Tabling Logic Programs in a Common Global Trie

Jorge Costa and Ricardo Rocha
DCC-FC & CRACS
University of Porto, Portugal

c0607002@alunos.dcc.fc.up.pt   ricroc@dcc.fc.up.pt
Motivation

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.
Motivation

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.

- The performance of tabled evaluation largely depends on the implementation of the table space. Arguably, the most successful data structure for tabling is **tries**.
Motivation

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a table space and then reused when a repeated call appears.

- The performance of tabled evaluation largely depends on the implementation of the table space. Arguably, the most successful data structure for tabling is **tries**.

- However, while tries are efficient for variant based tabled evaluation, they are limited in their ability to recognize and represent repeated answers for different calls.
Motivation

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.

- The performance of tabled evaluation largely depends on the implementation of the table space. Arguably, the most successful data structure for tabling is **tries**.

- However, while tries are efficient for variant based tabled evaluation, they are **limited in their ability to recognize and represent repeated answers for different calls**.

- In this work, we propose a new design for the table space where **all tabled subgoal calls and tabled answers are stored in a common global trie** instead of being spread over several different trie data structures.
Motivation

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.

- The performance of tabled evaluation largely depends on the implementation of the table space. Arguably, the most successful data structure for tabling is **tries**.

- However, while tries are efficient for variant based tabled evaluation, they are limited in their ability to recognize and represent repeated answers for different calls.

- In this work, we propose a new design for the table space where **all tabled subgoal calls and tabled answers are stored in a common global trie** instead of being spread over several different trie data structures.

- We will focus our discussion on a concrete implementation, the **YapTab system**, but our proposal can be generalized and applied to other tabling engines.
Table Space

➢ Can be accessed to:

♦ Look up if a subgoal is in the table, and if not insert it.
♦ Look up if a newly found answer is in the table, and if not insert it.
♦ Load answers for repeated subgoals.

➢ Implementation requirements:

♦ Fast look-up and insertion methods.
♦ Compactness in representation of logic terms.
Using Tries to Represent Terms

- Tries are trees in which common prefixes are represented only once.
Tries are trees in which common prefixes are represented only once.

The entry point is called the root node, internal nodes represent symbols in terms and leaf nodes specify completed terms.

Using Tries to Represent Terms
Tries are trees in which common prefixes are represented only once.

- The entry point is called the root node, internal nodes represent symbols in terms and leaf nodes specify completed terms.
- Each different path through the nodes in the trie corresponds to a term. Terms with common prefixes branch off from each other at the first distinguishing symbol.
Using Tries to Organise the Table Space

Subgoal Trie Structure

- Stores the tabled subgoal calls.
- Starts at a table entry and ends with subgoal frames.
- A subgoal frame is the entry point for the subgoal answers.

:- table t/2.
  t(X,Y) :- term(X), term(Y).
  term(a(1)).
  term(a(2)).
Using Tries to Organise the Table Space

Subgoal Trie Structure
- Stores the tabled subgoal calls.
- Starts at a table entry and ends with subgoal frames.
- A subgoal frame is the entry point for the subgoal answers.

Answer Trie Structure
- Stores the subgoal answers.
- Answer tries hold just the substitution terms for the free variables which exist in the argument terms of the corresponding subgoal call.
Common Global Trie

Global Trie Structure

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT)** instead of being spread over several different trie data structures.
- The GT data structure still is a tree structure where each different path through the trie nodes corresponds to a subgoal call and/or answer.
- However, here a path can end at any internal trie node and not necessarily at a leaf trie node.
The original subgoal trie and answer trie data structures are now represented by a unique level of trie nodes that point to the corresponding paths in the GT.

For the subgoal tries, each node is a pointer to the GT’s path representing the subgoal call.

For the answer tries, each node is a pointer to the GT’s path representing the answer.
Implementation Details: Tabling Operations

The table space can be accessed to:

- **Look up if a subgoal is in the table, and if not insert it.**
- **Look up if a newly found answer is in the table, and if not insert it.**
- **Load answers for repeated subgoals.**

```c
subgoal_check_insert(TABLE_ENTRY te, SUBGOAL_CALL call) {
    if (GT) { // GT table design
        leaf_gt_node = trie_check_insert(GT, call)
        leaf_st_node = trie_check_insert(te, leaf_gt_node)
    } else { // original table design
        leaf_st_node = trie_check_insert(te, call)
    }
    return leaf_st_node
}
```
Implementation Details: Tabling Operations

