Global Trie for Subterms

João Raimundo and Ricardo Rocha
CRACS & INESC TEC
University of Porto, Portugal

CICLOPS 2011, Lexington, Kentucky, USA, July 2011
Tabling is an implementation technique that overcomes some limitations of traditional Prolog systems in dealing with redundant sub-computations and recursion.

Tabling works by storing found answers in a memory area called the table space and then by reusing those answers in similar calls that appear during the resolution process.
Motivation

A critical component in the implementation of an efficient tabling system is the design of the table space. The most popular and successful data structure for representing tables is based on a two-level trie data structure, where one trie level stores the tabled subgoal calls and the other stores the tabled answers.
Motivation

- A critical component in the implementation of an efficient tabling system is the design of the table space. The most popular and successful data structure for representing tables is based on a two-level trie data structure, where one trie level stores the tabled subgoal calls and the other stores the tabled answers.

- The Global Trie (GT) is an alternative table space organization where tabled subgoal calls and tabled answers are represented only once in a global trie instead of being spread over several different trie data structures.
Motivation

- A critical component in the implementation of an efficient tabling system is the design of the table space. The most popular and successful data structure for representing tables is based on a two-level *trie data structure*, where one trie level stores the *tabled subgoal calls* and the other stores the *tabled answers*.

- The *Global Trie (GT)* is an alternative table space organization where tabled subgoal calls and tabled answers *are represented only once in a global trie* instead of being spread over several different trie data structures.

- In this work, we propose an extension to the GT organization, named *Global Trie for Subterms (GT-ST)*, where compound subterms in term arguments are represented as unique entries in the GT.
Motivation

A critical component in the implementation of an efficient tabling system is the design of the table space. The most popular and successful data structure for representing tables is based on a two-level trie data structure, where one trie level stores the tabled subgoal calls and the other stores the tabled answers.

The Global Trie (GT) is an alternative table space organization where tabled subgoal calls and tabled answers are represented only once in a global trie instead of being spread over several different trie data structures.

In this work, we propose an extension to the GT organization, named Global Trie for Subterms (GT-ST), where compound subterms in term arguments are represented as unique entries in the GT.

Our new design extends our previous approaches [PADL’09, ICLP’09], where we first introduced the idea of using a global trie.
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 similar 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.
- 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 token.
Using Tries to Represent Terms

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

![Trie Diagram]

```
        root
         ↓
         t/2
          ↓
          f/1
           ↓
           1
            ↓
           VAR0
```

Inserting $t(f(1), Y)$
Using Tries to Represent Terms

- Tries are trees in which common prefixes are represented only once.
- 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 token.
Using Tries to Represent the Table Space

Subgoal Trie

- 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.

```prolog
:- table t/2.
t(X, Y) :- term(X),
         term(Y).
term(f(1)).
term(f(2)).
```

![Subgoal Trie Diagram]

- Table entry for t/2
- Subgoal frame for t(VAR0, VAR1)
- Subgoal frame for t(f(1), VAR0)
- Subgoal trie
Using Tries to Represent the Table Space

Answer Trie

- Stores the subgoal answers.
- Answer tries hold just the substitution terms for the free variables which exist in the corresponding subgoal call.

```prolog
:- table t/2.
t(X,Y) :- term(X), term(Y).
term(f(1)).
term(f(2)).
```
Using Tries to Represent the Table Space

- Table entry for t/2
  - VAR0
  - VAR1

- Subgoal frame for t(VAR0,VAR1)
  - f/1
  - 2
  - 1
  - subgoal trie

- Subgoal frame for t(f(1),VAR0)
  - answer trie
  - f/1
  - 2
  - 1
GT-T: Global Trie for Terms [ICLP’09]

➢ The GT-T was designed in order to maximize the sharing of tabled data which is structurally equal.

