Dynamic Mixed-Strategy Evaluation of Tabled Logic Programs

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

Vítor Santos Costa
COPPE Systems & LIACC
Federal University of Rio de Janeiro, Brazil
vitor@cos.ufrj.br

Motivation

- During tabled execution, there are several points where we can choose between continuing forward execution, backtracking, consuming answers from the table, or completing subgoals. A choice is made by the **scheduling strategy**.
- There is no single best scheduling strategy [Freire, PhD].
- Best performance may be achieved by **using multiple strategies within the same evaluation** [Freire and Warren, 97].

Our Contribution

- **Mixed-strategy evaluation** for **batched and local scheduling**.
- Elegant extension of the original **YapTab system** design.
- Support **dynamic intermixing** of batched and local scheduling at the **subgoal level**.

Tabling Execution Model

Basic Execution Model

- Whenever a tabled subgoal is first called, 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, instead of being re-evaluated against the program clauses.
- Meanwhile, as new answers are found, they are inserted into the table and returned to all variant subgoals.

Nodes Classification

- **Generators**: nodes that first call a tabled subgoal.
- **Consumers**: nodes that consume answers from the table space.

Tabling Operations

- **Tabled Subgoal Call:** checks if a subgoal is in the table. If so, allocates a consumer and starts consuming the available answers. If not, adds a new entry to the table, and allocates a new generator node.
- **New Answer:** verifies whether a newly found answer is already in the table, and if not, inserts the answer. Otherwise, fails.
- **Answer Resolution:** verifies whether extra answers are available for a particular consumer and, if so, consumes the next one. Otherwise, **suspends** the current computation and schedules a possible resolution to continue the execution.
- **Completion:** determines whether a subgoal is **completely evaluated**, that is, when no more answers can be found. If so, closes the subgoal's table entry and reclaims space. Otherwise, moves to a consumer with unconsumed answers.

Tabling Operations

- **Tabled Subgoal Call:** checks if a subgoal is in the table. If so, allocates a consumer and starts consuming the available answers. If not, adds a new entry to the table, and allocates a new generator node.
- **New Answer:** verifies whether a newly found answer is already in the table, and if not, inserts the answer. Otherwise, fails.
- **Answer Resolution:** verifies whether extra answers are available for a particular consumer and, if so, consumes the next one. Otherwise, **suspends** the current computation and schedules a possible resolution to continue the execution.
- **Completion:** determines whether a subgoal is **completely evaluated**, that is, when no more answers can be found. If so, closes the subgoal's table entry and reclaims space. Otherwise, moves to a consumer with unconsumed answers.
  - A number of subgoals may be mutually dependent (**Strongly Connected Component or SCC**) and thus they can only be completed together. The youngest subgoal which does not depend on older subgoals is the **leader**. The leader defines the current completion point.

Batched Scheduling

- The batched strategy schedules the program clauses in a depth-first manner as does the WAM.
- When **new answers** are found for a particular tabled subgoal, they are added to the table space and the **evaluation continues**.
- Newly found answers are only returned to consumer nodes when all program clauses for the whole SCC were resolved.

