A Tabling Engine for the Yap Prolog System

Ricardo Rocha   Fernando Silva   Vítor Santos Costa

DCC-FC & LIACC, University of Porto, Portugal
{*ricroc,fds*}*@ncc.up.pt*

COPPE Systems Engineering, Federal University of Rio de Janeiro, Brazil
vitor@cos.ufrj.br

Overview

Tabling and Parallelism
Motivation, approach and integration issues.

Tabling Concepts
Execution model, tabled nodes and completion.

Extending Yap to Support Tabling
Data structures, leader nodes, completion and answer resolution.

Initial Performance Evaluation
Running times on a set of tabled and non tabled benchmarks.

Conclusions

Tabling and Parallelism: Motivation

Tabling (SLG resolution) has the advantage over Prolog (SLD resolution) in that:

- Avoids redundant subcomputations
- Deals with infinite loops

In tabling we still exploit alternatives for solving goals:

- We need to search like in Prolog;
- We can parallelise the search like in or-parallel Prolog:
  - Search both tabled and non-tabled goals in parallel;
  - Reuse or-parallel technology.

Develop an efficient or-parallel tabling system

Tabling and Parallelism: How to?

Key Ideas:

- Extract parallelism from both tabled and non-tabled subgoals;
- Separate tabling and parallelism as much as possible.

Starting Points:

- **Or-Parallel Models**: Environment Copying (Muse) / Binding Arrays (Aurora)
- **Tabling Models**: SLG-WAM / Chat (XSB)

OPTYap = YapOr + YapTab + Tabling/Parallelism Integration

Tabling Concepts I

Basic Execution Model

- Whenever a tabled subgoal is called for the first time, a new entry is allocated in the *table space*. This entry will collect all the answers generated for the subgoal.
- Variant calls to tabled subgoals are resolved by consuming the answers already stored in the table.
- Meanwhile, as new answers are founded, they are inserted into the table and returned to all variant subgoals.

Nodes Classification

- **Generator**: nodes that first call a tabled subgoal.
- **Consumer**: nodes that consume answers from the table space.
- **Interior**: nodes that are evaluated by the standard resolution.

Tabling Concepts II

Completion Operation

- A tabled subgoal is said to be *completely evaluated* when all possible resolutions have been made:
  - no more answers can be generated;
  - the variant subgoals have consumed all the available answers.
- A number of subgoals may be mutually dependent, forming a *strongly connected component (SCC)*. In such case, the SCC is said to be completely evaluated when each subgoal belonging to the SCC is completely evaluated.
- The completion operation is performed by the *leader node*, i.e., the generator node corresponding to:
  - the oldest subgoal in a SCC;
  - the subgoal, if not in a SCC.

Running Example I

Running Example II

Running Example III

Designing YapTab to Support Parallelism

Main Problems:

- Tabling suspensions management;
- Completion detection.

