YapTab: A Tabling Engine Designed to Support Parallelism

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.

Extending Yap to Support Tabling
Tabled nodes, 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

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;
- Advantages:
  - Search both tabled and non-tabled goals in parallel;
  - Reuse or-parallel technology.
- Problems:
  - Tabled suspensions and completion detection must work with parallelism.

Develop an efficient or-parallel tabling system

Tabling and Parallelism: How to?

Our Starting Points:

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

Our Approach:

OPTYap = YapOr + YapTab + Tabling/Parallelism Integration

Tabling and Parallelism: The OPTYap Model

Separate tabling and parallelism as much as possible:

- Run separate YapTab engines in parallel;
- Each YapTab engine operates the stacks, i.e.:
  - allocates all types of nodes;
  - fully implements suspension of tabled subgoals;
  - resumes subcomputations to consume newly found answers;
  - completes private subgoals.
- YapOr is triggered to:
  - assign work to the YapTab engines (stack-copying);
  - synchronize access to shared branches;
  - complete shared subgoals.

Parallelism stems from both tabled and non-tabled subgoals.

Tabling and Parallelism: Integration

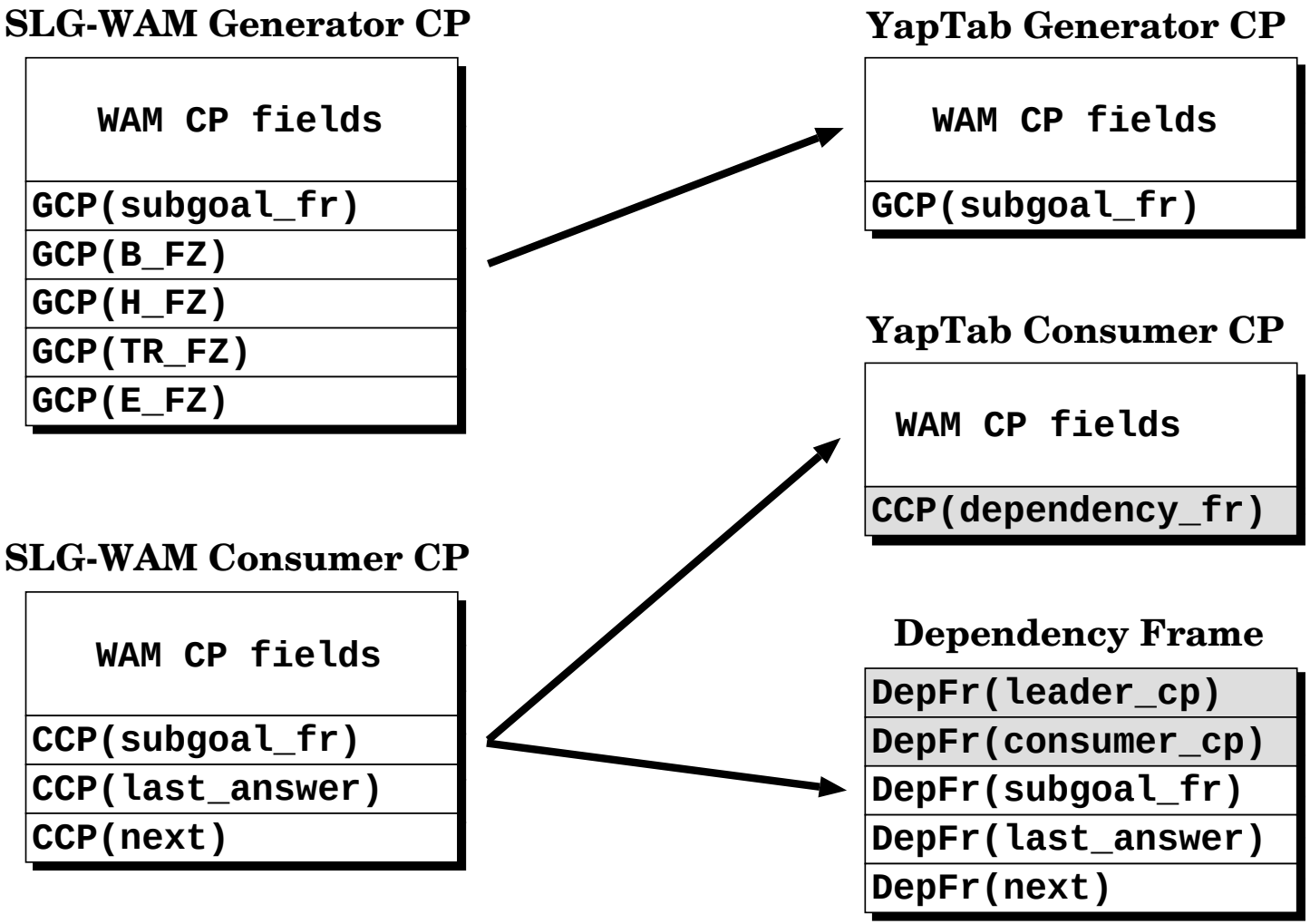
Potential sources of overhead:

- Data related with tabling suspensions;
- Completion stack.

