A Subterm-Based Global Trie for Tabled Evaluation of Logic Programs

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

EPIA 2011, Lisboa, Portugal, October 2011

Tabling in Logic Programming

- **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:

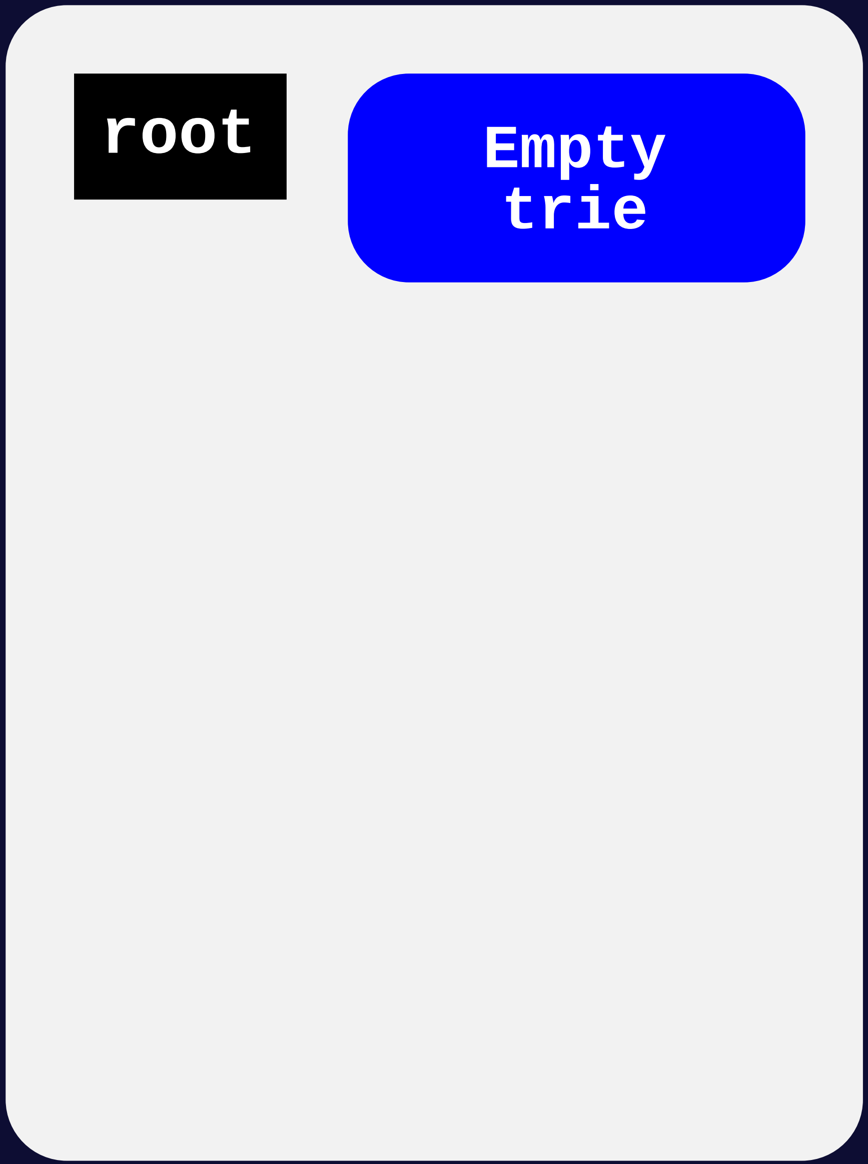
- Look up if a subgoal is in the table and, if not, insert it.
- Look up if a newly found answer is in the table and, if not, insert it.
- Load answers for similar subgoals.

➤ Implementation requirements:

- Fast look-up and insertion methods.
- Compactness in representation of logic terms.

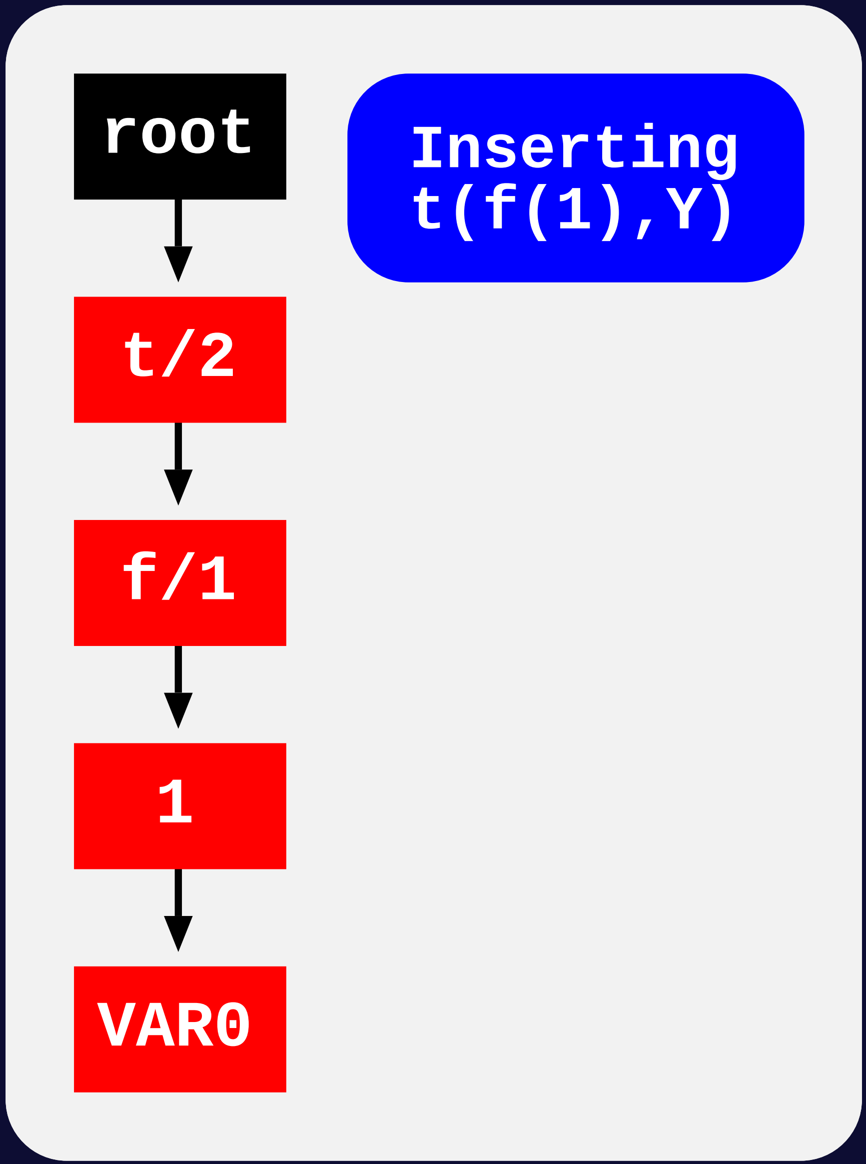
Using Tries to Represent Terms

- 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.



