Or-Parallelism within Tabling

Ricardo Rocha Fernando Silva Vítor Santos Costa
{*ricroc,fds,vsc*}*@ncc.up.pt*

Department of Computer Sciences & LIACC
University of Porto
Portugal

Summary

Parallel Execution of Tabled Programs
The fundamental issues in supporting parallelism for tabling systems.

Alternative approaches
Two computational models to combine or-parallelism and tabling.

- **Or-Parallelism within Tabling (OPT)**
- **Tabling unified with Or-Parallelism (TOP)**

Implementing the OPT Approach
Data areas, data structures, leader nodes and public completion.

Conclusions

Parallel Execution of Tabled Programs

An important advantage of LP is that **parallelism can be exploited implicitly**:

- Or-Parallelism
- And-Parallelism

An interesting observation is that **tabling is still about exploiting alternatives** for solving goals:

- It should be amenable for parallel execution within traditional parallel models;
- No need to restrict parallelism to tabled or non-tabled subgoals.

Our Goal: exploit **maximum parallelism** and take **maximum advantage** of current technology for parallel and tabling systems.

Problems: synchronization within tabling operations and scheduling strategies.

Or-Parallelism within Tabling (OPT)

OPT = Sequential Tabling Engine + Parallel Component

Tabling is the base component of the system: workers spend most of their time executing as if they were sequential tabling engines.

Parallel exploitation: all unexploited alternatives should be amenable for parallel execution, be they from generator, consumer or interior nodes.

Parallel tabling synchronization: accomplished by a new data structure to form a dependency graph between consumer nodes to efficiently check for resumption and completion points.

Tabling unified with Or-Parallelism (TOP)

TOP = Standard Prolog + Tabling/Parallel Component

Workers are considered WAM engines: they only manage a logical branch, not a whole part of the tree.

The notion of suspension is unified: the system handles suspensions from parallelism and from tabling in the same framework. A branch can be suspended because:

- It is speculative;
- It is not leftmost;
- It contains consumer nodes waiting for solutions.

Suspended branches are public branches: when a worker suspends a consumer node, the corresponding branch becomes shared work that anyone can take.

TOP Advantages

Workers have a clearly defined state
A worker always occupies the tip of a single branch in the search tree.

Less memory should be spent
A suspended branch will only appear once, instead of possibly several times for several workers.

TOP Disadvantages

A suspended branch is a public branch
Large amount of tabling suspensions may increase overheads.

A different tabling engine is required
To efficiently support the unified suspension and to reduce the overlap between parallelism and tabling.

