

# Tabling Logic Programs in a Database

Pedro Costa, Ricardo Rocha and Michel Ferreira

DCC-FC & LIACC

University of Porto, Portugal

*c0370061@dcc.fc.up.pt*

*{ricroc,michel}@ncc.up.pt*

# 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 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 applications:
  - ◆ Model Checking
  - ◆ Program Analysis
  - ◆ Deductive Databases
  - ◆ Non-Monotonic Reasoning
  - ◆ Natural Language Processing

# Motivation

## ➤ Problem

- ◆ In general, tables are **in-memory** data structures.
- ◆ Applications that build **very many or very large tables** can quickly **run out of memory**.

# Motivation

## ➤ Problem

- ◆ In general, tables are **in-memory** data structures.
- ◆ Applications that build **very many or very large tables** can quickly **run out of memory**.

## ➤ Solution I

- ◆ Have a set of primitives that the programmer can use to dynamically abolish some of the tables.
- ◆ Difficult to use and to decide what are the **potentially useless tables** that should be deleted.

# Motivation

## ➤ Problem

- ◆ In general, tables are **in-memory** data structures.
- ◆ Applications that build **very many or very large tables** can quickly **run out of memory**.

## ➤ Solution I

- ◆ Have a set of primitives that the programmer can use to dynamically abolish some of the tables.
- ◆ Difficult to use and to decide what are the **potentially useless tables** that should be deleted.

## ➤ Solution II

- ◆ YapTab's memory management algorithm that **automatically** recovers space from the **least recently used tables** when the system runs out of memory.

# Motivation

## ➤ Problem with Solutions I and II

- ◆ We lose the already found answers for the deleted tables.
- ◆ When a repeated call appears, we need to **recompute** the set of answers from the beginning.

# Motivation

## ➤ Problem with Solutions I and II

- ◆ We lose the already found answers for the deleted tables.
- ◆ When a repeated call appears, we need to **recompute** the set of answers from the beginning.

## ➤ Our Proposal

- ◆ Store tables externally using a **relational database management system**.
- ◆ When a repeated call appears, we load the stored answers from the database hence avoiding recomputing them.
- ◆ With this approach, we can still use YapTab's memory management algorithm when the system runs out of memory, but instead of deleting tables, we can use it to decide what tables we should move to database storage.

# 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.

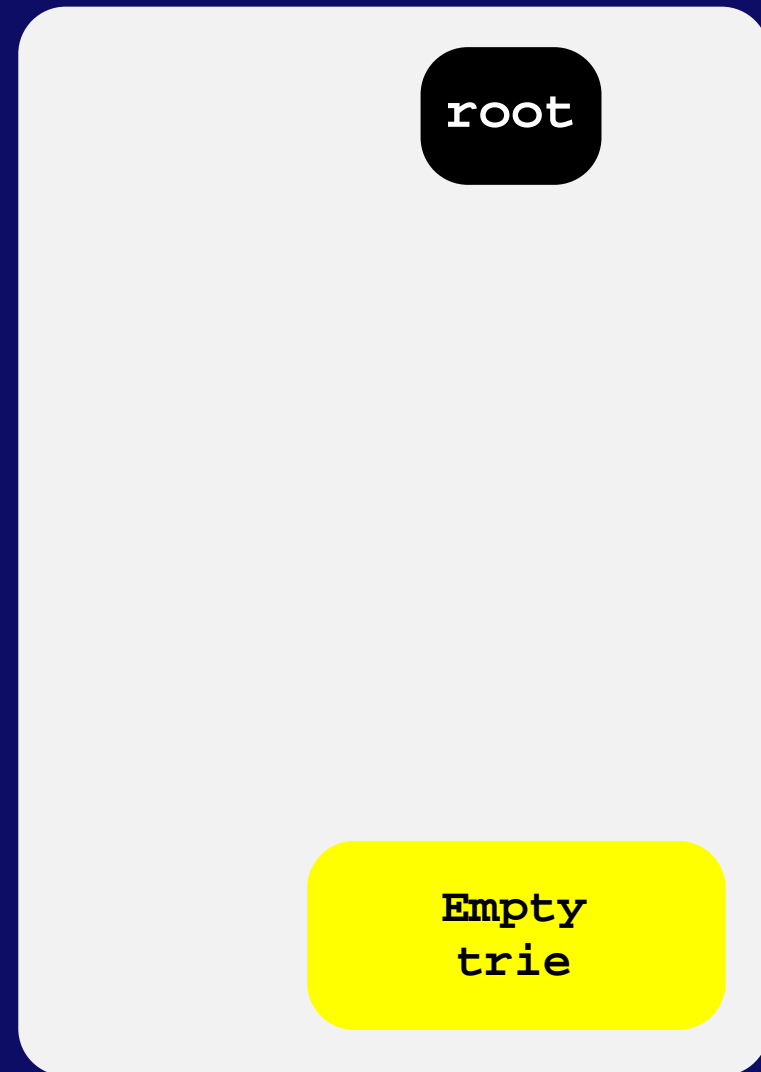
## ➤ Implementation requirements:

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



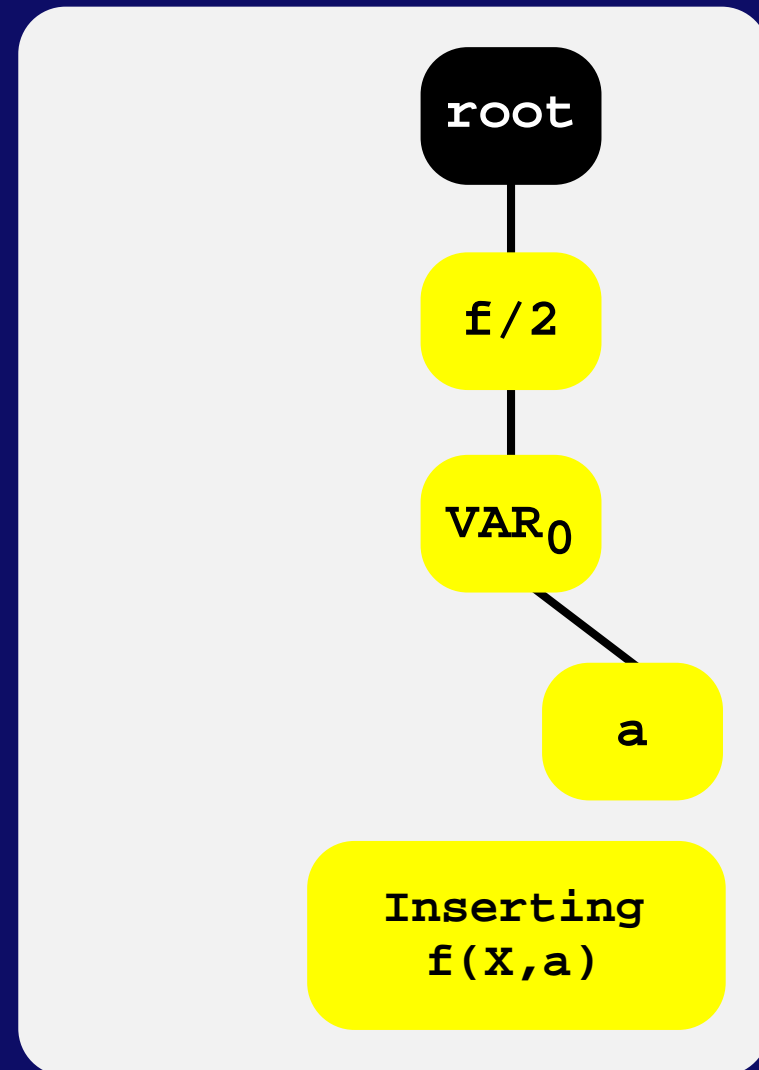
# Using Tries to Represent Terms

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



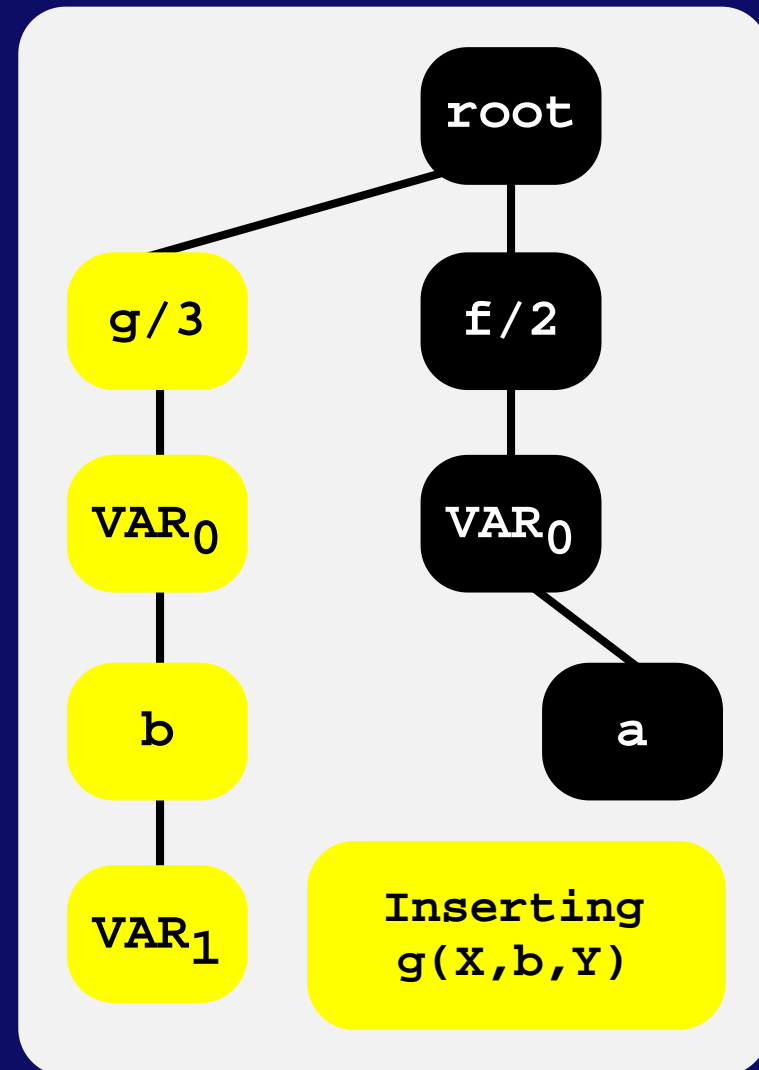
# Using Tries to Represent Terms

- 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.



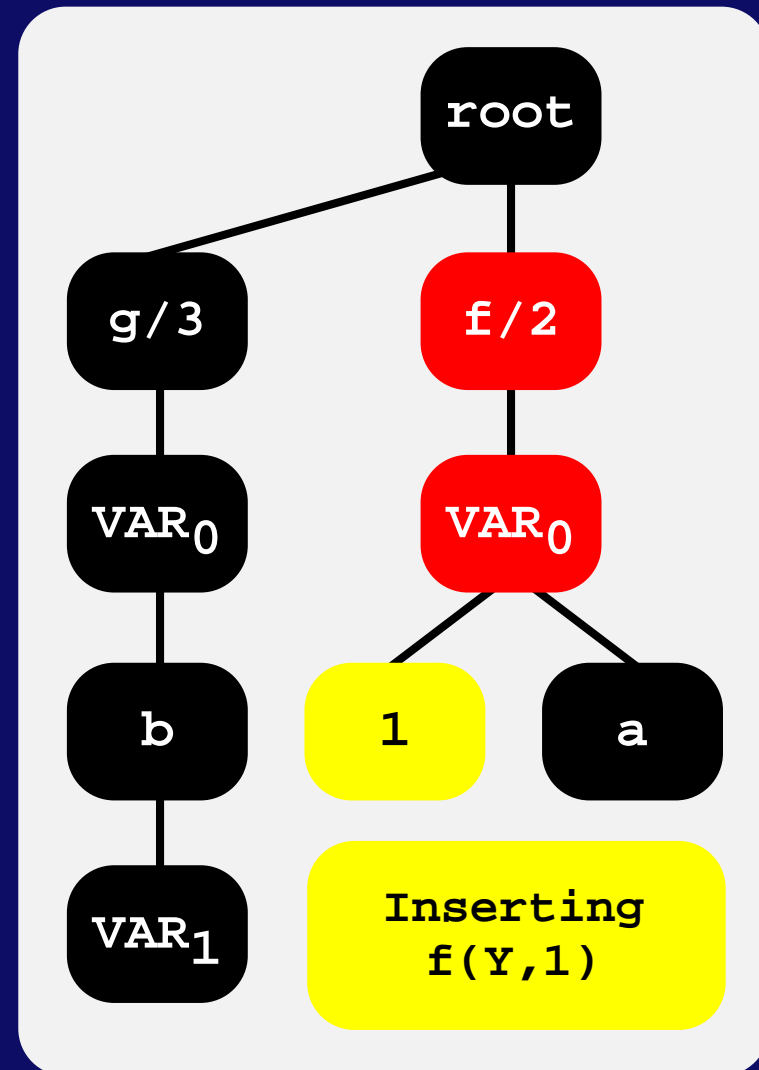
# Using Tries to Represent Terms

- 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.



# Using Tries to Represent Terms

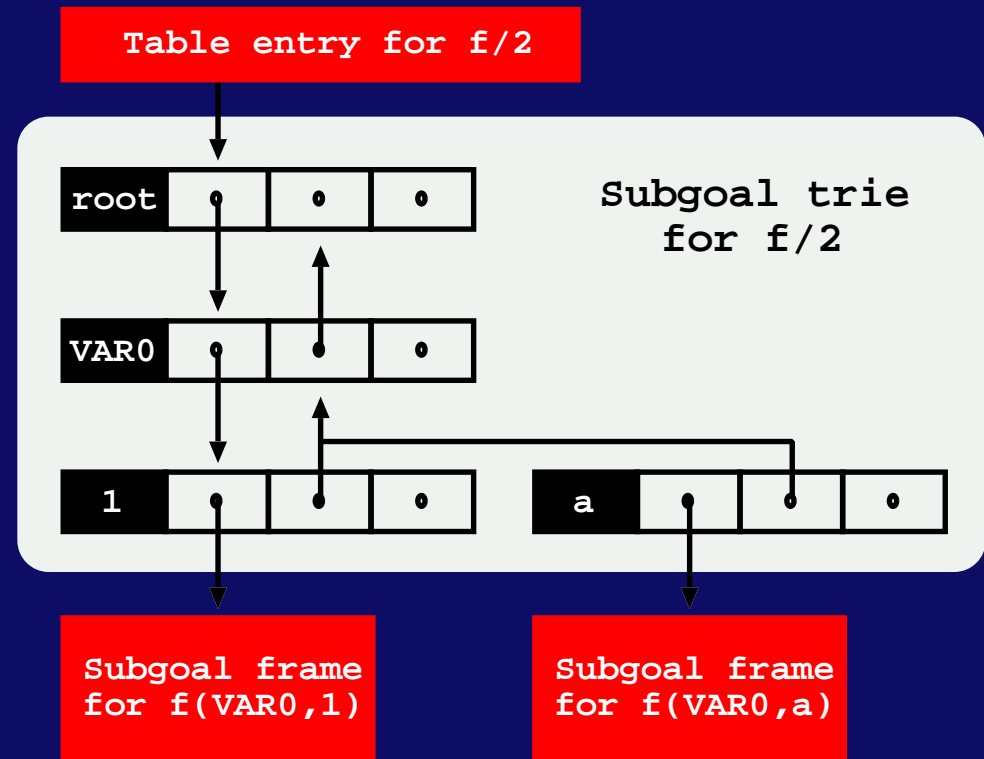
- Tries are trees in which common prefixes are represented only once.
  - ◆ The entry point is called the root node, internal nodes represent symbols in terms and leaf nodes specify completed terms.
  - ◆ Each different path through the nodes in the trie corresponds to a term.
  - ◆ Terms with common 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.