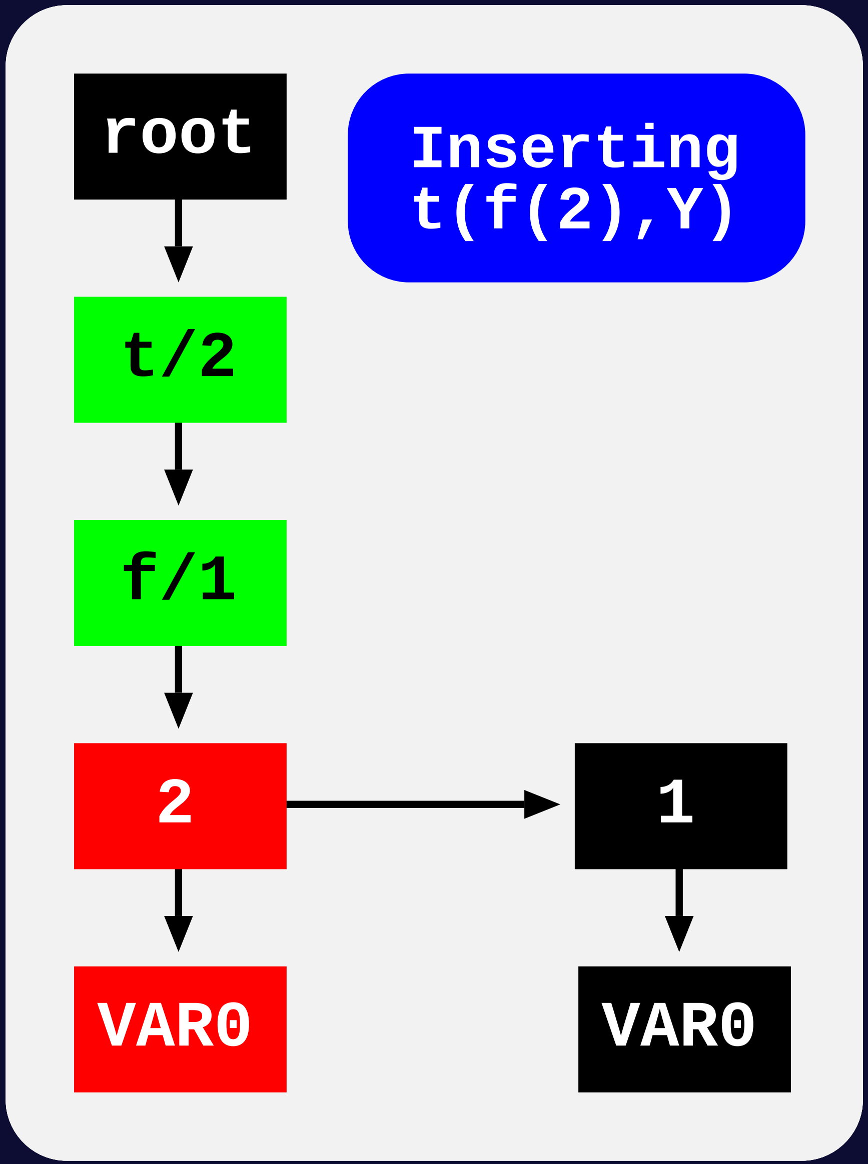
Using Tries to Represent Terms

- 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.



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.

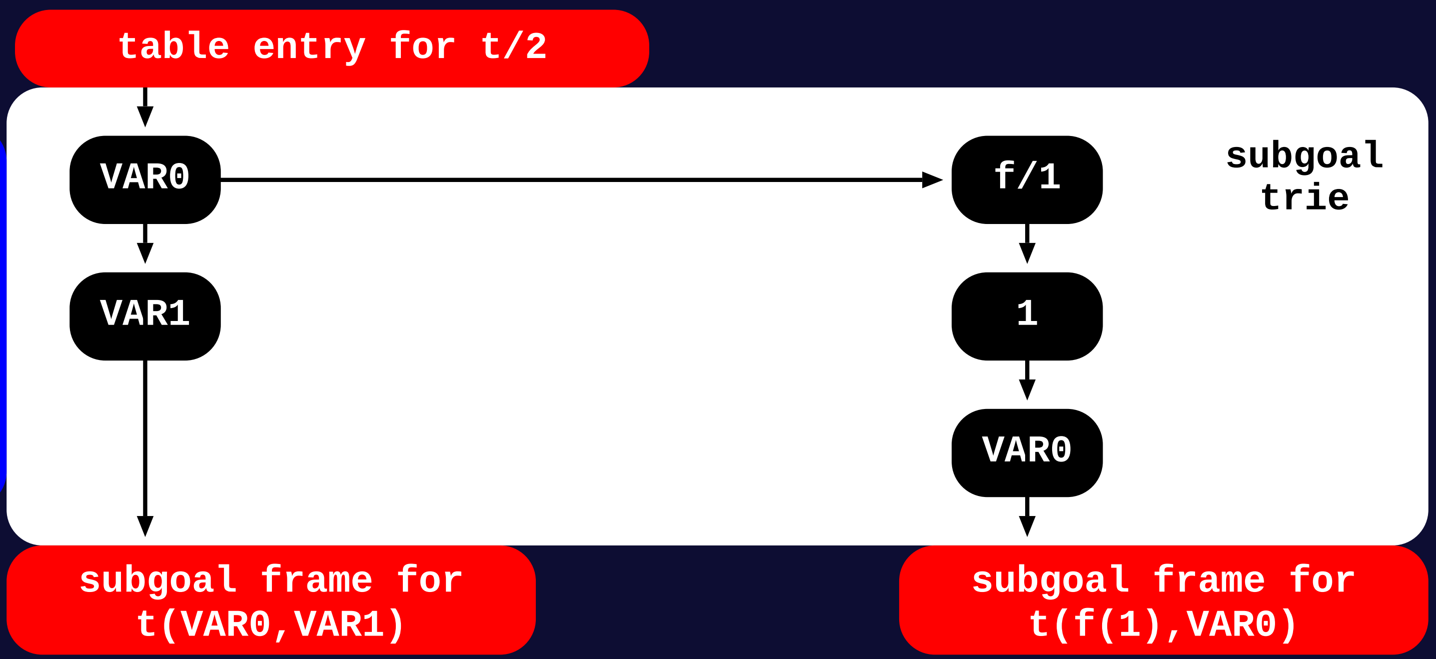


Using Tries to Represent the Table Space

➤ 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.