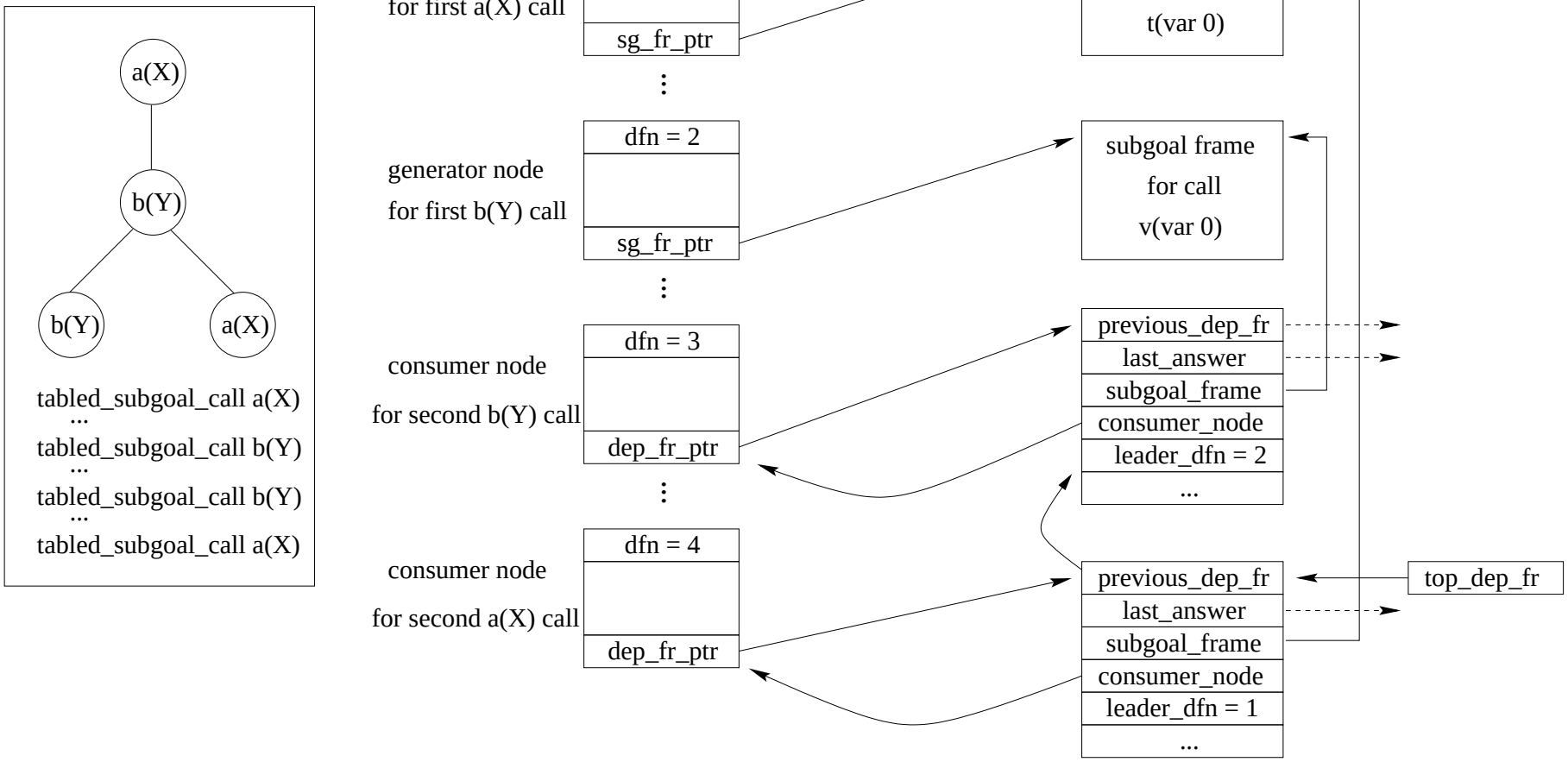
Overview of OPTYAP

OPTYAP extends the **YapOr** environment copy or-parallel system to support tabling in a manner close to the **SLG-WAM** engine.

A set of workers will execute a tabled program by traversing its search tree, whose nodes are entry points for parallelism:

- Each worker physically owns a copy of the environment and shares a large area related to tabling and scheduling;
- The incremental copy technique is used when the workers with unexploited private alternatives or unconsumed answers share work;
- Whenever a worker backtracks to a public node it synchronizes to guarantee the correctness of the sequential tabling execution;
- When there are no alternatives or no unconsumed answers left in a shared node, the public completion operation may be executed.

Running Example

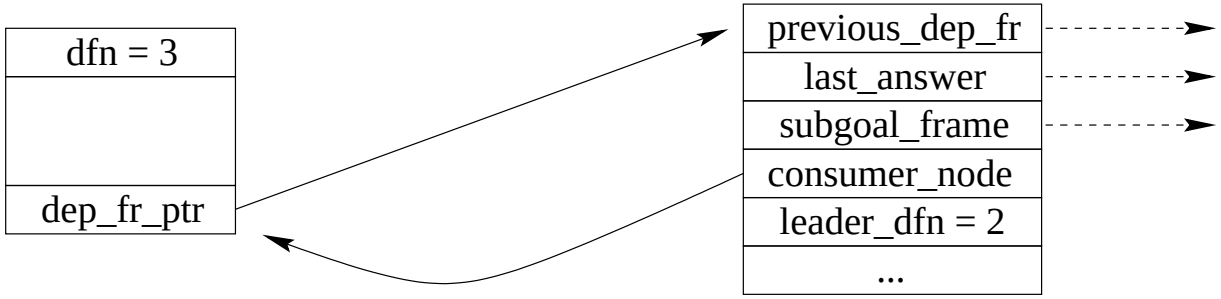


Dependency Frames

Key data structure to control suspension, resumption and completion of subgoals.