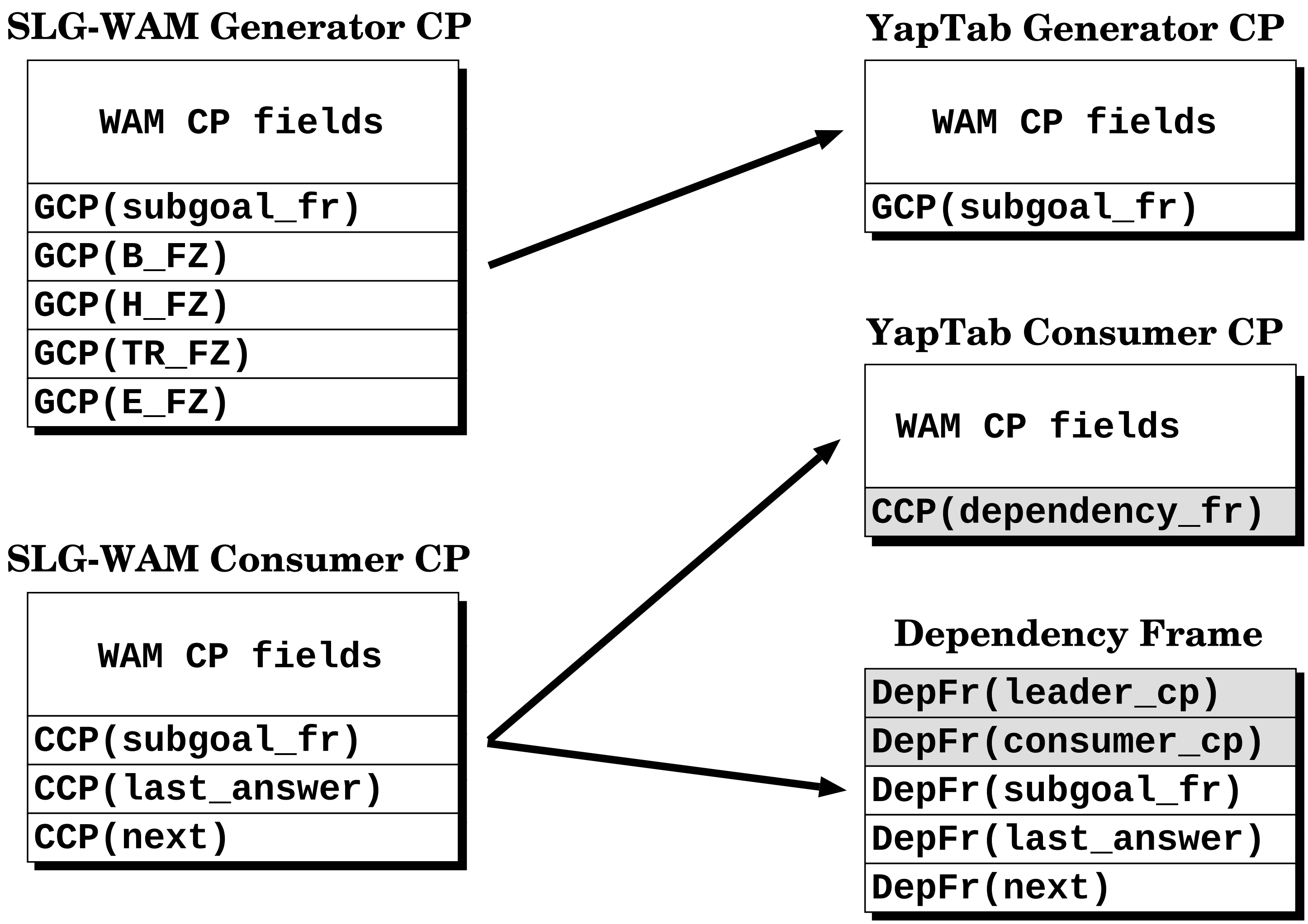
Potential sources of overhead:

- Data related with tabling suspensions;
- Completion stack.