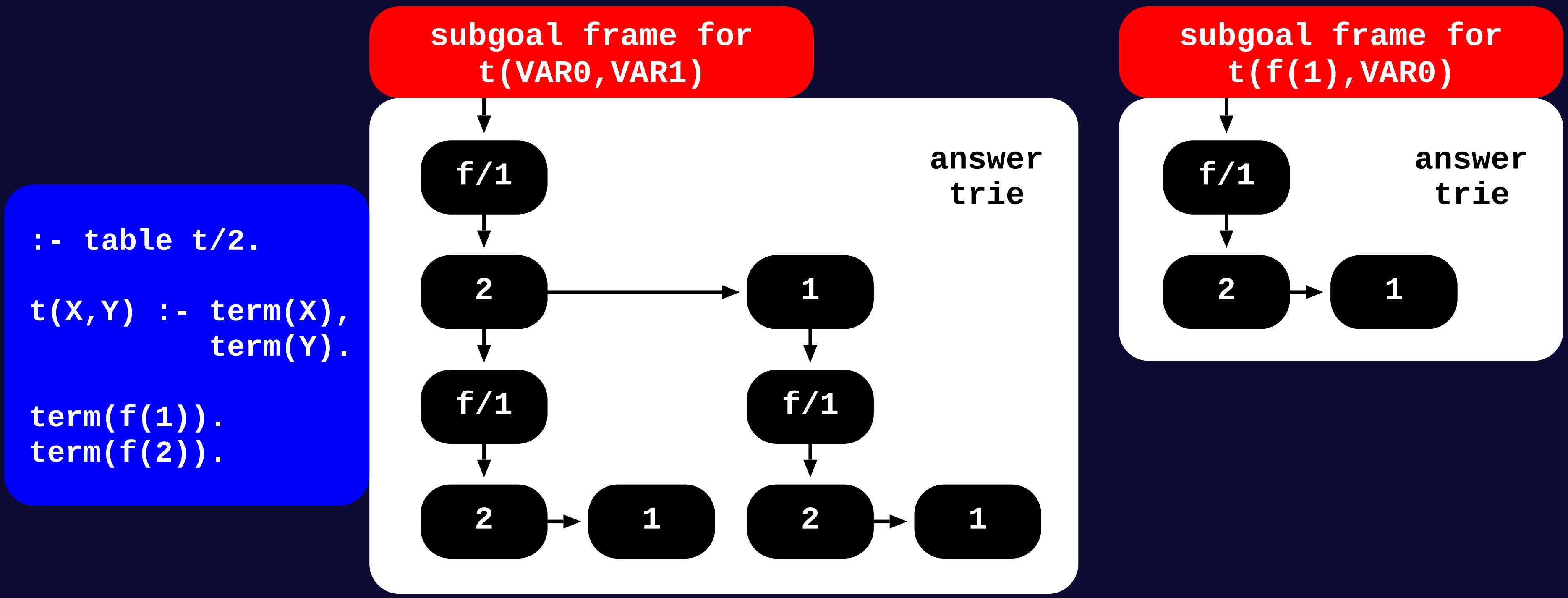
```
:- table t/2.

t(X,Y) :- term(X),
          term(Y).

term(f(1)).
term(f(2)).
```



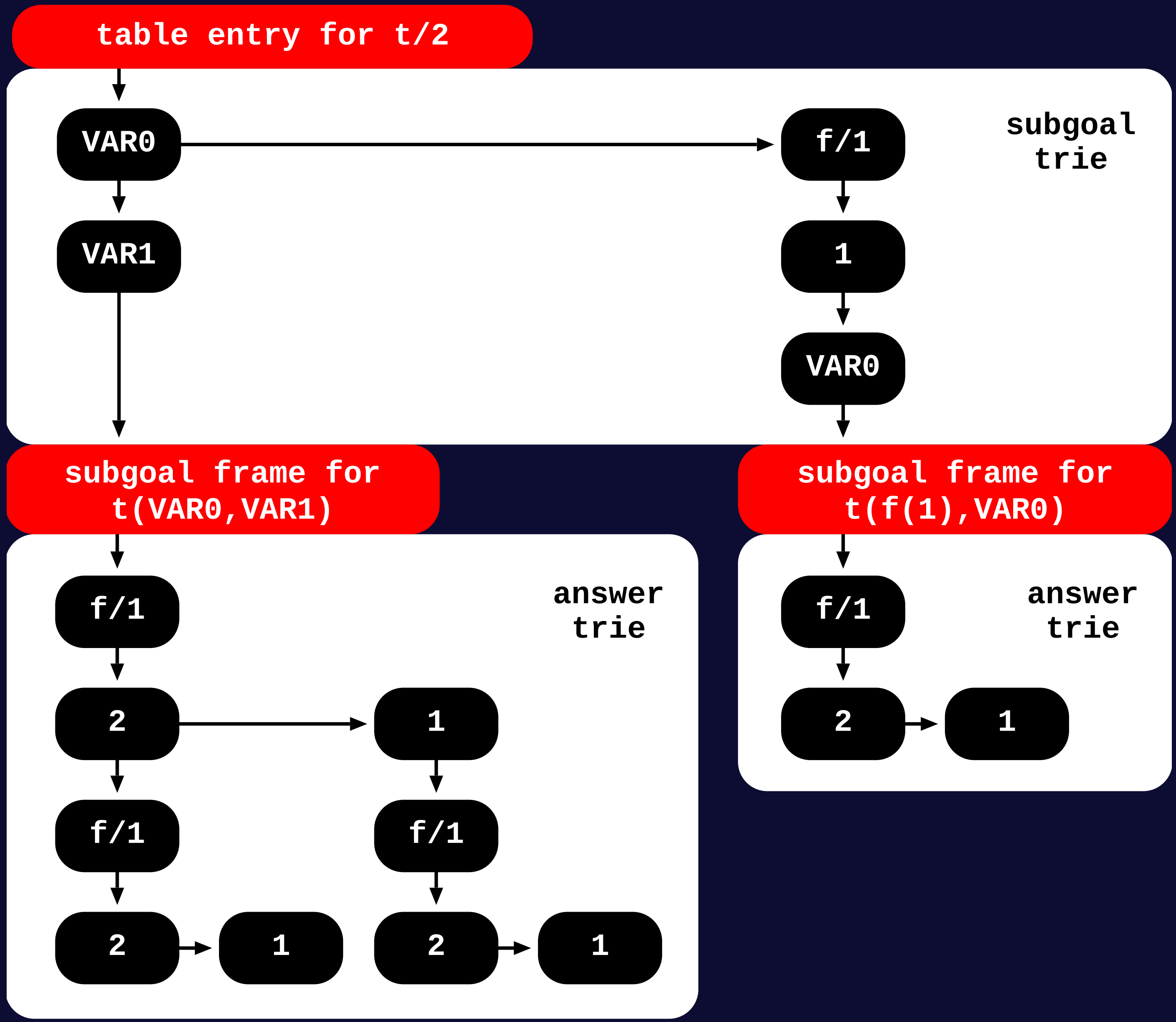
Using Tries to Represent the Table Space

