A Term-Based Global Trie for Tabled Logic Programs

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

ICLP 2009, Pasadena, California, USA, July 2009

Tabling in Logic Programming

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

Tabling in Logic Programming

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.
- Tabling has proven to be particularly effective in logic (**Prolog**) programs:
  - Avoids recomputation, thus reducing the search space.
  - Avoids infinite loops, thus ensuring termination for a wider class of programs.

Tabling in Logic Programming

- **Tabling** is an implementation technique where intermediate answers for subgoals are stored in a **table space** and then reused when a repeated call appears.
- Tabling has proven to be particularly effective in logic (**Prolog**) programs:
  - Avoids recomputation, thus reducing the search space.
  - Avoids infinite loops, thus ensuring termination for a wider class of programs.
- Tabling has been successfully applied to real-world applications:
  - Deductive Databases
  - Knowledge Based Systems
  - Model Checking
  - Program Analysis
  - Theorem Proving
  - Non-Monotonic Reasoning
  - Natural Language Processing
  - Inductive Logic Programming

Motivation

- 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

- 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

- 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 **terms in tabled subgoal calls and tabled answers are represented only once in a common global trie** instead of being spread over several different trie data structures.

Motivation

- 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 **terms in tabled subgoal calls and tabled answers are represented only once in a common global trie** instead of being spread over several different trie data structures.
- Our new design can be seen as an extension of a previous approach [PADL09], where we first introduced the idea of using a common global trie.

Table Space

➤ Can be accessed to:

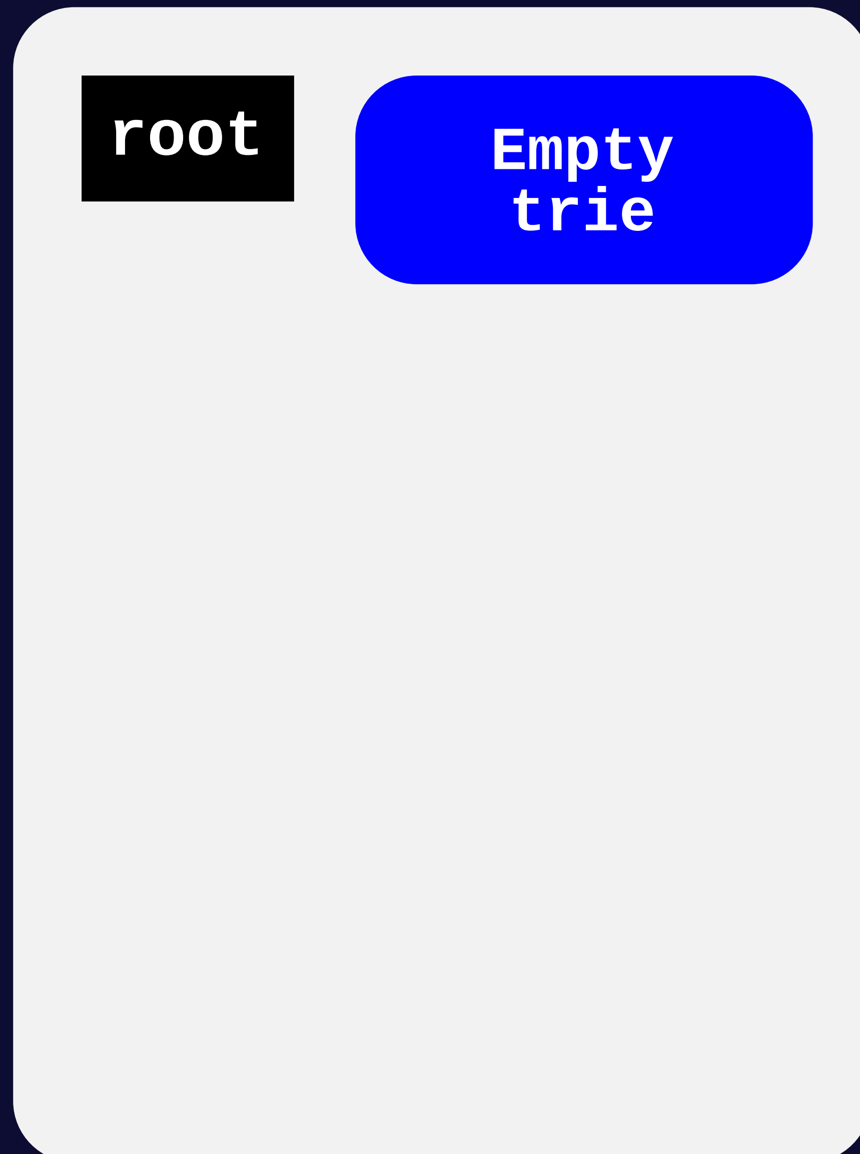
- Look up if a subgoal is in the table and, if not, insert it.
- Look up if a newly found answer is in the table and, if not, insert it.
- Load answers for repeated subgoals.

➤ Implementation requirements:

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

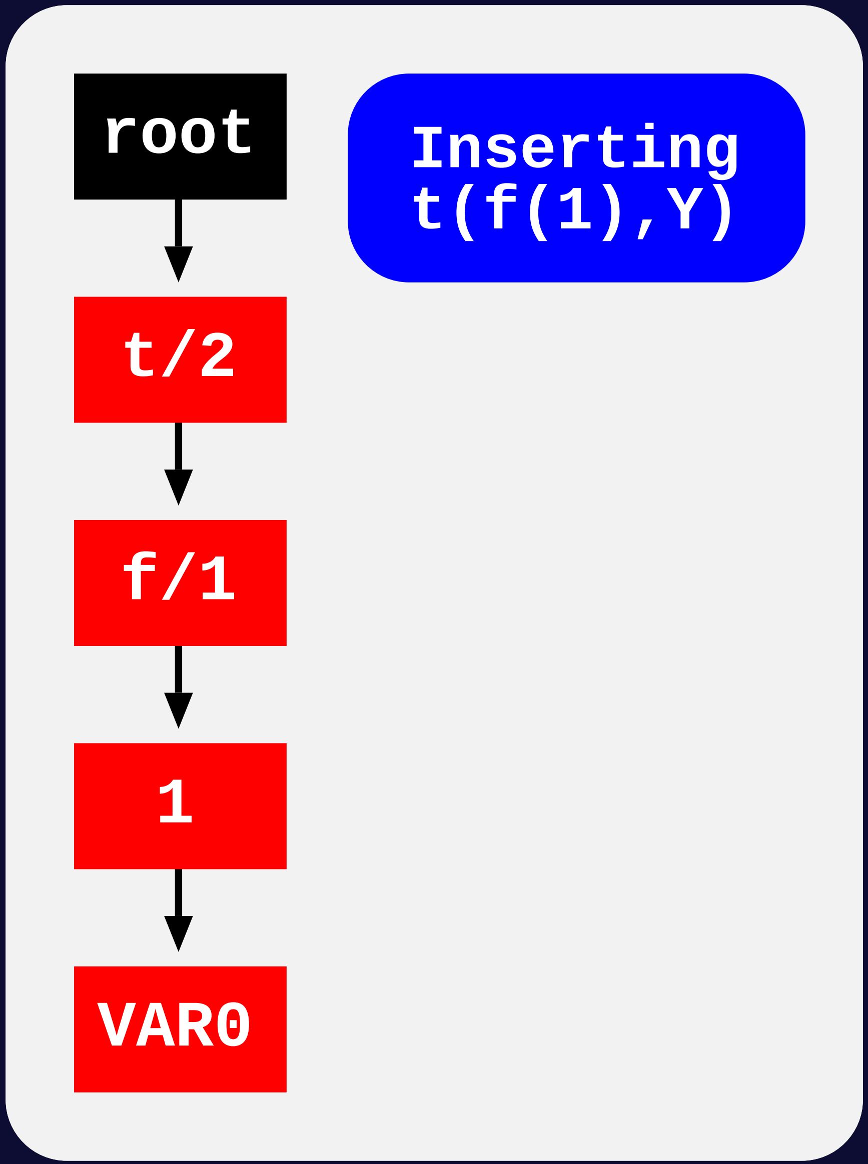
Using Tries to Represent Terms

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



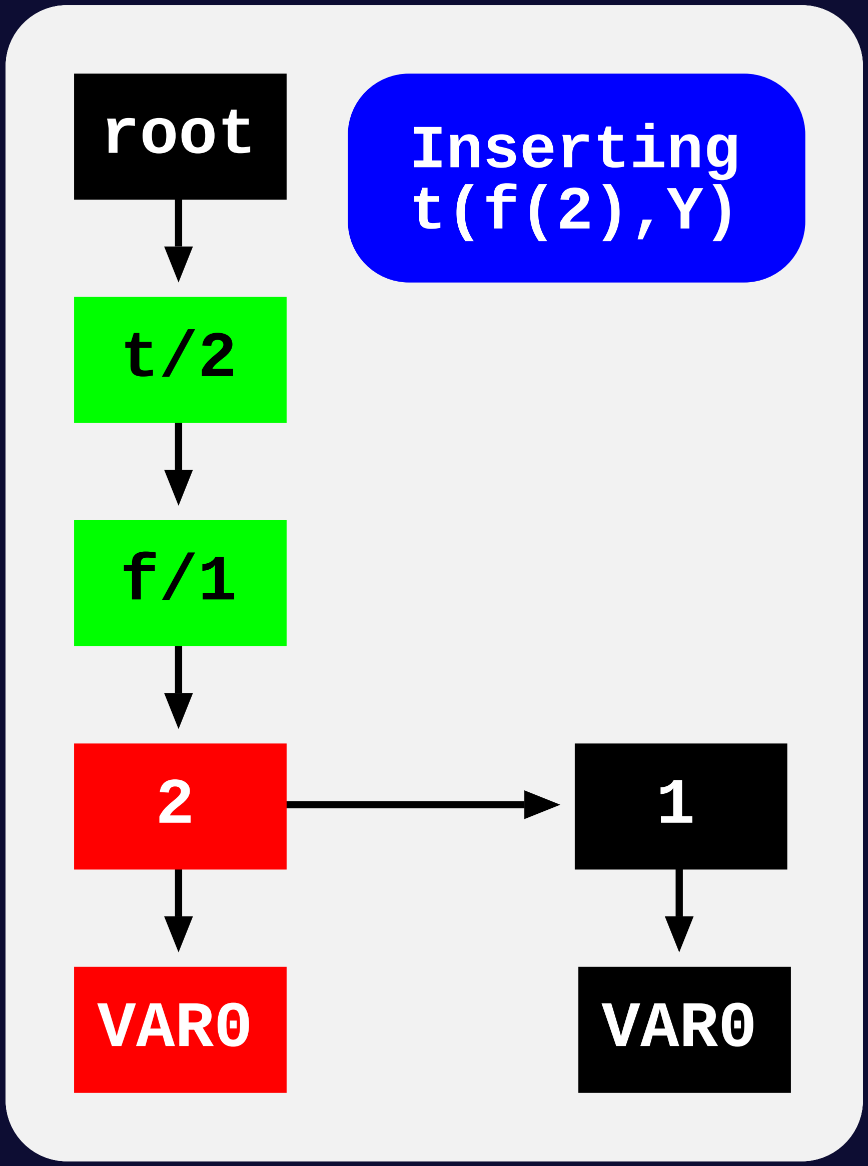
Using Tries to Represent Terms

- 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

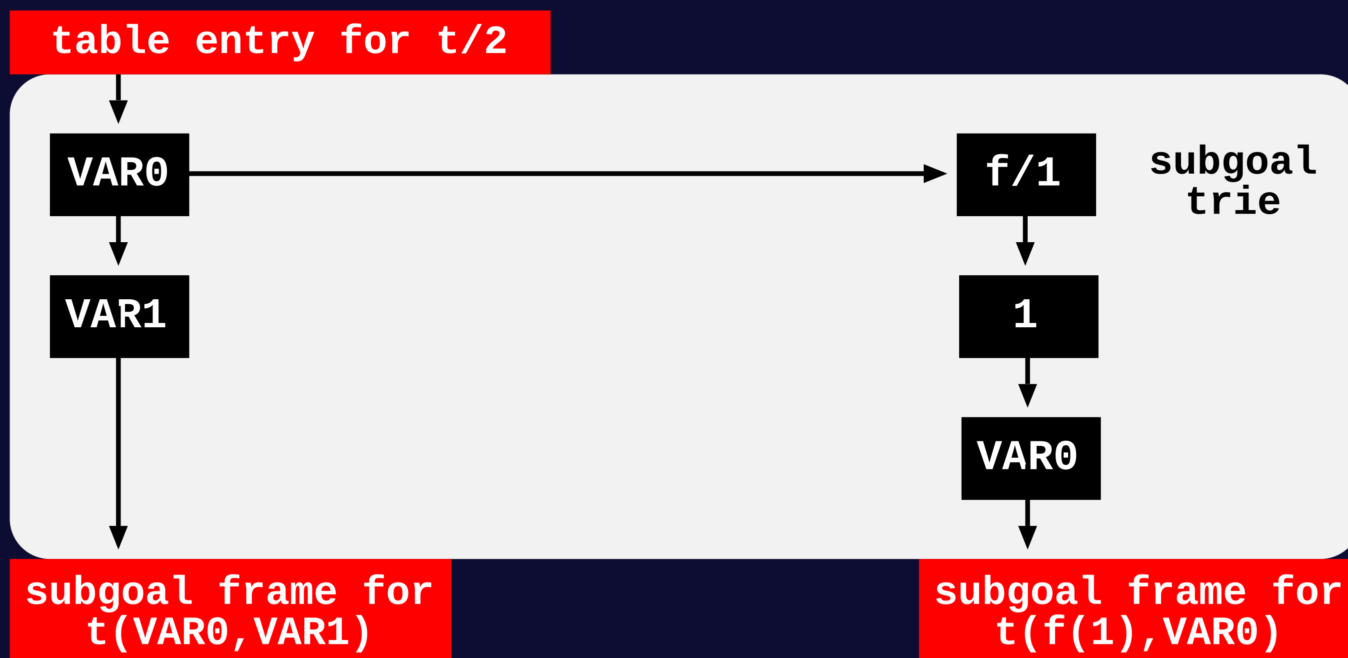
➤ Subgoal Trie

- 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

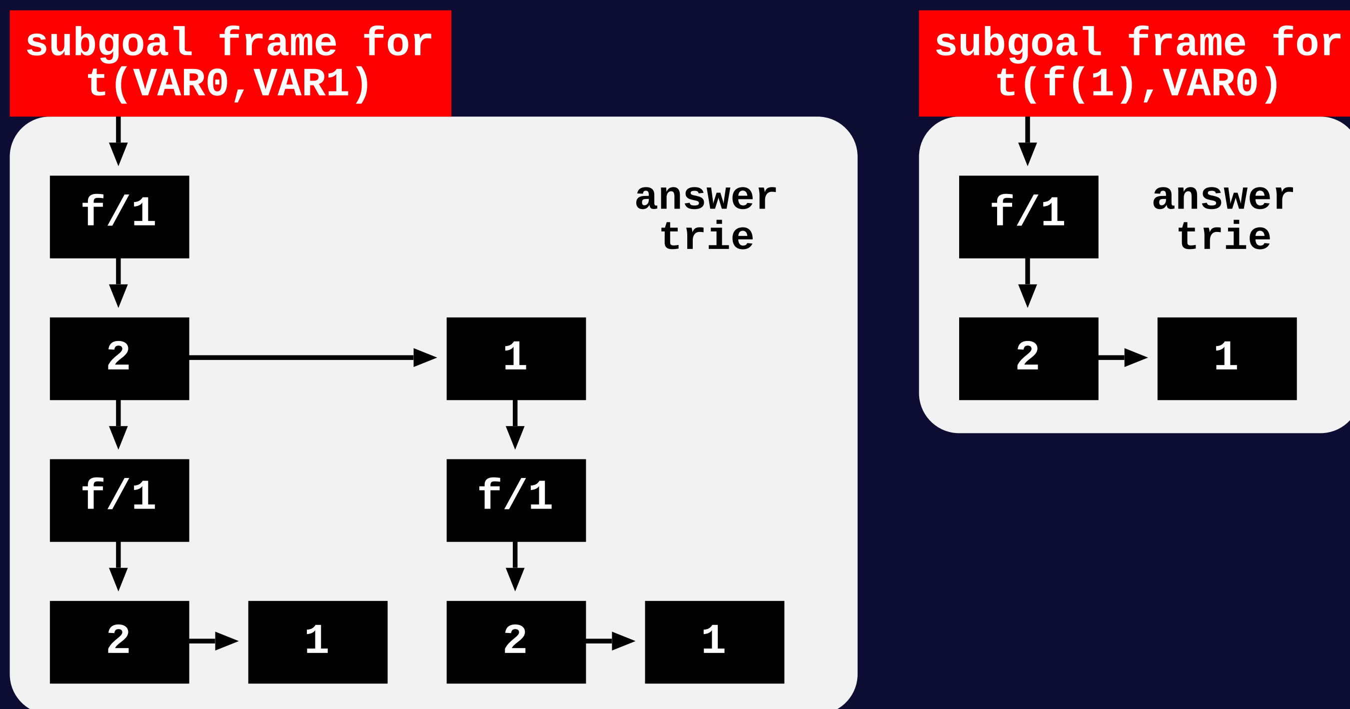
➤ Answer Trie

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

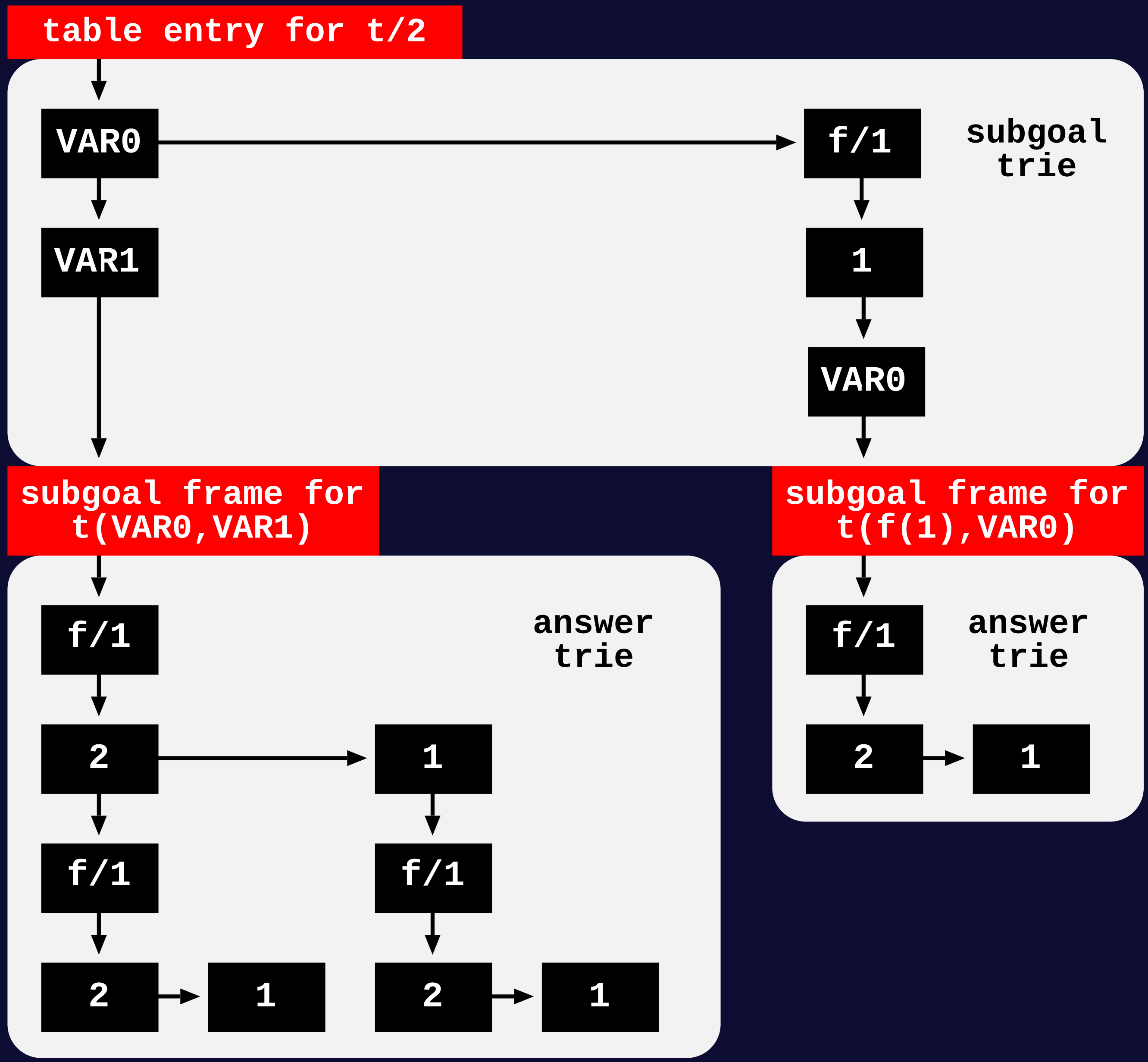
```
:- table t/2.

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

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

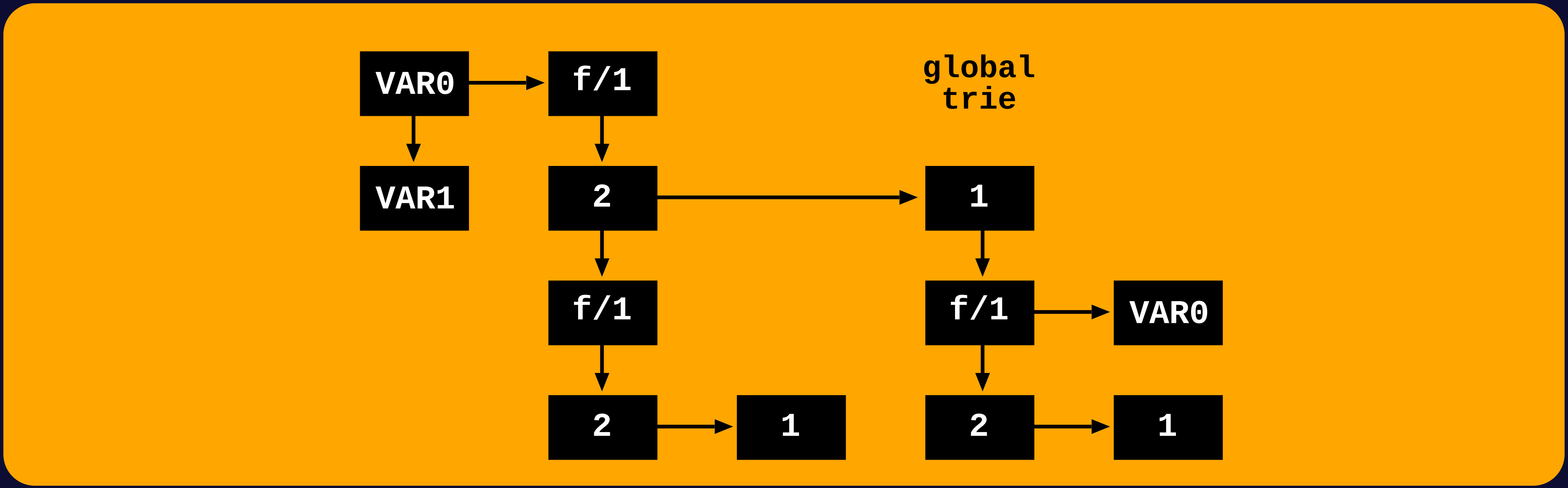


Using Tries to Represent the Table Space



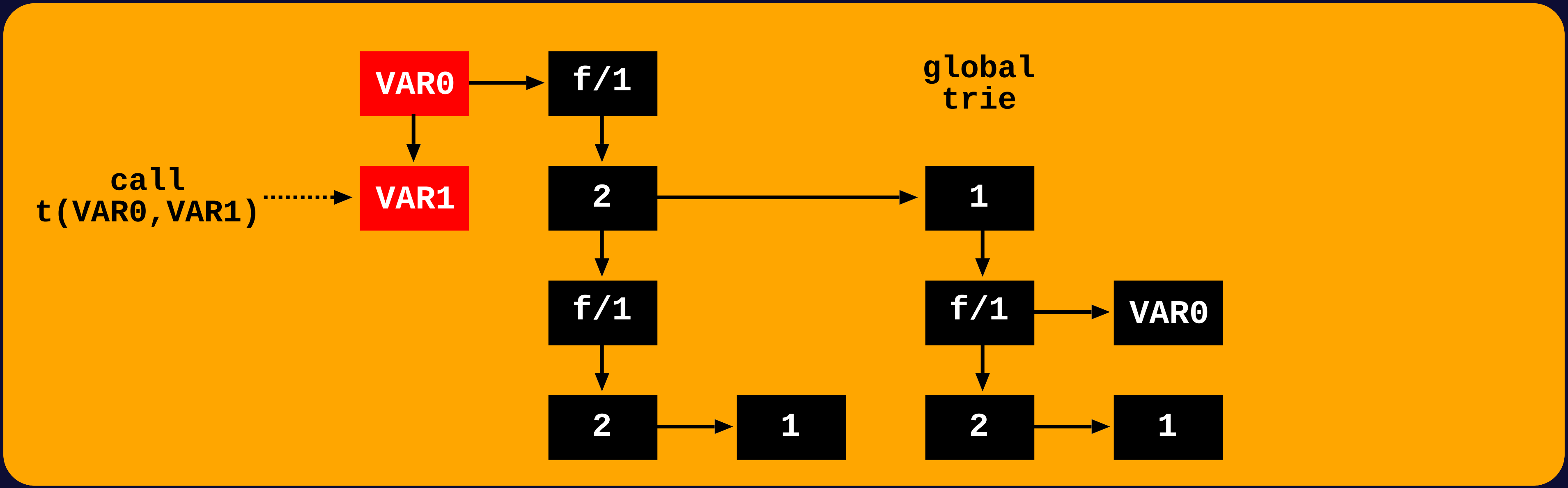
GT-CA: Global Trie for Calls and Answers [PADL09]

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT-CA)** instead of being spread over several different trie data structures.



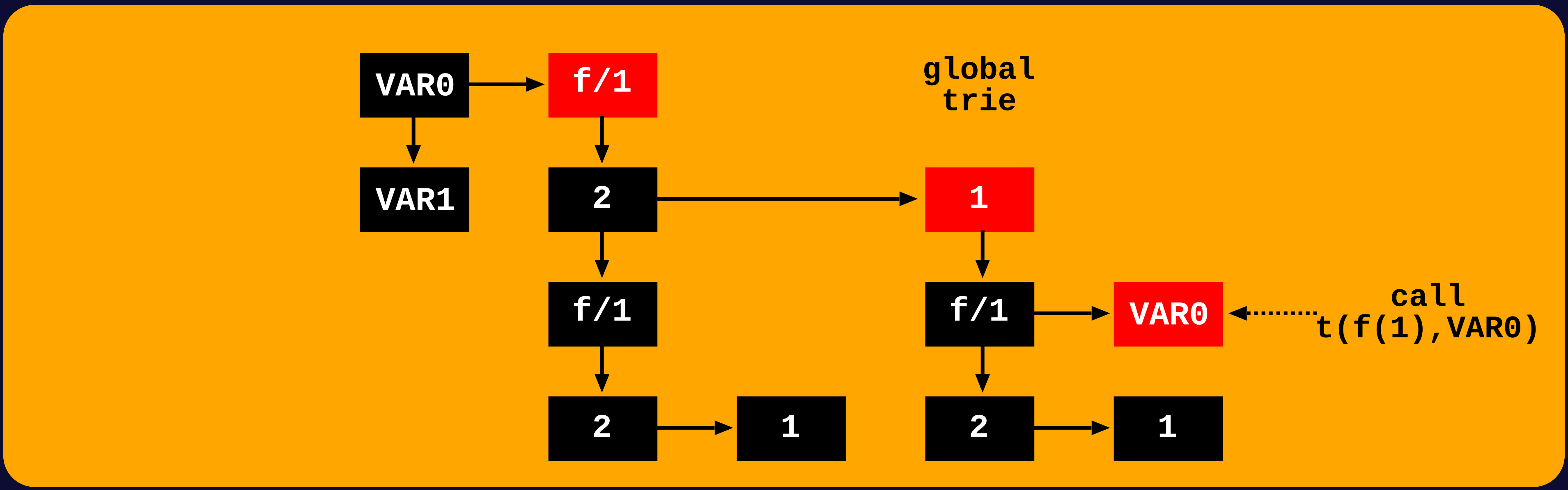
GT-CA: Global Trie for Calls and Answers [PADL09]

