

Implementation of Suspension-Based Tabling in Prolog using External Primitives

Ricardo Rocha, Cláudio Silva and Ricardo Lopes
DCC-FC & LIACC

University of Porto, Portugal

ricroc@dcc.fc.up.pt

ccaldas@dcc.online.pt

rslopes@dcc.fc.up.pt

Tabling in Logic Programming

- **Tabling** is an implementation technique where answers for subcomputations are stored and then reused when a repeated computation appears.

Tabling in Logic Programming

- **Tabling** is an implementation technique where answers for subcomputations are stored and then reused when a repeated computation appears.
- Tabling has proven to be particularly effective in logic (**Prolog**) programs:
 - ◆ **Avoids recomputation**, thus reducing the search space.
 - ◆ **Avoids infinite loops**, thus ensuring termination for a wider class of programs.

Tabling in Logic Programming

- **Tabling** is an implementation technique where answers for subcomputations are stored and then reused when a repeated computation appears.
- Tabling has proven to be particularly effective in logic (**Prolog**) 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

Tabling in Logic Programming

➤ Example 1

```
% reachability relation in a graph
path(X,Z) :- path(X,Y), edge(Y,Z).
path(X,Z) :- edge(X,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

Tabling in Logic Programming

➤ Example 1

```
% reachability relation in a graph
path(X,Z) :- path(X,Y), edge(Y,Z).
path(X,Z) :- edge(X,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

```
% p(1,Z) calls itself --> infinite loop
```

Tabling in Logic Programming

➤ Example II

```
% reachability relation in a graph
path(X,Z) :- edge(X,Y), path(Y,Z).
path(X,Z) :- edge(X,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

Tabling in Logic Programming

➤ Example II

```
% reachability relation in a graph
path(X,Z) :- edge(X,Y), path(Y,Z).
path(X,Z) :- edge(X,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

```
% p(1,Z) calls p(2,Z) that calls p(1,Z) --> infinite loop
```


Tabling in Logic Programming

➤ Example III

```
% reachability relation in a graph
path(X,Z) :- edge(X,Z).
path(X,Z) :- edge(X,Y), path(Y,Z).

% graph definition
edge(1,2).
edge(2,1).

% query goal
?- p(1,Z).
```

Tabling in Logic Programming

➤ Example III

```
% reachability relation in a graph
path(X,Z) :- edge(X,Z).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

Z = 2

Tabling in Logic Programming

➤ Example III

```
% reachability relation in a graph
path(X,Z) :- edge(X,Z).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

Z = 2

Z = 1

Tabling in Logic Programming

► Example III

```
% reachability relation in a graph
path(X,Z) :- edge(X,Z).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

Z = 2

Z = 1

Z = 2

Tabling in Logic Programming

► Example III

```
% reachability relation in a graph
path(X,Z) :- edge(X,Z).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

```
% graph definition
edge(1,2).
edge(2,1).
```

```
% query goal
?- p(1,Z).
```

Z = 2

Z = 1

Z = 2

Z = 1 --> infinite loop of answers

Tabling in Logic Programming

➤ Basic Execution Model

- ◆ Tabled subgoals are evaluated by storing their answers in an appropriate data space called the **table space**.
- ◆ Variant calls to tabled subgoals are not re-evaluated against the program clauses, instead they are resolved by **consuming** the answers already stored in their table entries.
- ◆ When a variant call exhausts the set of available answers, the computation state for the variant call is **suspended**.
- ◆ During this process and as further new answers are found, they are stored in their tables and returned to all variant calls.

Tabling Implementations

- The common approach used to include tabling support into existing Prolog systems is to modify and **extend the low-level engine**.
 - ◆ More efficient implementations.
 - ◆ Not easily portable to other Prolog systems (requires changing important components of the system).