➤ Answer Trie

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



Using Tries to Represent the Table Space



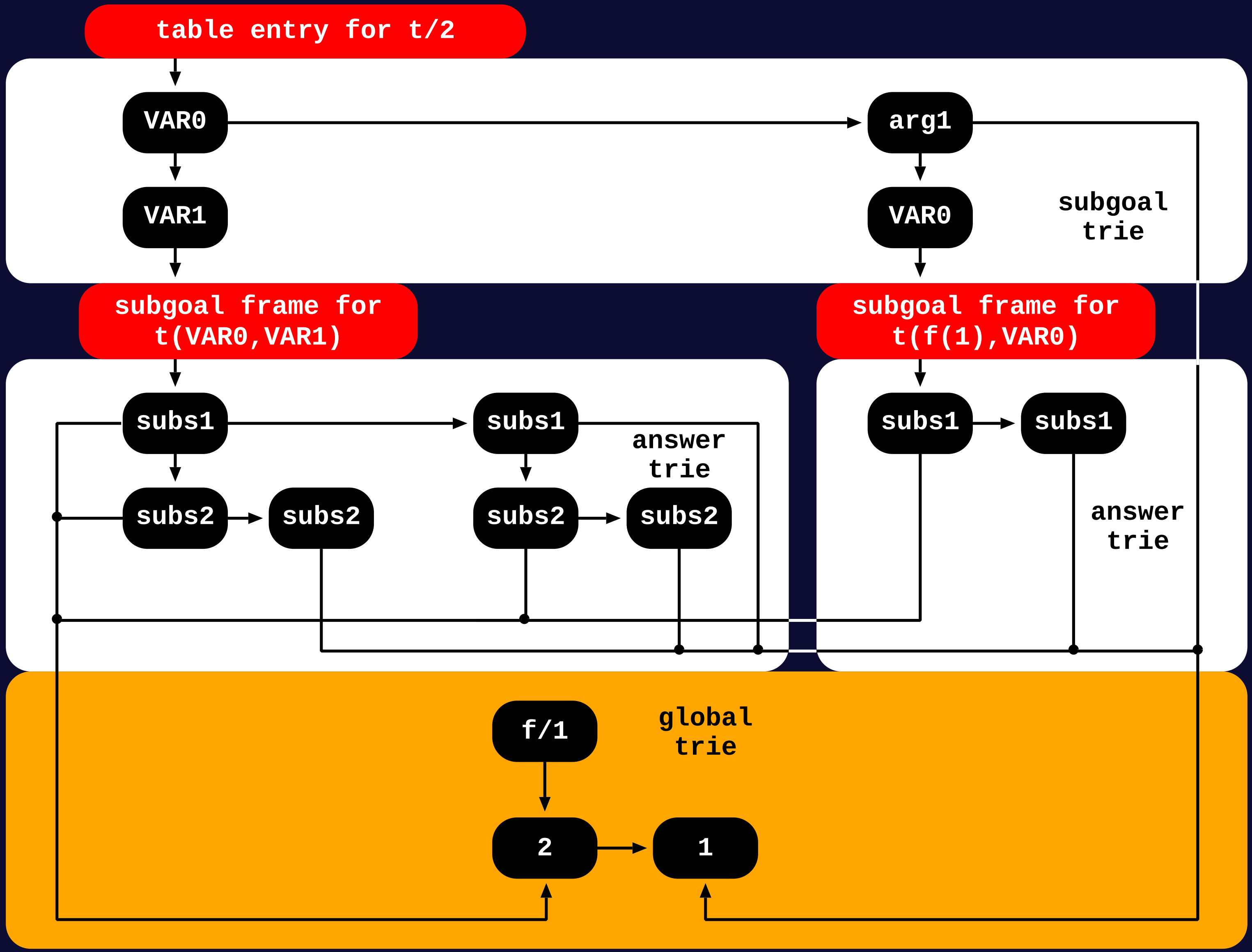
GT-T: Global Trie for Terms [ICLP'09]

- 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]

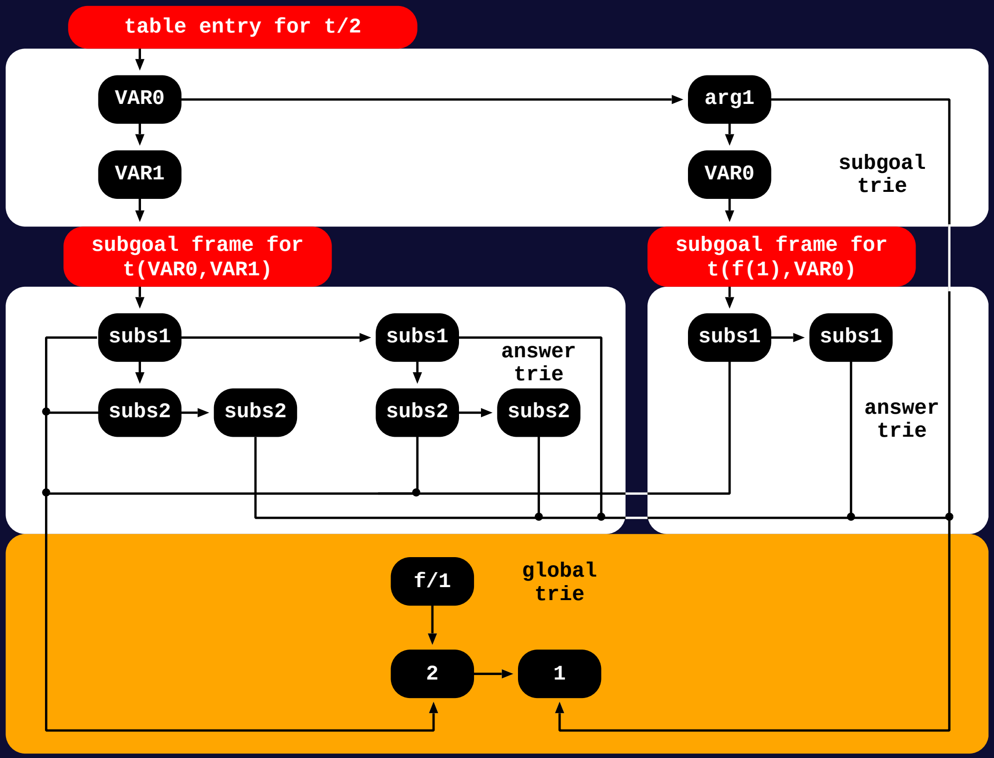
- The GT-T was designed in order to **maximize the sharing of tabled data which is structurally equal**.
- In GT-T, all **argument and substitution compound terms** appearing in tabled subgoal calls and/or answers are **represented only once in the GT**, thus preventing situations where these terms are 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]



GT-T: Global Trie for Terms [ICLP'09]

- 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.

