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*

CICLOPS 2008, Udine, Italy, December 2008

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

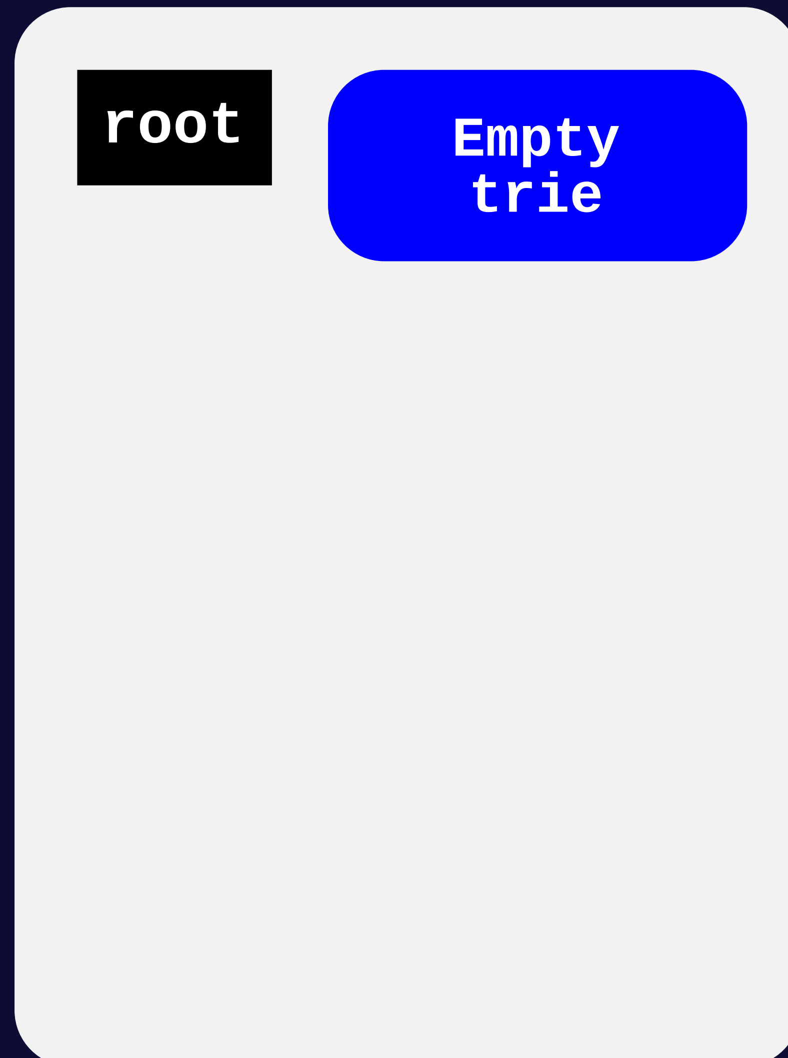
- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.
- The performance of tabled evaluation largely depends on the implementation of the table space. Arguably, the most successful data structure 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.