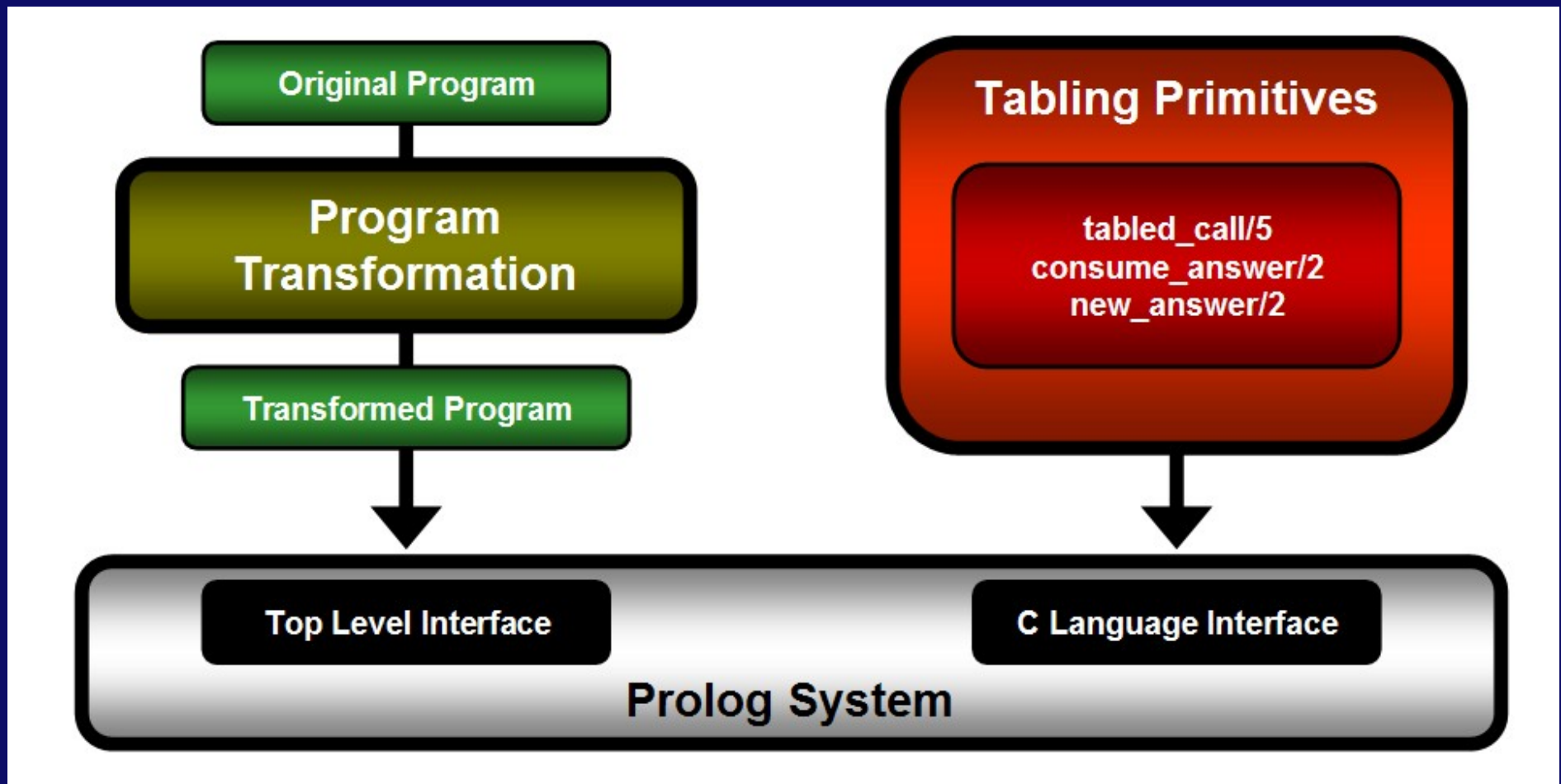
Tabling Implementations

- The common approach used to include tabling support into existing Prolog systems is to modify and **extend the low-level engine**.
 - ◆ More efficient implementations.
 - ◆ Not easily portable to other Prolog systems (requires changing important components of the system).