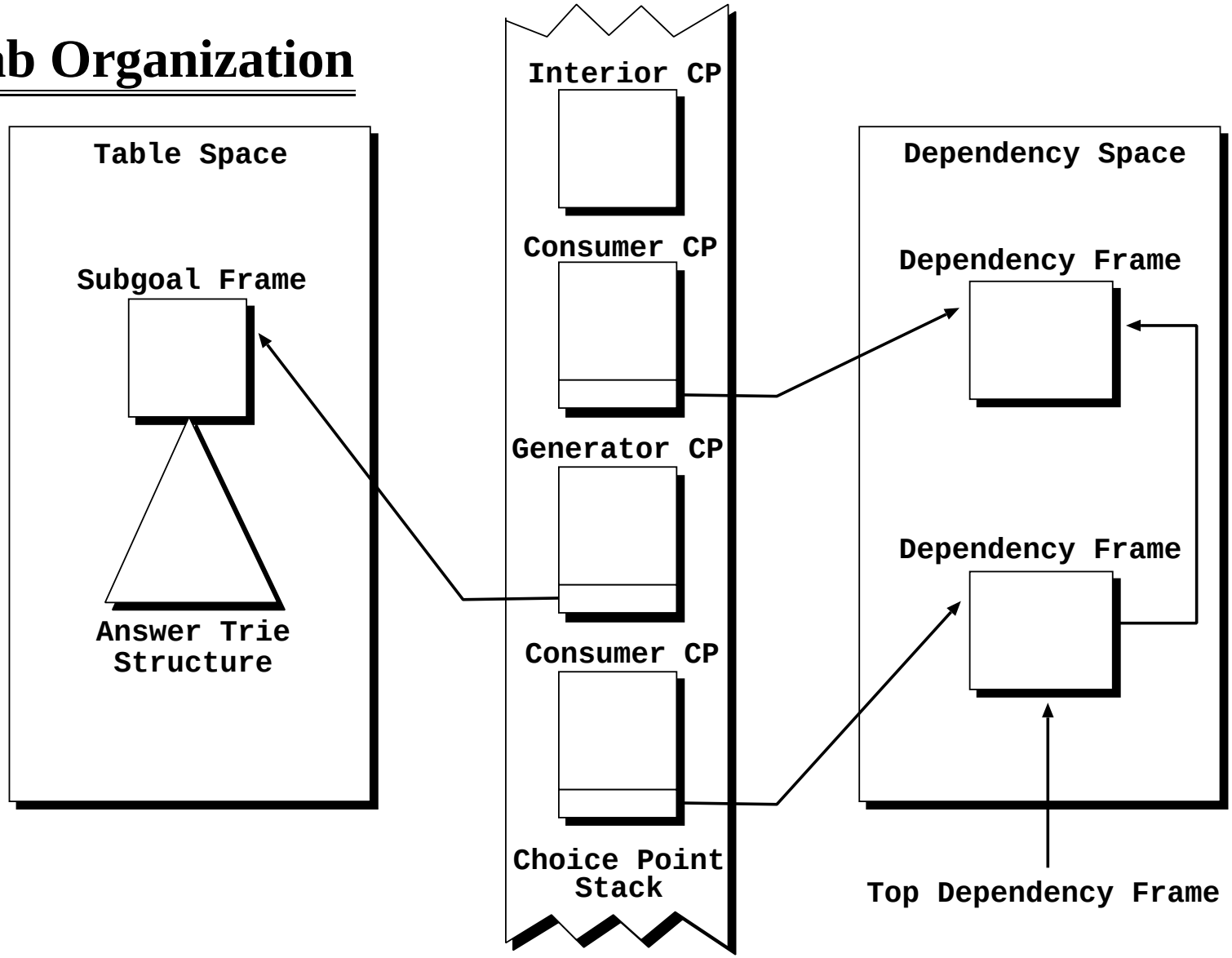
The dependency frame data structure:

- 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.