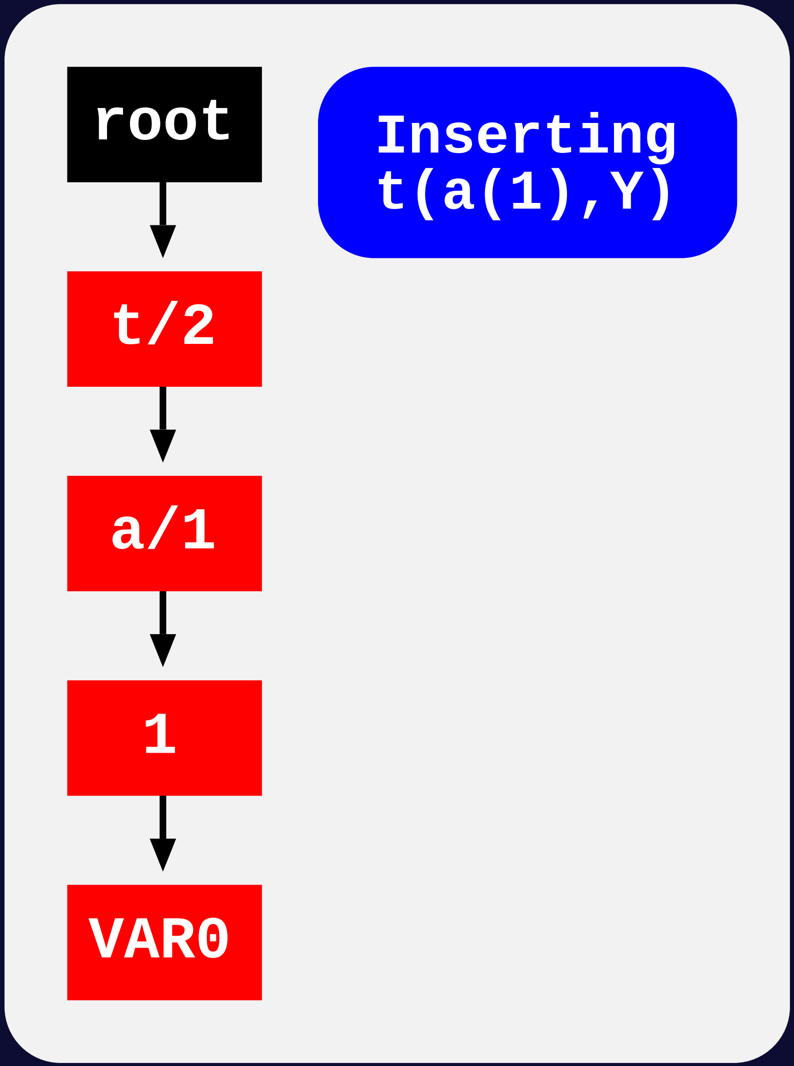
Using Tries to Represent Terms

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



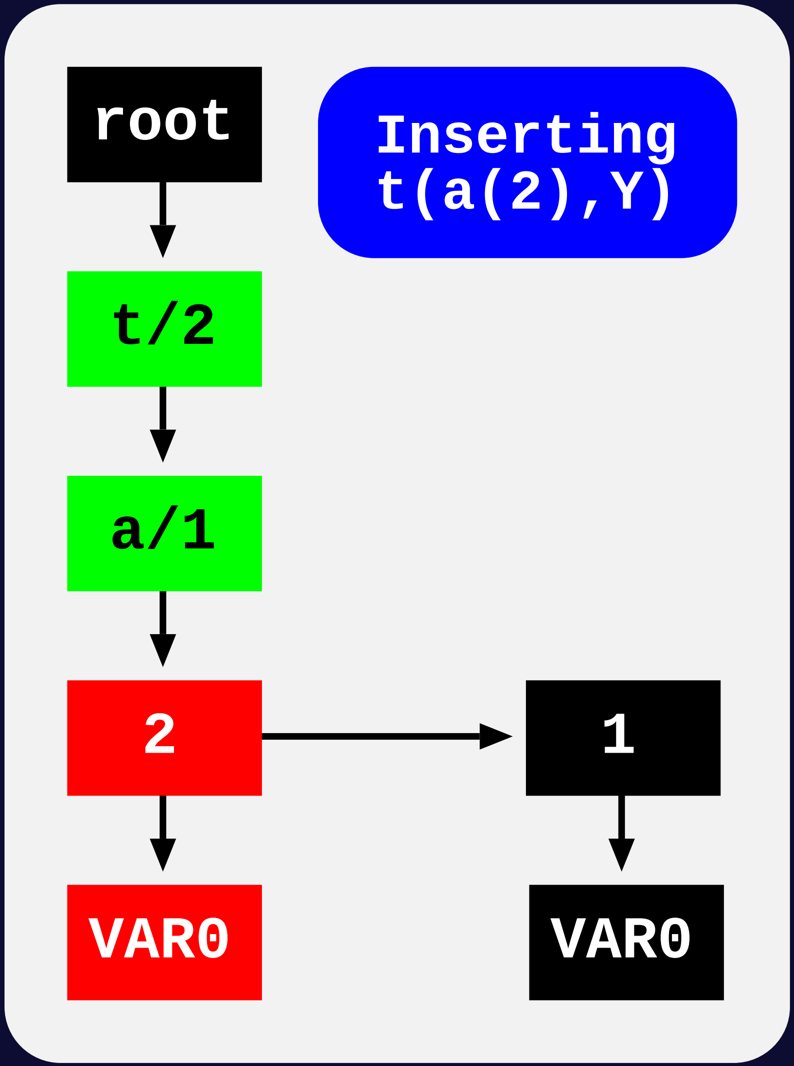
Using Tries to Represent Terms

- 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.