- A different approach is to **apply source level transformations** to a tabled program and then use **external tabling primitives** to implement tabled evaluation.
 - ◆ Source level transformations can be written in Prolog.
 - ◆ Tabling primitives can be implemented using the C language interface available in most Prolog systems.
 - ◆ Less efficient implementations.

Our Approach

- A suspension-based tabling mechanism based on program transformation with tabling primitives implemented in C.



Our Approach

- The program transformation module is fully **written in Prolog**.
- The tabling primitives module uses the C language interface of the Yap Prolog system. It implements a **local scheduling** search strategy and it uses **tries** to implement the table space.
- Suspension is implemented by leaving the **continuation call** for the current computation in the table entry corresponding to the variant call being suspended. Further new answers are returned to all variant calls by calling the previously stored continuation calls.

Program Transformation

```
% original tabled predicate
```

```
:- table p/2.
```

```
p(X,Z) :- e(X,Y), p(Y,Z).
```

```
p(X,Z) :- e(X,Z).
```

Program Transformation

```
% original tabled predicate
```

```
:- table p/2.
```

```
p(X,Z) :- e(X,Y), p(Y,Z).
```

```
p(X,Z) :- e(X,Z).
```

```
% step I: create a bridge predicate for p/2
```

```
p0(p(X,Z),Sid) :- e(X,Y), p(Y,Z).
```

```
p0(p(X,Z),Sid) :- e(X,Z).
```

Program Transformation

```
% original tabled predicate
```

```
:- table p/2.
```

```
p(X,Z) :- e(X,Y), p(Y,Z).
```

```
p(X,Z) :- e(X,Z).
```

```
% step I: create a bridge predicate for p/2
```

```
p0(p(X,Z),Sid) :- e(X,Y), p(Y,Z).
```

```
p0(p(X,Z),Sid) :- e(X,Z).
```

```
% step II: add primitive new_answer/2 at the end of each clause
```

```
p0(p(X,Z),Sid) :- e(X,Y), p(Y,Z), new_answer(p(X,Z),Sid).
```

```
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).
```

Program Transformation

```
% step III: use primitive tabled_call/5 to break recursive calls
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).
```

Program Transformation

```
% step III: use primitive tabled_call/5 to break recursive calls
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).
```

```
% step IV: adapt p/2 to execute with the bridge predicate
p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true),
          consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).
```

Tabled Evaluation

```

p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true), consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).

```

e(1,2).
e(2,1).

Sid	Subgoal	Answers	Continuation calls

```

?- p(1,Z).
|
1. tabled_call(p(1,Z),Sid,_,p0,true), consume_answer(p(1,Z),Sid).

```


Tabled Evaluation

```

p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true), consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).

```

e(1,2).
e(2,1).

Sid	Subgoal	Answers	Continuation calls
sid1	1. p(1,Z)		

1. tabled_call(p(1,Z),Sid,_,p0,true).

2. p0(p(1,Z),sid1).

3. e(1,Y), tabled_call(p(Y,Z),sid1,[1,Z,Y],p0,p1).