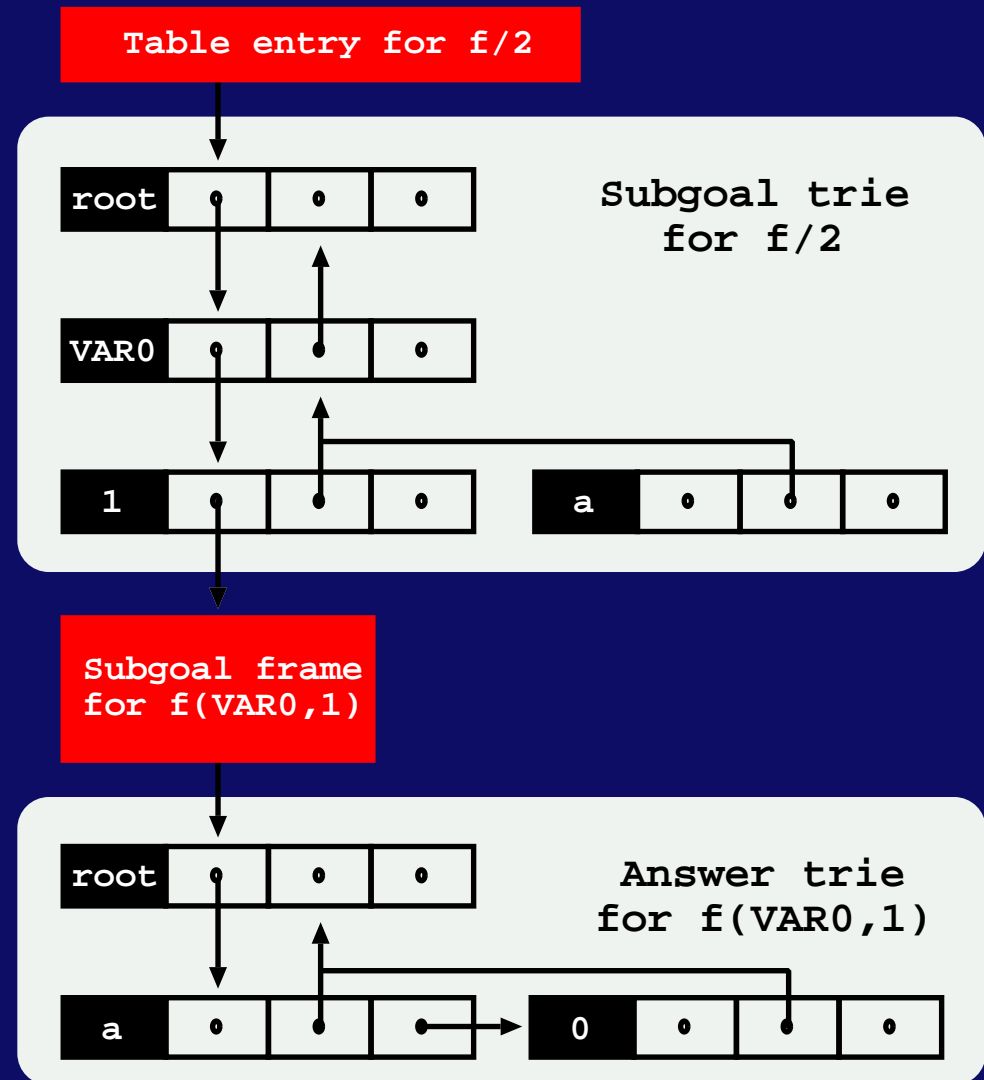
# Using Tries to Organise the Table Space

## ➤ 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

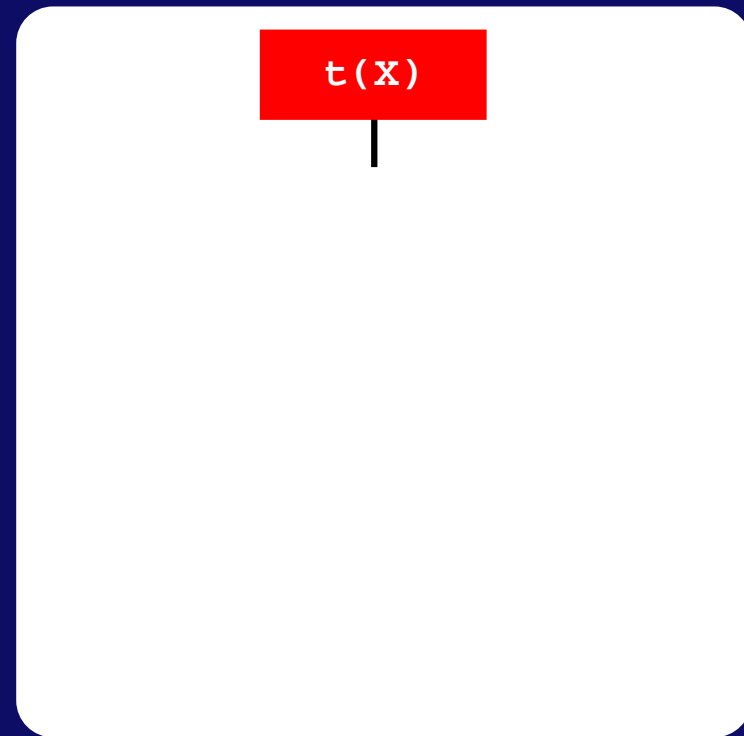
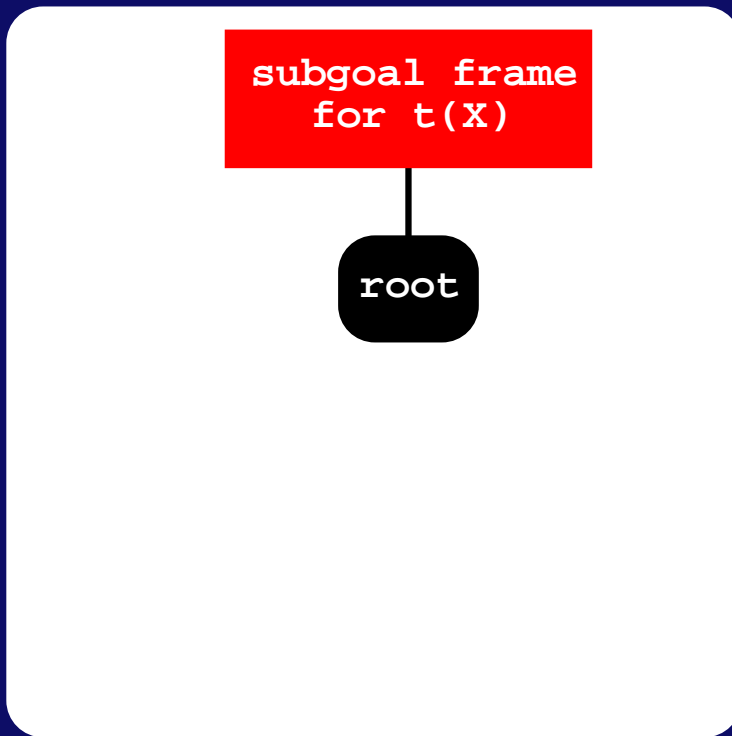
- ◆ Stores the subgoal answers.