Designed to:

- Save information about suspension points;
- Connect consumer nodes with the table space;
- Search for and pick up new answers;
- Form a dependency graph between consumer nodes;
- Efficiently check for leader nodes and perform completion.

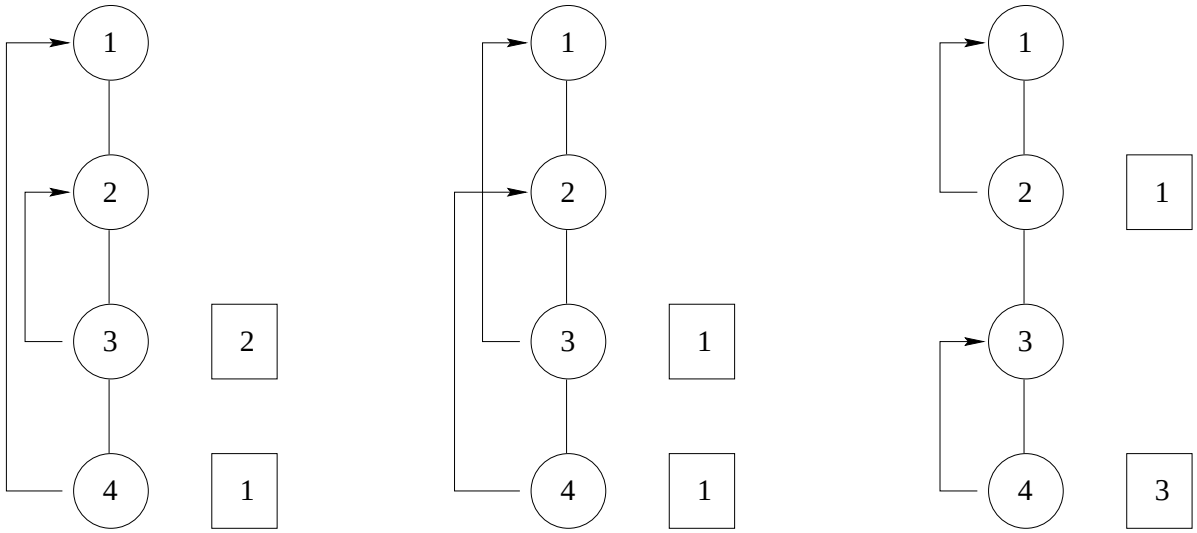


Leader Nodes

Definition

A leader node is a node where a worker can perform completion.

Leader Detection



Remark

In parallel tabling, all kinds of nodes can be leaders in a worker's branch.

Public Completion

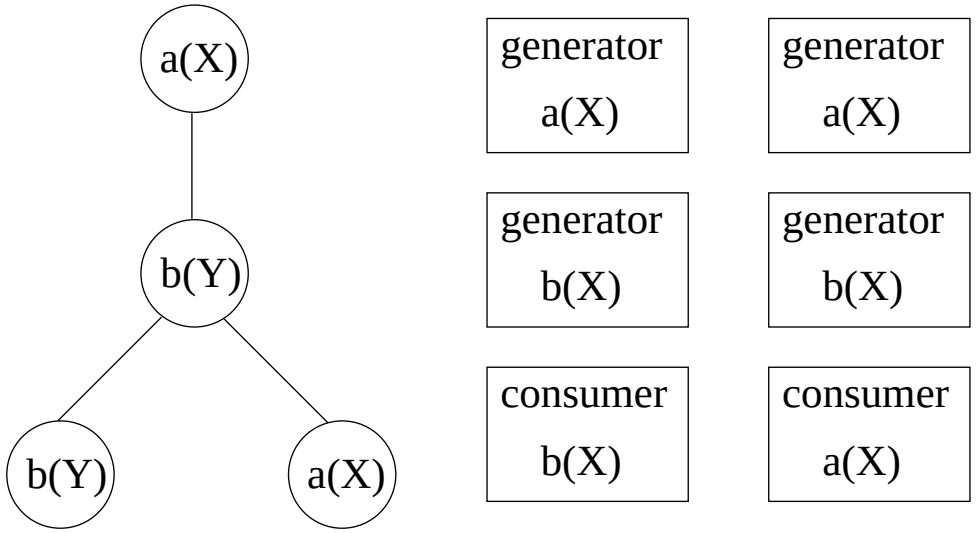
Fundamental Idea
Avoid explicit communication between workers and reduce suspension points.