➢ In GT-T, all argument and substitution compound terms appearing in tabled subgoal calls and/or answers are represented only once in the GT, thus preventing situations where these terms are represented more than once in different trie data structures.
GT-T: Global Trie for Terms [ICLP’09]

- The GT-T was designed in order to maximize the sharing of tabled data which is structurally equal.
- In GT-T, all argument and substitution compound terms appearing in tabled subgoal calls and/or answers are represented only once in the GT, thus preventing situations where these terms are represented more than once in different trie data structures.
- Each path in the original subgoal and answer tries is composed of a fixed number of trie nodes representing the number of argument or substitution terms in the corresponding tabled subgoal call or answer.
  - For example, for the subgoal tries, the node representing the argument term ARGi stores either ARGi, if ARGi is a simple term (an atom, integer or variable term), or the reference to the path’s leaf node in the GT representing ARGi, if ARGi is a compound (non-simple) term.
GT-T: Global Trie for Terms [ICLP’09]

Table entry for \( t/2 \)

Subgoal frame for \( t(\text{VAR0, VAR1}) \)

Subgoal frame for \( t(f(1), \text{VAR0}) \)

Global trie

f/1

2 → 1

Subgoal trie

CICLOPS 2011, Lexington, Kentucky, USA, July 2011
The completed table optimization, that implements answer recovery by executing specific WAM-like instructions to top-down traversing the answer tries, in the GT-T design, works at the level of the substitution terms, instead of working at the level of the term’s tokens as in the original table design.

Regarding space reclamation, GT-T uses the child field of the leaf nodes (that is always NULL) to count the number of references to the path it represents.
GT-ST: Global Trie for Subterms

- The GT-ST design maintains most of the GT-T features, but tries to optimize GT’s memory usage by representing compound subterms in term arguments as unique entries in the GT.

- The GT-ST maximizes the sharing of the tabled data that is structurally equal at a second level, by avoiding the representation of equal compound subterms, and thus preventing situations where the representation of those subterms occur more than once.
GT-ST: Global Trie for Subterms

- The GT-ST design maintains most of the GT-T features, but tries to optimize GT’s memory usage by representing compound subterms in term arguments as unique entries in the GT.

- The GT-ST maximizes the sharing of the tabled data that is structurally equal at a second level, by avoiding the representation of equal compound subterms, and thus preventing situations where the representation of those subterms occur more than once.

- Although GT-ST uses the same GT-T’s tree structure for implementing the GT, every different path in the GT can now represent a complete term or a subterm of another term, but still being an unique term.
Consider, for example, the insertion of the terms \( f(g(1),g(1)) \) and \( f(g(2),g(2)) \) in the GT-T...

```prolog
:- table t/2.

\( t(X,Y) :\neg \text{term}(X), \\text{term}(Y). \)
\( \text{term}(f(g(1),g(1))). \)
\( \text{term}(f(g(2),g(2))). \)
```

![Global Trie for Subterms Diagram]
... and in the GT-ST.

```prolog
:- table t/2.
t(X, Y) :- term(X),
          term(Y).
term(f(g(1), g(1))).
term(f(g(2), g(2))).
```
GT-ST: Global Trie for Subterms

- In more detail, after storing the node representing \( f/2 \), the process is suspended and the subterm \( g(1) \) is inserted as an individual term in the GT.

- After the complete insertion of subterm \( g(1) \) in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents \( g(1) \) in the GT.

- The construction of the main term then continues by applying an analogous procedure to the second argument \( g(1) \).
In more detail, after storing the node representing $f/2$, the process is suspended and the subterm $g(1)$ is inserted as an individual term in the GT.

After the complete insertion of subterm $g(1)$ in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents $g(1)$ in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument $g(1)$. 
In more detail, after storing the node representing \( f/2 \), the process is suspended and the subterm \( g(1) \) is inserted as an individual term in the GT.

After the complete insertion of subterm \( g(1) \) in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents \( g(1) \) in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument \( g(1) \).
In more detail, after storing the node representing $f/2$, the process is suspended and the subterm $g(1)$ is inserted as an individual term in the GT.

