Extending Tabled Logic Programming with Multi-Threading: A Systems Perspective

Rui Marques
CITI, Dep. Informatica FCT, Universidade Nova de Lisboa

Terrance Swift
CENTRIA Universidade Nova de Lisboa

José Cunha
CITI, Dep. Informatica FCT, Universidade Nova de Lisboa

Overview

- What would we like to make multi-threaded?
  - Tabled negation (WFS + residual program)
  - Call variance vs. Call subsumption
  - Answer variance vs. Answer subsumption
  - Tabled constraints
  - Incremental tabling
  - Space reclamation
- How far have we gotten?
  - Thread Private Tables in XSB
  - Thread Shared Tables in XSB
    - The effect of Local and Batched Evaluation

Goals for Multi-Threaded XSB

- Support multi-threading ISO/IEC DTR 13211–5:2007
- Any tabling function should be available to any active thread using tables that are private to a thread.
- Any tabling function should be available to any active thread using tables that are shared among threads.
- Private tables should be highly scalable up to the number of cores available.
- For problems that support both tabling and parallelism, shared tables should be able to provide speedup proportional to the number of cores available
- MT-TLP can be used by others for real work

These goals are ambitious and not yet fulfilled!

Benchmarking Tabling

- Paucity of good tabling benchmarks
  - Some (e.g. graph reachability) test well parts of an engine, but don't have a realistic mixture of tabled and non-tabled code
  - Others for analysis and model-checking test only definite programs
- We introduce some new benchmarks based on reachability in Petri or workflow nets
  - Reachability is a central problem for Petri Net analysis, to which problems such as liveness, deadlock-freedom, and the existence of home states can be reduced.
- The following slide illustrates the *entire* program for 1-safe (elementary) Petri Nets as a tabled definite program
  - Configuration is maintained as a ordered set of terms (terms differ for elementary nets, place-transition nets, color nets, preference nets, etc.)
- A definite program for analyzing workflow nets is approximately twice the size.
  - The program supports nearly all standard workflow control patterns in (van der Aalst, ter Hofstede, 2003). This includes splits, synchronized merges, discriminators, cancellation, etc.
- All benchmark code can be found in CVS `mttests` module of XSB on `sourceforge.net`.

Tabling: Definite Programs

```
% Prolog representation of the Producer-Consumer Net
:- index(trans/2,trie).
trans([p1],[p2],t1).          trans([b2,p2],[p1,b1],t2).
trans([b1,c1],[b2,c2],t3).    trans([c2],[c1],t4).

% Program to determine reachability of an elementary net
:- table reachable/2.
reachable(InConf,NewConf):-
   reachable(InConf,Conf),
   hasTransition(Conf,NewConf).
reachable(InConf,NewConf):-
   hasTransition(InConf,NewConf).

hasTransition(Conf,NewConf):-
   get_trans_for_conf(Conf,AllTrans),
   member(Trans,AllTrans),
   apply_trans_to_conf(Trans,Conf,NewConf).

get_trans_for_conf(Conf,Flattrans):-
   get_trans_for_conf_1(Conf,Conf,Trans),
   flatten(Trans,Flattrans).

get_trans_for_conf_1([],_Conf,[]).
get_trans_for_conf_1([H|T],Conf,[Trans1|RT]):-
   findall(trans([H|In],Out,Tran),trans([H|In],Out,Tran),Trans),
   check_concession(Trans,Conf,Trans1),
   get_trans_for_conf_1(T,Conf,RT).

check_concession([],_,[]).
check_concession([trans(In,Out,Name)|T],Input,[trans(In,Out,Name)|T1]):-
   ord_subset(In,Input),
   ord_disjoint(Out,Input),!,
   check_concession(T,Input,T1).
check_concession([_Trans|T],Input,T1):-
   check_concession(T,Input,T1).

apply_trans_to_conf(trans(In,Out_Name),Conf,NewConf):-
   ord_subtract(Conf,In,Diff),
   flatsort([Out|Diff],Temp),
```

