Efficient Retrieval of Subsumed Subgoals in Tabled Logic Programs

Flávio Cruz and Ricardo Rocha

CRACS & INESC-Porto LA
Faculty of Sciences, University of Porto

INForum/CoRTA 2010

Tabling in Logic Programming

Prolog and SLD Resolution

- The operational semantics of Prolog are based on **SLD resolution**, where clauses are evaluated in a top-down fashion, from left to right.
- However, some perfectly logical programs can't be evaluated due to the limitations of traditional Prolog systems based on SLD resolution.
  - Programs with **positive loops**.
  - Programs with **negative loops**.

Tabling in Logic Programming

Programs with Positive Loops

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

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

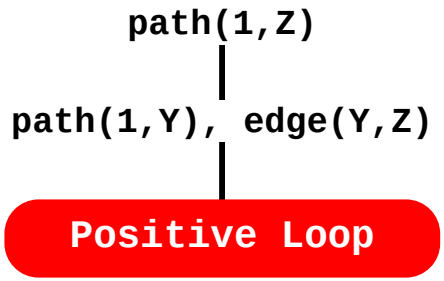
```
path(1,Z)
```

Tabling in Logic Programming

Programs with Positive Loops

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

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



Tabling in Logic Programming

Programs with Positive Loops

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

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

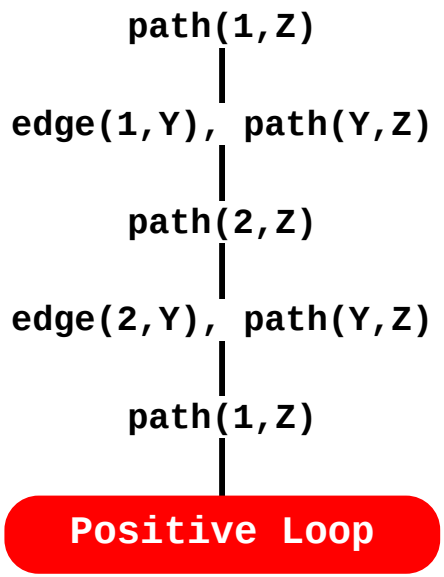
```
path(1,Z)
```

Tabling in Logic Programming

Programs with Positive Loops

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

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



Tabling in Logic Programming

Tabled Resolution

- Tabling is an implementation technique that overcomes some limitations of traditional Prolog systems in dealing with recursion and redundant sub-computations.
- It extends the standard SLD resolution method by adding new tabling operations.

Tabling in Logic Programming

Tabled Resolution

- Tabling is an implementation technique that overcomes some limitations of traditional Prolog systems in dealing with recursion and redundant sub-computations.
- It extends the standard SLD resolution method by adding new tabling operations.
  - First calls to tabled subgoals are evaluated as usual through the execution of code but answers are inserted into a **table space**.

Tabling in Logic Programming

Tabled Resolution

- Tabling is an implementation technique that overcomes some limitations of traditional Prolog systems in dealing with recursion and redundant sub-computations.
- It extends the standard SLD resolution method by adding new tabling operations.
  - First calls to tabled subgoals are evaluated as usual through the execution of code but answers are inserted into a **table space**.
  - **Similar calls** are evaluated by consuming answers from the table space that were generated by the corresponding similar subgoal, instead of re-evaluating them against the program clauses.

Tabling in Logic Programming

Tabled Resolution

- Tabling is an implementation technique that overcomes some limitations of traditional Prolog systems in dealing with recursion and redundant sub-computations.
- It extends the standard SLD resolution method by adding new tabling operations.
  - First calls to tabled subgoals are evaluated as usual through the execution of code but answers are inserted into a **table space**.
  - **Similar calls** are evaluated by consuming answers from the table space that were generated by the corresponding similar subgoal, instead of re-evaluating them against the program clauses.
  - Similar calls are found by using a **call similarity test** which determines if a subgoal will be a **generator** or a **consumer**.