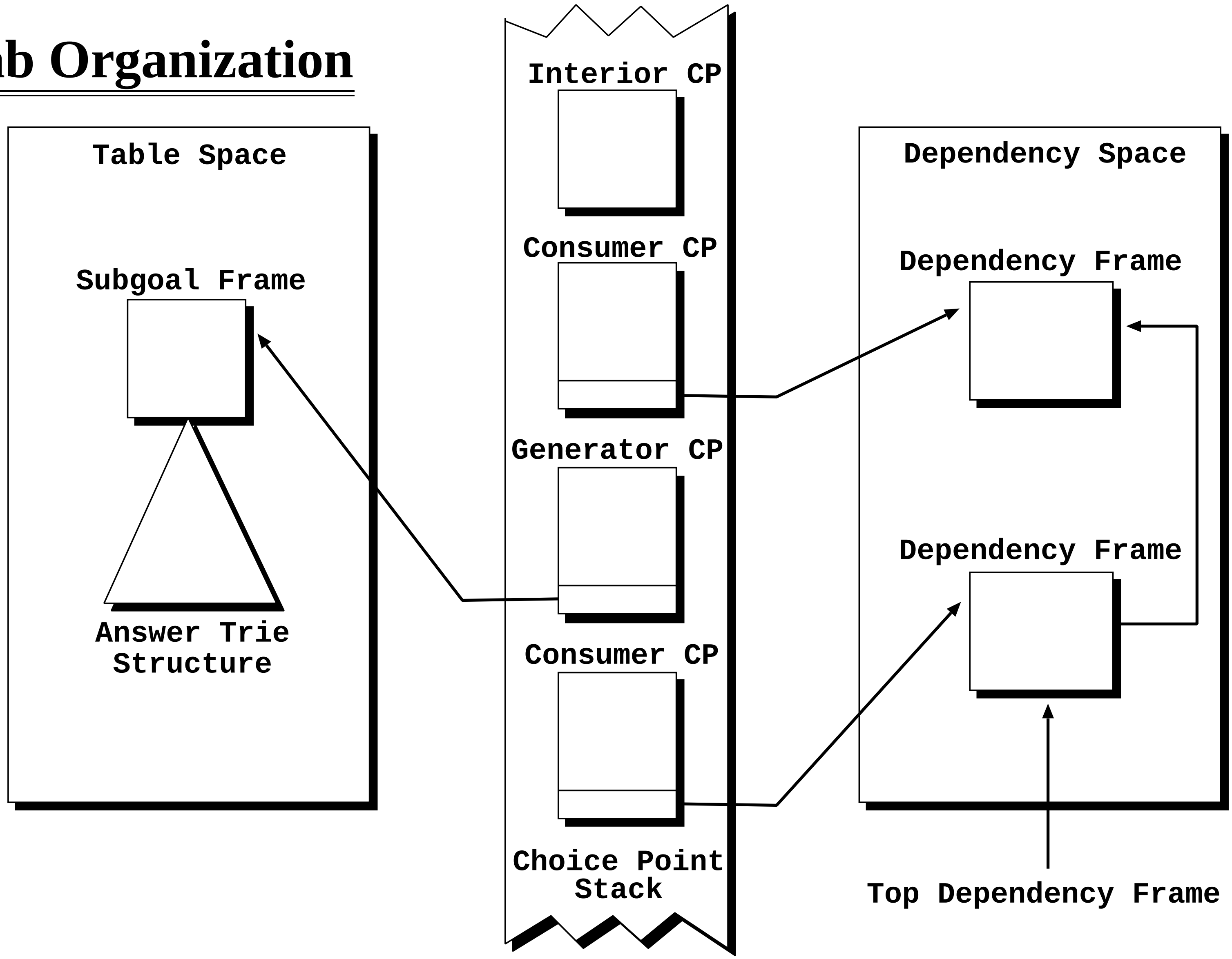
The dependency frame data structure:

- Keeps track of all data related with a particular tabling suspension;
- Reduces the number of extra fields in tabled choice points;
- Eliminates the need for a completion stack area;
- Very useful for parallelism.

