Incremental Tabling in Support of Knowledge Representation and Reasoning

Terrance Swift

July 19, 2014

Definition

Incremental Tabling [SR05, Sah06] ensures that tables correctly reflect changes in dynamic rules or facts.

Overview

1. Context: Tabling for KRR Systems
2. Previous Work: Manual Incremental Tabling
3. New Work: Transparent Incremental Tabling
4. Performance and Scalability Overview

Some Traditional Use Cases for Tabling

The majority of predicates are not tabled

- As an extension of Prolog
    - Parts of a Prolog program are tabled for termination, efficiency, or semantic support.
    - Parsers, graph search algorithms...
        - XSB, Inc's CDF-system uses tabling with stratified negation to efficiently traverse inheritance structures.
- As a means to implement specialized deduction
    - Tabled predicates implement inference rules as a module within a larger system
    - Process logics: CCS, π-calculus, Petri Nets
    - Temporal Logics: CCL, modal μ-calculus
    - Probabilistic reasoning: PITA, Problog, PRISM
- These use cases are neither completely distinct nor exhaustive

KRR Systems that use Tabling

- Description logics may be of high complexity (e.g., $\mathcal{ALC}$ and extensions, $\mathcal{SHOIQ}$); or low-complexity (e.g., $\mathcal{EL}$ or various flavors of DL-Lite).
- Logical rules also may be of high complexity (ASP); or of low complexity e.g., *Flora-2* (open-source), Silk (Vulcan, Inc), Ergo (Coherent Knowledge Systems)
- Silk and Ergo are extensions of *Flora-2*, and so are implemented using XSB and Tabling. Ergo supports
    - Lists and structures as with Prolog
    - Monotonic and non-monotonic inheritance;
    - Hilog
    - "Mix-ins" of defeasibility theories
    - Partial implementation of Transaction Logic
    - "Omni-rules" that permit Lloyd-Topor transformations in the body *and* head, and allow some existential reasoning

Uses of Ergo

- One of the main applications involves the automatic processing of text into rules
    - The sentence: *A contractile vacuole is inactive in an isotonic environment* [RUC+10] is translated to

```
forall(?x6)^contractile(vacuole)(?x6))
    ==> forall(?x9)^isotonic(environment)(?x9)
    ==> inactive(in(?x9))(?x6);
```

- Another is to use loosely-coordinated teams to construct knowledge bases

Pervasive Tabling

- *Flora-2*, Silk, and Ergo all make use of *Pervasive Tabling*: A user rule is tabled unless it is explicitly declared not tabled.
  - Rules that have side-effects should not be tabled
  - Facts are not tabled
  - Uses tabling with well-founded negation, attributed variables, call abstraction, answer abstraction (restraint) and table space reclamation
- Behavior of a computation differs greatly from Prolog and starts to resemble a deductive database.
  - Often, 10's of millions of tables, if domain is not well restricted.

The Need for Incremental Tabling in KRR Systems

- Would like to support easier interactive rule development – adding or deleting rules and/or facts
- Would like to support hypothetical reasoning (used in question answering)
- Would like to support use of Ergo, etc. in reactive systems

In short, want to make lots of things incrementally tabled!

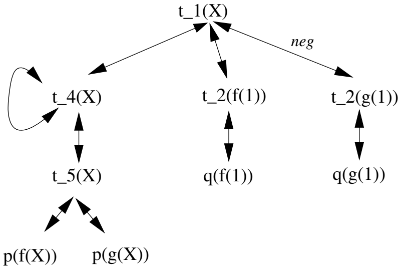
Manual Incremental Tabling

- Incremental Tabling [SR05, Sah06] provides for a table to be updated when a fact or rule upon which it depends is updated
- Used to support a deductive spreadsheet [RRW07]
- Relies on the notion of a dynamic *Incremental Dependency Graph* (IDG)
  - In the next slide arrows represent direct dependency
  - $Goal_1$ depends on $Goal_2$ iff $Goal_2$ affects $Goal_1$
  - A leaf node depends on no other node
- Descriptions of all algorithms are highly simplified. Exact algorithms are in the paper.