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.

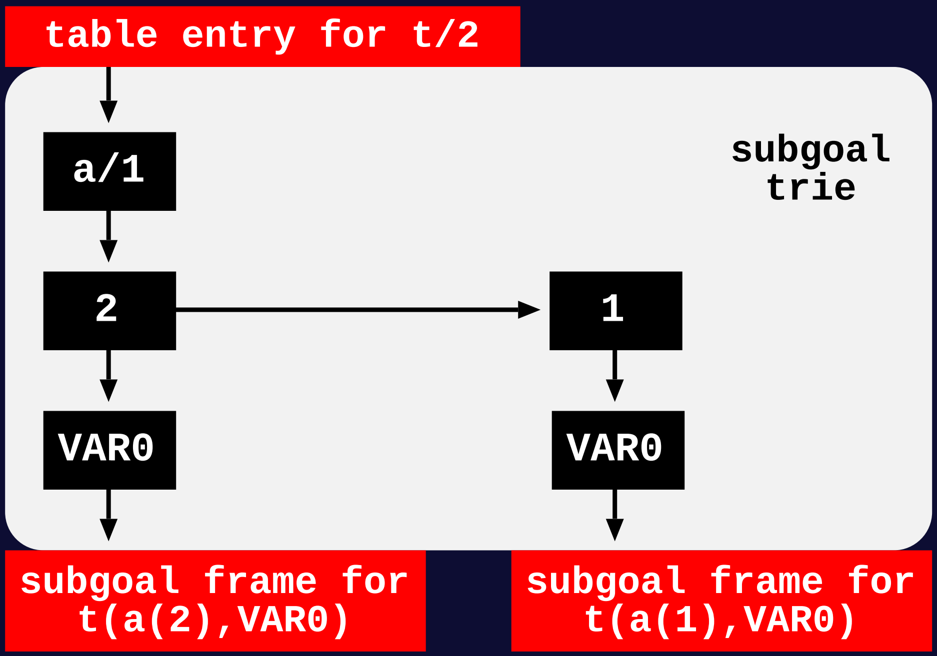


Using Tries to Organise the Table Space

➤ 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

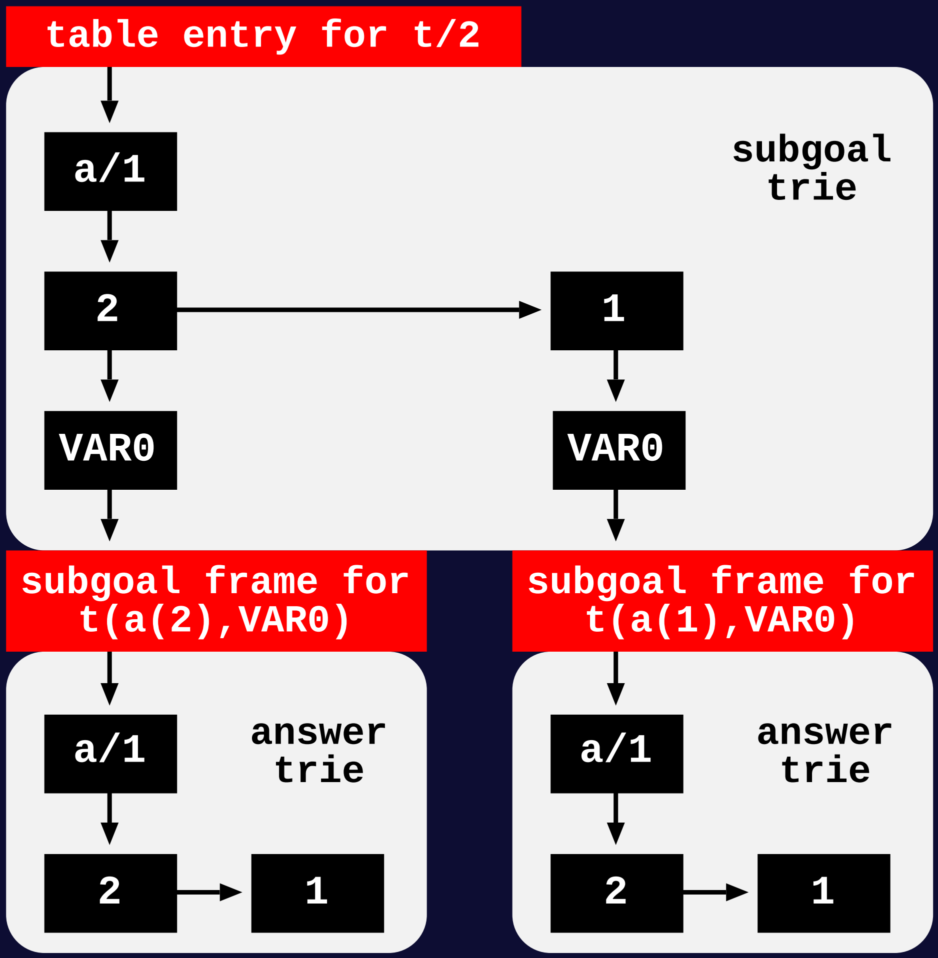