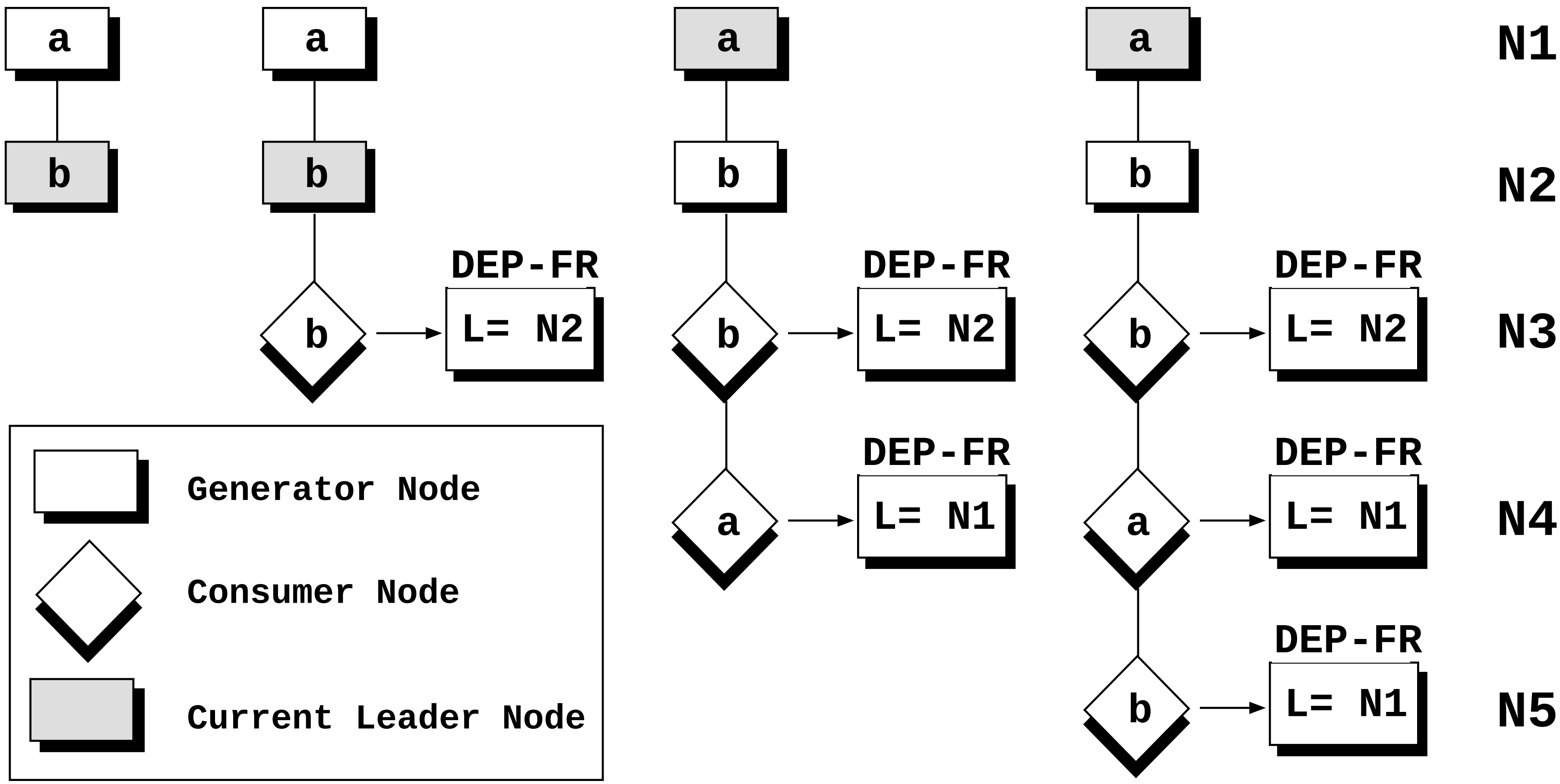
From SLG-WAM To YapTab



YapTab Organization



Computing the Leader Node Information



Completion and Answer Resolution Instructions

Completion Instruction in GN $\rightarrow$ GN is a leader node ?

- **No** $\rightarrow$ Backtrack
- **Yes** $\rightarrow$ Younger consumer node CN with unconsumed answers ?
  - **Yes** $\rightarrow$ Resume computation to CN
  - **No** $\rightarrow$ Perform completion operation

Answer Resolution Instruction in CN $\rightarrow$ Unconsumed answers ?

- **Yes** $\rightarrow$ Load the next available answer and proceed execution
- **No** $\rightarrow$ Resume computation to the younger node of
  - Previous consumer node with unconsumed answers
  - Last leader node when the completion instruction was executed

Initial Performance Evaluation

Benchmark	YapTab	Yap Prolog	XSB Prolog
9-queens	740	740(1.00)	1819(2.46)
cubes	210	210(1.00)	589(2.80)
ham	460	430(0.93)	1139(2.48)
nsort	390	370(0.95)	1101(2.82)
puzzle	2430	2120(0.87)	5819(2.39)
Average		(0.95)	(2.59)

Running Times (in milliseconds) on a Set of Non Tabled Benchmarks.

Benchmark	YapTab	XSB Prolog
binary tree (depth 10)	180	451(2.50)
chain (64 nodes)	130	399(3.06)
cycle (64 nodes)	390	1121(2.87)
grid (4x4 nodes)	1330	5740(4.31)
Average		(3.18)

Running Times (in milliseconds) on a Four Version Tabled Benchmark.

Results obtained on a 200 MHz PentiumPro, 128 MB RAM, 256 KB cache, Linux-2.2.5 kernel.

Conclusions

- We presented the design and implementation of YapTab, an extension of the Yap Prolog system that implements sequential tabling.
- YapTab reuses the principles of the SLG-WAM engine, but innovates in introducing the dependency space and in proposing a new completion detection algorithm.
- YapTab first results are very encouraging. Overheads over standard Yap are low and performance in tabling benchmarks is quite satisfactory.
- YapTab includes all the machinery required to extend the system to execute tabled programs in or-parallel.
- We have obtained very initial timings for parallel execution on a shared memory PentiumPro machine. The results show significant speedups for a tabled application increasing up to the four processors.