Incremental Dependency Graph (IDG)

```
:- table t_1/1, t_2/1, t_4/1, t_5/1 as incremental.
 t_1(X) :- t_4(X),tnot(t_2(X)).
t_4(X)  :- t_5(X).          t_4(X):- t_4(Y),p(X,Y).
t_2(X)  :- q(X).            t_5(X):- nt_1(X).
nt_1(X):- p(f(X)).          nt_1(X):- p(g(X)).

:- dynamic p/1, q/1 as incremental.
p(f(1)). q(f(1)).
```



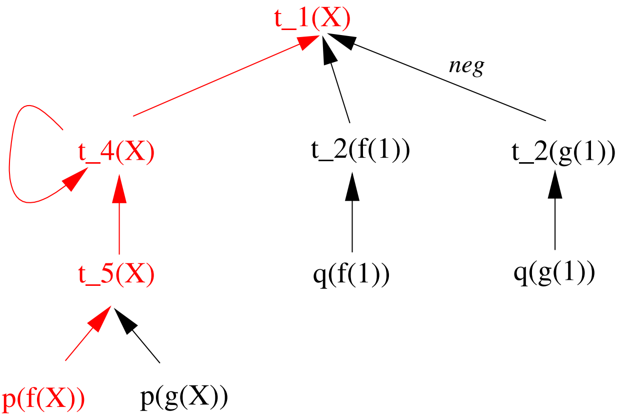
Manual Incremental Tabling: Invalidation

- *Invalid* means that a subgoal may not be correct given the current state of the program.
- Perform immediately after updating a dynamic incremental predicate
- In practice, a depth-first algorithm is used

```
/* Let A be the head of the clause that was updated */
Use the IDG to determine LeafSet, the set of leaf nodes that unify with $A$
Let SubgoalSet be the set of nodes that directly depend on some
                                                   $leaf \in LeafSet$
For each $S \in SubgoalSet$ until SubgoalSet is empty
    Increment S.invalid_children
    If S.invalid_children is now 1 /* S was made invalid */
‘v      Add S to a global InvalList
        Add to SubgoalSet all nodes that S directly affects
```

IDG Invalidation

- Suppose that *p(f(2))* were asserted. Then the invalidation phase would invalidate all nodes affected by the leaf *p(f(X))*.



Manual Incremental Tabling: *InvalList* Recomputation

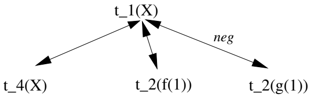
- The recomputation step makes subgoals valid again
- If $S.invalid\_children = 0$, this means that no tables or dynamic facts on which S depends have been changed by the update

---

```
/* The dependency partial order is preserved by InvalList */
Traverse InvalList and for each node $S$
    If $S.invalid\_children > 0$
        Recompute $S$, and set $S.invalid\_children = 0$
        If the extension of $S$ has changed
            For each node $S'$ that $S$ directly affects, decrement $S'.invalid\_chil$
            Recursively propagate the validity if $S'.invalid\_children$ is now 0
```

Manual Incremental Tabling: *InvalList* Recomputation

- Invalidation also sets an *invalid_children* field containing the number of immediate children that are currently invalid
- If this number is set to 0, a node does not need to be recomputed



Manual Incremental Tabling: Subgoal Recomputation

How to determine if the extension of a subgoal S has changed

```
Mark all answers for $S$ as deleted
Set $S.nbr\_new\_answers = 0$; set $S.new\_answer = false$
Whenever an answer $A$ is derived for $S$
    Increment $S.nbr\_new\_answers$
    If $A$ was already in the table remove the deleted mark
    Otherwise set $S.new\_answer = true$
When $S$ is completed remove deleted answers
```

If $S.new\_answer = false$ and $S.nbr\_answers = S.nbr\_new\_answers$ then the extension of S has *not* changed

Manual Incremental Tabling: Summary

- Incremental Tabling works at the subgoal level, with optimizations to reduce cost of graph traversal during the invalidation phase, and to avoid recomputations of goals whose *invalid_children* becomes 0.
- Because it works at a subgoal level and invalidation represents abstract "change" incremental update works
  - For both asserts and retracts
  - For both facts and rules
  - For positive and negative dependencies – as long as the program is stratified
- Invalidation immediately follows an assert or retract
- Recomputation can happen
  - Immediately after an assert or retract to a dynamic incremental predicate; or
  - May be invoked by a user command

Manual Incremental Tabling: Issues for KRR Systems

1. Works for stratified programs, but not for full WFS
2. Invoking recomputation is problematic
   - Immediately after an assert or retract is too inefficient in many cases
   - Using explict commands to invoke recomputation forces a "programming" burden on the KE, and allows invalid results to be derived
3. Assumes a programmer will only invoke recomputation when there are no choice points to incremental tables – no notion of view consistency
4. IDG can grow very large for some programs

Transparent Incremental Tabling: Support for WFS

- Atoms with a truth-value of *u* are represented in XSB as *conditional answers*, e.g., $p(a)$:- $tnot(q(b))|$.
- For propagation purposes the incremental update system needs to keep track of changes in truth value
- In stratified programs, only changes between *t* and *f* need to be maintained i.e., whether an answer has been added or not.
- For non-stratified programs, need to keep track of
    - *informational strenghening*: $u \Rrightarrow t$ or $u \Rrightarrow f$
    - *informational weakening*: $t \Rrightarrow u$ or $f \Rrightarrow u$
    - *truth strengthening*: $u \Rrightarrow t$
    - *truth weakening*: $u \Rrightarrow f$