After the complete insertion of subterm $g(1)$ in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents $g(1)$ in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument $g(1)$.
In more detail, after storing the node representing $f/2$, the process is suspended and the subterm $g(1)$ is inserted as an individual term in the GT.

After the complete insertion of subterm $g(1)$ in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents $g(1)$ in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument $g(1)$.
In more detail, after storing the node representing \( f/2 \), the process is suspended and the subterm \( g(1) \) is inserted as an individual term in the GT.

After the complete insertion of subterm \( g(1) \) in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents \( g(1) \) in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument \( g(1) \).
In more detail, after storing the node representing $f/2$, the process is suspended and the subterm $g(1)$ is inserted as an individual term in the GT.

After the complete insertion of subterm $g(1)$ in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents $g(1)$ in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument $g(1)$.
GT-ST: Global Trie for Subterms

In more detail, after storing the node representing \( f/2 \), the process is suspended and the subterm \( g(1) \) is inserted as an individual term in the GT.

After the complete insertion of subterm \( g(1) \) in the GT, the insertion of the main term is resumed by storing a node referencing the leaf node that represents \( g(1) \) in the GT.

The construction of the main term then continues by applying an analogous procedure to the second argument \( g(1) \).
GT-ST: Global Trie for Subterms

- Despite these structural differences in the GT design, all the remaining data structures remain unaltered.

  - Each path in the original subgoal and answer tries is composed of a fixed number of trie nodes representing the number of argument or substitution terms in the corresponding tabled subgoal call or answer.

  - The completed table optimization and the space reclamation mechanisms are implemented as for the GT-T design.
Global Trie for Subterms

GT-ST: Implementation Details

trie_subgoal_check_insert(TABLE_ENTRY te, SUBGOAL_CALL call) {
    sg_node = te->subgoal_trie_root_node
    arity = get_arity(call)
    for (i = 1; i <= arity; i++) {
        t = get_argument_term(call, i)
        if (is_simple_term(t)) {
            sg_node = trie_token_check_insert(sg_node, t)
        } else {
            // t is a compound term
            gt_node = trie_term_check_insert(GT_ROOT_NODE, t)
            sg_node = trie_token_check_insert(sg_node, gt_node)
        }
    }
    return sg_node
}

Global Trie for Subterms

GT-ST: Implementation Details

trie_term_check_insert(TRIE_NODE gt_node, TERM t) {
    if (is_simple_term(t)) {
        gt_node = trie_token_check_insert(gt_node, t)
    }
    else { // t is a compound term
        if (gt_node == GT_ROOT_NODE) {
            name = get_name(t)
            arity = get_arity(t)
            gt_node = trie_token_check_insert(gt_node, name)
            for (i = 1; i <= arity; i++) {
                sub_t = get_argument_term(t, i)
                gt_node = trie_term_check_insert(gt_node, sub_t)
            }
        } else { // t is a compound subterm of a compound term
            sub_gt_node = trie_term_check_insert(GT_ROOT_NODE, t)
            gt_node = trie_token_check_insert(gt_node, sub_gt_node)
        }
    }
    return gt_node
}
### Experimental Results

