subgoal trie* com um campo chamado *in_eval* que regista o número de subgolos sobre aquele ramo da trie que estão a executar.

Subgoal trie



Novo gerador



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.

Tabulação por subsumção

- Avaliou-se o desempenho do motor de tabulação por subsumção 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 <i>Speedup médio</i>	Yap Prolog <i>Speedup médio</i>
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 subsumção tradicional e tabulação por variantes.

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 subsumção tradicional e tabulação por variantes.

Programa	Yap Prolog	
	<i>Retro / Var</i>	<i>Retro / Sub</i>
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
<i>Média Total</i>	1.10	1.05

Ganhos da TSR

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

Ganhos da TSR

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

Programa	Yap Prolog	
	<i>Var / Retro</i>	<i>Sub / Retro</i>
left_first	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

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

Conclusões

- Contribuições:

Conclusões

- Contribuições:
 - ▶ YapTab suporta tabulação por subsumção.

Conclusões

- Contribuições:
 - ▶ YapTab suporta tabulação por subsumção.
 - ▶ Mecanismos e algoritmos que controlam a execução retroactiva.

Conclusões

- Contribuições:
 - ▶ YapTab suporta tabulação por subsumção.
 - ▶ Mecanismos e algoritmos que controlam a execução retroactiva.
 - ▶ Algoritmo de pesquisa de subgolos específicos.

Conclusões

- Contribuições:

- ▶ YapTab suporta tabulação por subsumção.
- ▶ Mecanismos e algoritmos que controlam a execução retroactiva.
- ▶ Algoritmo de pesquisa de subgoals específicos.
- ▶ Espaço de tabelas inovador que permite uma maior reutilização de respostas.

Conclusões

- Contribuições:

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

Conclusões

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

Conclusões

- Contribuições:

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

- Trabalho futuro:

- ▶ Integrar o trabalho na distribuição oficial do Yap Prolog.

Conclusões

- Contribuições:

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

- Trabalho futuro:

- ▶ Integrar o trabalho na distribuição oficial do Yap Prolog.
- ▶ Melhoramento dos algoritmos do espaço das tabelas.

Conclusões

- Contribuições:

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

- Trabalho futuro:

- ▶ Integrar o trabalho na distribuição oficial do Yap Prolog.
- ▶ Melhoramento dos algoritmos do espaço das tabelas.
- ▶ Maior experimentação com aplicações reais.

Conclusões

- Contribuições:

- ▶ YapTab suporta tabulação por subsumção.
- ▶ 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 subsumção.

- 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*.

Conclusões

- Contribuições:

- ▶ YapTab suporta tabulação por subsumção.
- ▶ 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 subsumção.

- 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.