Tabling in Logic Programming

Call Similarity

- In general, we can distinguish two main approaches to determine similarity between tabled subgoals.

Tabling in Logic Programming

Call Similarity

- In general, we can distinguish two main approaches to determine similarity between tabled subgoals.
  - **Variant-Based Tabling:** subgoal A is similar to B if they are the same by renaming the variables.

Example

p(X,1,Y) and **p(Y,1,Z)** are **variants** because both can be transformed into **p(VAR0,1,VAR1)**.

Tabling in Logic Programming

Call Similarity

- In general, we can distinguish two main approaches to determine similarity between tabled subgoals.
  - **Variant-Based Tabling:** subgoal A is similar to B if they are the same by renaming the variables.

Example

p(X,1,Y) and **p(Y,1,Z)** are **variants** because both can be transformed into **p(VAR0,1,VAR1)**.

  - **Subsumption-Based Tabling:** subgoal A is similar to B if A is more specific than B (or B is more general than A).

Example

p(X,1,2) is more specific than **p(Y,1,Z)** because there is a **substitution** {**Y=X, Z=2**} that makes **p(X,1,2)** an **instance** of **p(Y,1,Z)**.

Subsumption-Based Tabling

Advantages

- **Less code is executed** because subsumed subgoals can reuse answers instead of executing their own code.
- **More answers are shared** across subgoals, therefore there is less redundancy in the table space.

Subsumption-Based Tabling

Disadvantages

- The mechanisms to support subsumption-based tabling are **harder to implement**.
- If a more general subgoal is called before specific subgoals, answer reuse will happen, but if more specific subgoals are called before a more general subgoal, **no reuse will occur**.

Example

If **p(1,X)** is called **before p(X,Y)**, **p(1,X) will not reuse** the answers from **p(X,Y)**, but will execute code to generate its own answers.

Subsumption-Based Tabling

Retroactive Call Subsumption

- We have developed a new resolution extension called **Retroactive Call Subsumption (RCS)** that supports subsumption-based tabling by allowing full sharing of answers among subsumptive subgoals, independently of the order they are called.

Example

If **p(1,X)** is called **before or after p(X,Y)**, **p(1,X) will reuse** the answers from **p(X,Y)**.

Subsumption-Based Tabling

Retroactive Call Subsumption

- We have developed a new resolution extension called **Retroactive Call Subsumption (RCS)** that supports subsumption-based tabling by allowing full sharing of answers among subsumptive subgoals, independently of the order they are called.

Example

If **p(1,X)** is called **before or after p(X,Y)**, **p(1,X) will reuse** the answers from **p(X,Y)**.

- RCS selectively prunes the evaluation of a subgoal G' when a more general subgoal G appears later on.

Subsumption-Based Tabling

Retroactive Call Subsumption

- We have developed a new resolution extension called **Retroactive Call Subsumption (RCS)** that supports subsumption-based tabling by allowing full sharing of answers among subsumptive subgoals, independently of the order they are called.

Example

If **p(1,X)** is called **before or after p(X,Y)**, **p(1,X) will reuse** the answers from **p(X,Y)**.

- RCS selectively prunes the evaluation of a subgoal G' when a more general subgoal G appears later on.
- RCS works by pruning the execution branch of G' and then by restarting the evaluation of G' as a consumer. By doing that, we **save execution time** by not executing code that would generate a subset of the answers we can find by executing G.

Retroactive Call Subsumption

Challenges

- Compute the set of subsumed subgoals that are currently executing.
  - New algorithms and extensions to the table space to efficiently retrieve the set of subsumed subgoals.

Retroactive Call Subsumption

Challenges

- Compute the set of subsumed subgoals that are currently executing.
  - New algorithms and extensions to the table space to efficiently retrieve the set of subsumed subgoals.
- Keep execution consistent after pruning evaluation.
  - Build a subgoal dependency tree.
  - Update the low-level stacks related to the pruned subgoals.
  - New operations and evaluation strategies that can handle multiple scenarios in order to ensure correct completion.