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT-CA)** instead of being spread over several different trie data structures.



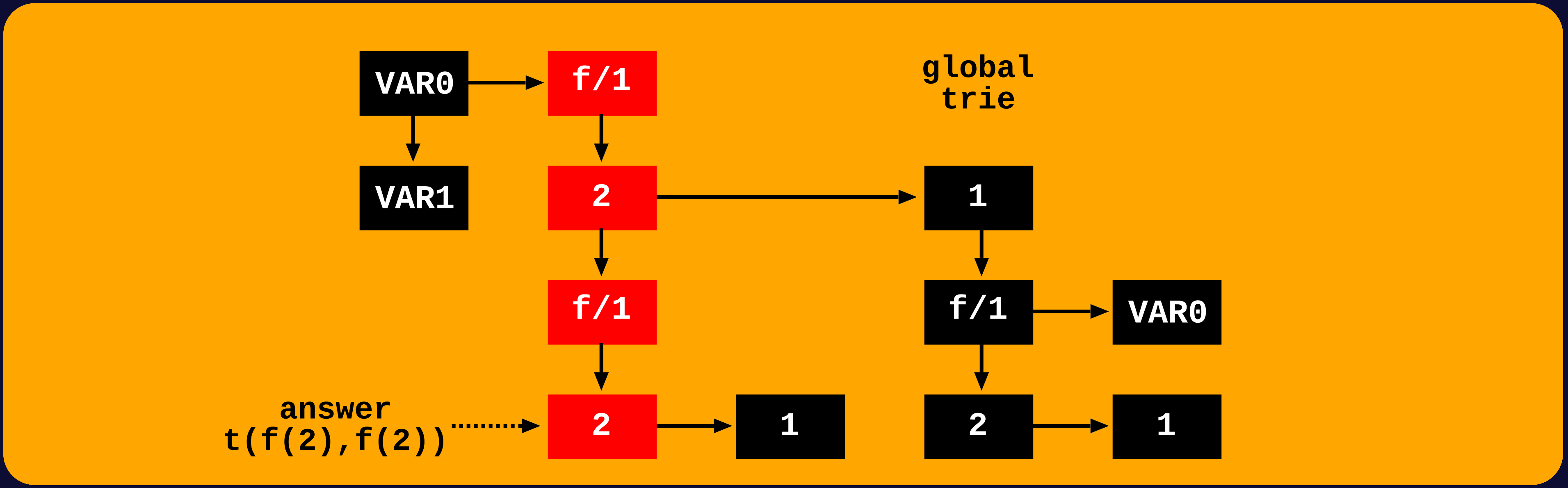
GT-CA: Global Trie for Calls and Answers [PADL09]

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT-CA)** instead of being spread over several different trie data structures.



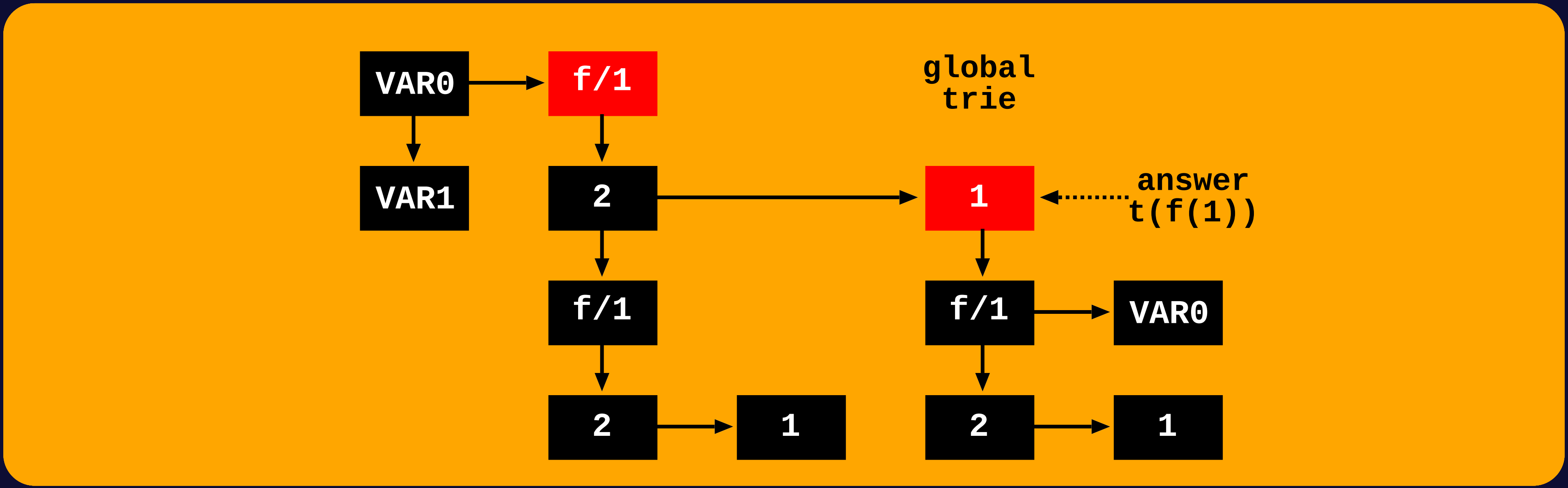
GT-CA: Global Trie for Calls and Answers [PADL09]

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT-CA)** instead of being spread over several different trie data structures.



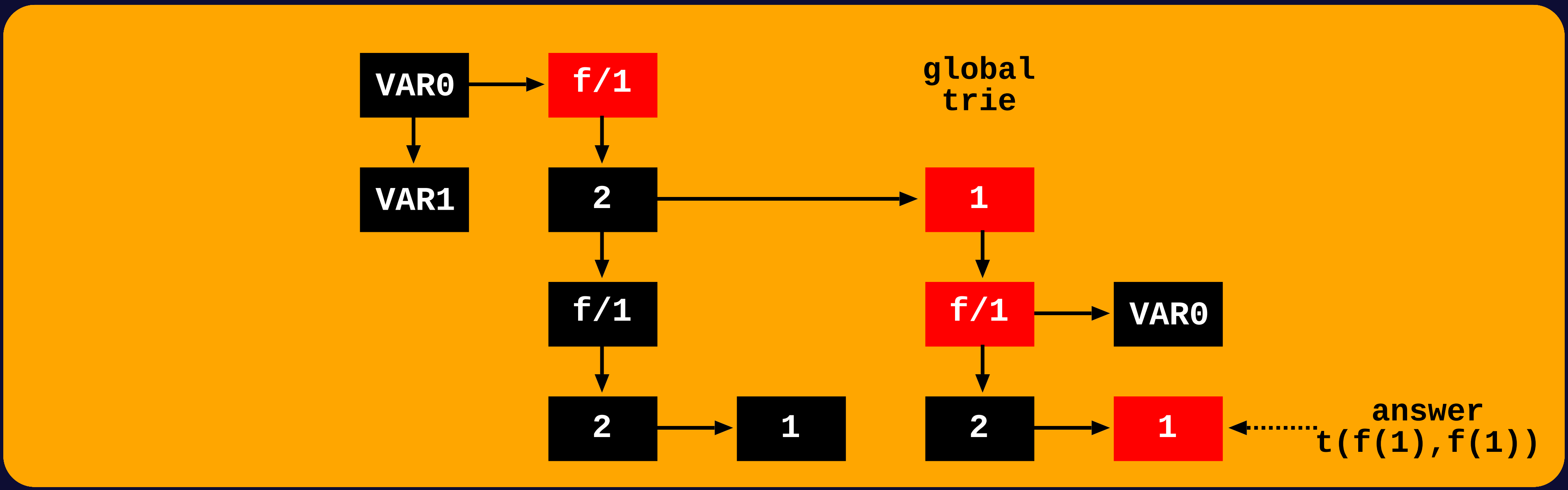
GT-CA: Global Trie for Calls and Answers [PADL09]

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT-CA)** instead of being spread over several different trie data structures.



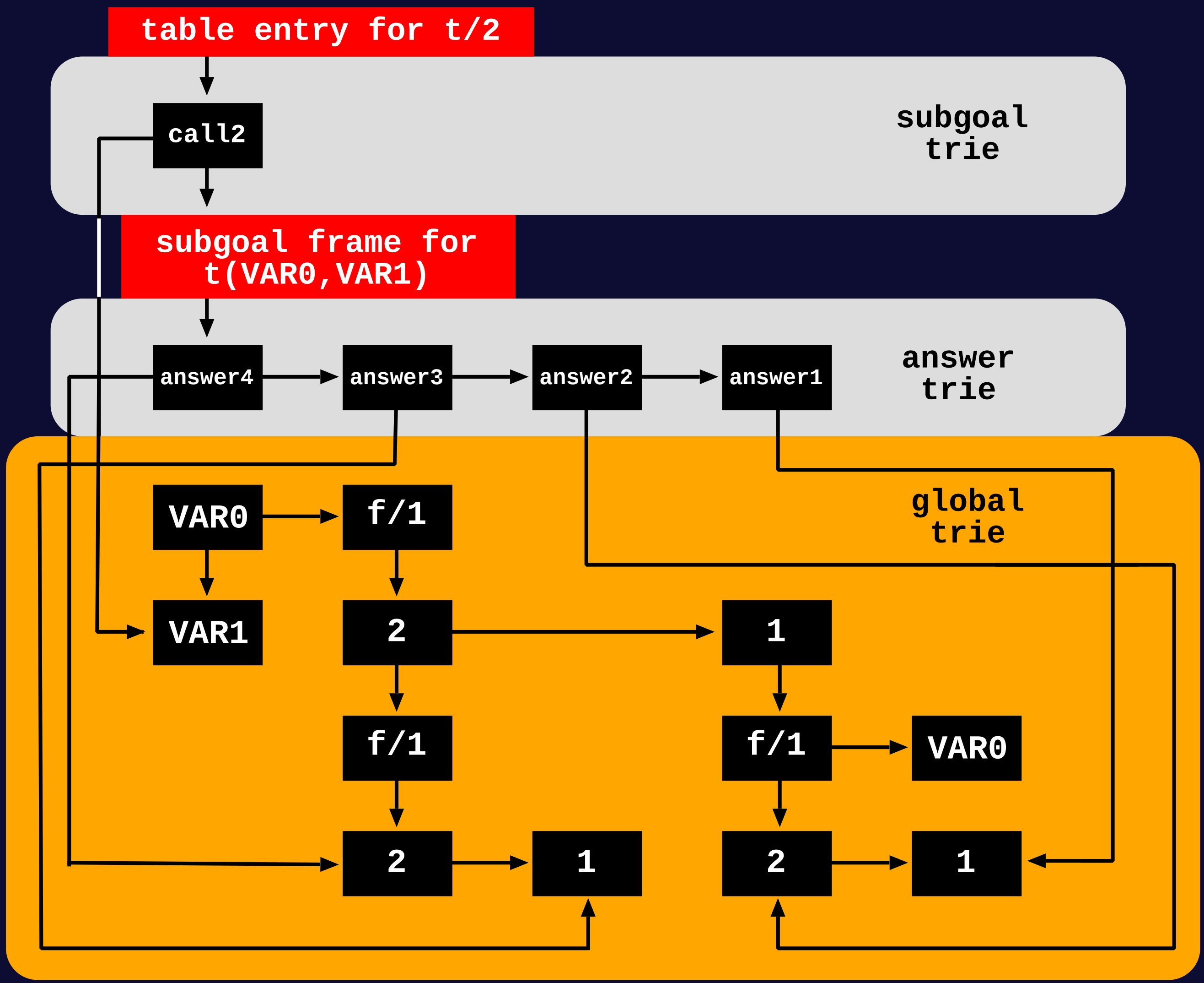
GT-CA: Global Trie for Calls and Answers [PADL09]

- All tabled subgoal calls and tabled answers are stored in a **common global trie (GT-CA)** instead of being spread over several different trie data structures.



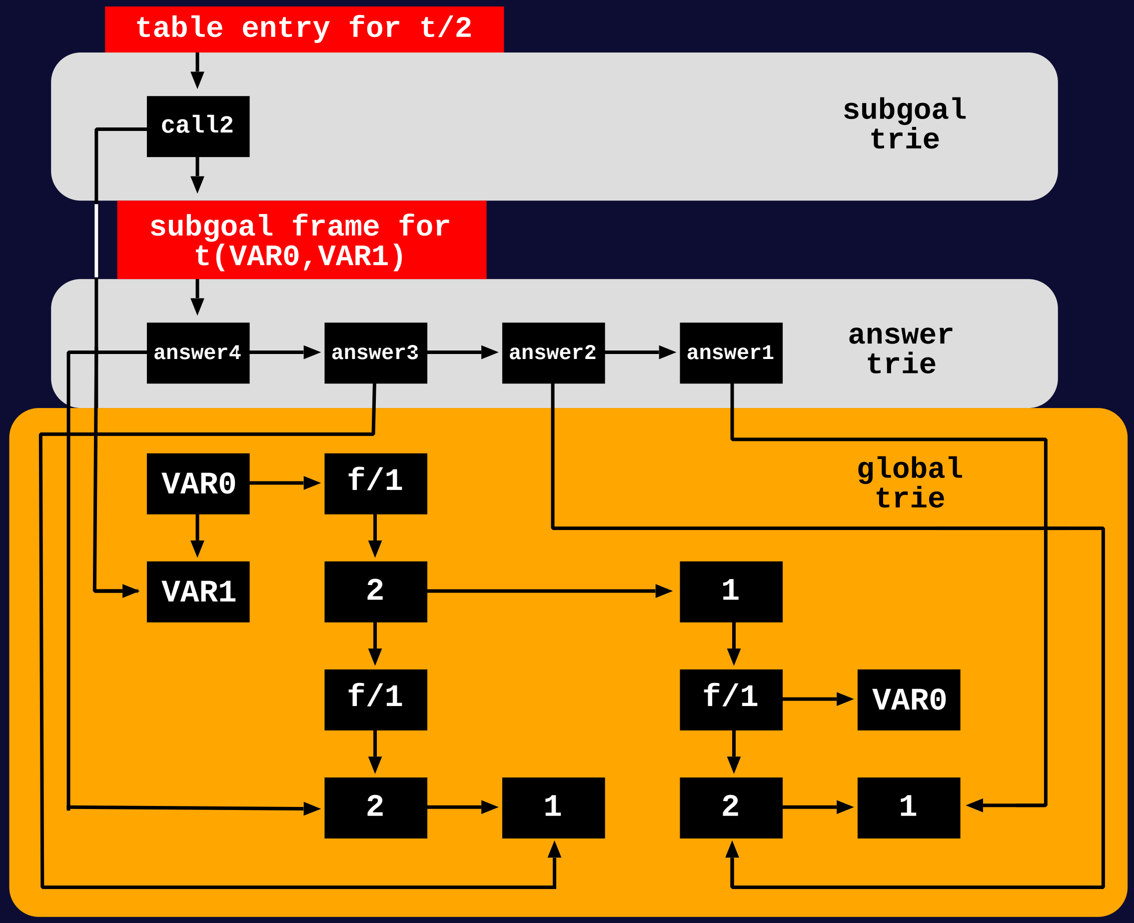
