Efficient Handling of Large Sets of Tuples
With Sharing Trees
Denis Zampunikris & Baudouin Le Charlier
Facultés Universitaires Notre-Dame de la Paix
{dza,ble}@info.fundp.ac.be

Computing with sets of tuples (n-ary relations) is often required in programming, while being a major cause of performance degradation as the size of sets increases.

We present a new data structure dedicated to the manipulation of large sets of tuples, dubbed sharing tree. The main idea to reduce memory consumption is to share some sub-tuples of the set represented by a sharing tree: Prefixes are represented only once, and suffixes are additionally shared whenever possible.

A sharing tree is a rooted acyclic graph \((N, V, \text{val}, \text{succ})\) such that \(N = N_0 + \ldots + N_k, k \geq 0\), is a finite set called the set of nodes (the nodes are organized in layers, \(N_i\) is the set of nodes of layer \(i\)); \(\text{val} : N \rightarrow V + \{\top, \bot\}\) is the valuation function; \(\text{succ} : N \rightarrow 2^N\) gives the successors of a node; and the following rules are confirmed: \(\forall 0 \leq i < k, \forall n \in N_i, \text{succ}(n) \subseteq N_{i+1}\); each node has all its successors on the next layer; \(\forall 0 < i < k, \forall n_1,n_2 \in N_i, n_1 \neq n_2, \text{val}(n_1) = \text{val}(n_2) \Rightarrow \text{succ}(n_1) \neq \text{succ}(n_2)\): two equal nodes in the same layer do not have the same sons; \(\forall n \in N, \forall s_1,s_2 \in \text{succ}(n), s_1 \neq s_2 \Rightarrow \text{val}(s_1) \neq \text{val}(s_2)\): a node does not have equal sons; \#\(N_0 = 1\) and \(\forall n \in N, \text{val}(n) = \top \Leftrightarrow n \in N_0\): the first layer \(N_0\) contains only one element (called the root), the only one with value \(\top\); \(\text{val}(n) = \bot \Rightarrow \text{succ}(n) = \emptyset\) and \(\text{succ}(n) = \emptyset \Rightarrow (\text{val}(n) = \bot \lor \text{val}(n) = \top)\).

The elements of \(ST = (N, V, \text{val}, \text{succ})\) are the tuples of values on all the different paths starting from the root node \(r \in N_0\): \(\text{Elem}(ST) = \text{Set}(r)\) and \(\forall n \in N, \text{Set}(n) = \{()\}\) if \(\text{val}(n) = \bot\) or \(\text{Set}(n) = \bigcup_{s \in \text{succ}(n)} \text{Set}(s)\) if \(\text{val}(n) = \top\) or \(\{\text{val}(n)\} \times \bigcup_{s \in \text{succ}(n)} \text{Set}(s)\) otherwise.

The organization in layers and the first condition enable some improvements in the algorithms by allowing direct access to any projection of the set, while the overhead required by the structure of layers is small, for sharing trees representing large sets of tuples have usually much more nodes than layers. The second condition ensures some suffix merging of tuples that share equal ends, while the third condition ensures the prefix merging of tuples that share equal beginnings. Note that tuples of different lengths can be represented in the same sharing tree.

We developed algorithms for common set operations: membership, insertion, equality, union, intersection, ... that have theoretical complexities proportional to the sizes of the sharing trees given as arguments, which are usually much smaller than the sizes of the represented sets.

As future work we plan to investigate the use of sharing trees as the main data structure to record tables of relational databases.