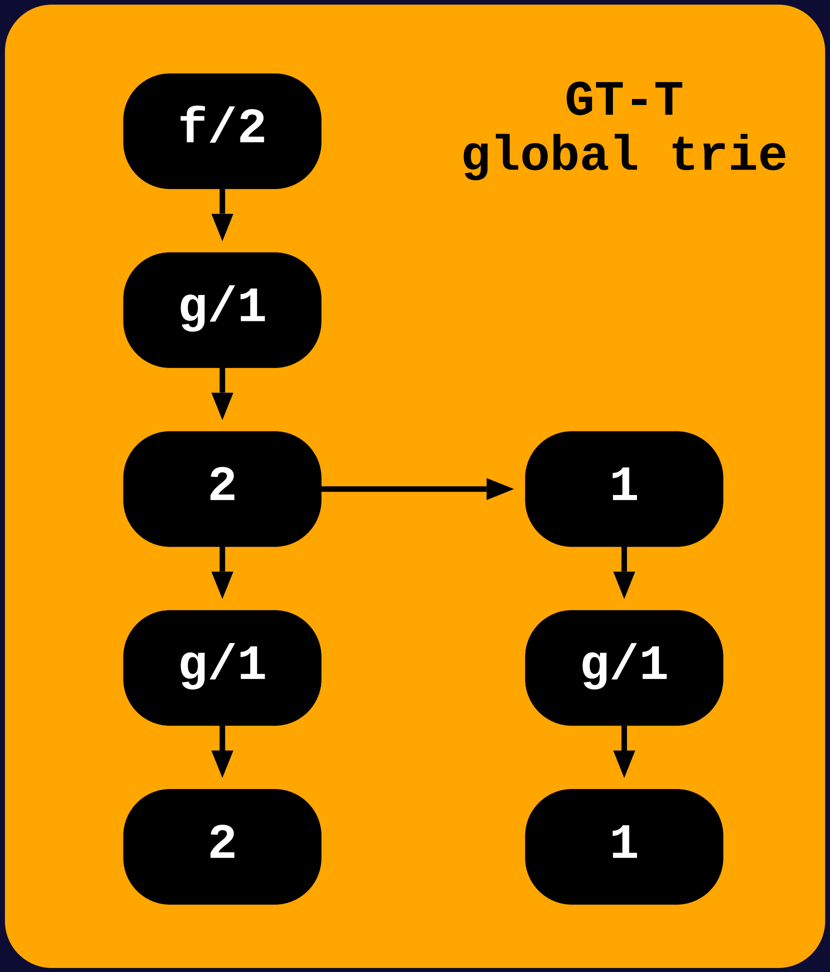
GT-ST: Global Trie for Subterms

- 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**.

GT-ST: Global Trie for Subterms

- Consider, for example, the insertion of the terms **f(g(1),g(1))** and **f(g(2),g(2))** in the GT-T...

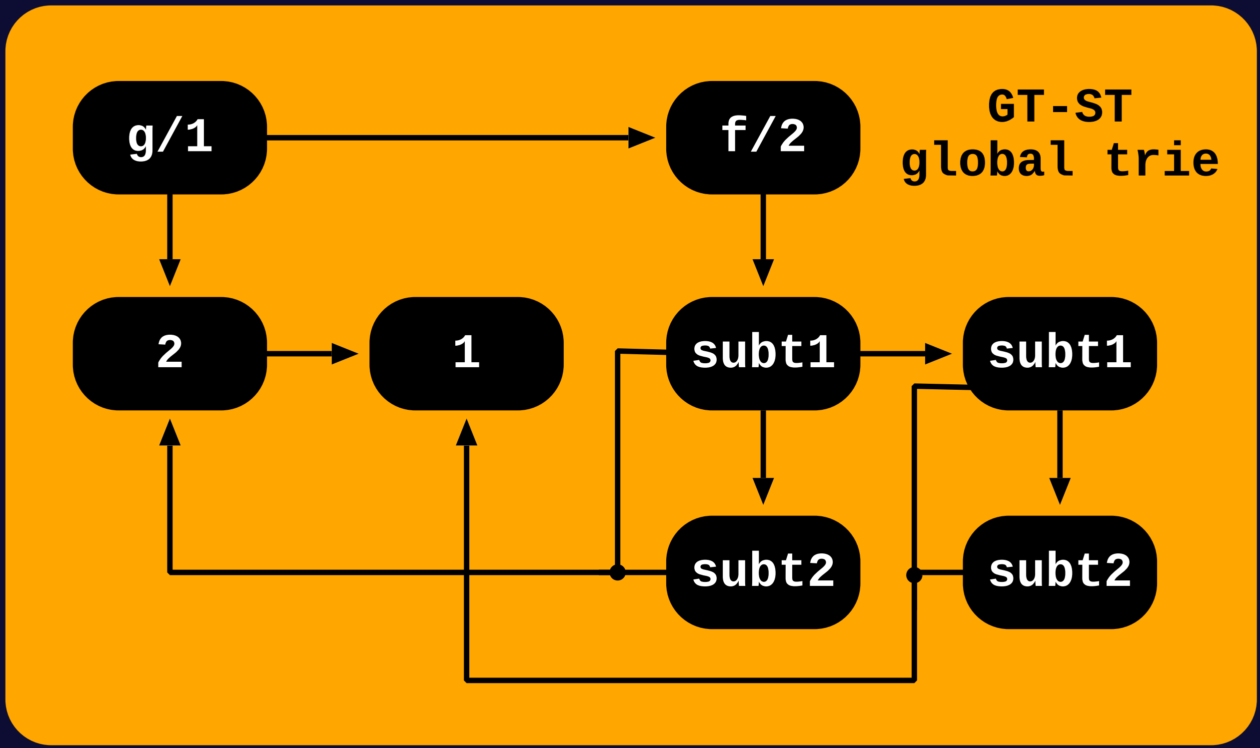
```
:- table t/2.

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

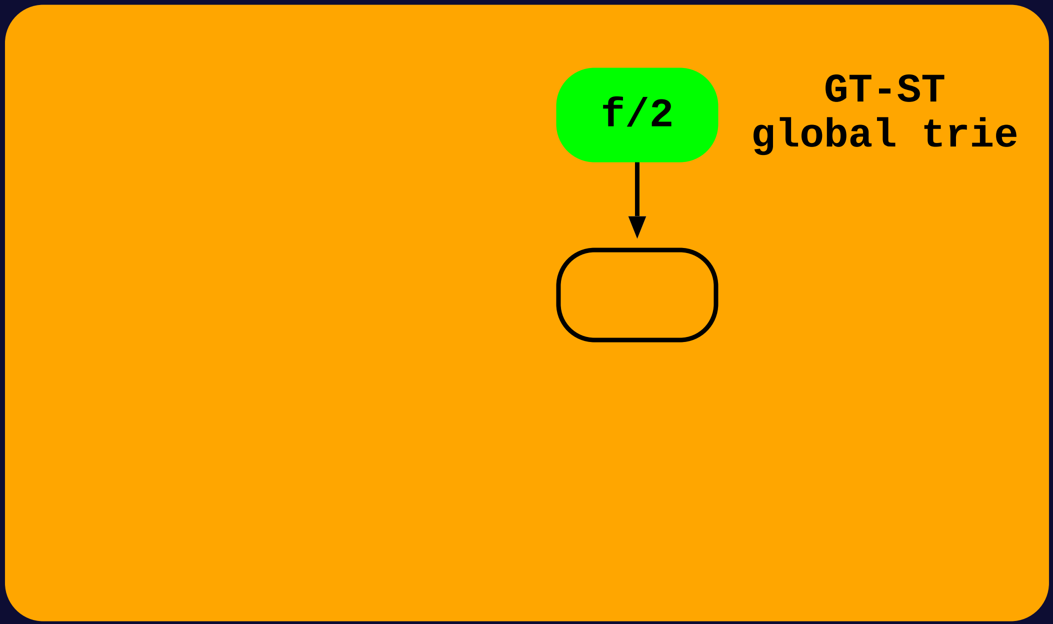
- ... and in the GT-ST.