GT-CA: Global Trie for Calls and Answers [PADL09]

- 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-CA.



GT-CA: Global Trie for Calls and Answers [PADL09]

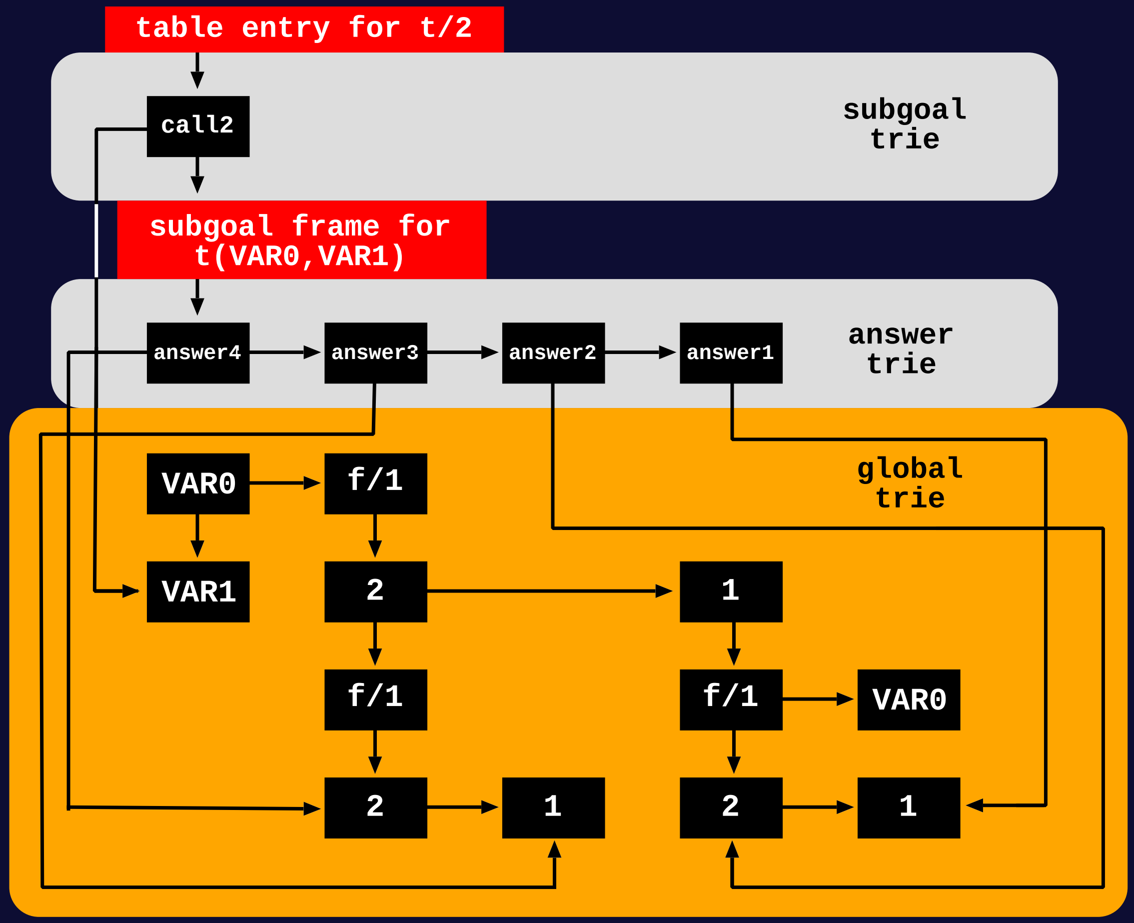
- **Implementation Problems**
  - How to deal with **table abolish operations**.



GT-CA: Global Trie for Calls and Answers [PADL09]

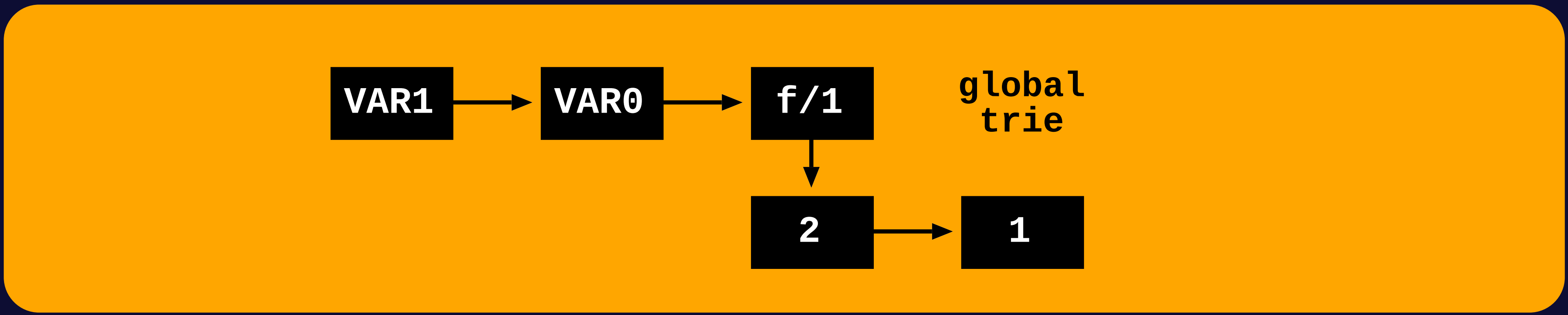
➤ Implementation Problems

- How to deal with **table abolish operations**.
- How to support the **completed table optimization**, an optimization that loads answers by executing specific WAM-like code by top-down traversing the completed answer trie.



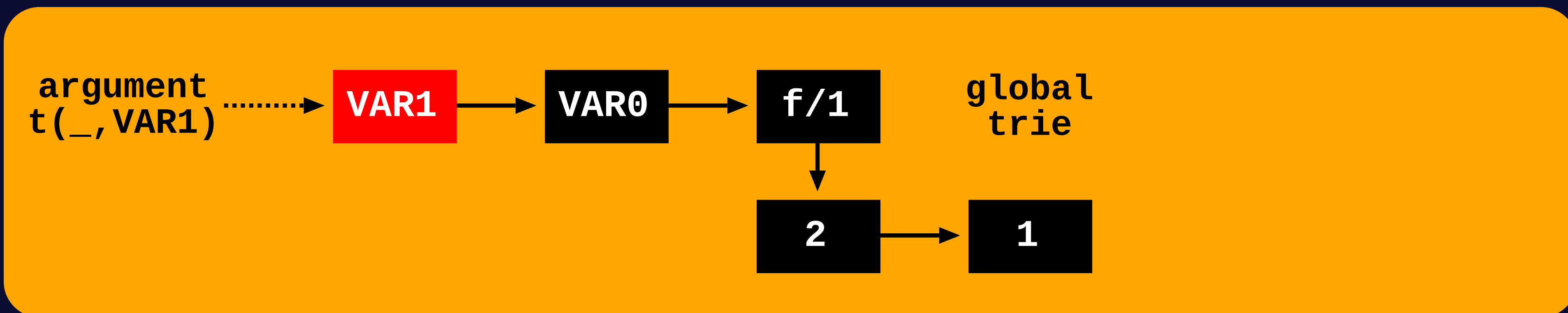
GT-T: Global Trie for Terms

- All **argument** and **substitution terms** appearing in tabled subgoal calls and/or answers are now represented only once in the **common global trie (GT-T)**.
- Each path through the trie nodes represents a **unique** argument and/or substitution term, therefore always ending at a leaf trie node.



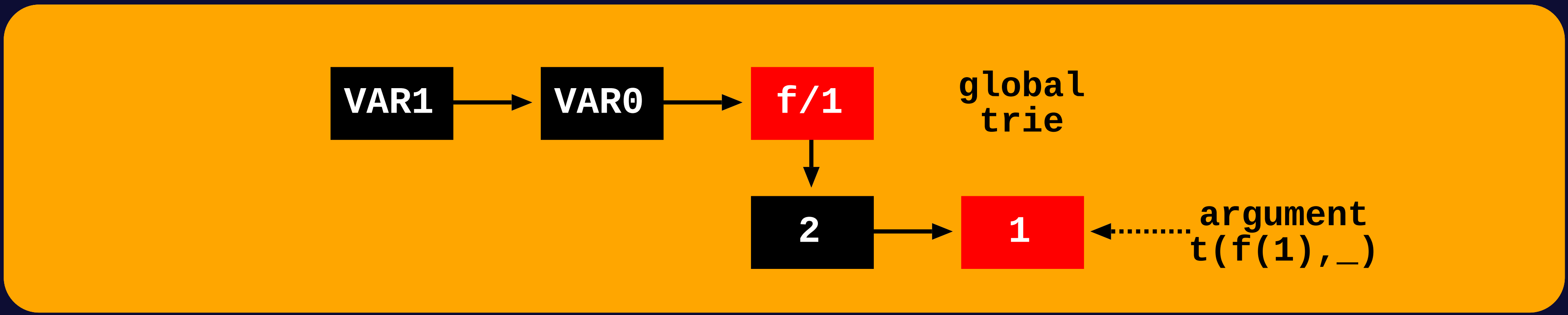
GT-T: Global Trie for Terms