Tabled Negation

- Well-founded negation (WFS) can be used
  - to compute inheritance in object-logics (e.g. Flora, CDF)
  - to code preferences (e.g. Preference Logic Grammars)
  - to compute a residual program to be sent to an ASP solver (e.g. XASP/Smodels)
- For the Workflow Net example, WFS can be used to ensure that a transition T is taken from a configuration C only if no other transition is preferred to T for C.
  - Lookaheads from a given state can be performed
  - Preferences can be added to a workflow in a modular way, to support local policies or requirements

```
hasTransition(Conf,NewConf):-
   get_trans_for_conf(Conf,AllTrans),
   member(Trans,AllTrans),
   sk_not(unpreferred(Trans,AllTrans,Conf)),
   apply_trans_to_conf(Trans,Conf,NewConf).
```

Answer Variance vs. Answer Subsumption

- For calls, subsumption is useful on the partial order (lattice) of terms.
- For answers, this is not usually very useful, but other partial orders or lattices are:
  - Interval [0,1] for fuzzy sets
  - Lattice of uncertainty intervals in Dempster-Schaeffer inference
  - Lattices for paraconsitent logics
  - Monotonic recursive aggregates (min, max)
  - Comparing abductive solutions
- Example. Naive reachability for Place/Transition Nets may not terminate. Instead, a technique called ω-abstraction must be used. A place marked with ω indicates that it may contain any number of tokens.
- In XSB, the left-recursive `reachable/2` predicate is rewritten as:

```
reachable(InConf,NewConf):-
    filterPOA(reachable(InConf),Conf,gte_omega,omega_abstr,call_abstr),
    hasTransition(Conf,NewConf).
```

`filterPOA/5` calls `reachable(InConf,-ConfTemp)` and succeeds if there is no answer for the table (returning `Conf`) or if `Conftemp` is greater than any element in the table for `reachable(InConf,-ConfTemp)` that has key `reachable(Conf,_)`. `ConfTemp` may be abstracted by `omega_abstr/2` and before returning it as `Conf`.

Tabled Constraints

- Many Constraint Logic Programs do not benefit from tabling, as the logic program is used primarily to set up the constraints.
- Tabling *can* help search through a state space where the states can be labelled with constraints.
  - Examples in natural language analysis, program analysis, verification, ILP
- In XSB, attributed variables are copied into and out of tables as any other term
- Need to handle constraint interrupts before accessing the table for a tabled subgoal or answer
- Example. Constraints can encode a type of Colored Petri net. Rather than collecting tokens, a transition collects constraints until the constraint set entails a formula (and does not fire otherwise). Once it fires, new constraints may be applied to the resulting configuration.

```
apply_trans_to_conf(trans(In,Entailment,Out),Conf,NewConf):-
    unify_for_entailment(In,Conf,MidConf),
    entailed(Entailment),
    call_new_constraints(Out,OutPlaces),
    flatsort([OutPlaces|MidConf],NewConf).
```

`unify_for_entailment/3` simply unifies variables in the transition with those of the configuration to produce a constraint store for entailment checking.

Call Variance vs. Call Subsumption

- When should a tabled subgoal reuse a table and when should it create a new one?
- The most common approach is call variance which uses a table if a table exists whose subgoal is a variant of a selected subgoal
  - E.g. the subgoal p(X,a,X) can use the table for p(Y,a,Y)
- Call subsumption is useful for computing fixed-points of programs, e.g. program analysis, RDF inferences, certain types of deductions in OWL (e.g. OWL wine example).
  - E.g. the subgoal p(X,a,X) can use the table for p(Y,a,Z)

Incremental Tabling

- Answers for a tabled subgoal may depend on dynamic facts used in the derivation of the answers. When these facts change, the answers may become invalid.
  - Incremental tabling provides automatic recomputation of answers as dynamic facts are asserted or retracted.
  - Useful for maintaining graphical views of an underlying logical model. For instance, user interfaces driven by Interprolog/XJ or deductive spreadsheets.

Table management and space reclamation

- Tables can be explicitly abolished (truncated), and are implicitly abolished when a declaring the tabled predicates is reconsulted.
  - Tables can be abolished at the subgoal or predicate level. In addition, all tables for predicates in a given module can be abolished, as well as all tables present in the system.
  - Space for abolished tables may not be immediately reclaimable if there are choice points that will backtrack through these tables. In XSB, garbage collection is performed at the subgoal and predicate level.
  - When well-founded evaluation creates a residual program (with clauses) deleting a single table could lead to dangling pointers. E.g. if a residual clause were `p(a):- tnot(q(b))`, deleting the subgoal `q(b)` or the predicate `q/2` may lead to dangling pointers in the table for `p(a)`. In such a case, XSB will perform a *cascading abolish* that also abolishes `q(a)`, and so on.
  - YAP allows automatic reclamation via a least-recently-used algorithm.