➤ Subgoal Trie Structure

- ♦ 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.

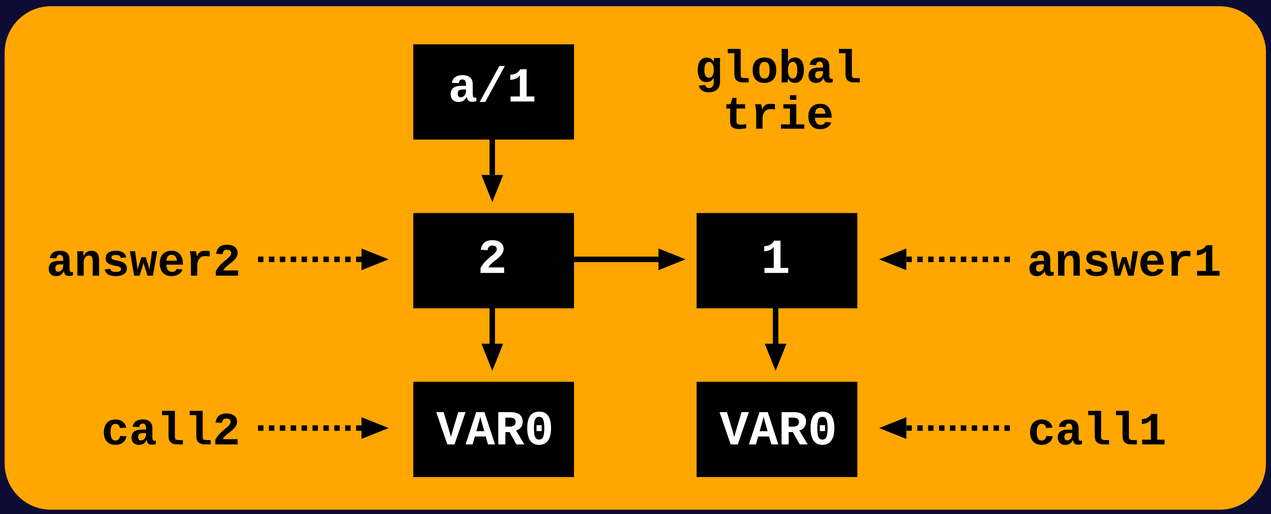
```
:- table t/2.
t(X,Y) :- term(X), term(Y).

term(a(1)).    term(a(2)).
```