- All **argument** and **substitution terms** appearing in tabled subgoal calls and/or answers are now represented only once in the **common global trie (GT-T)**.
- Each path through the trie nodes represents a **unique** argument and/or substitution term, therefore always ending at a leaf trie node.



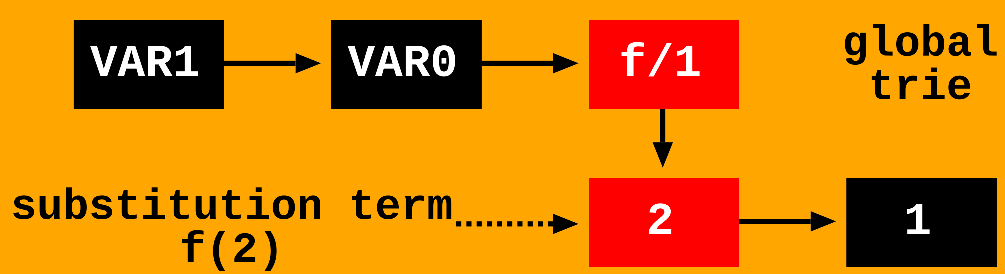
GT-T: Global Trie for Terms

- All **argument** and **substitution terms** appearing in tabled subgoal calls and/or answers are now represented only once in the **common global trie (GT-T)**.
- Each path through the trie nodes represents a **unique** argument and/or substitution term, therefore always ending at a leaf trie node.



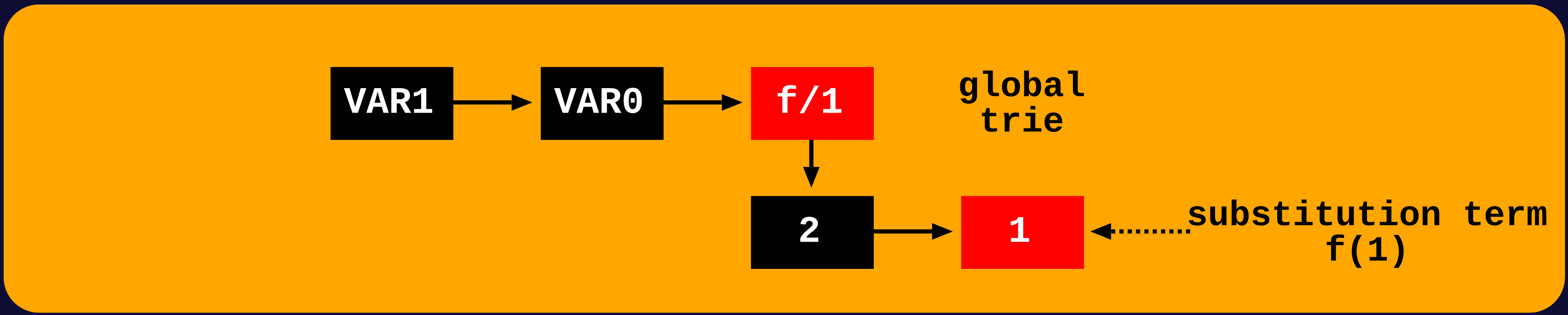
GT-T: Global Trie for Terms

- All **argument** and **substitution terms** appearing in tabled subgoal calls and/or answers are now represented only once in the **common global trie (GT-T)**.
- Each path through the trie nodes represents a **unique** argument and/or substitution term, therefore always ending at a leaf trie node.



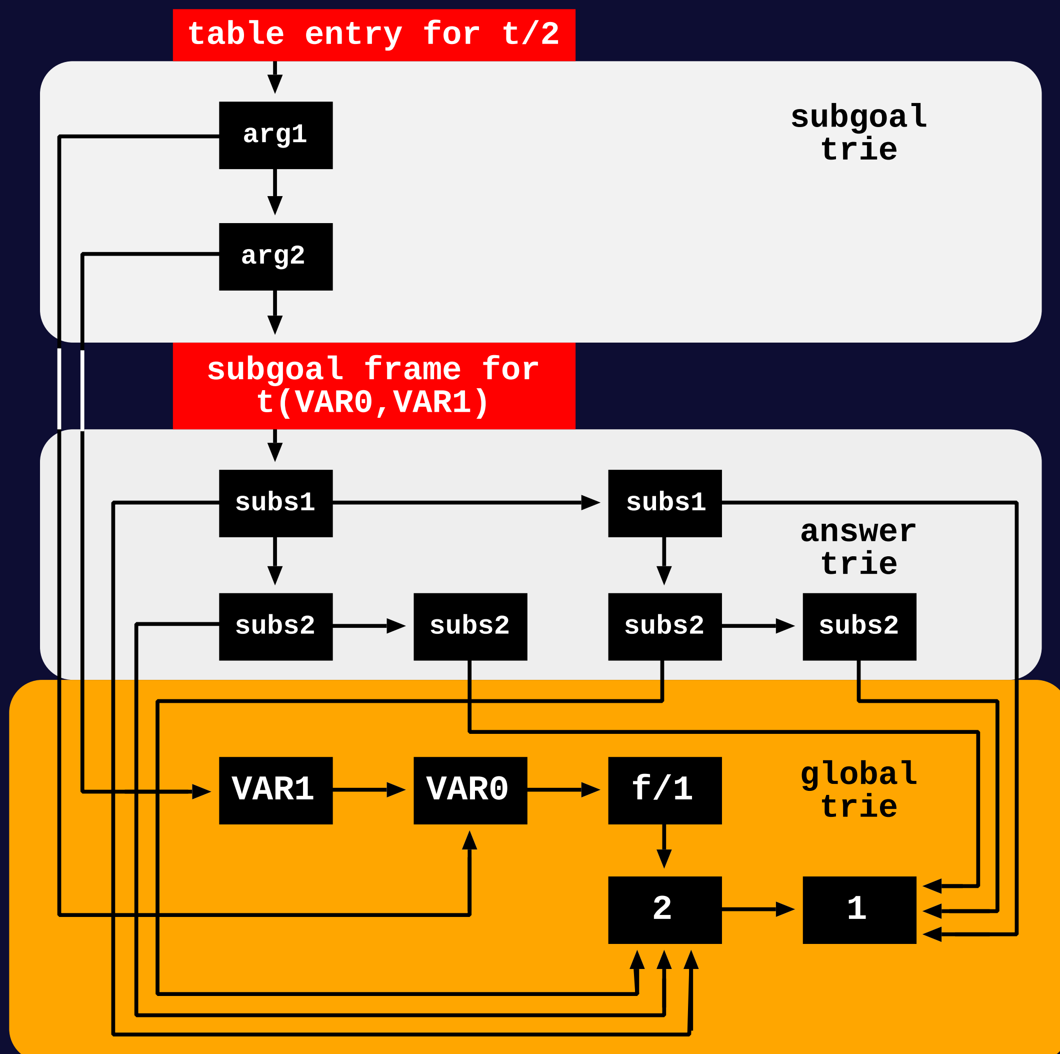
GT-T: Global Trie for Terms