Basic Consideration
When a leader node is public and contains consumer nodes below it, this means that it depends on branches explored by other workers.

Consequence
In certain conditions, it becomes necessary to suspend the leader branch.

Completion Conditions

- Be the **last worker** in the node;
- No **hidden workers** in the node;
- No **suspended branches** to resume.



Conclusions

Presentation
We suggested two major approaches to combine or-parallelism and tabling.

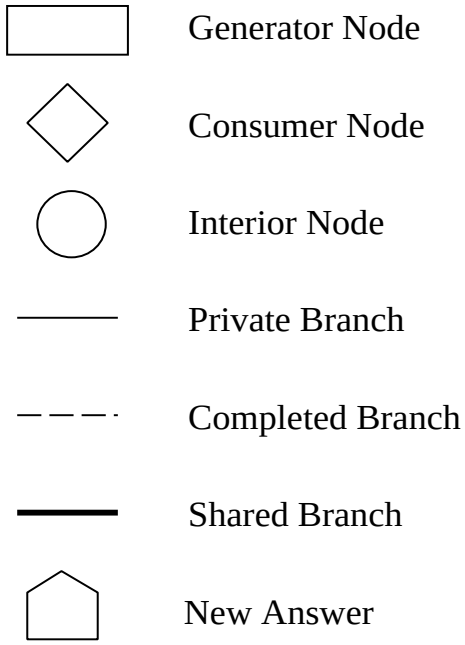
We presented the fundamental concepts on the design and implementation of the OPT approach.

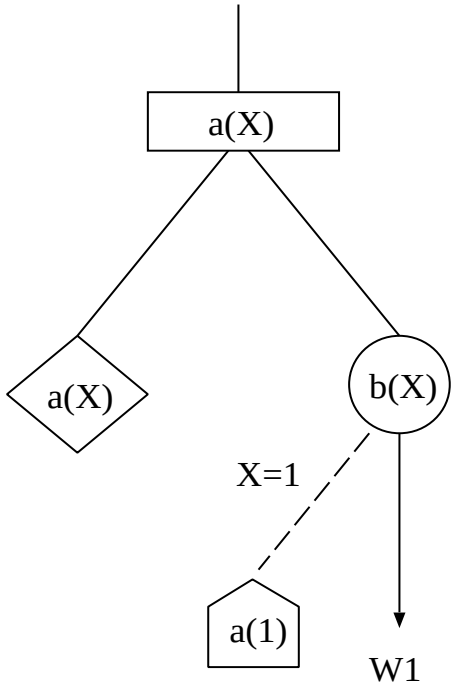
Current and Further Work
Currently, we have sequential tabling and or-parallelism functioning separately within the same system.

We are now working on adjusting the system, mainly the basic or-scheduler, to support parallel tabling execution.

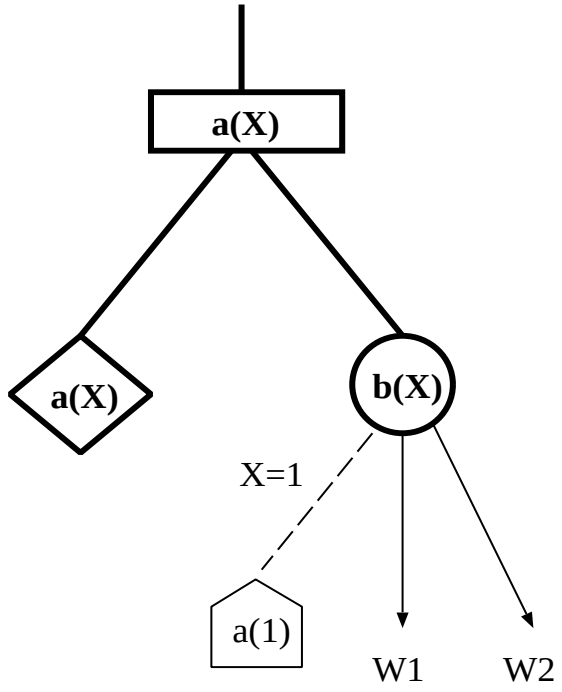
Practical Significance
"..although I believe that the **potential** practical significance of the work is substantial, the **current** practical significance is quite limited."