Batched Scheduling

```
:- table t/1.

t(1).
t(2).
```

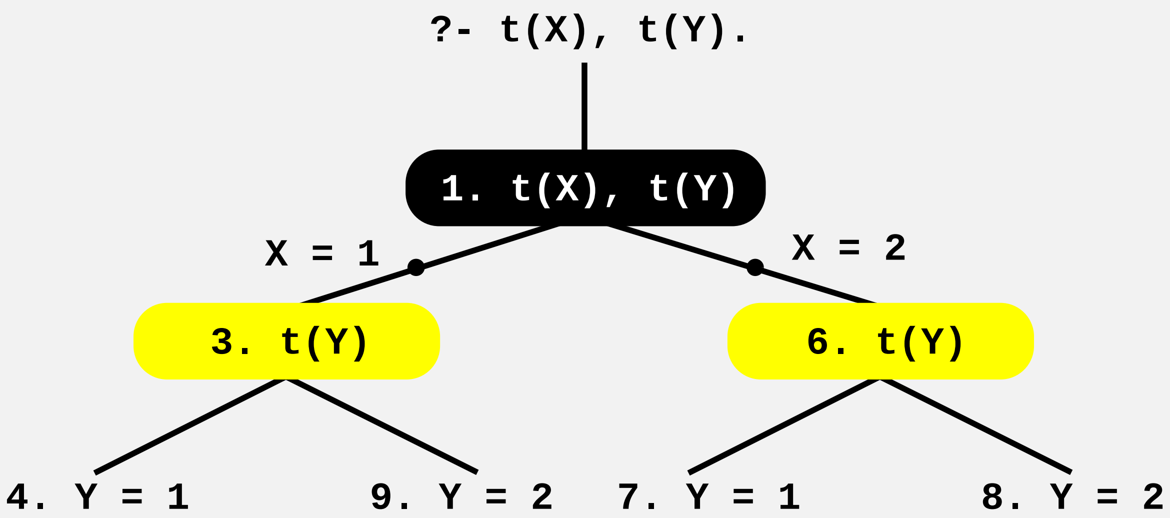
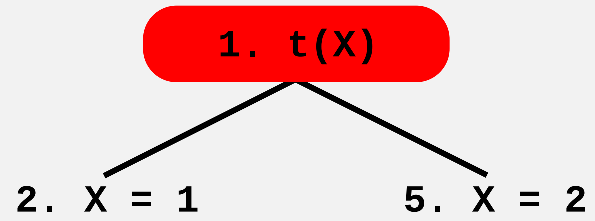


Table space

subgoal	answers
1. t(X)	2. X = 1 5. X = 2 10. complete



Local Scheduling

- The local strategy tries to complete subgoals as soon as possible, that is, evaluation is done one SCC at a time.
- The key idea is that when **new answers** are found, they are added to the table space and the **evaluation fails**.
- Answers are only returned outside the SCC when the whole SCC is completed.

Local Scheduling

```
:- table t/1.

t(1).
t(2).
```

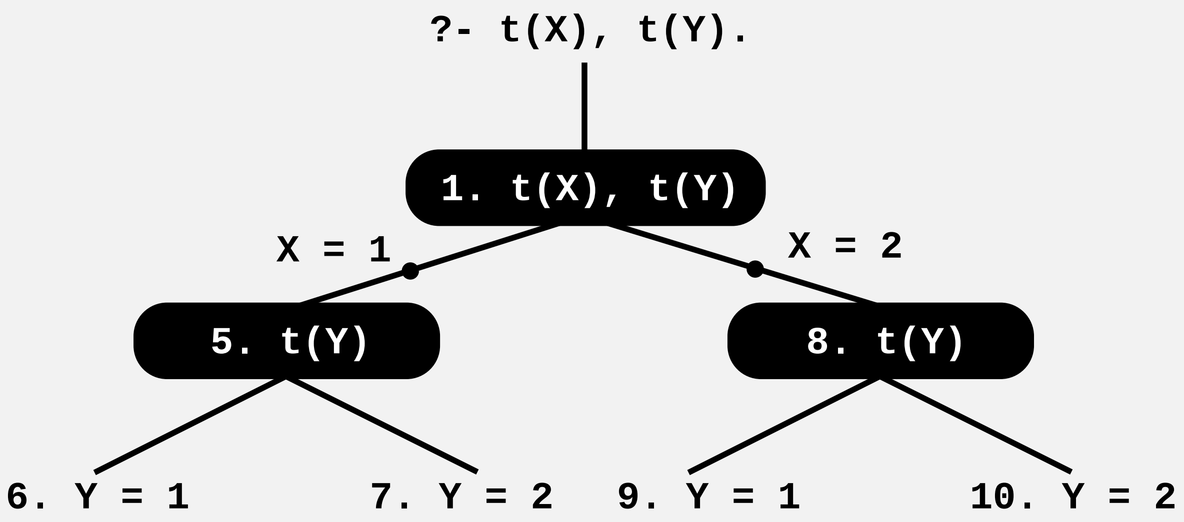
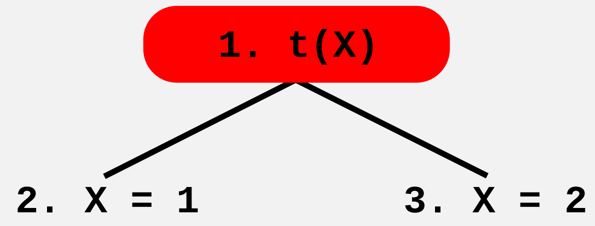