- The table space can be accessed to:
  - Look up if a subgoal is in the table, and if not insert it.
  - **Look up if a newly found answer is in the table, and if not insert it.**
  - Load answers for repeated subgoals.

```c
answer_check_insert(SUBGOAL_FRAME sf, ANSWER answer) {
    if (GT) {
        leaf_gt_node = trie_check_insert(GT, answer)
        leaf_at_node = trie_check_insert(sf, leaf_gt_node)
    } else { // original table design
        leaf_at_node = trie_check_insert(sf, answer)
    }
    return leaf_at_node
}
```
Implementation Details: Tabling Operations

The table space can be accessed to:

- Look up if a subgoal is in the table, and if not insert it.
- Look up if a newly found answer is in the table, and if not insert it.
- **Load answers for repeated subgoals.**

```c
answer_load(ANSWER_TRIE_NODE leaf_at_node) {
    if (GT) { // GT table design
        leaf_gt_node = leaf_at_node->symbol
        answer = trie_load(leaf_gt_node)
    } else // original table design
        answer = trie_load(leaf_at_node)
    return answer
}
```
Implementation Details: Two Small Problems ;-) 

How to deal with table abolish operations.
Implementation Details: Two Small Problems ;-)

- How to deal with **table abolish operations**.
- How to support the **completed table optimization**, an optimization that implements answer recovery by top-down traversing the completed answer trie and by executing specific WAM-like code from the answer trie nodes.
## Experimental Results

<table>
<thead>
<tr>
<th>Terms</th>
<th>YapTab Mem</th>
<th>YapTab Store</th>
<th>YapTab Load</th>
<th>YapTab+GT Mem</th>
<th>YapTab+GT Store</th>
<th>YapTab+GT Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 ints</td>
<td>49074</td>
<td>238</td>
<td>88</td>
<td>1.08</td>
<td>1.29</td>
<td>1.05</td>
</tr>
<tr>
<td>500 atoms</td>
<td>49074</td>
<td>256</td>
<td>88</td>
<td>1.08</td>
<td>1.18</td>
<td>1.05</td>
</tr>
<tr>
<td>500 f/1</td>
<td>49172</td>
<td>336</td>
<td>176</td>
<td>1.07</td>
<td>1.33</td>
<td>0.77</td>
</tr>
<tr>
<td>500 f/2</td>
<td>98147</td>
<td>430</td>
<td>190</td>
<td>0.58</td>
<td>1.16</td>
<td>0.82</td>
</tr>
<tr>
<td>500 f/3</td>
<td>147122</td>
<td>554</td>
<td>220</td>
<td>0.41</td>
<td>1.04</td>
<td>0.80</td>
</tr>
<tr>
<td>500 f/4</td>
<td>196097</td>
<td>596</td>
<td>210</td>
<td>0.33</td>
<td>1.07</td>
<td>0.94</td>
</tr>
<tr>
<td>500 f/5</td>
<td>245072</td>
<td>676</td>
<td>258</td>
<td>0.28</td>
<td>1.00</td>
<td>0.84</td>
</tr>
<tr>
<td>500 f/6</td>
<td>294047</td>
<td>796</td>
<td>290</td>
<td>0.25</td>
<td>1.01</td>
<td>0.83</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>0.64</td>
<td>1.14</td>
<td>0.89</td>
</tr>
</tbody>
</table>

Memory usage in KBytes and store/load times in milliseconds for a t/5 tabled predicate that simply stores in the table space terms defined by term/1 facts, called recursively with all combinations of one and two free variables in the arguments.
Conclusions and Further Work

- We have presented a new design for the table space organization where all tabled subgoal calls and tabled answers are stored in a common global trie instead of being spread over several different trie data structures.

- Our goal is to reduce redundancy in term representation, thus saving memory by sharing data that is structurally equal.

- Our preliminary experiments showed very significant reductions on memory usage.

- As further work we intend to study how alternative designs for the table space organization can efficiently solve our two small problems and/or further reduce redundancy in term representation.
Conclusions and Further Work

- Table entry for t/2
  - VAR0
  - VAR1

- Subgoal frame for \( t(VAR0, VAR1) \)
  - a/1

- Answer trie
  - 2
  - 2
  - 1
  - 1
Conclusions and Further Work

**Table entry for t/2**

- **arg1**
- **arg2**

**Subgoal frame for t(VAR0, VAR1)**

- **subs1**
- **subs2**

**Answer trie**

**Global trie**

- **VAR1**
- **VAR0**
- **a/1**
- **2**
- **1**