- All **argument** and **substitution terms** appearing in tabled subgoal calls and/or answers are now represented only once in the **common global trie (GT-T)**.
- Each path through the trie nodes represents a **unique** argument and/or substitution term, therefore always ending at a leaf trie node.



GT-T: Global Trie for Terms

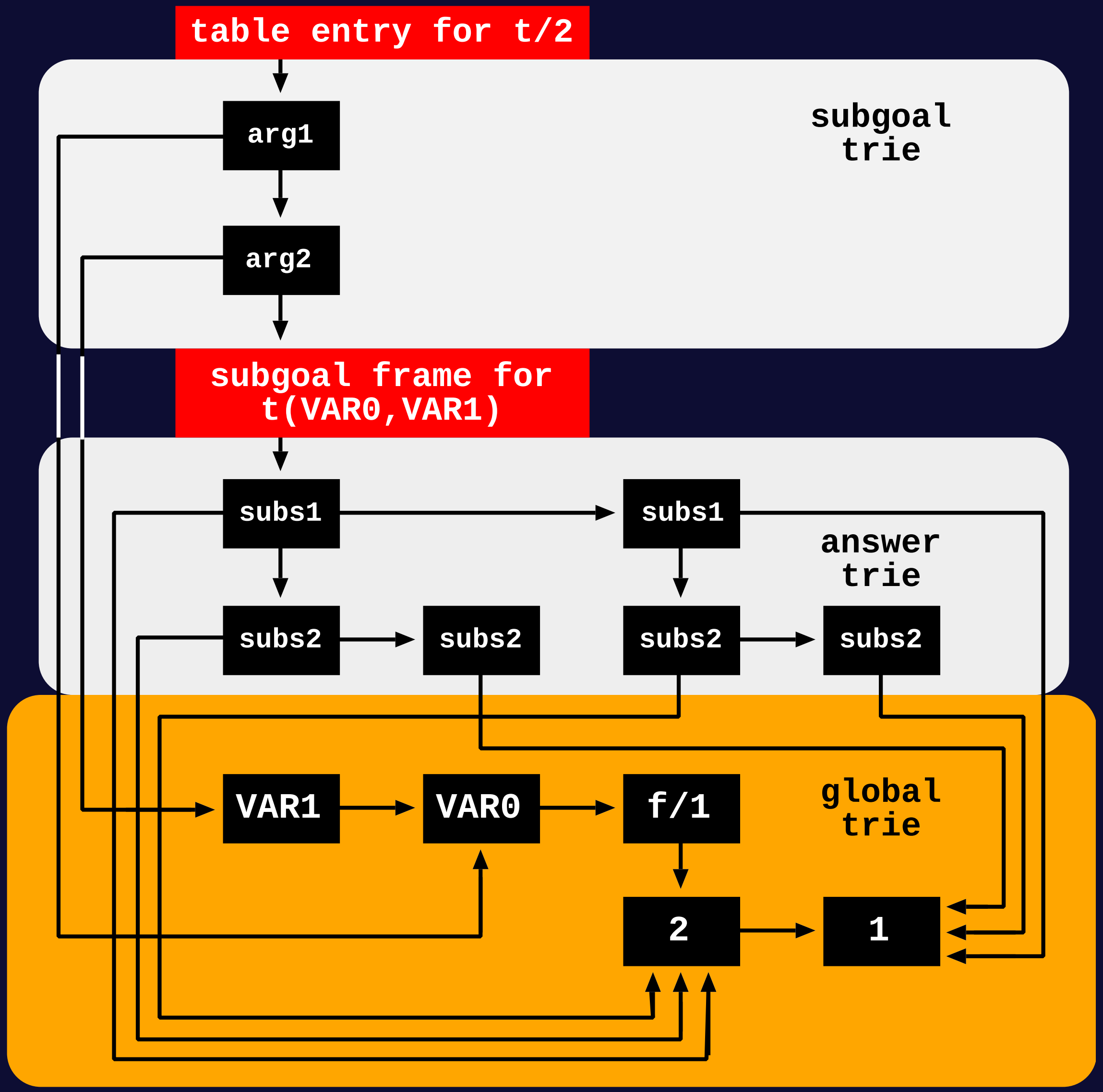
- The original subgoal trie and answer trie data structures are now composed of a fixed number of trie nodes representing the argument or substitution terms in the corresponding tabled subgoal call or tabled answer.



GT-T: Global Trie for Terms

➤ Solving GT-CA Problems

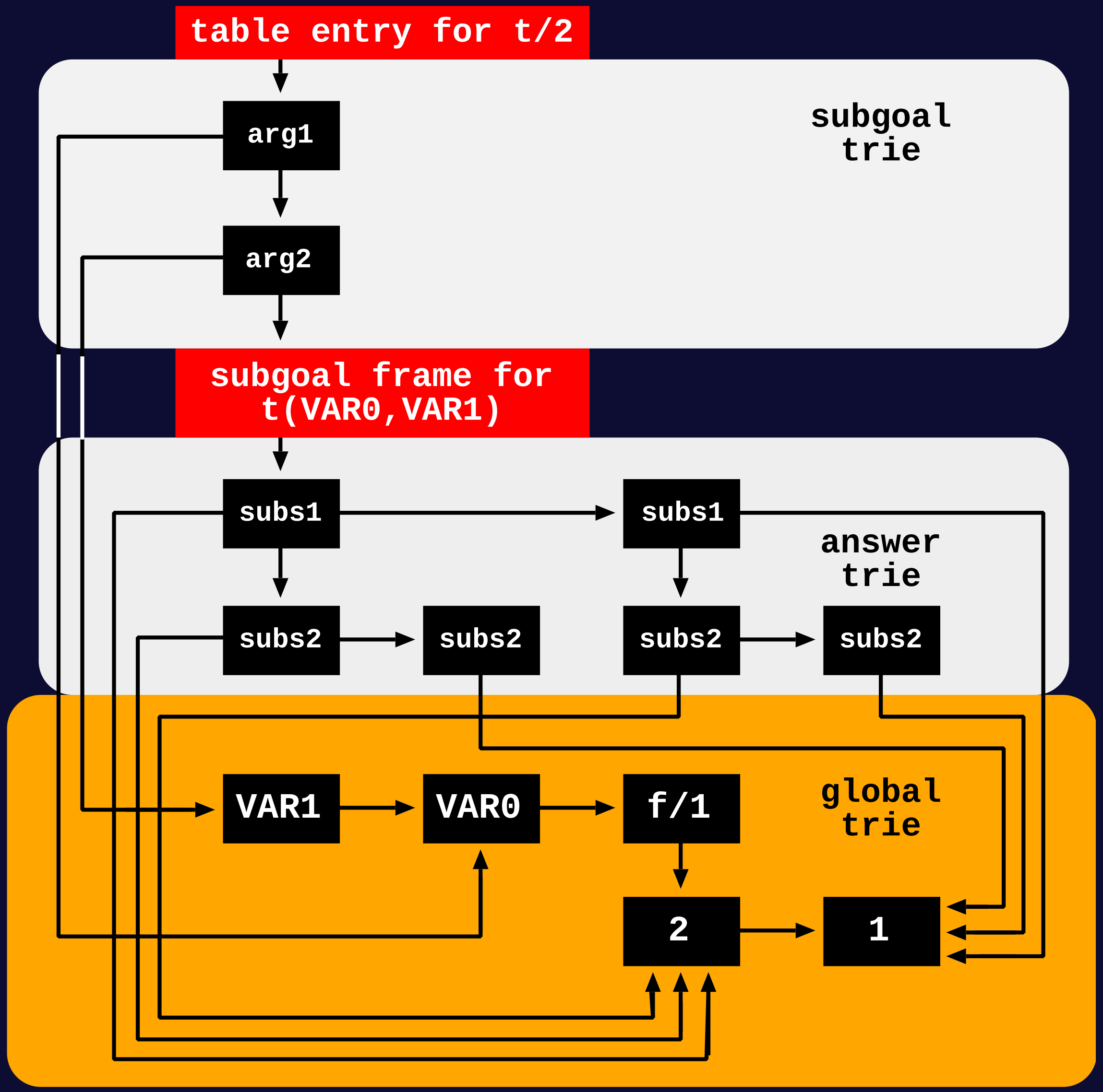
- ♦ Regarding **space reclamation**, we can use the child field of leaf nodes to count the number of paths it represents.



GT-T: Global Trie for Terms

➤ Solving GT-CA Problems

- Regarding **space reclamation**, we can use the child field of leaf nodes to count the number of paths it represents.
- Regarding **compiled tries**, the idea is to use WAM-like instructions that work at the level of the substitution terms.



Implementation Details: Tabling Operations