Transparent Incremental Tabling: Support for WFS

The subgoal recomputation algorithm is changed as follows

```
Mark all answers for $S$ as deleted
Mark all unconditional answers for $S$ as unconditional
Set $S.nbr\_new\_answers = 0$; set $S.new\_answer = false$
Whenever an answer $A$ is derived for $S$
    Increment $S.nbr\_new\_answers$
    If $A$ was already in the table remove the deleted mark
    Else if $A.unconditional$ was false, but $A$ is now unconditional
        /* Informational strengthening $u \Rrightarrow t$ */
        $S.new\_answer = true$; invoke simplification
    Otherwise set $S.new\_answer = true$
After completion of $S$ traverse answers
    If $A.deleted = true$ and $A.unconditional = false$
        /* Informational strengthening $u \Rrightarrow f$ */
        $S.new\_answer = true$; invoke simplification
    If $A.unconditional = true$ and $A$ is now conditional
        /* Informational weakening $t \Rrightarrow u$ */
        $S.new\_answer = true$
```

Transparent Incremental Tabling: Support for WFS

Summary

- Changes for WFS need affect only the subgoal recomputation code
  - Propagate changes of truth values – additions or deletions of conditional answers that do not affect truth values does not spark propagation
- Strengthening w.r.t. truth order handled during recomputation; Weakening w.r.t. truth order handled in post-completion traversal
- Strengthening w.r.t. information order handled by simplification to maintain consistency of the residual program
- Changes are actually lighter-weight than may appear from slides (see paper)

Transparent Incremental Tabling Features

- WFS Support
- **Lazy Incremental Tabling** (avoids need for explicit command)
- View Consistency
- IDB Abstraction (reduces the size of IDBs)

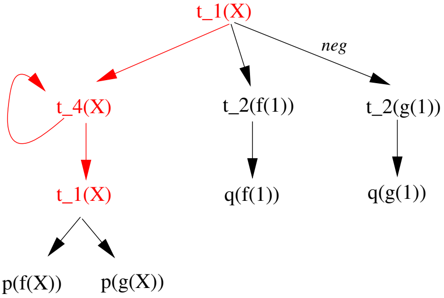
Lazy Incremental Tabling

Why not update table on demand? I.e., when calling a tabled subgoal S

```
If $S$ is (incremental and) invalid
    If $S.reeval\_ready = compute\_dependencies\_first$
        Set $S.re\_eval\_ready$ to $true$
        Construct $InvalList$ by traversing dependent nodes starting from $S$
        Call routine to incrementally update $InvalList$, with continuation $S$
```

IDG Invalidation

- If *t_1(X)* were called after the assert of *p(f(2))* in a previous slide, the dependency edges would be traversed to construct an *InvalList* that would give a bottom-up order of recomputation.



Lazy Incremental Tabling

- If $S.re\_eval\_ready = compute\_dependencies\_first$ the computation is interrupted to construct *InvalList* for S and recompute subgoals
- Later, when the continuation to S is taken, S will no longer be invalid and it will be safe to use its answers
- The interrupt mechanism is the same as that used for handling unifications to attributed variables; thread signalling, etc.
- Now, a new call to an incrementally tabled subgoal will *always* be correct — transparently
- Can be more efficient than manual approach
  - Avoids extra recomputations if multiple updates are made between calls to S
  - Avoids recomputation if S is never called again

Transparent Incremental Tabling Features

- WFS Support
- Lazy Incremental Tabling (avoids need for explicit command)
- **View Consistency**
- IDB Abstraction (reduces the size of IDBs)

Supporting View Consistency

- Suppose there are choicepoints into a completed incremental table S and S is updated. What about these choicepoints
- Previous version didn't handle this ("core-dump" semantics)
- Thinking as a deductive database, these choicepoints are similar to cursors traversing a materialized view
- Want to ensure view consistency for choicepoints into an updated table
- These choice points are called OCCPs – Open Cursor Choice Points

Supporting View Consistency

- View consistency should impose no significant overhead on the speed of non-incremental tables, or on incremental tables when there are no OCCPs
- First, keep track of the number of OCCPs to a completed incremental subgoal S
    - Increment number when calling the completed subgoal S
    - Decrement the number on failure. cuts and throws