Table space

subgoal	answers
1. t(X)	2. X = 1 3. X = 2 4. complete



Batched x Local Scheduling

➤ Main Differences

- In batched, when a new answer is found, the evaluation continues. In local, the evaluation fails.
- In batched, when a SCC is completed, the evaluation fails. In local, the leader starts acting like a consumer and consumes the first available answer.

Batched x Local Scheduling

Main Differences

- In batched, when a new answer is found, the evaluation continues. In local, the evaluation fails.
- In batched, when a SCC is completed, the evaluation fails. In local, the leader starts acting like a consumer and consumes the first available answer.

Questions

- Can we have **different predicates** being evaluated by different strategies?
- Can we have **different subgoals** being evaluated by different strategies?

Our Approach

➤ Previous YapTab Version

- Compile Yap with **-DTABLING_BATCHED_SCHEDULING=1** to enable tabling support with batched scheduling.
- Compile Yap with **-DTABLING_LOCAL_SCHEDULING=1** to enable tabling support with local scheduling.

Our Approach

Previous YapTab Version

- Compile Yap with **-DTABLING_BATCHED_SCHEDULING=1** to enable tabling support with batched scheduling.
- Compile Yap with **-DTABLING_LOCAL_SCHEDULING=1** to enable tabling support with local scheduling.

Current YapTab Version

- Compile Yap with **-DTABLING=1** to enable tabling support with both batched and local scheduling.
- Use the standard **yap_flag/2** predicate to define the scheduling strategy for the whole computation.
- Use the new **tabling_mode/2** predicate to define the scheduling strategy of a particular tabled predicate. The default scheduling strategy is batched.