```
:- table t/2.

t(X,Y) :- term(X),
          term(Y).

term(f(g(1),g(1))).
term(f(g(2),g(2))).
```



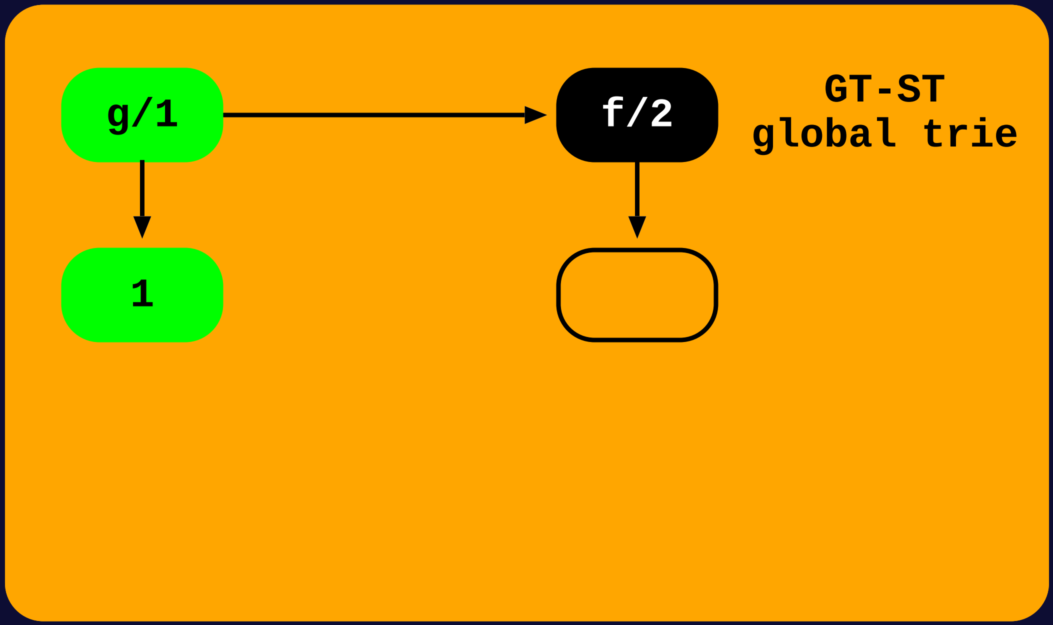
GT-ST: Global Trie for Subterms

- 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)**.



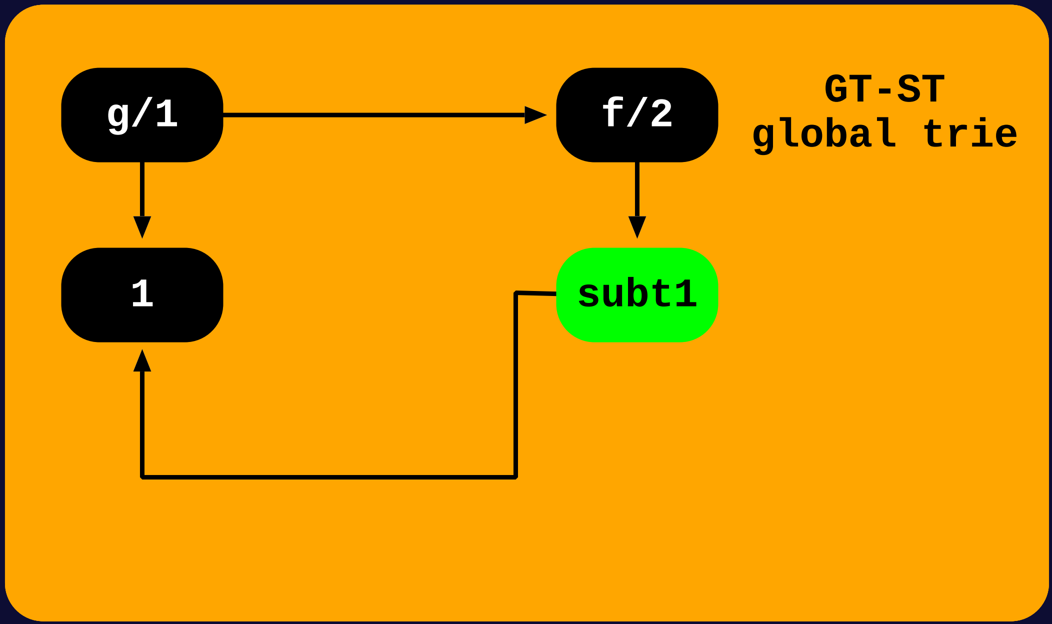
GT-ST: Global Trie for Subterms

- 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)**.



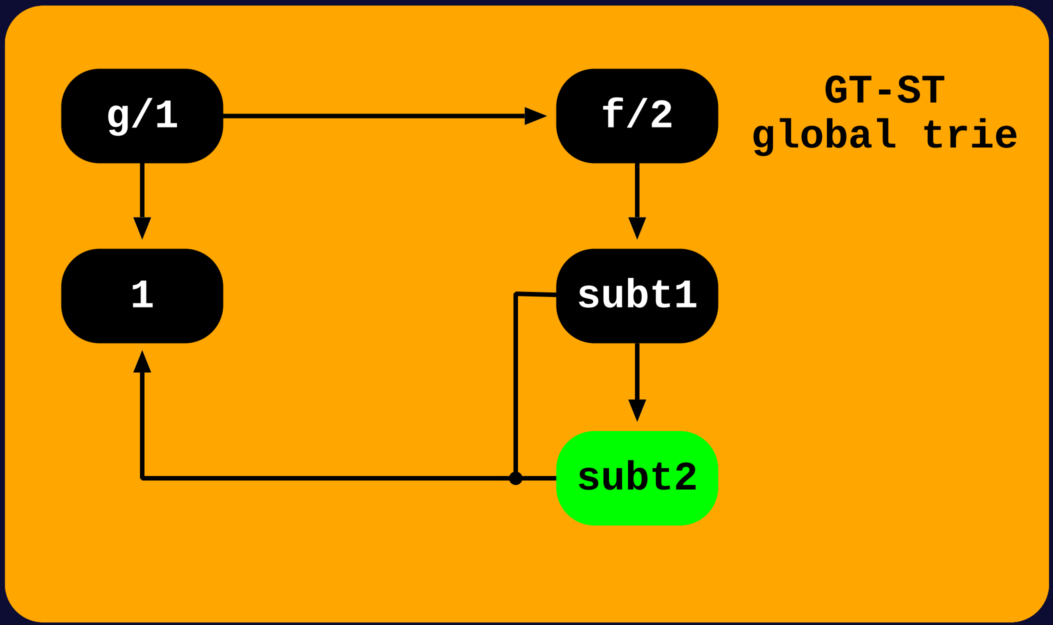
GT-ST: Global Trie for Subterms

- 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)**.



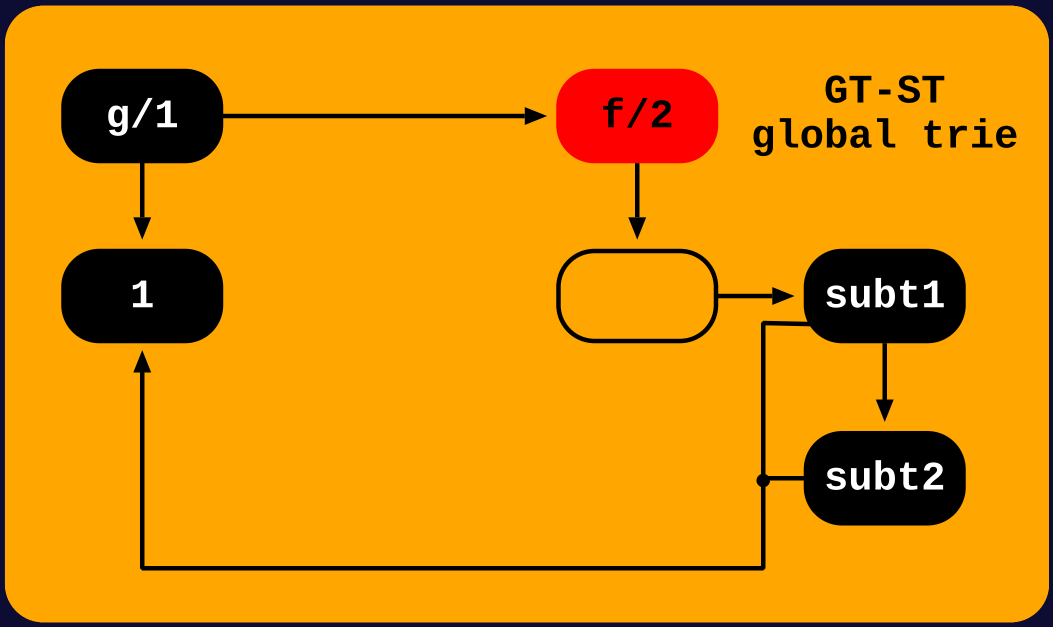