OPT Example



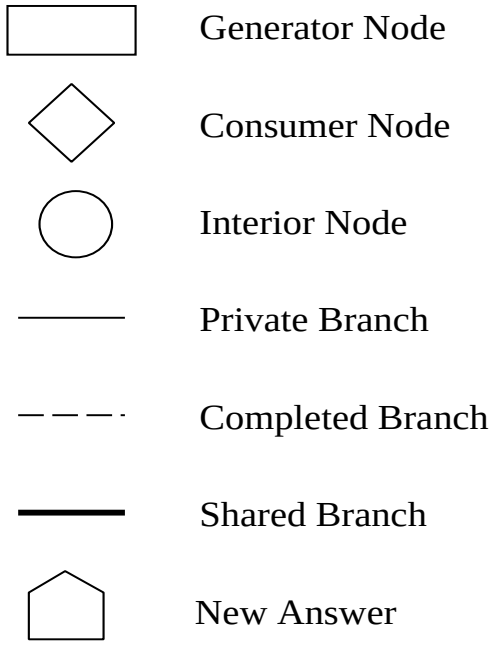


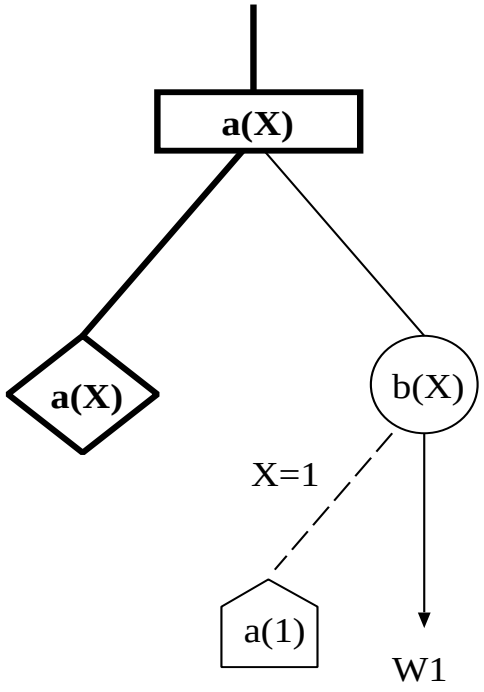
One Worker (W1)



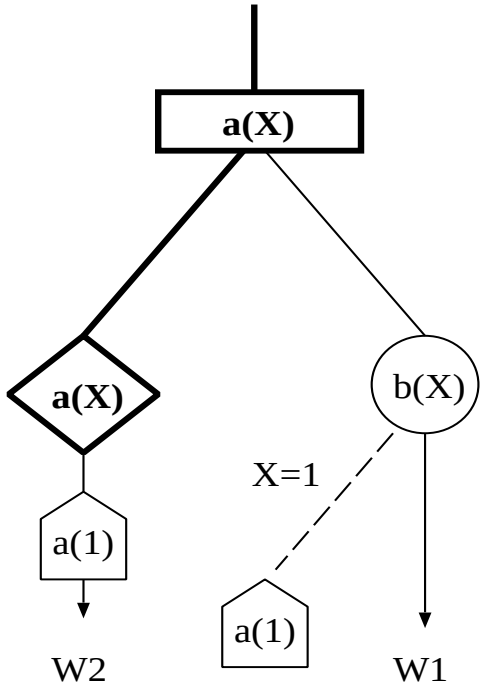
Two workers (W1 and W2)

TOP Example





One Worker (W1)



Two workers (W1 and W2)

Public Completion Pseudo-Code

```
public_completion (node N)

   if (last worker in node N)
      for all suspension branches SB stored in node N
         if (exists unconsumed answers for any consumer node in SB)
            collect (SB) /* to be resumed later */
   if (N is a leader node)
      if (exists unconsumed answers for any consumer node below node N)
         backtrack_through_new_answers() /* as in sequential tabling */
      if (suspension branches collected)
         suspend_current_branch()
         resume (a suspension branch)
      else if (not last worker in node N)
         suspend_current_branch()
      else if (hidden workers in node N)
         suspend_current_branch()
      else
         complete_all()
   else   /* not leader */
      if (consumer nodes below node N)
         increment hidden workers in node N
   backtrack
```