```
subgoal_check_insert(TABLE_ENTRY te, SUBGOAL_CALL call, N_ARGS a) {
  if (GT) {                                                // GT-T table design
    st_node = te->subgoal_trie
    for (i = 1; i <= a; i++) {
      t = get_argument_term(call, i)
      leaf_gt_node = trie_check_insert(GT, t)
      leaf_gt_node->child++  // increase number paths represented
      st_node = trie_check_insert(st_node, leaf_gt_node)
    }
    leaf_st_node = st_node
  } else                                               // original table design
    leaf_st_node = trie_check_insert(te->subgoal_trie, call)
  return leaf_st_node
}
```

Implementation Details: Tabling Operations

```
answer_check_insert(SUBGOAL_FRAME sf, ANSWER answer, N_SUBS a) {
  if (GT) {                                              // GT-T table design
    at_node = sf->answer_trie
    for (i = 1; i <= a; i++) {
      t = get_substitution_term(answer, i)
      leaf_gt_node = trie_check_insert(GT, t)
      leaf_gt_node->child++  // increase number paths represented
      at_node = trie_check_insert(at_node, leaf_gt_node)
    }
    leaf_at_node = at_node
  } else                                             // original table design
    leaf_at_node = trie_check_insert(sf->answer_trie, answer)
  return leaf_at_node
}
```

Implementation Details: Tabling Operations