Retroactive Call Subsumption

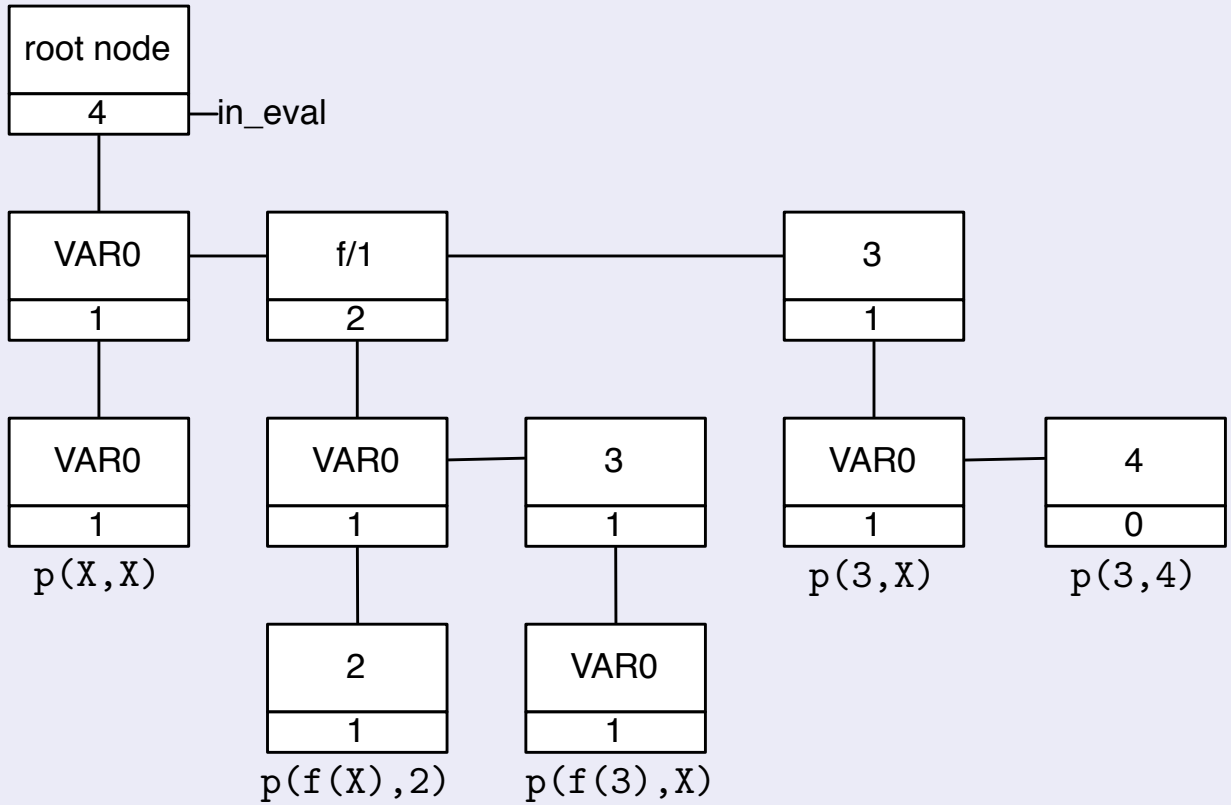
Challenges

- Compute the set of subsumed subgoals that are currently executing.
  - New algorithms and extensions to the table space to efficiently retrieve the set of subsumed subgoals.
- Keep execution consistent after pruning evaluation.
  - Build a subgoal dependency tree.
  - Update the low-level stacks related to the pruned subgoals.
  - New operations and evaluation strategies that can handle multiple scenarios in order to ensure correct completion.
- Ensure that new consumers will not consume repeated answers.
  - New table space organization where answers are represented only once.