Supporting View Consistency

```
When an invalid incremental subgoal $S$ is about to be recomputed
If there are OCCPs
        Find each such OCCP $C_{OCCP}$ in the CP stack
        Copy the unconsumed answers for $C_{OCCP}$ from the table
                                                  to a list in the heap
        Alter $C_{OCCP}$ so that it has a new instruction
                                                  and protects the used heap space
```

- All of this is done in C, so its reasonably fast, although it may require a lot of heap space if tables are large or there are a large number of OCCPs
- Once an OCCP has been altered, it is protected and need not be considered by further updates – you pay the price once per OCCP

Transparent Incremental Tabling Features

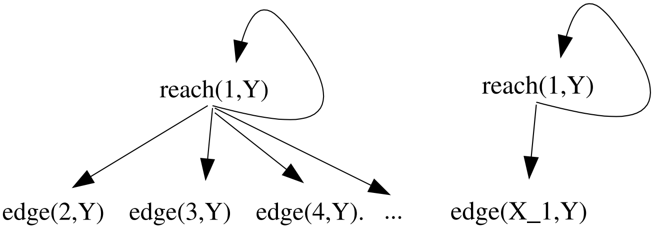
- WFS Support
- Lazy Incremental Tabling (avoids need for explicit command)
- View Consistency
- **IDB Abstraction** (reduces the size of IDBs)

IDG Abstraction

- What if you want to use incremental tabling always and everywhere. Is that feasible?
- If there are no updates, the main overhead of incremental tabling w.r.t. non-incremental tabling is maintenance of the IDG
- Sometimes we need to abstract what is kept in the IDG.
  - This is different than subgoal abstraction as it does not affect indexing or what is maintained in the table, just the IDG
- *:- dynamic edge/2 as incremental, abstract(0)*.
- Consider the following program

IDG Abstraction

:- table reach/2 as incremental.
:- dynamic edge/2 as incremental.
reach(X,Y):- edge(X,Y).
reach(X,Y):- reach(X,Z),edge(Z,Y).



- Left side without IDG abstraction; Right side with IDG abstraction

Performance Summary: Recursions

- As a first benchmark the overhead of incremental tabling over tabling was tested for left-linear recursion on randomly generated graphs
    - Without IDG abstraction: 50% overhead for time; 200% overhead for space
    - With IDG abstraction: essentially no overhead for time; 28% overhead for space
- For 3-valued recursion time overheads were similar
    - Without IDG abstraction, 66% overhead for space
    - With IDG abstraction, less than 10% overhead for space

Performance Summary: (Pseudo-)KRR Benchmarks

- A pseudo-KRR program was evaluated. This used stratified negation, but its main computational issue was its use of equality between constants and functional terms (similar to a description logic).
- From an implementation perspective, the KRR program used tabled negation, the *u* truth value for answer abstraction [GS13]. and subgoal abstraction [RS14]
- EDBs from 10,000 facts to 10,000,000 facts were tested
- Tests showed invalidation did not take a significant amount of time
  - The larger IDGs contained up to 750 million edges during the invalidation phase
  - After recomputation, the IDGs contained over 1 billion edges
- Recomputation time depended on whether the search space was expanded (i.e., if additional EDB facts added many new answers)

Summary

- All features described are in version 3.5 of XSB
- More engineering work on incremental tabling would be useful: for instance to integrate it with call subsumption, which can also be useful for KRR applications.
- Work is needed to help decide when, e.g. IDG abstraction will be useful.
- Can incremental tabling be *adaptive*? Can it perform IDG abstraction dynamically during a computation if it detects that the IDG space is growing to fast?
- Focus needs to be on fully integrated tools rather than on research prototypes

References I

[GS13] B. Grosof and T. Swift.
Radial restraint: A semantically clean approach to bounded rationality for logic programs.
In *American Association for Artificial Intelligence Press*, 2013.

[RRW07] C.R. Ramakrishnan, I.V. Ramakrishnan, and David S. Warren.
XcelLog: A deductive spreadsheet system.
Knowledge Engineering Review, 22(3):269–279, 2007.

[RS14] F. Riguzzi and T. Swift.
Terminating evaluation of logic programs with finite three-valued models.
ACM Trans. on Computational Logic, 2014.
To Appear.

[RUC+10] J. Reece, L. Urry, M. Cain, S. Wasserman, P. Minorsky, and R. Jackson.
Campbell Biology.
B. Cummings, 2010.
9th Edition.

[Sah06] D. Saha.
Incremental Evaluation of Tabled Logic Programs.
PhD thesis, SUNY Stony Brook, 2006.

References II

[SR05] D. Saha and C.R. Ramakrishnan.
Incemental and demand-driven points-to analysis using logic programming.
In *Principles and Practice of Decl. Prog.*, pages 117–128, 2005.