```
answer_load(ANSWER_TRIE_NODE leaf_at_node, SUBS_ARITY a) {
  if (GT) {                                      // GT-T table design
    at_node = leaf_at_node
    for (i = a; i >= 1; i--) {
      leaf_gt_node = at_node->token
      t = trie_load(leaf_gt_node)
      put_substitution_term(t, answer)
      at_node = at_node->parent
    }
  } else                                     // original table design
    answer = trie_load(leaf_at_node)
  return answer
}
```

Experimental Results

Terms	GT-CA/YapTab				GT-T/YapTab			
	Mem	Store	Load	Comp	Mem	Store	Load	Comp
1000 ints	1.08	1.56	1.30	n.a.	1.00	1.32	1.18	1.69
1000 atoms	1.08	1.54	1.41	n.a.	1.00	1.26	1.24	1.54
1000 f/1	1.08	1.35	1.33	n.a.	1.00	1.28	1.11	1.88
1000 f/2	**0.58**	1.25	1.37	n.a.	**0.50**	1.11	1.18	1.58
1000 f/4	**0.33**	1.21	1.35	n.a.	**0.25**	1.07	1.16	1.14
1000 f/6	**0.25**	1.12	1.29	n.a.	**0.17**	1.01	1.05	1.08
1000 [ ]/1	**0.58**	1.32	1.44	n.a.	**0.50**	1.17	1.21	1.29
1000 [ ]/2	**0.33**	1.06	1.55	n.a.	**0.25**	**0.93**	1.20	1.48
1000 [ ]/4	**0.20**	1.10	1.57	n.a.	**0.13**	**0.81**	1.01	1.28
1000 [ ]/6	**0.16**	1.02	1.05	n.a.	**0.08**	**0.71**	**0.58**	**0.68**
Average	**0.57**	**1.25**	**1.37**	**n.a.**	**0.49**	**1.07**	**1.09**	**1.36**

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

Data Sets	GT-CA/YapTab Mem	GT-CA/YapTab Store	GT-CA/YapTab Load	GT-CA/YapTab Comp	GT-T/YapTab Mem	GT-T/YapTab Store	GT-T/YapTab Load	GT-T/YapTab Comp
Pred								
Carc_P1	**0.82**	1.35	1.34	n.a.	**0.62**	1.07	1.05	1.03
Carc_P2	**0.87**	1.42	1.44	n.a.	**0.51**	1.23	1.30	1.22
Muta_P1	**0.73**	1.20	1.19	n.a.	**0.63**	**0.91**	1.00	**0.94**
Muta_P2	**0.73**	1.26	1.47	n.a.	**0.63**	**0.96**	1.22	1.10
Average	**0.79**	**1.31**	**1.36**	**n.a.**	**0.60**	**1.04**	**1.14**	**1.07**
Conj								
Carc_C1	**0.53**	1.57	1.63	n.a.	**0.39**	1.20	1.22	1.08
Carc_C2	**0.50**	1.50	1.50	n.a.	**0.14**	1.11	1.09	**0.82**
Muta_C1	**0.66**	1.30	1.65	n.a.	**0.53**	**0.99**	1.22	1.35
Muta_C2	**0.16**	1.25	1.42	n.a.	**0.16**	**0.98**	1.10	**0.78**
Average	**0.46**	**1.41**	**1.55**	**n.a.**	**0.31**	**1.07**	**1.16**	**1.01**

Memory usage and store/load times for two well-known ILP benchmarks: the *carcinogenesis* (**Carc**) and *mutagenesis* (**Muta**) data sets, evaluated by tabling individual predicates (**Pred**) and by tabling literal conjunctions (**Conj**).

Conclusions and Further Work

- We have presented a new design for the table space organization that uses a common global trie to store all the terms appearing in tabled subgoal calls and/or answers.
- Our goal is to reduce redundancy in term representation, thus saving memory by sharing data that is structurally equal.
- Our experiments showed that our approach has potential to achieve significant reductions on memory usage without compromising running time.
- As further work we intend to study how alternative/complementary designs for the table space can further reduce redundancy in term representation.

Questions ?