<table>
<thead>
<tr>
<th>Terms</th>
<th>Mem</th>
<th>GT-T/YapTab</th>
<th></th>
<th></th>
<th>GT-ST/YapTab</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Store</td>
<td>Load</td>
<td>Comp</td>
<td>Mem</td>
<td>Store</td>
<td>Load</td>
</tr>
<tr>
<td>1,000 ints</td>
<td>1.00</td>
<td>1.05</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.09</td>
<td>1.11</td>
</tr>
<tr>
<td>1,000 atoms</td>
<td>1.00</td>
<td>1.04</td>
<td>1.01</td>
<td>1.02</td>
<td>1.00</td>
<td>1.04</td>
<td>1.03</td>
</tr>
<tr>
<td>1,000 f/1</td>
<td>1.00</td>
<td>1.32</td>
<td>1.16</td>
<td>2.10</td>
<td>1.00</td>
<td>1.34</td>
<td>1.17</td>
</tr>
<tr>
<td>1,000 f/2</td>
<td>0.50</td>
<td>1.10</td>
<td>1.14</td>
<td>1.84</td>
<td>0.50</td>
<td>1.06</td>
<td>1.11</td>
</tr>
<tr>
<td>1,000 f/4</td>
<td>0.25</td>
<td>0.81</td>
<td>0.98</td>
<td>1.44</td>
<td>0.25</td>
<td>0.78</td>
<td>1.04</td>
</tr>
<tr>
<td>1,000 f/6</td>
<td>0.17</td>
<td>0.72</td>
<td>0.72</td>
<td>1.38</td>
<td>0.17</td>
<td>0.66</td>
<td>0.71</td>
</tr>
<tr>
<td>1,000 [ ]/1</td>
<td>0.50</td>
<td>1.08</td>
<td>1.05</td>
<td>1.61</td>
<td>0.50</td>
<td>1.10</td>
<td>1.02</td>
</tr>
<tr>
<td>1,000 [ ]/2</td>
<td>0.25</td>
<td>0.80</td>
<td>0.94</td>
<td>1.38</td>
<td>0.25</td>
<td>1.00</td>
<td>1.05</td>
</tr>
<tr>
<td>1,000 [ ]/4</td>
<td>0.13</td>
<td>0.63</td>
<td>0.54</td>
<td>0.96</td>
<td>0.13</td>
<td>0.89</td>
<td>0.66</td>
</tr>
<tr>
<td>Average</td>
<td>0.53</td>
<td>0.95</td>
<td>0.95</td>
<td>1.42</td>
<td>0.53</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Terms</th>
<th>GT-T</th>
<th>GT-ST/GT-T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem (MB)</td>
<td>Times (ms)</td>
</tr>
<tr>
<td></td>
<td>Total/GT</td>
<td>Str/Ld/Cmp</td>
</tr>
<tr>
<td>f/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500,000 g/1</td>
<td>17.17/7.63</td>
<td>126/28/51</td>
</tr>
<tr>
<td>500,000 g/3</td>
<td>32.43/22.89</td>
<td>198/34/61</td>
</tr>
<tr>
<td>500,000 g/5</td>
<td>47.68/38.15</td>
<td>293/47/83</td>
</tr>
<tr>
<td>f/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500,000 g/1</td>
<td>32.43/22.89</td>
<td>203/38/71</td>
</tr>
<tr>
<td>500,000 g/3</td>
<td>62.94/53.41</td>
<td>45/60/103</td>
</tr>
<tr>
<td>500,000 g/5</td>
<td>93.46/83.92</td>
<td>438/111/146</td>
</tr>
<tr>
<td>f/3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500,000 g/1</td>
<td>47.68/38.15</td>
<td>296/50/89</td>
</tr>
<tr>
<td>500,000 g/3</td>
<td>93.46/83.92</td>
<td>616/142/164</td>
</tr>
<tr>
<td>500,000 g/5</td>
<td>139.24/129.7</td>
<td>832/197/224</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory usage and store/load times for a $t/1$ tabled predicate that simply stores in the table space terms defined by term/1 facts.
Conclusions

➢ We have presented a new design for the table space organization, named **Global Trie for Subterms (GT-ST)**, that optimizes GT’s memory usage by avoiding the representation of equal compound subterms, thus maximizing the sharing of the tabled data that is structurally equal at a second level.

➢ Experiments results, using the YapTab tabling system, showed that GT-ST support has potential to achieve significant reductions on memory usage and execution time for programs with increasing compound subterms in term arguments, without compromising the execution time for other programs.

➢ As further work we intend to study how alternative/complementary designs for the table space can further reduce redundancy in term representation. We also plan to study how GT-ST can be used with co-induction.
Questions ?