Commom Global Trie

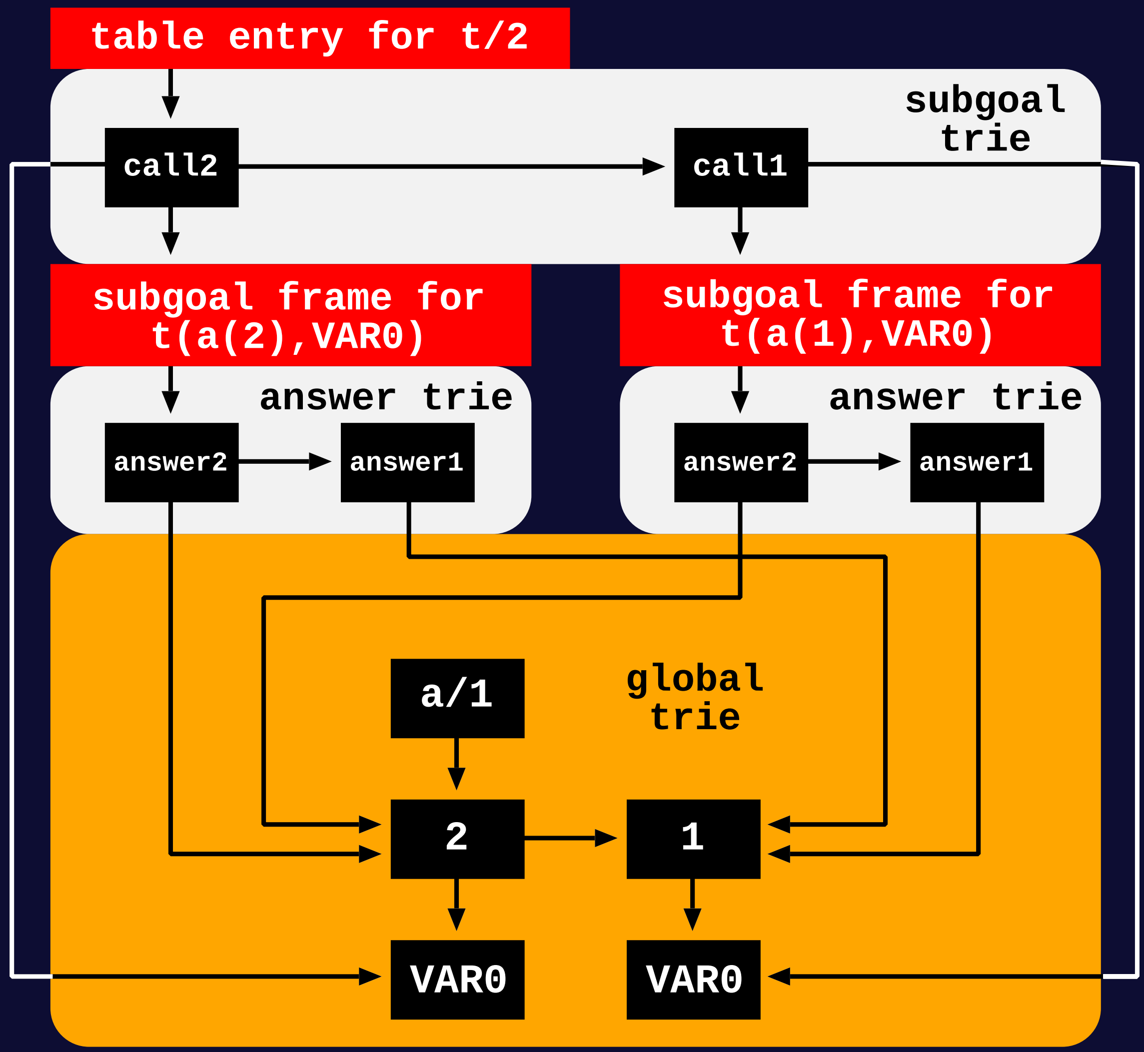
➤ Global Trie Structure

- 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.



Commom Global Trie

- 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.

```
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.