MT-TLP: Private Tables

- Private tables are suitable to ensure query completeness or to support a particular semantics (WFS, GAPs, Preference Logics).
- Private tables use sequential tabling algorithms, but with thread-safe code. They generally require no synchronization above the level of memory management.
- Each thread can access the residual program for its private tables
  - Each thread must be able to reclaim its own table space upon exit or explicit abolish (including delay listes, etc. to support WFS)
  - Must ensure that memory allocation for table space data structures is efficient

MT-TLP: Summary of Private Table Functionality

Feature	Private Tables
Tabled constraints	Supported
Answer subsumption	Supported
Tabled Dynamic Code	Supported
Tabled negation	Supported
Space reclamation	Supported
Call subsumption	Supported
Incremental recomputation	"Almost" supported
XASP (Residual Program → Smodels)	"Almost" supported

Goal: *Any tabling function should be available to any active thread using tables that are private to a thread.* is "almost" supported.

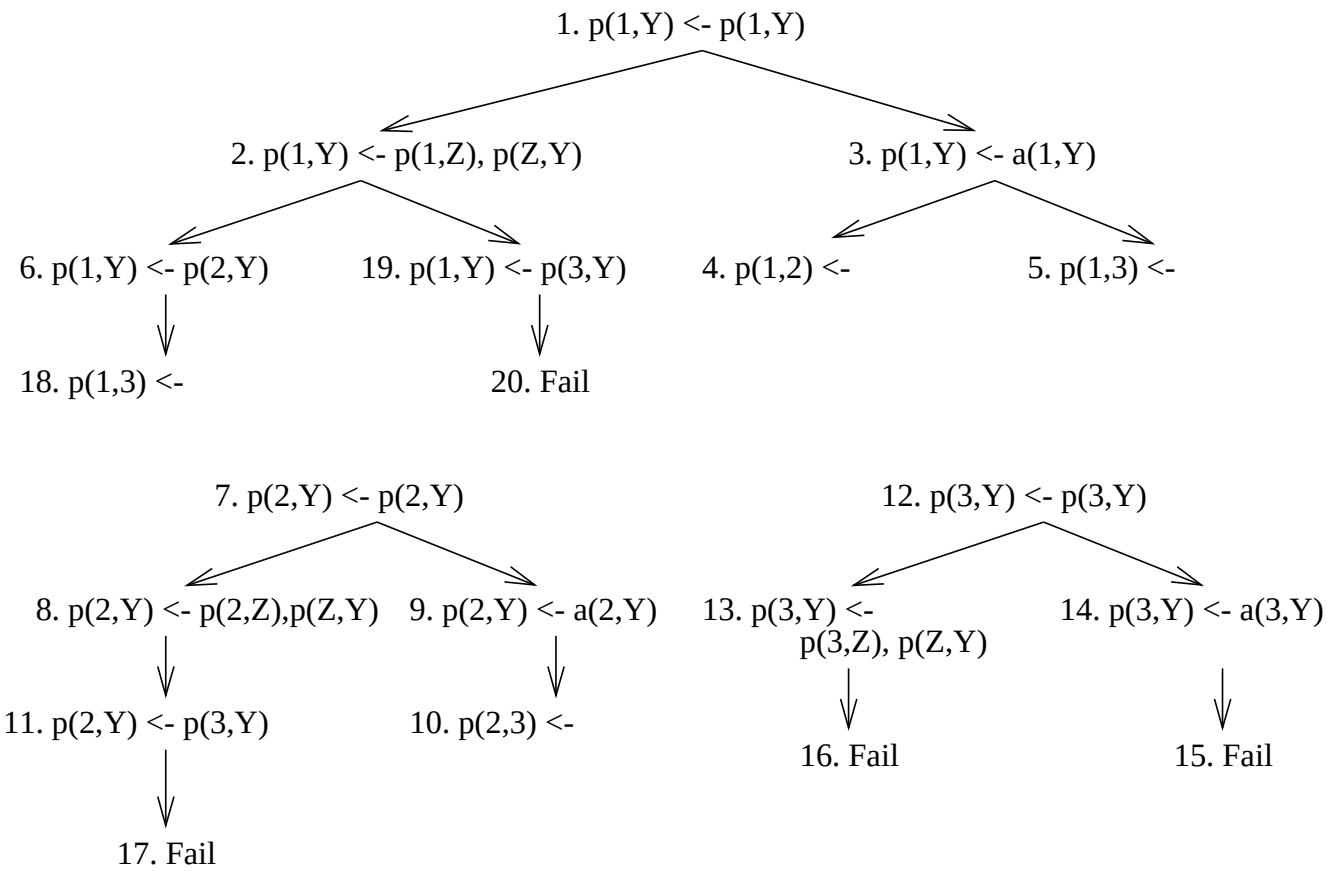
Private Table Performance

N. threads	1	2	Overhead	4	Overhead
`Elementary Net`	5.94	6.23	4.8%	6.25	5.2%
`Dynamic Elementary Net`	6.03	6.03	0%	6.03	0%
`Workflow Net`	19.21	19.68	2.4%	19.95	3.8%
`Omega Net`	7.18	8.33	16.0%	10.3	46.0%