Efficient Retrieval of Subsumed Subgoals

New Algorithms and Extensions

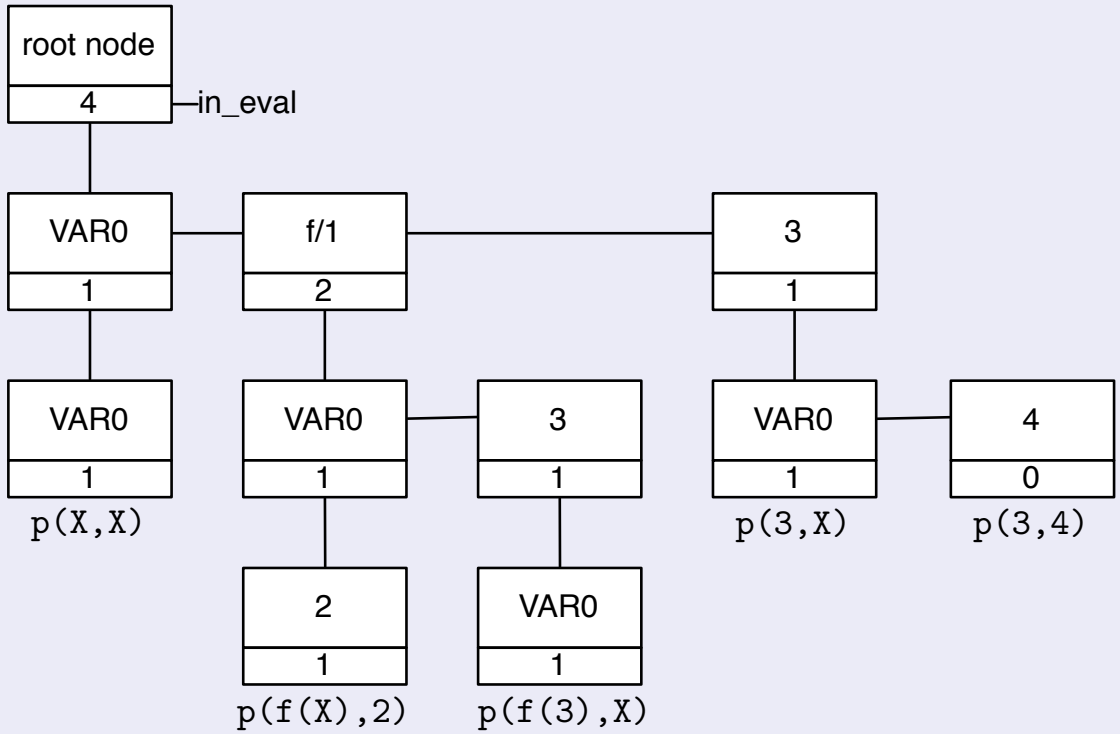
- Each subgoal table node has a field called **in_eval** to indicate how many running subgoals are under it. The matching algorithm then uses this information to prune subgoals during search.



Efficient Retrieval of Subsumed Subgoals

New Algorithms and Extensions

- The matching algorithm finds the running subgoals that are subsumed by a more general subgoal G by matching the arguments of G against the subgoals in the table space (for example, **p(f(X),2)** and **p(f(3),X)** are both subsumed by **p(f(X),Y)**).



Efficient Retrieval of Subsumed Subgoals

Preliminary Results

Program	Yap Prolog	
	Var / RCS	Sub / RCS
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**

Efficient Retrieval of Subsumed Subgoals

Conclusions

- We presented a new algorithm for the efficient retrieval of subsumed subgoals in tabled logic programs.
- Our proposal takes advantage of the existent tabling engine machinery and table space data structures.
- Preliminary results show that our proposal can achieve good results for programs where retroactive subsumption happens.
- Further work will include improvements to the table space design and refinements to the evaluation algorithms. We also plan to explore how this new strategy impacts with other applications.