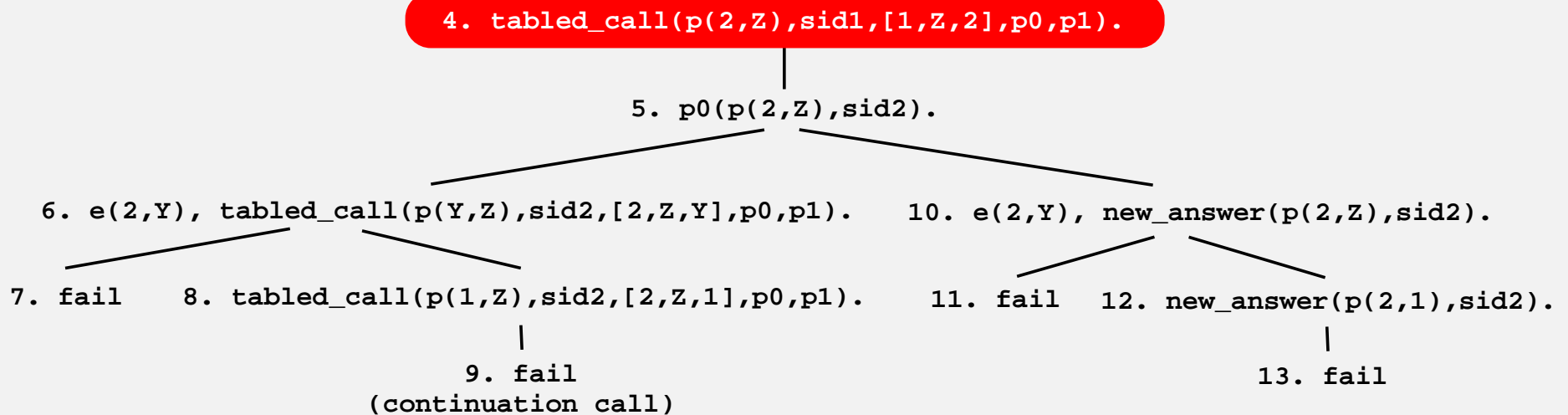
4. tabled_call(p(2,Z),sid1,[1,Z,2],p0,p1).

Tabled Evaluation

```
p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true), consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).
```

e(1,2).
e(2,1).

Sid	Subgoal	Answers	Continuation calls
sid1	1. p(1,Z)		9. p1(?ANS?,sid2,[2,Z,1])
sid2	4. p(2,Z)	12. p(2,1)	



Tabled Evaluation

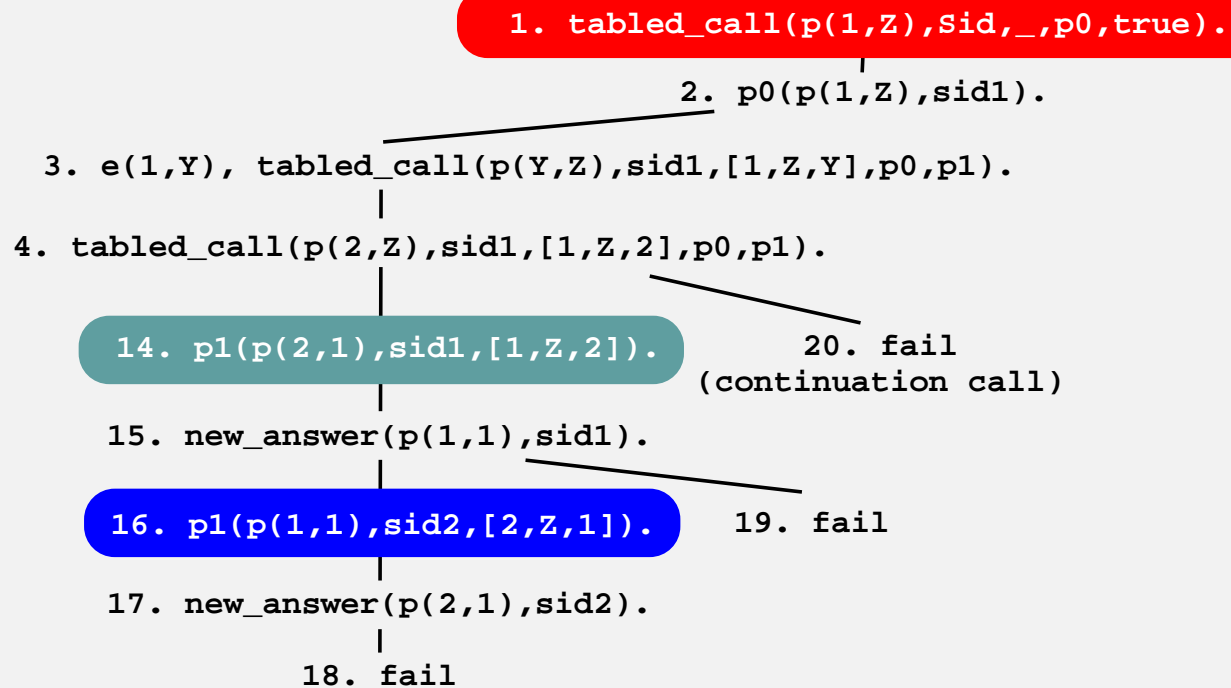
```

p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true), consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).