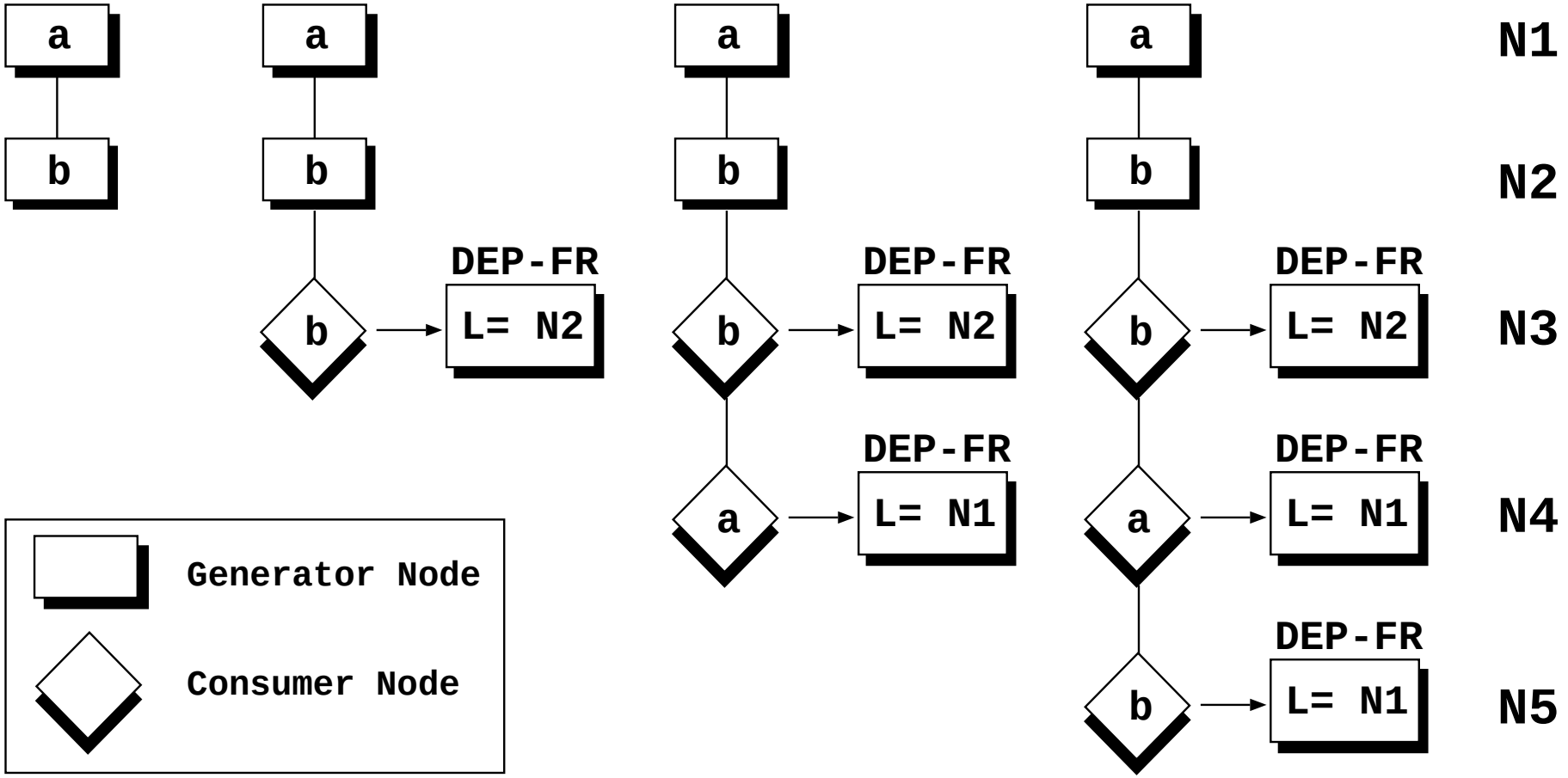
From SLG-WAM To YapTab



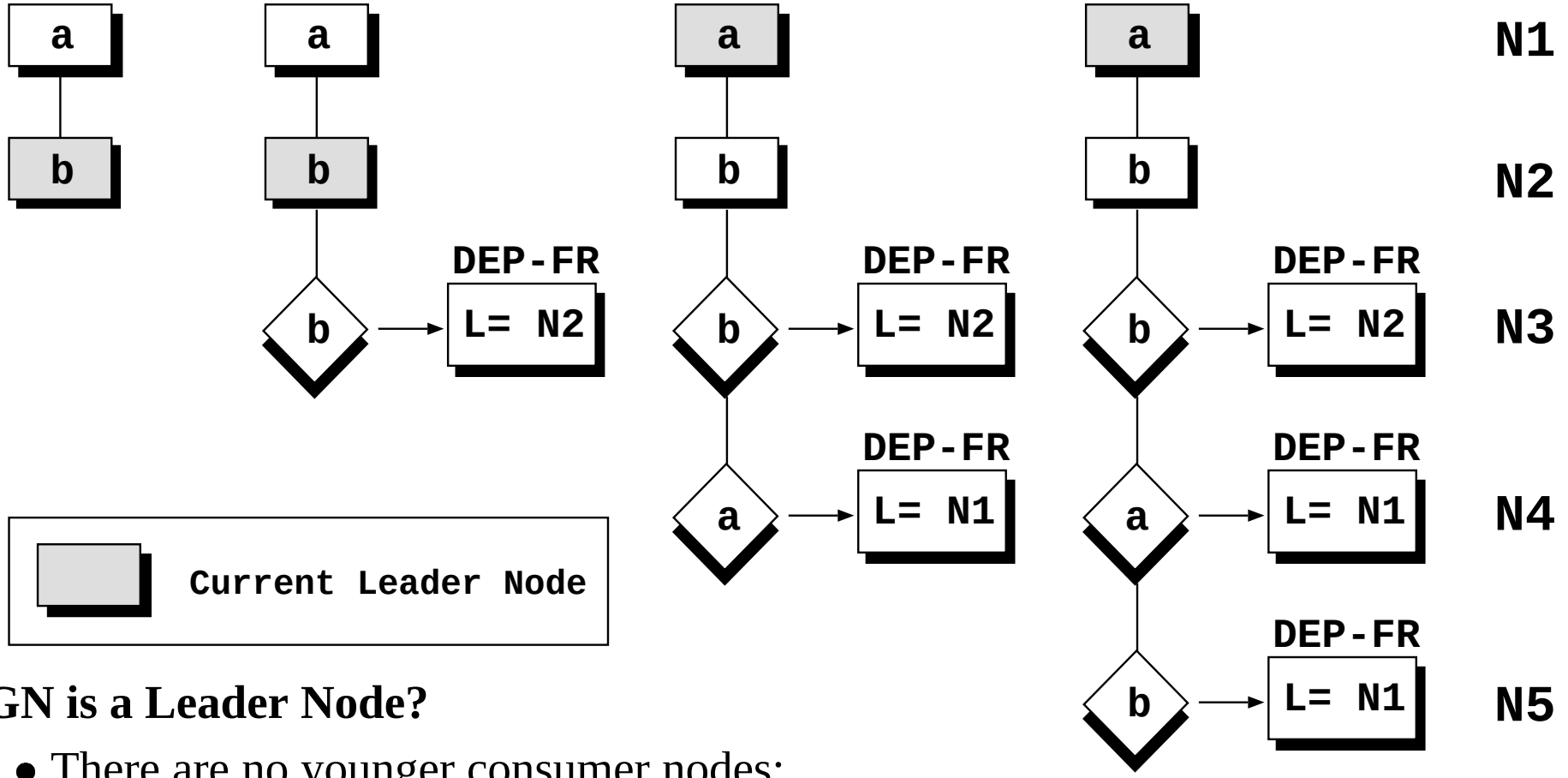
YapTab Organization



Computing the Leader Node Information



Leader Nodes



GN is a Leader Node?

- There are no younger consumer nodes;
- GN is the leader node referred in the top dependency frame.

Completion and Answer Resolution Instructions

Completion Instruction in GN → GN is a leader node ?

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

Answer Resolution Instruction in CN → Unconsumed answers ?

- **Yes** → Load the next available answer and proceed execution
- **No** → First time that backtracking from CN takes place ?
  - **Yes** → Backtrack
  - **No** → 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

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

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)

- Results obtained on a 200 MHz PentiumPro, 128 MB RAM, 256 KB cache, Linux-2.2.5 kernel.
- YapTab is based on the Yap4.2.1 engine.
- Same compilation flags for Yap and for YapTab.
- XSB version 2.2 with the default confi guration and the default execution parameters (chat engine and batched scheduling).

Initial Performance Evaluation

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

Benchmark	YapTab	XSB Prolog
binary tree (depth 10)	440	451(1.03)
chain (64 nodes)	120	399(3.33)
cycle (64 nodes)	380	1121(2.95)
grid (4x4 nodes)	1270	5740(4.52)
Average		(2.96)

Tries without hashing

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)

Tries using hashing optimization

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.