`Omega Net Specialized`	6.37	6.37	0%	6.37	0.0%
`Constraint Net`	2.75	2.84	3.2%	2.85	3.6%
`Preference Net`	3.74	3.77	0.8%	3.82	2.1%
`Call Subsumption`	.86	1.04	20.0%	1	43%

- `Elementary Net` uses tabled definite programs for Petri Net reachablilty; `Dynamic Elementary Net` is the same, but uses dynamic clauses. `Workflow Net` uses definite programs to test workflows.
- `Constraints Net` tests the constraint-based colored Petri nets, and `Preferences Net` tests extension for well-founded preferences.
- `Omega Net` tests ω-abstraction using answer subsumption. `filterPOA/5` relies on `call/N`, which accesses XSB's shared predicate table, causing poor performance. A specialized version, `Omega Net Specialized` avoids this problem
- `Call Subsumption` tests an unbound call to right recursion over a graph. Its poor scalability is apparently due to high demands for memory management
- Goal: *Private tables should be highly scalable up to the number of cores available* is mostly satisfied (for 4 cores), although more work needs to be done on call subsumption.

Scheduling: Local Evaluation

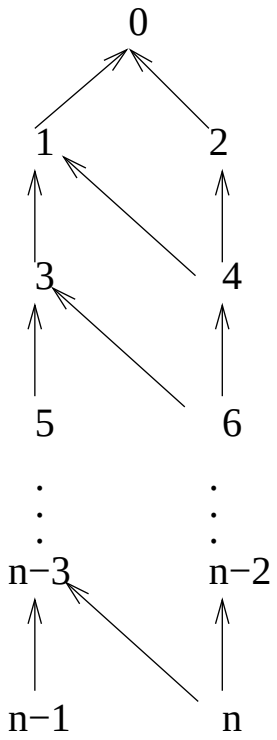
- Local Evaluation: Completely evaluate each mutually dependent set of subgoals before returning an answer to a subgoal not in that set.




```
:- table p/2.
p(X,Y) :- p(X,Z), p(Z,Y).
p(X,Y) :- a(X,Y).
a(1,2).  a(1,3).  a(2,3).
```