```
answer_check_insert(SUBGOAL_FRAME sf, ANSWER answer) {
  if (GT) {                                      // GT table design
    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
}
```

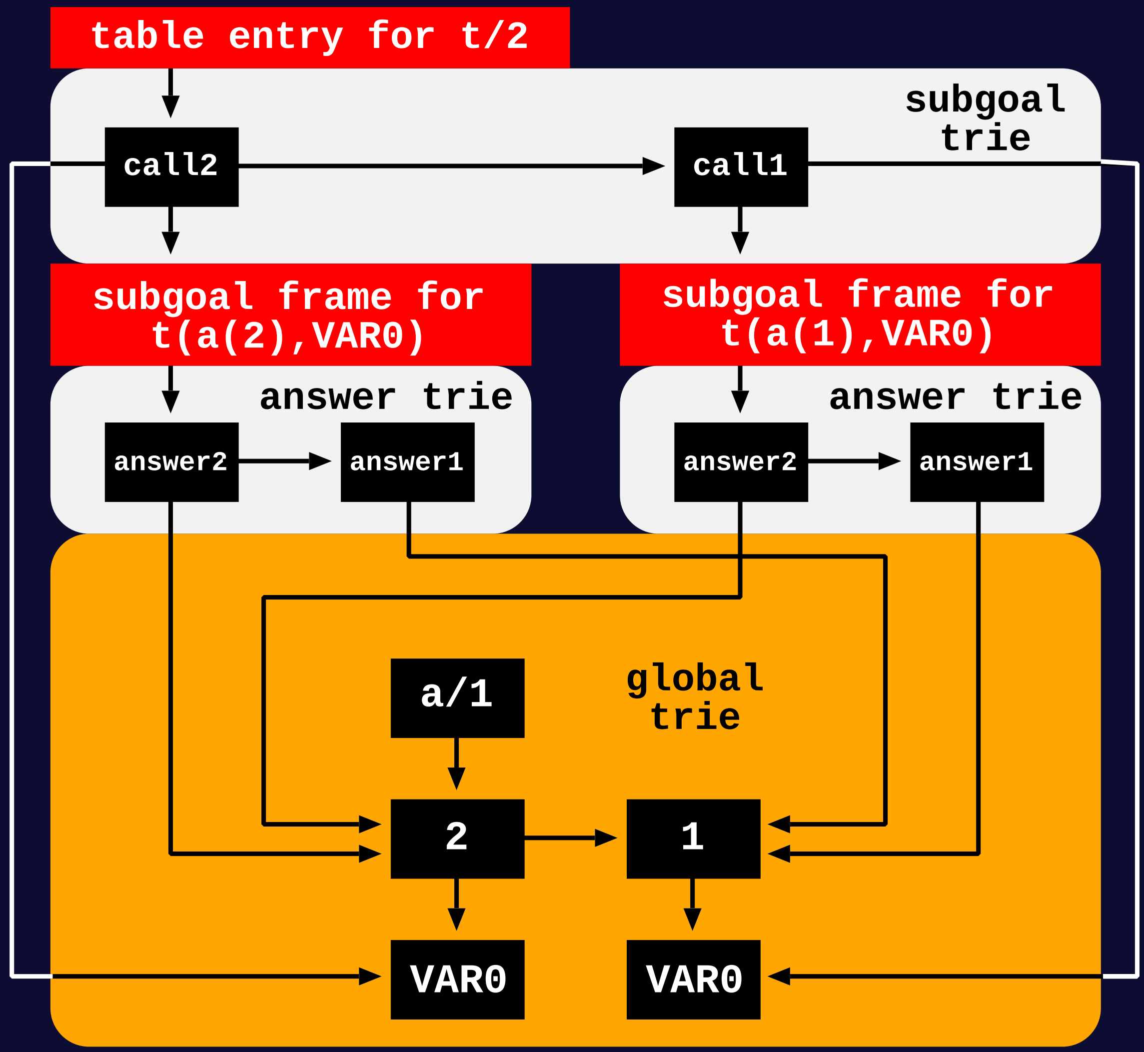
Implementation Details: Tabling Operations

- **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.**