# Using Tries to Organise the Table Space

## ➤ Answer Trie Structure

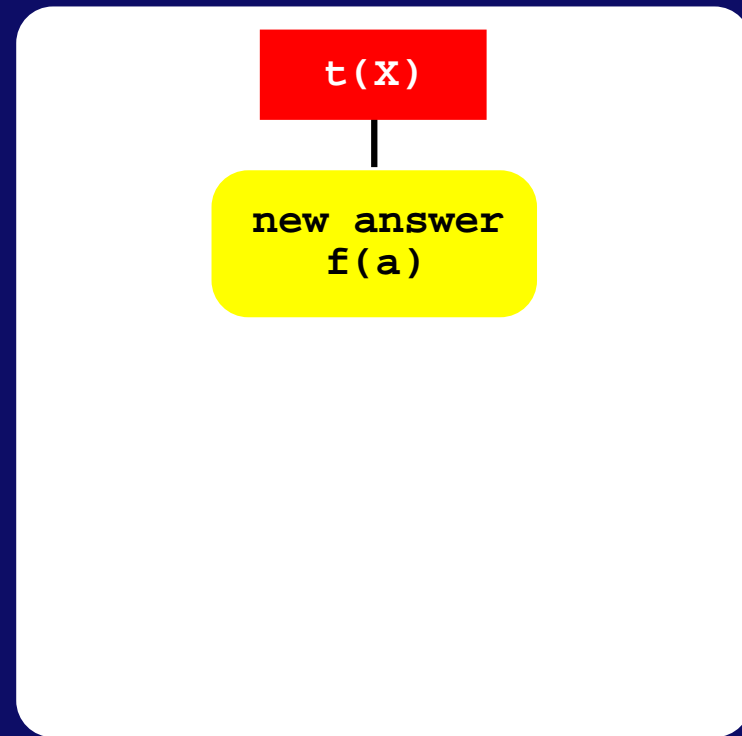
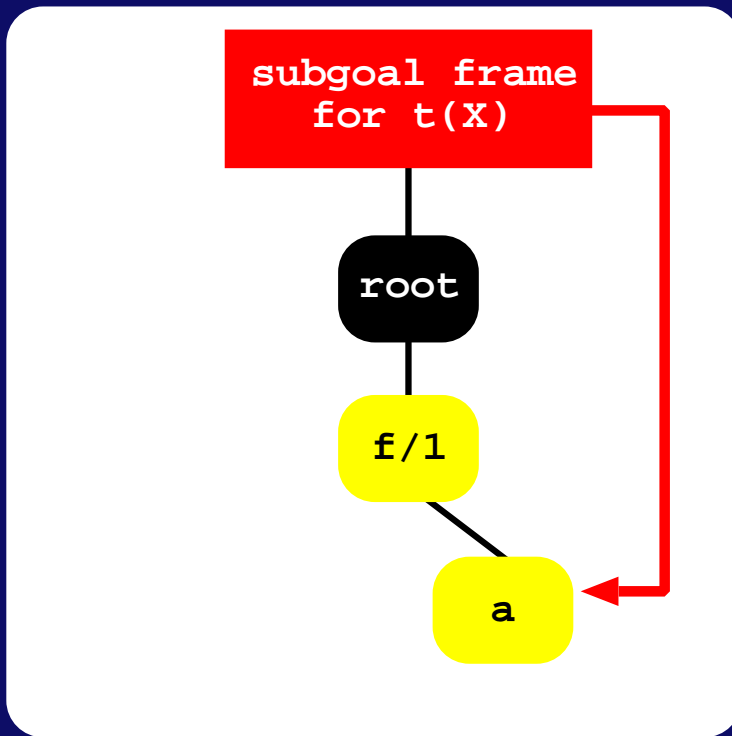
- ◆ Leaf nodes are chained in insertion time order.



# Using Tries to Organise the Table Space

## ➤ Answer Trie Structure

- ◆ Leaf nodes are chained in insertion time order.

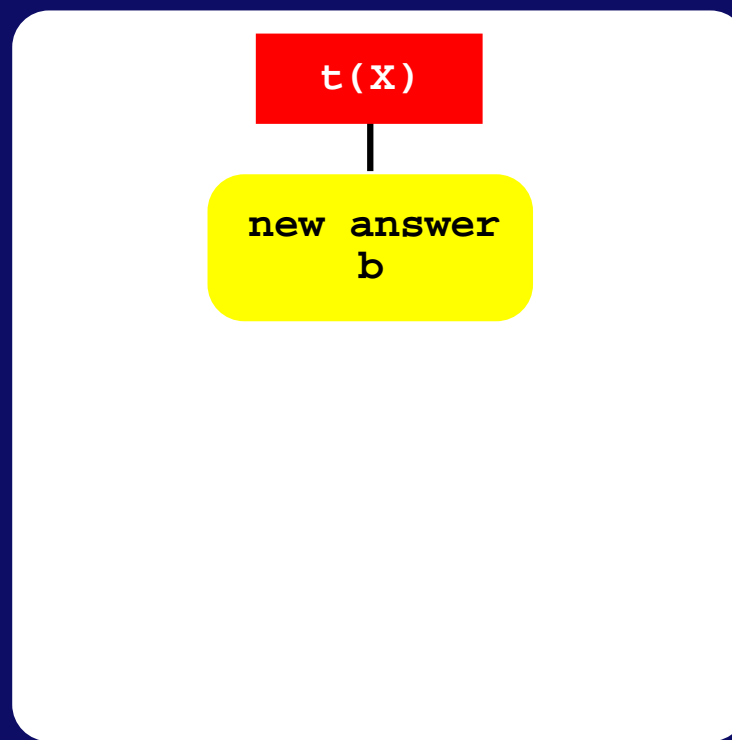
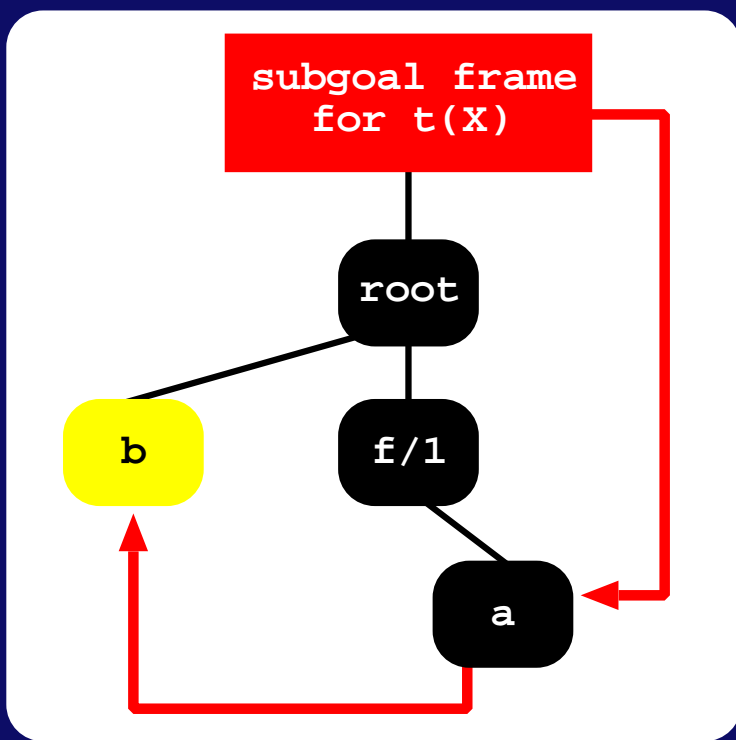