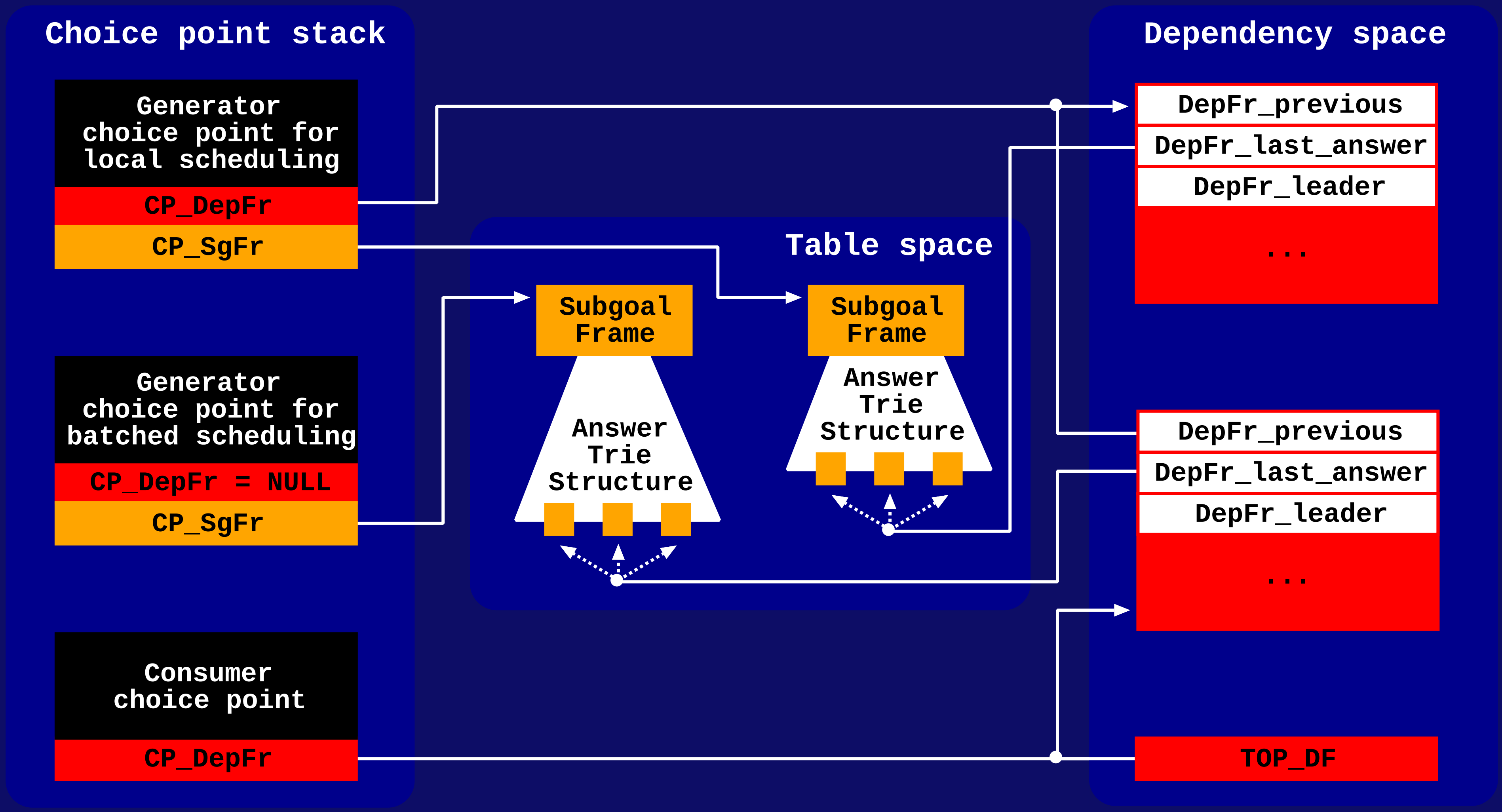
Our Approach

- Consider, for example, two tabled predicates, **t/1** and **t/2**, and the query goals:
  - :- t(1).
  - :- **yap_flag(tabling_mode,local)**, t(2,2).
  - :- t(3), **yap_flag(tabling_mode,default)**, t(3,3).
  - :- **tabling_mode([t/1,t/2],local)**, t(X), t(X,Y).
  - :- **tabling_mode(t/1,batched)**, t(Y).
- Subgoals evaluated with **batched scheduling**:
  - t(1)
  - t(3,3)
  - t(Y)
- Subgoals evaluated with **local scheduling**:
  - t(2,2)
  - t(3)
  - t(X)
  - t(X,Y)

Implementation

- In YapTab, applying batched or local scheduling to an evaluation mainly depends on the way **generator nodes** are handled.
- At the engine level, this includes minor changes to the operations **tabled subgoal call**, **new answer** and **completion**.
- All the other tabling extensions are common across both strategies.

Tabled Nodes



Tabled Subgoal Call

```
tabled_subgoal_call(subgoal call SC) {
  if (first_call_to(SC)) {
    GN = allocate_new_generator_node()
    CP_SgFr(GN) = add_new_table_entry(SC)
#ifdef TABLING_LOCAL_SCHEDULING
    CP_DepFr(GN) = allocate_new_dependency_frame()
#endif
  } else {
    ...
  }
}
```

Tabled Subgoal Call

```
tabled_subgoal_call(subgoal call SC) {
  if (first_call_to(SC)) {
    GN = allocate_new_generator_node()
    CP_SgFr(GN) = add_new_table_entry(SC)
    if (tabling_mode(SC) == batched)        // batched scheduling
      CP_DepFr(GN) = NULL
    else                                      // local scheduling
      CP_DepFr(GN) = allocate_new_dependency_frame()
  } else {
    ...
  }
}
```

New Answer

```
new_answer(answer A, generator node GN) {
  insert_answer(A, CP_SgFr(GN))
#ifdef TABLING_BATCHED_SCHEDULING
  proceed()
#else                                    // TABLING_LOCAL_SCHEDULING
  fail()
#endif
}
```