```
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
}
```

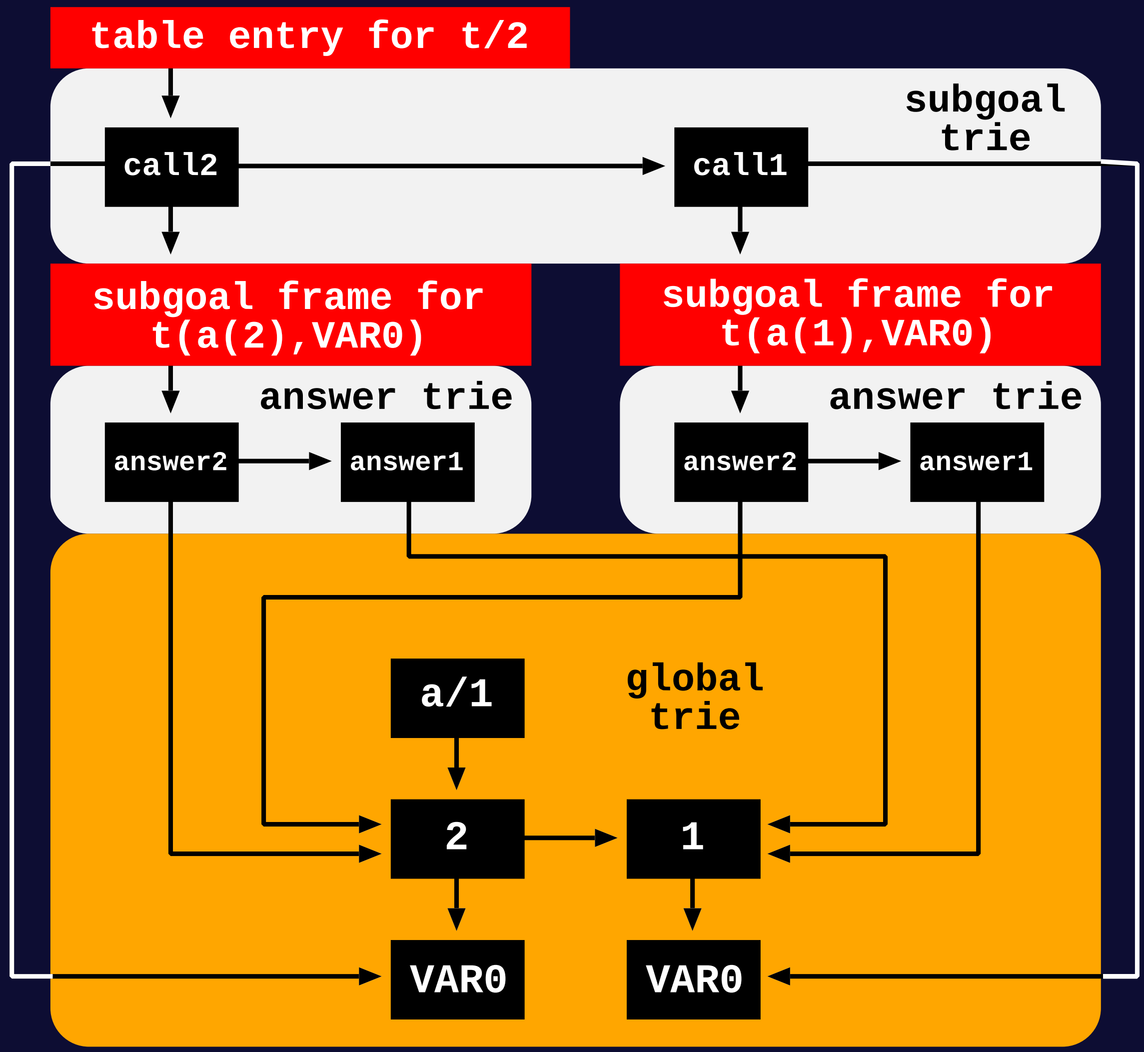
Implementation Details: Two Small Problems ;-)

- 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

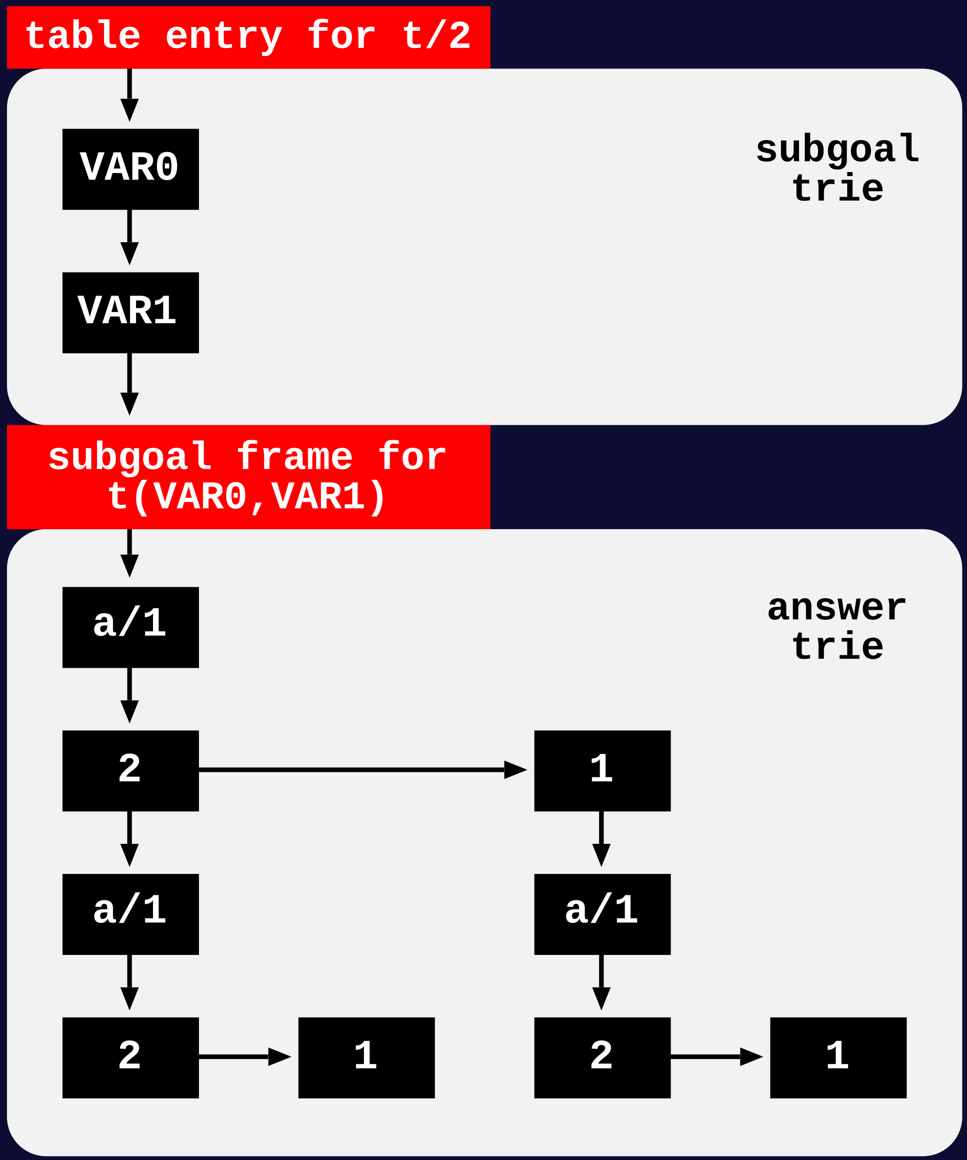
Terms	YapTab			YapTab+GT / YapTab		
	Mem	Store	Load	Mem	Store	Load
500 ints	49074	238	88	1.08	1.29	1.05
500 atoms	49074	256	88	1.08	1.18	1.05
500 f/1	49172	336	176	1.07	1.33	**0.77**
500 f/2	98147	430	190	**0.58**	1.16	**0.82**
500 f/3	147122	554	220	**0.41**	1.04	**0.80**
500 f/4	196097	596	210	**0.33**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

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



Conclusions and Further Work