```

e(1,2).
e(2,1).

Sid	Subgoal	Answers	Continuation calls
sid1	1. p(1,Z)	15. p(1,1)	9. p1(?ANS?,sid2,[2,Z,1])
sid2	4. p(2,Z)	12. p(2,1)	20. p1(?ANS?,sid1,[1,Z,2])



Tabled Evaluation

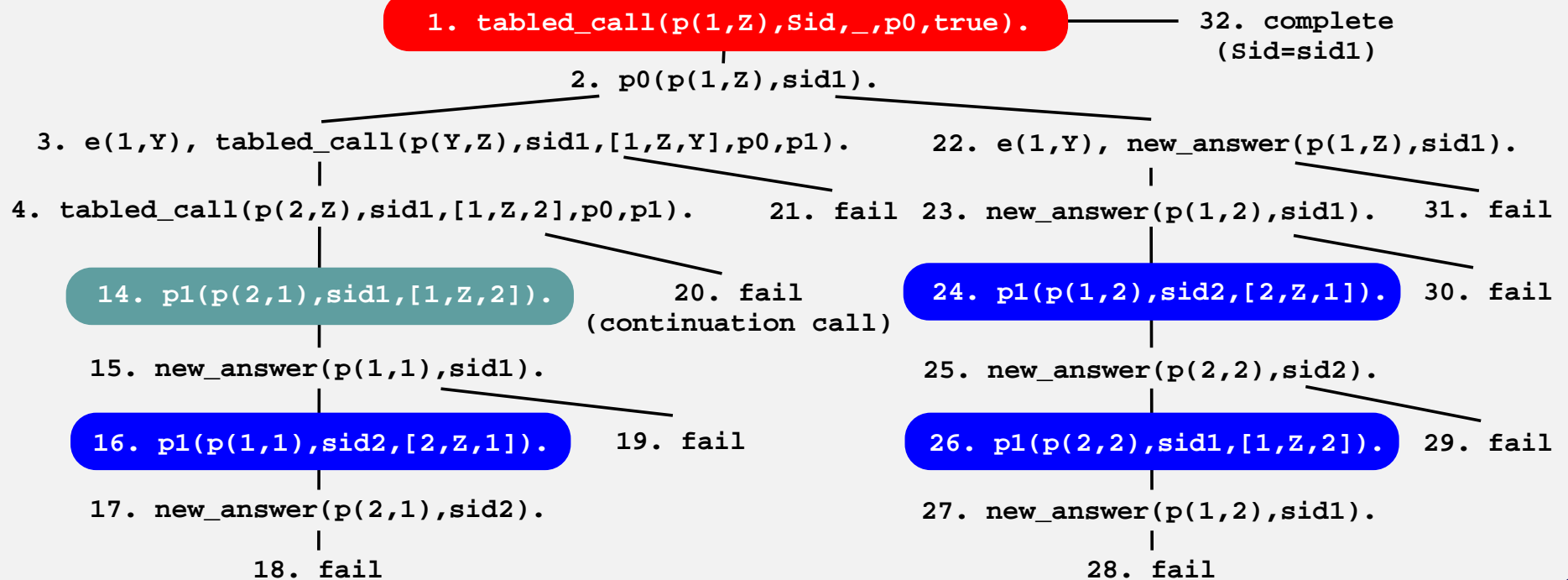
```

p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true), consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).