New Answer

```
new_answer(answer A, generator node GN) {
  insert_answer(A, CP_SgFr(GN))
  if (CP_DepFr(GN) == NULL)                      // batched scheduling
    proceed()
  else                                             // local scheduling
    fail()
}
```

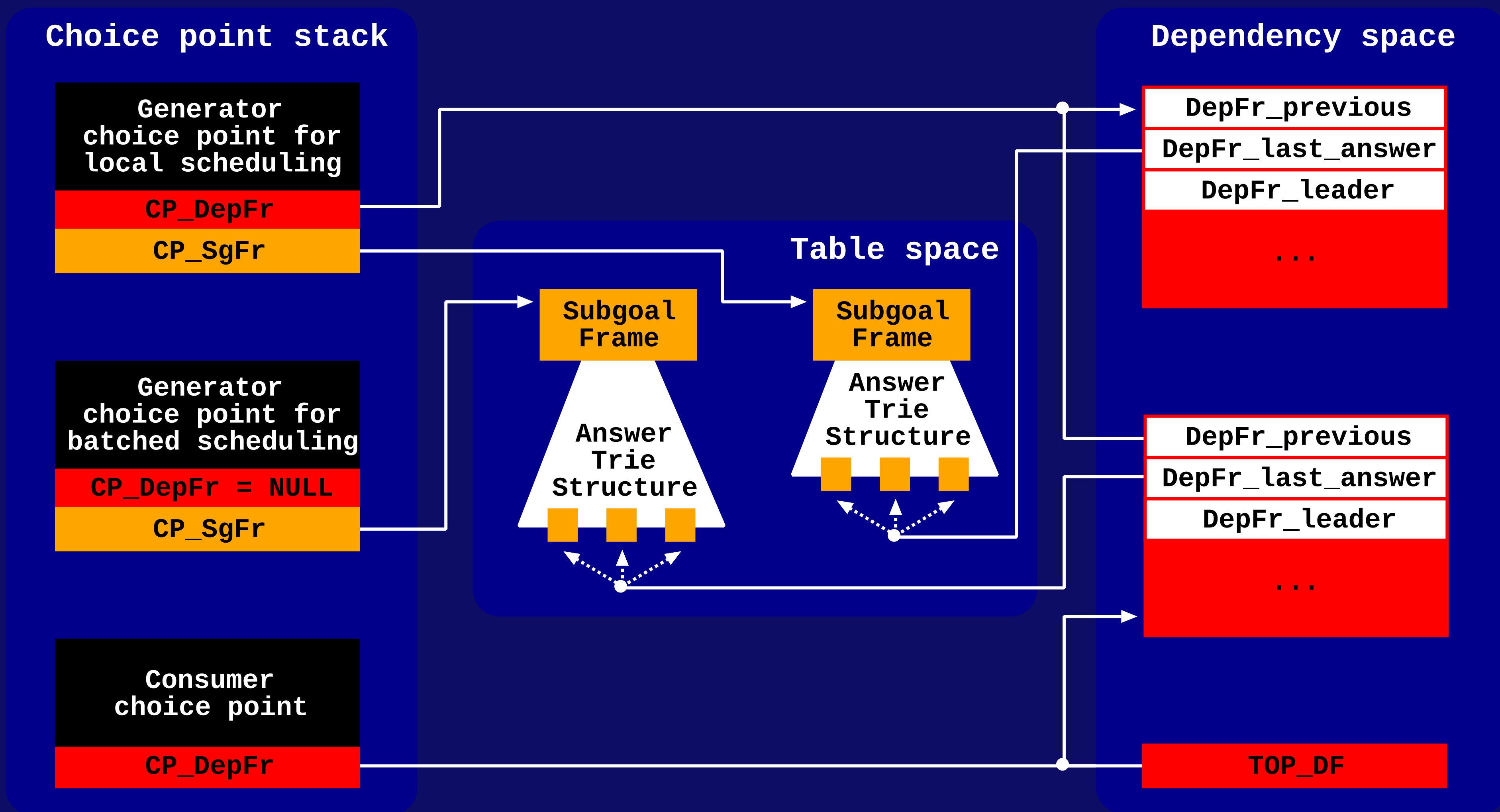
Completion

```
completion(generator node GN) {
  if (GN is the current leader node) {
    ...
    perform_completion()
  }
#ifdef TABLING_BATCHED_SCHEDULING
  fail()
#else                                        // TABLING_LOCAL_SCHEDULING
  CP_AP(GN) = answer_resolution
  load_first_answer_and_proceed()
#endif
}
```