GT-ST: Global Trie for Subterms

- 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)**.



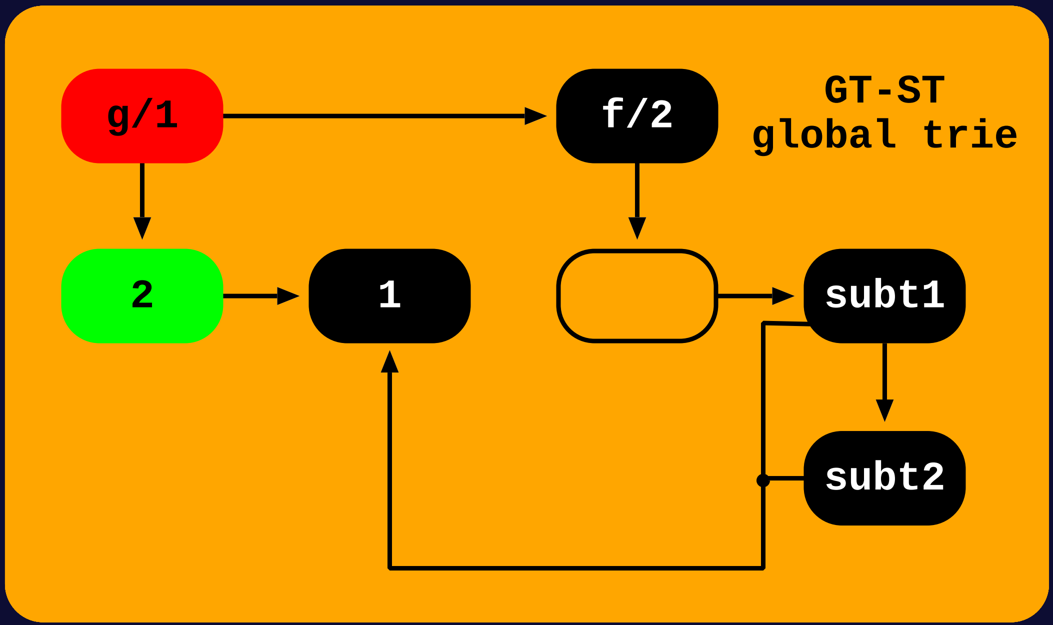
GT-ST: Global Trie for Subterms

- 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)**.



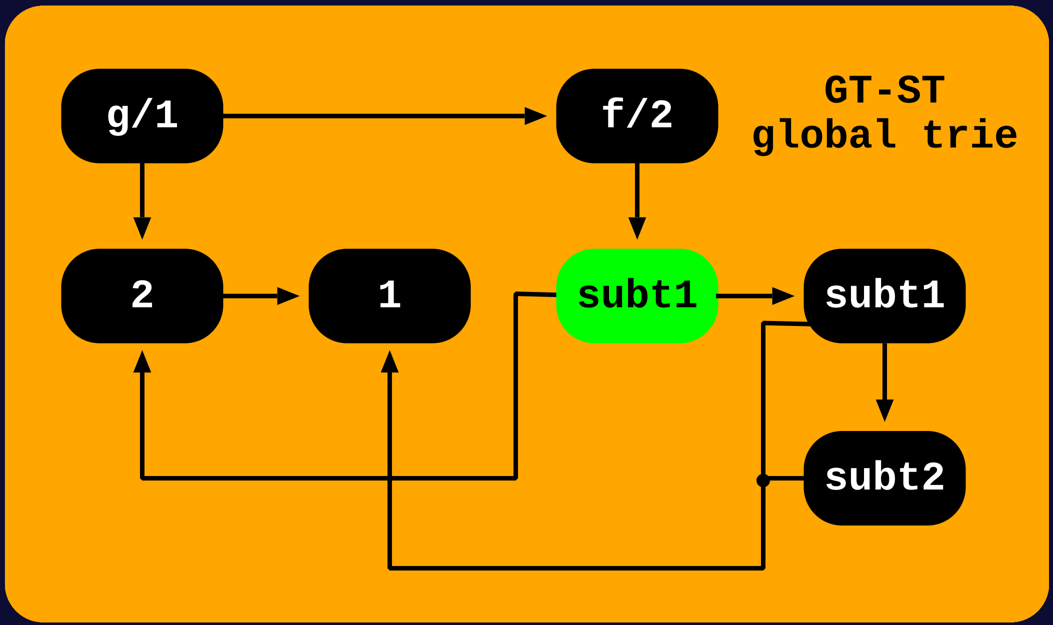
GT-ST: Global Trie for Subterms

- 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)**.



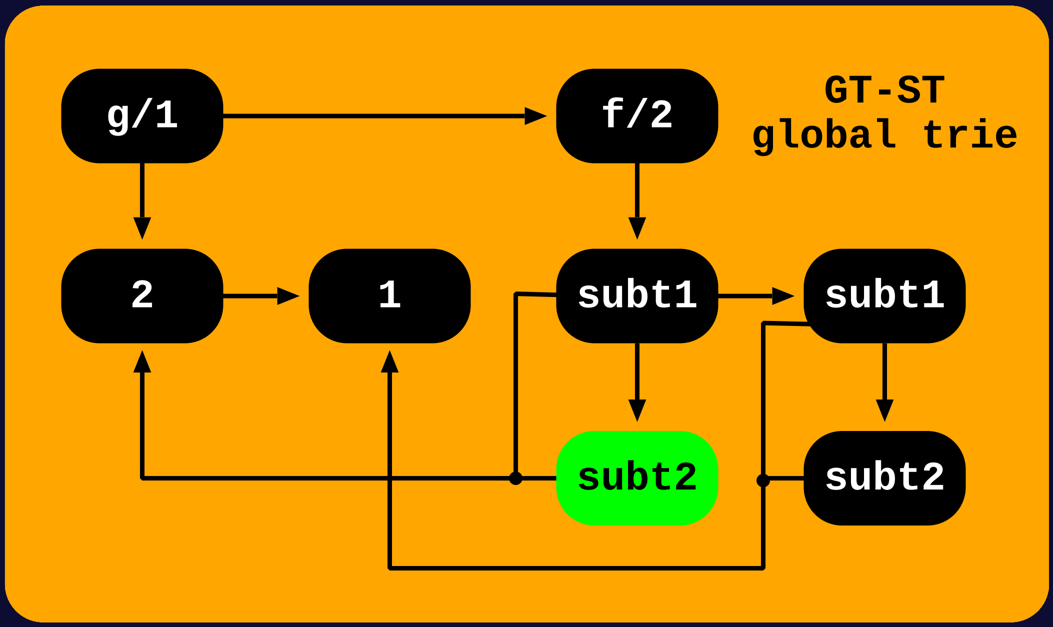
GT-ST: Global Trie for Subterms

- 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.

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
}
```