# Using Tries to Organise the Table Space

## ➤ Answer Trie Structure

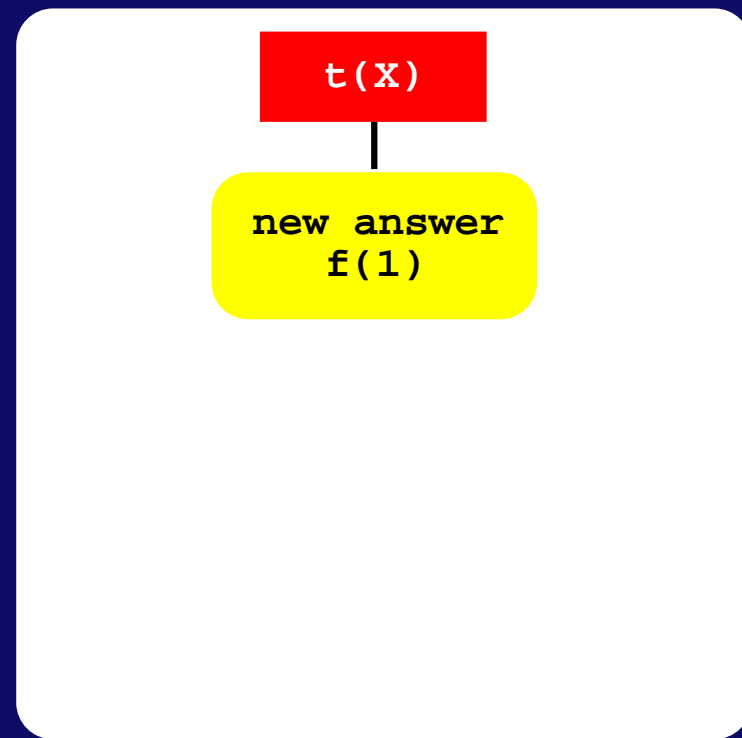
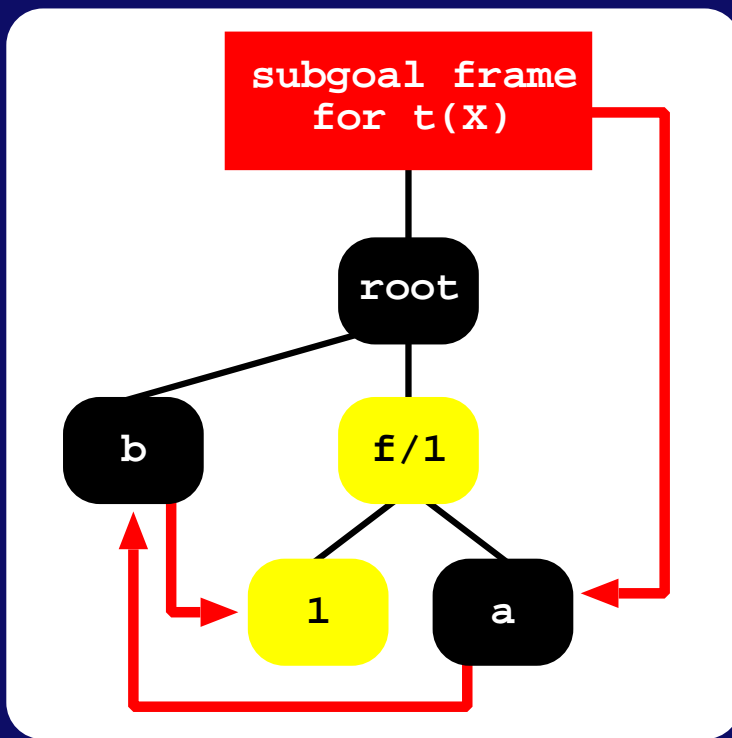
- ◆ Leaf nodes are chained in insertion time order.



# Using Tries to Organise the Table Space

## ➤ Answer Trie Structure

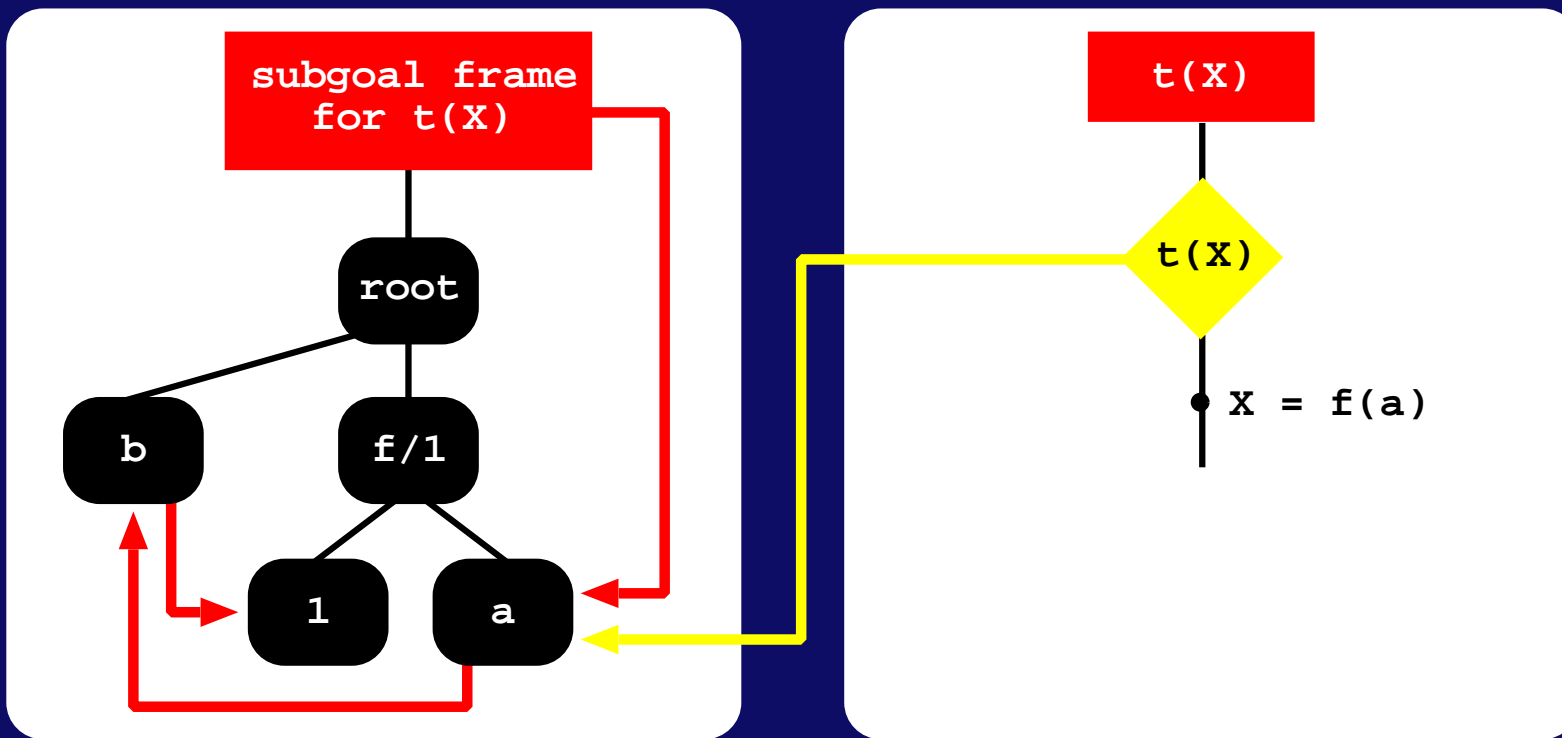
- ◆ Leaf nodes are chained in insertion time order.



# Using Tries to Organise the Table Space

## ➤ Answer Trie Structure

- ◆ Leaf nodes are chained in insertion time order.
- ◆ Repeated calls keep a reference to the leaf node of the last consumed answer, hence can consume more answers by following the chain.