Completion

```
completion(generator node GN) {
  if (GN is the current leader node) {
    ...
    perform_completion()
  }
  if (CP_DepFr(GN) == NULL)                  // batched scheduling
    fail()
  else {                                       // local scheduling
    CP_AP(GN) = answer_resolution
    load_first_answer_and_proceed()
  }
}
```

Tabled Nodes



Experimental Results

Overhead of supporting mixed-strategy evaluation

	Batched Scheduling		Local Scheduling	
Program	**Single**	**Mixed**	**Single**	**Mixed**
mc-iproto	2.495	2.519 (**1.009**)	2.668	2.689 (**1.007**)
mc-leader	8.452	8.467 (**1.001**)	8.385	8.403 (**1.002**)
mc-sieve	21.568	21.325 (**0.988**)	21.797	21.217 (**0.973**)
lgrid	0.850	0.870 (**1.023**)	1.012	1.031 (**1.018**)
rgrid	1.250	1.332 (**1.065**)	1.075	1.141 (**1.061**)
samegen	0.020	0.020 (**1.000**)	0.021	0.021 (**1.000**)
Average		(**1.014**)		(**1.010**)

Running times in seconds.

Experimental Results

Intermixing batched and local scheduling at the predicate level

Predicates	Running Time (s)
Without tabling	$>$ 1 day
All batched (11 predicates)	283
All local (11 predicates)	147
Some batched (7 predicates), others local (4 predicates)	127

The running times include the time to run the whole ILP system.

Experimental Results

Intermixing batched and local scheduling at the predicate level

Predicates	Running Time (s)
Without tabling	$>$ **1 day**
All batched (11 predicates)	**283**
All local (11 predicates)	**147**
Some batched (7 predicates), others local (4 predicates)	**127**

The running times include the time to run the whole ILP system.

- Better performance is still possible if we use YapTab's flexibility to intermix batched and local scheduling at the subgoal level.
- From the programmer point of view, it is very difficult to define the subgoals to table using one or another strategy.
- Further work is still needed to study how to use this flexibility to, in runtime, automatically adjust the system to the best approach.

Experimental Results

Intermixing batched and local scheduling at the subgoal level

Query Goal	Running Time(s)
:- **go_batched**, path(X,Y), reach_both(X,Y), fail.	**141**
:- **go_local**, path(X,Y), reach_both(X,Y), fail.	**60**
:- **go_local**, path(X,Y), **go_batched**, reach_both(X,Y), fail.	**19**

```
:- table path/2.
path(X,Y) :- path(X,Z), edge(Z,Y).
path(X,Y) :- edge(X,Y).

reach_both(X,Y) :- path(F,X), path(F,Y), !.

go_batched :- tabling_mode(path/2,batched).
go_local :- tabling_mode(path/2,local).
```

Conclusions

- We presented the design and implementation of YapTab to support dynamic mixed-strategy evaluation of tabled logic programs.
- Our approach proposes the ability to combine batched scheduling with local scheduling at the subgoal level with minor changes to the tabling engine.
- Our results show that dynamic mixed-strategies can be very important to improve the performance of some applications.

Further Work

- Design a more *aggressive* approach for applications that generate large tables and/or do a lot of pruning over the table space, such as ILP applications.
  - Automatically recover space from unused tables.
  - Support incomplete tables.
- Support alternative approaches for declaring the tabling mode.
  - tabling_mode(**path(1,2)**,local).
  - tabling_mode(**path(1,*)**,local).
- Investigate the impact of combining mixed-strategy evaluation in other application areas.