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

Terms	GT-T/YapTab				GT-ST/YapTab			
	Mem	Store	Load	Comp	Mem	Store	Load	Comp
1,000 ints	1.00	1.05	1.00	1.00	1.00	1.09	1.11	1.07
1,000 atoms	1.00	1.04	1.01	1.02	1.00	1.04	1.03	1.08
1,000 f/1	1.00	1.32	1.16	2.10	1.00	1.34	1.17	2.13
1,000 f/2	**0.50**	1.10	1.14	1.84	**0.50**	1.06	1.11	1.88
1,000 f/4	**0.25**	**0.81**	**0.98**	1.44	**0.25**	**0.78**	1.04	1.53
1,000 f/6	**0.17**	**0.72**	**0.72**	1.38	**0.17**	**0.66**	**0.71**	1.36
1,000 [ ]/1	**0.50**	1.08	1.05	1.61	**0.50**	1.10	1.02	1.58
1,000 [ ]/2	**0.25**	**0.80**	**0.94**	1.38	**0.25**	1.00	1.05	1.48
1,000 [ ]/4	**0.13**	**0.63**	**0.54**	**0.96**	**0.13**	**0.89**	**0.66**	1.14
Average	**0.53**	**0.95**	**0.95**	**1.42**	**0.53**	**0.99**	**0.99**	**1.47**

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

Terms	GT-T Mem (MB) Total/GT	GT-T Times (ms) Str/Ld/Cmp	GT-ST/GT-T Mem Total/GT	GT-ST/GT-T Times Str/Ld/Cmp
f/1				
500,000 g/1	17.17/7.63	126/28/51	1.44 / 2.00	1.55 / 1.14 / 1.00
500,000 g/3	32.43/22.89	198/34/61	1.24 / 1.33	3.29 / 1.12 / 1.25
500,000 g/5	47.68/38.15	293/47/83	1.16 / 1.20	1.46 / 1.00 / **0.99**
f/2				
500,000 g/1	32.43/22.89	203/38/71	1.00 / 1.00	1.28 / 1.13 / 1.09
500,000 g/3	62.94/53.41	45/60/103	**0.76** / **0.71**	1.18 / **0.84** / **0.95**
500,000 g/5	93.46/83.92	438/111/146	**0.67** / **0.64**	1.10 / **0.67** / **0.80**
f/3				
500,000 g/1	47.68/38.15	296/50/89	**0.84** / **0.80**	2.87 / 1.02 / 1.03
500,000 g/3	93.46/83.92	616/142/164	**0.59** / **0.55**	1.25 / **0.80** / **0.85**
500,000 g/5	139.24/129.7	832/197/224	**0.51** / **0.47**	**0.96** / **0.67** / **0.74**
Average			**0.96** / **0.97**	**0.93** / **0.97** / **0.91**

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.

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.
  - This is the first viable approach that makes **tabling technology applicable to predicates that process lists recursively**:

```
append([],L,L).
append([H|T],L,[H|L1]) :- append(T,L,L1).
```

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.
  - This is the first viable approach that makes **tabling technology applicable to predicates that process lists recursively**:

```
append([],L,L).
append([H|T],L,[H|L1]) :- append(T,L,L1).
```

- 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 ?