# The DBTAB Relational Storage Model

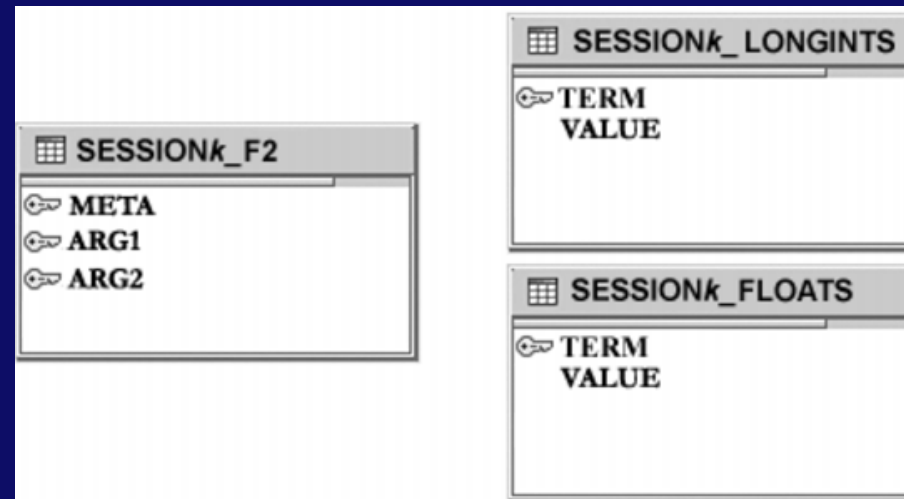
- We have developed two different database schemes:
  - ◆ Single Table Schema
  - ◆ Multiple Table Schema



A diagram of a single table schema. It shows a window titled 'SESSIONk\_F2' containing a list of attributes: META, ARG1, LINT\_ARG1, FLT\_ARG1, ARG2, LINT\_ARG2, and FLT\_ARG2.

SESSIONk_F2
META
ARG1
LINT_ARG1
FLT_ARG1
ARG2
LINT_ARG2
FLT_ARG2

**Single Table Schema**



A diagram of a multiple table schema. It shows three windows: 'SESSIONk\_F2', 'SESSIONk\_LONGINTS', and 'SESSIONk\_FLOATS'. 'SESSIONk\_F2' contains attributes META, ARG1, and ARG2, each with a key icon. 'SESSIONk\_LONGINTS' and 'SESSIONk\_FLOATS' each contain a 'TERM VALUE' attribute with a key icon.

SESSIONk_F2
META
ARG1
ARG2

SESSIONk_LONGINTS
TERM VALUE

SESSIONk_FLOATS
TERM VALUE

**Multiple Table Schema**

- The initial implementation of DBTAB only handles integers, long integers, atoms and floating-point numbers.

# The DBTAB Relational Storage Model

## ➤ Atomic Terms

- ◆ Integer terms within the non-mask part of a term.
- ◆ Atom terms (pointers to the internal symbol addressing space).
- ◆ Their values are directly stored within the corresponding **ARG<sub>i</sub>** record fields.

## ➤ Non-Atomic Terms

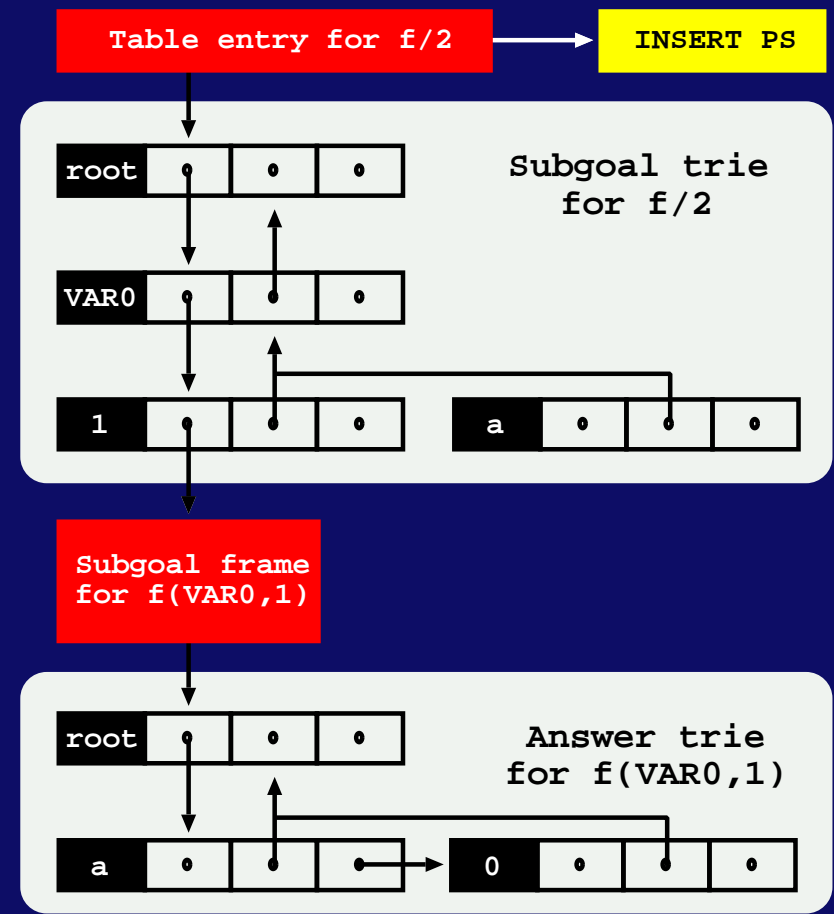
- ◆ Long integer terms (integers larger than the non-mask part of a term).
- ◆ Floating-point terms.
- ◆ For the single table schema they are stored in specifically typed fields placed after each **ARG<sub>i</sub>** record field.
- ◆ For the multiple table schema they are substituted in the **ARG<sub>i</sub>** record fields by unique sequential values that work as a foreign key to the **TERM** field of the auxiliary tables.