Local Evaluation: Example

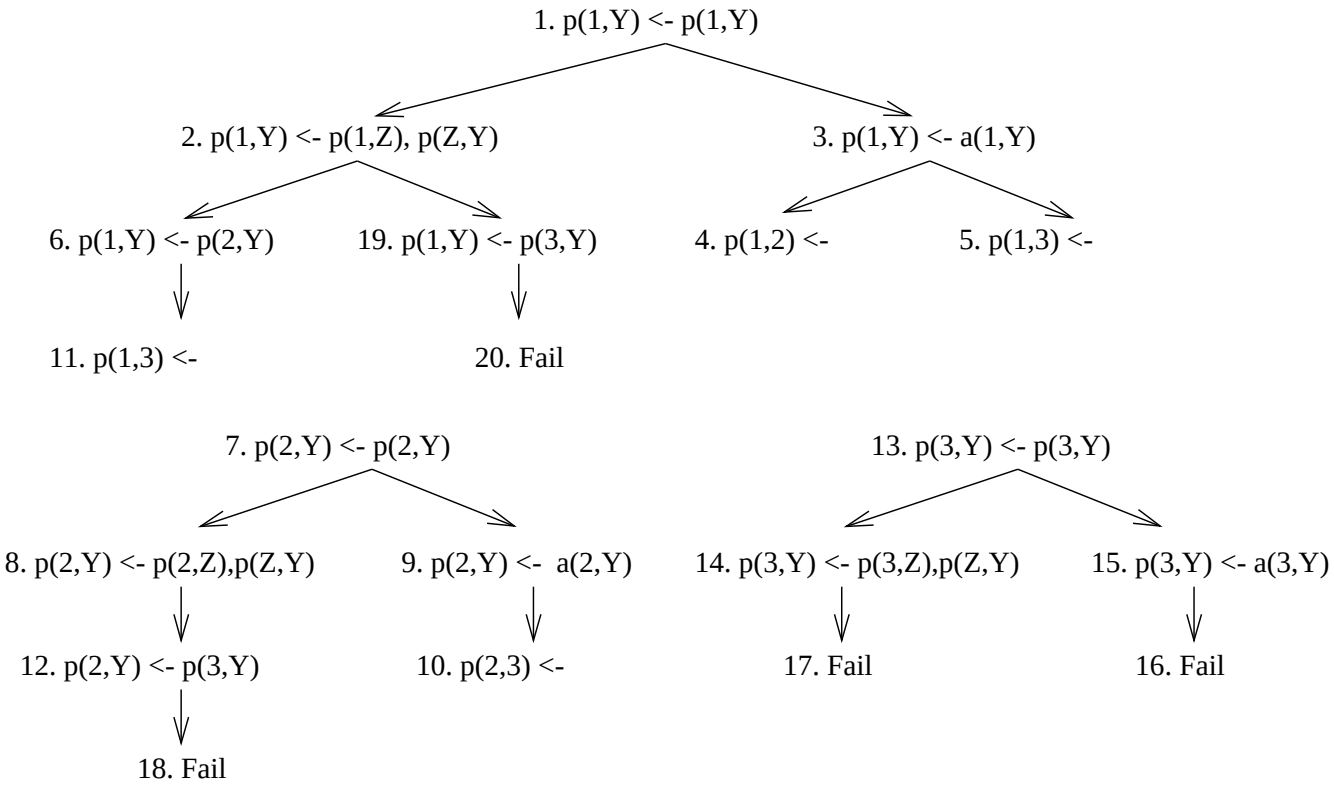
```
sgi(X,Y)(D) :- arc(X,Y).
sgi(X,Y)(D) :-
     arc(X,Z), subsumes(min)(sgi(Z,Z1),D1),
     arc(Y,Z1), D is D1+1.
```



- Time for `?-p(bound,free)` is linear in edges for Local Evaluation, Linear in size of paths for Batched Evaluation

Scheduling: Batched Evaluation

- Batched Evaluation: Return an answer to parent environments as soon as it is derived.




```
:- table p/2.
p(X,Y) :- p(X,Z), p(Z,Y).
p(X,Y) :- a(X,Y).
a(1,2).  a(1,3).  a(2,3).
```

Scheduling Strategies

- *Answer Subsumption* Local Evaluation is superior as only optimal answers are returned out of a maximal SCC.
- *Negation* Local Evaluation can be more efficient for non-stratified negation as it may allow delayed answers that are later simplified away to avoid being propagated.
- *Time to first answer* Batched Evaluation is faster, because it returns answers out of an SCC eagerly.
- *Time for left recursion* Batched Evaluation is somewhat faster than Local Evaluation for left recursion as Local Evaluation imposes overhead to prevent answers from being returned outside of an independent SCC.
- *Stack space* Local Evaluation often requires less space than Batched Evaluation as it fully explores a maximal independent SCC, completes the SCC's subgoals, reclaims space, and then moves on to a new SCC.
- *Call subsumption* Local Evaluation can be faster than Batched Evaluation, as subsumed calls may make use of completed subsuming tables.
- ***Scope for Parallelism*** Batched Evaluation allows one thread to consume answers as they are produced by another thread