```

e(1,2).
e(2,1).

Sid	Subgoal	Answers	Continuation calls
sid1	1. p(1,Z)	15. p(1,1) 23. p(1,2) 32. complete	9. p1(?ANS?,sid2,[2,Z,1])
sid2	4. p(2,Z)	12. p(2,1) 25. p(2,2) 32. complete	20. p1(?ANS?,sid1,[1,Z,2])



Tabled Evaluation

```

p(X,Z) :- tabled_call(p(X,Z),Sid,_,p0,true), consume_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Y), tabled_call(p(Y,Z),Sid,[X,Z,Y],p0,p1).
p1(p(Y,Z),Sid,[X,Z,Y]) :- new_answer(p(X,Z),Sid).
p0(p(X,Z),Sid) :- e(X,Z), new_answer(p(X,Z),Sid).

```

e(1,2).
e(2,1).

Sid	Subgoal	Answers	Continuation calls
sid1	1. p(1,Z)	15. p(1,1) 23. p(1,2) 32. complete	9. p1(?ANS?,sid2,[2,Z,1])
sid2	4. p(2,Z)	12. p(2,1) 25. p(2,2) 32. complete	20. p1(?ANS?,sid1,[1,Z,2])

```

?- p(1,Z).
|
1. tabled_call(p(1,Z),Sid,_,p0,true), consume_answer(p(1,Z),Sid).
|
33. consume_answer(p(1,Z),sid1).
|
34. Z=1   35. Z=2   36. no

```

Experimental Results

- We ran our approach against the YapTab system that implements tabling support at the low-level engine. YapTab also implements a suspension-based mechanism, uses tries to implement the table space and is implemented on top of Yap.

Predicate	Binary Tree			Cycle			Grid		
	12	14	16	200	300	400	10x10	15x15	20x20
p_right_first/2	4.00	3.73	3.62	4.36	3.99	3.89	7.75	6.41	6.11
p_right_last/2	3.73	3.59	3.70	4.56	4.00	3.98	8.55	6.27	6.42
p_left_first/2	2.65	2.39	2.34	3.05	2.65	2.26	3.11	2.46	2.12
p_left_last/2	5.00	4.31	4.25	5.13	4.34	4.24	5.67	4.73	4.15
p_doubly_first/2	8.13	7.72	7.68	10.45	11.57	11.22	10.34	9.66	10.40
p_doubly_last/2	15.05	13.96	13.68	20.36	22.23	21.72	19.74	18.25	19.53

Overheads over the YapTab running times

Concluding Remarks

- As expected, YapTab outperformed our mechanism in all programs tested. Best performance was achieved for left recursive tabled predicates with the recursive clause first, with an average overhead between 2 and 3.
- In general, our approach scales well when we increase the complexity of the problem. In particular, for some configurations, the overhead over YapTab shows a generic tendency to decrease as the complexity of the problem increases.

Concluding Remarks

- As expected, YapTab outperformed our mechanism in all programs tested. Best performance was achieved for left recursive tabled predicates with the recursive clause first, with an average overhead between 2 and 3.
- In general, our approach scales well when we increase the complexity of the problem. In particular, for some configurations, the overhead over YapTab shows a generic tendency to decrease as the complexity of the problem increases.
- Considering that Yap and YapTab are two of the fastest Prolog and tabling engines currently available, these results are very interesting and very promising.
- We thus argue that our approach is a good alternative to incorporate tabling into other Prolog systems with a C language interface. Currently, we have already a port of our implementation running as a module of the Ciao Prolog system [CICLOPS 2007, PADL 2008].