# Exporting Answers

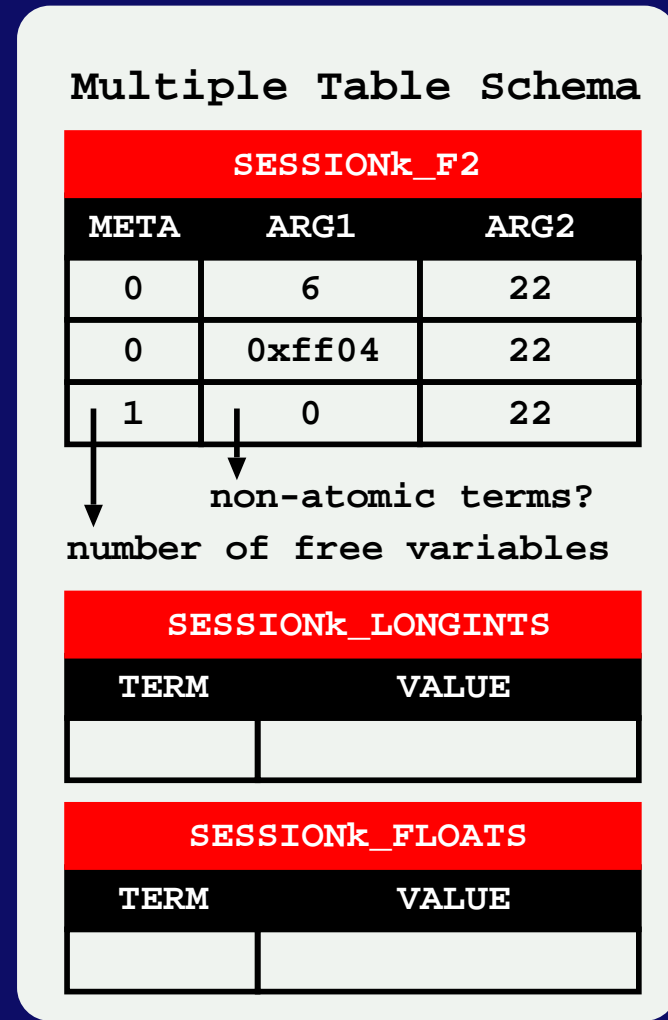
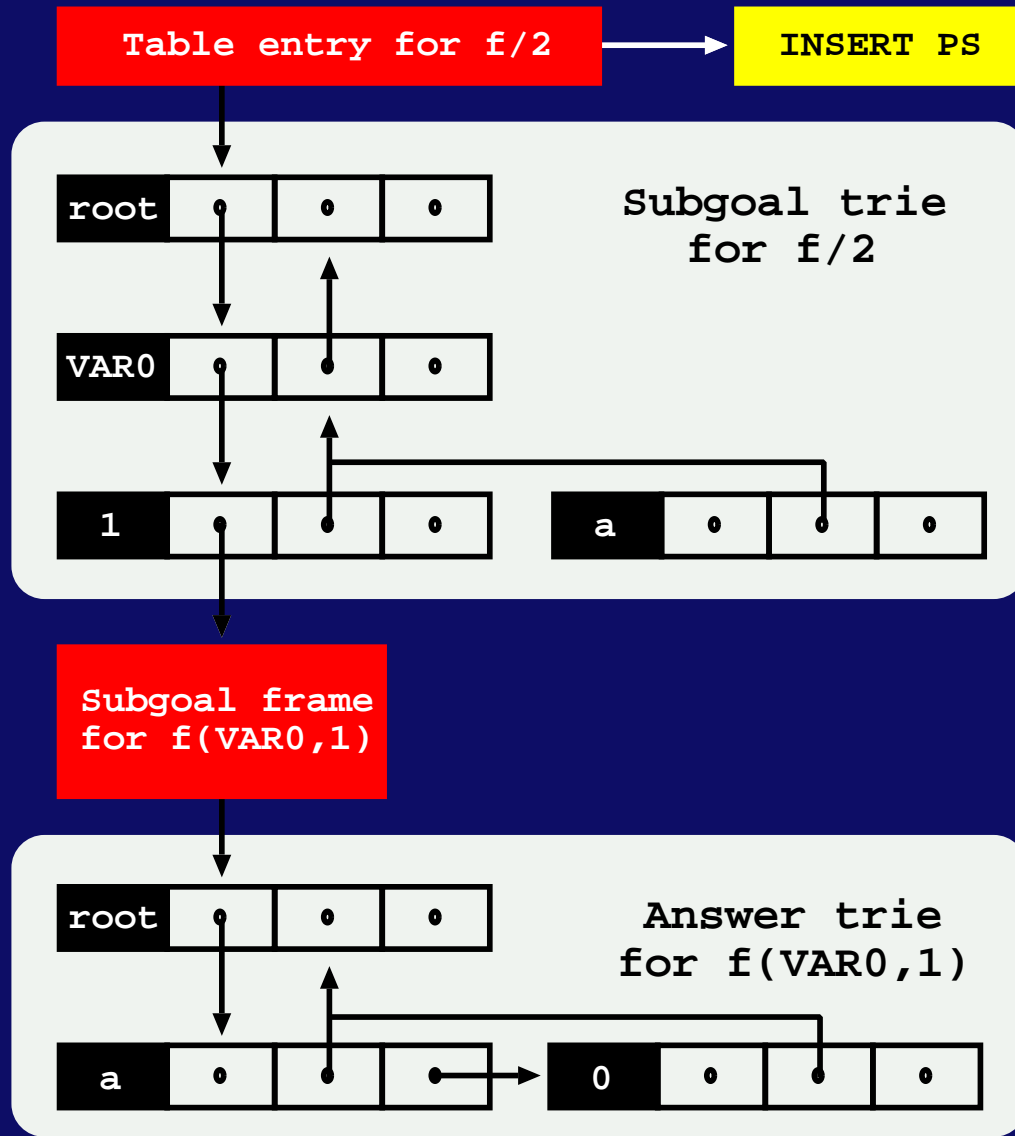
➤ Table entry structure extended with an INSERT prepared statement:  
**INSERT IGNORE INTO SESSION $k$ \_F2(META,ARG1,ARG2) VALUES (?,?);**

## ➤ Basic Idea

1. Bind all terms from subgoal trie branch to prepared statement parameters.
2. Bind all terms from answer trie branch to the remaining prepared statement parameters.
3. Execute prepared statement.
4. Goto 2 until no more answers.
5. Store meta-data for the subgoal trie branch.



# Exporting Answers



# Importing Answers

- Subgoal frame structure extended with two SELECT statements:

```
SELECT ARG1 FROM SESSION $k$ _F2 WHERE META>1 AND ARG2=22;  
SELECT ARG1 FROM SESSION $k$ _F2 WHERE META=0 AND ARG2=22;
```

- **Basic Idea**

1. Execute statement 1 to retrieve meta-data info.
2. Execute statement 2 to retrieve answers.
3. Load answers back to the YapTab engine:
  - ◆ Rebuild the answer trie.
  - ◆ Browse the result-set.



# Importing Answers

- Subgoal frame structure extended with two SELECT statements:

```
SELECT ARG1 FROM SESSIONk_F2 WHERE META>1 AND ARG2=22;  
SELECT ARG1 FROM SESSIONk_F2 WHERE META=0 AND ARG2=22;
```

- **Basic Idea**

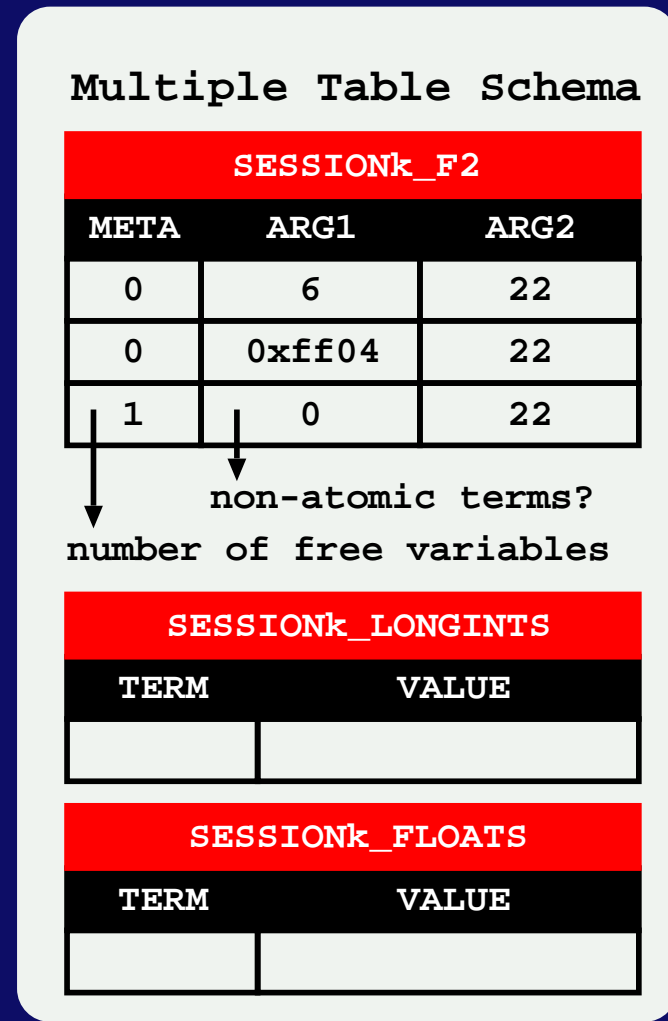
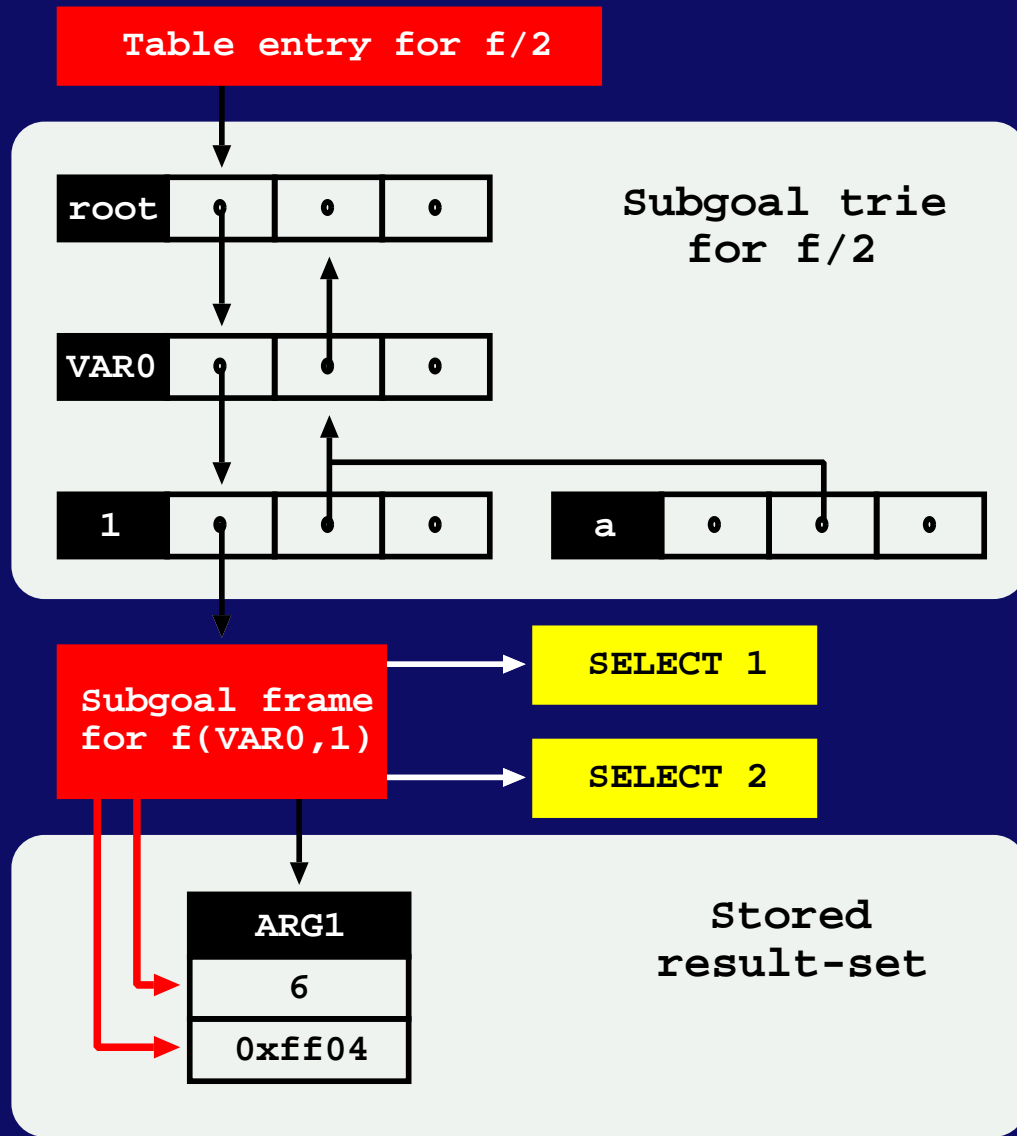
1. Execute statement 1 to retrieve meta-data info.
2. Execute statement 2 to retrieve answers.
3. Load answers back to the YapTab engine:
  - ◆ Rebuild the answer trie.
  - ◆ Browse the result-set.