Shared Table Functionality for Local Evaluation

Feature	Shared Tables (Local)
Tabled Dynamic Code	Supported
Tabled constraints	Supported
Answer subsumption	Supported
Tabled negation	Supported
XASP (Residual Program → Smodels)	"Almost" supported
Space reclamation	Partially Supported
Call subsumption	Not supported
Incremental recomputation	Not supported

- The generality of Concurrent Local SLG means that it can support most tabling functionality
- Full space reclamation for abolished tables is not yet fully supported because of lack of resources to implement a table garbage collector in a multi-threaded environment.
- Further work needs to be done to extend Concurrent Local SLG to call subsumption and to assess its efficiency.
- Incremental recomputation is difficult for shared tables, since each thread that is using a table must maintain its view of the table throughout a query

Shared Tables: Efficiency of Concurrent Local SLG

N. threads	1	2	Speedup	4	Speedup
`Shared Elementary`	25.12	13.00	1.93	6.55	3.83
`Shared Dynamic Elemtary`	24.8	13.02	1.90	6.59	3.76
`Shared Workflow`	41.25	20.78	1.98	10.58	3.89
`Shared Omega`	19.58	10.38	1.88	5.57	3.51
`Shared Constraint`	11.13	5.56	2.00	2.83	3.93
`Shared Preferences`	3.73	1.86	1.99	0.95	3.92

- Speedup is based on partitionable search space
- Overhead of shared tables over private tables is about 10-40%‘ depending on benchmark and platform
- Overheads are mostly due to handling mutual exclusion for tables and to memory allocation. Further work is needed in these areas.

Shared Tables: Batched Evaluation

- For Concurrent Batched Evaluation, if one thread calls an (incomplete) table owned by another thread, it backtracks through the existing answers, rather than suspending
- This fixes Producer/Consumer example, and is all you need unless two threads own subgoals in the same SCC
- When there are intra-SCC dependencies among threads, Concurrent Batched Evaluation is a generalization of Sequential Batched Evaluation.
- In a sequential (or thread-private) computation, when the engine backtracks to the oldest subgoal in an SCC, it schedules the return of unconsumed answers for each consuming node in the SCC by creating a chain of choice points, and then backtracks into the newly created chain.

Shared Tables: Batched Evaluation

In a concurrent computation, let $\mathcal{S}$ be an SCC with subgoals owned by different threads

- Suppose a thread T_1 computing subgoals in $\mathcal{S}$ backtracks to the oldest subgoal that it "owns" in $\mathcal{S}$.
- If another thread computing $\mathcal{S}$ is active, T_1 will suspend and will be wakened when a thread performs batched scheduling for $\mathcal{S}$;
- Tf T_1 is the last unsuspended thread computing subgoals in $\mathcal{S}$, T_1 itself will perform a fixed point check and batched scheduling and awaken the other threads computing $\mathcal{S}$ — either to return further answers or to complete their tables.
- As implemented in XSB, Concurrent Batched Evaluation thus allows parallel computation of subgoals, but has a sequential fixpoint check that synchronizes multiple threads when they compute the same SCC.

Shared Table Functionality for Batched Evaluation

Feature	Shared Tables (Batched-β)
Tabled Dynamic Code	Supported
Tabled constraints	Supported
Answer subsumption	Supported
XASP (Residual Program $\rightarrow$ Smodels)	"Almost" supported
Tabled negation	Partially Supported
Space reclamation	Partially Supported
Call subsumption	Not Supported
Incremental recomputation	Not Supported

- Unlike Concurrent Batched Evaluation, Concurrent Local Evaluation only supports LRD-stratified programs.
- Currently adjusting heuristics to coordinate behavior between threads

Goals for Multi-Threaded XSB

- Support multi-threading ISO/IEC DTR 13211–5:2007
  - *Almost complete*
- Any tabling function should be available to any active thread using tables that are private to a thread.
  - *Almost complete*
- Any tabling function should be available to any active thread using tables that are shared among threads.
  - *Partly achieved under Local Evaluation*
- Private tables should be highly scalable up to the number of cores available.
  - *Mostly achieved, at least for small number of cores*
- For problems that support both tabling and parallelism, shared tables should be able to provide speedup proportional to the number of cores available
  - *Mostly achieved under Local Evaluation, at least for small number of cores*
- MT-TLP can be used by others for real work
  - *Still trying, but parts ready for real work with implementor participation*