- Statement 2 with floating-point values on **ARG1**:

```
SELECT DISTINCT ARG1,FLT_ARG1 FROM SESSIONk_F2  
WHERE META=0 AND ARG2 = 22;
```

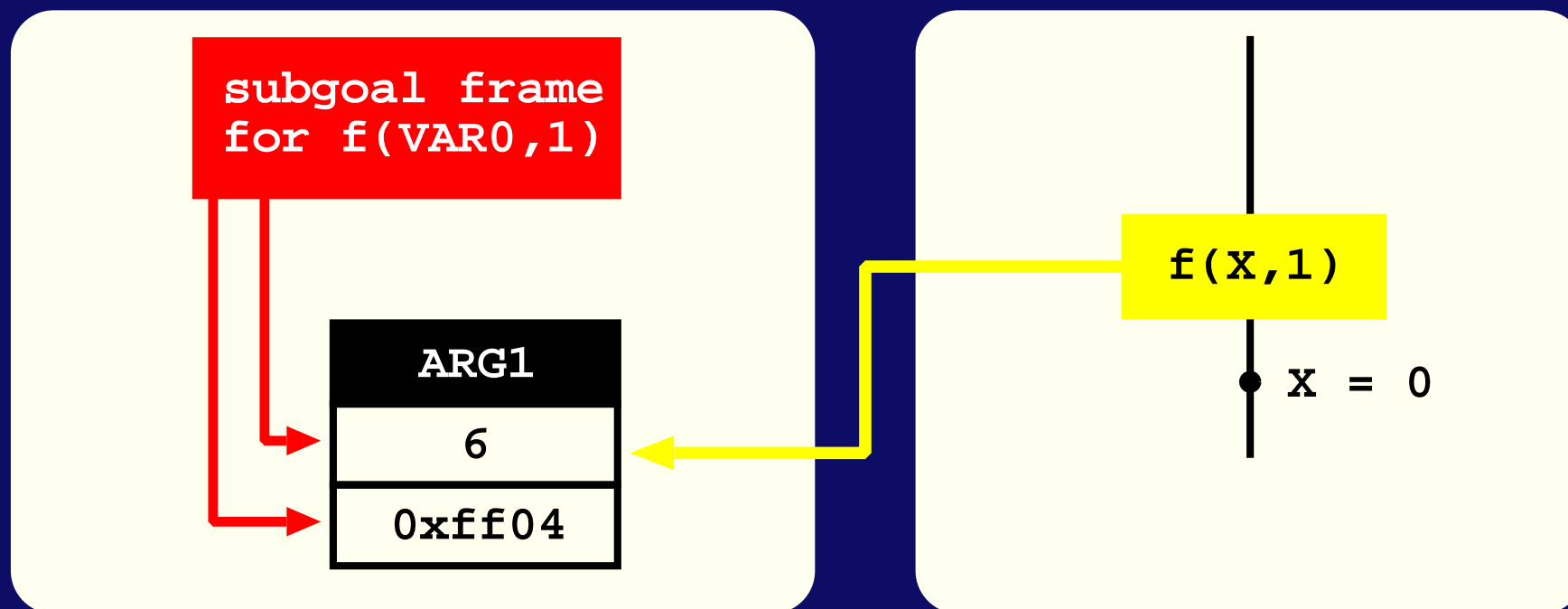
```
SELECT F2.ARG1, FLT.VALUE FROM SESSIONk_F2 AS F2  
LEFT JOIN SESSIONk_FLOATS AS FLT ON (F2.ARG1 = FLT.TERM)  
WHERE F2.META=0 AND F2.ARG2 = 22;
```

# Importing Answers



# Browsing the Result-Set

- Repeated calls now keep a reference to the **offset** of the last consumed answer.



# Experimental Results

Answers	Terms	YapTab		DBTAB				
		Generate	Browse	Multiple Table		Single Table		Browse
				Export	Import	Export	Import	
10,000	integers	65	1	1055	16	1048	34	2
	floats	103	2	10686	44	1112	47	6
50,000	integers	710	6	4911	76	5010	195	12
	floats	1140	8	83243	204	5012	282	27
100,000	integers	1724	11	9576	153	9865	392	20
	floats	1792	14	215870	418	11004	767	55

Running times in milliseconds

# Discussion

- Most of the execution time is spent in data transactions (export and import).
- Due to the extra search and insertion on auxiliary tables in the multiple table approach, the single table approach is clearly preferable for storage purposes.
- On the other hand, the import time for the single table approach is, on average, the double of the time spent by the multiple table approach.
- Browsing in stored data-sets is about to 2 times slower than browsing in tries. May become interesting when re-computation for a tabled predicate raises over this factor.

# Further and Ongoing Work

- Evaluate the impact of DBTAB on a more representative set of programs.
- Fit within the context of YapTab's memory management algorithm.
- Expand term representation to lists and functors.
- **Export answers as trie nodes (slower than the single table schema when exporting but faster than the single and multiple table schema when importing).**
- **Clustering answers when exporting (with a 25 tuple cluster we obtained an average speedup of 5 for the single table schema and an average speedup of 7 for the multiple table schema).**