Handling Incomplete and Complete Tables in Tabled Logic Programs

Ricardo Rocha
DCC-FC & LIACC
University of Porto, Portugal
ricroc@ncc.up.pt

ICLP 2006, Seattle, Washington, USA, August 2006

Motivation

- This work was motivated by our recent attempt of **applying tabling to Inductive Logic Programming (ILP)** [Rocha *et al.*, ECML'05].
- ILP applications are an excellent case study for tabling because they have **huge search spaces** and do a **lot of re-computation**.
- In particular, in this work we focus on the table space and how to **efficiently handle incomplete and complete tables**.

Tabling and ILP: Incomplete Tables

- Tabling is about storing answers for subgoals so that they can be reused when a repeated call appears.
- On the other hand, most ILP algorithms are interested in query satisfiability, not in the answers: query evaluation stops as soon as an answer is found. This is usually implemented by **pruning** at the Prolog level.

Tabling and ILP: Incomplete Tables

- Tabling is about storing answers for subgoals so that they can be reused when a repeated call appears.
- On the other hand, most ILP algorithms are interested in query satisfiability, not in the answers: query evaluation stops as soon as an answer is found. This is usually implemented by **pruning** at the Prolog level.
  - Pruning over tabled computations results in **incomplete tables**: we may have found several answers but not the complete set.

Tabling and ILP: Incomplete Tables

- Tabling is about storing answers for subgoals so that they can be reused when a repeated call appears.
- On the other hand, most ILP algorithms are interested in query satisfiability, not in the answers: query evaluation stops as soon as an answer is found. This is usually implemented by **pruning** at the Prolog level.
  - Pruning over tabled computations results in **incomplete tables**: we may have found several answers but not the complete set.
  - Thus, when a repeated call appears we cannot simply trust the answers from an incomplete table because we may loose part of the computation.

Tabling and ILP: Incomplete Tables

- Tabling is about storing answers for subgoals so that they can be reused when a repeated call appears.
- On the other hand, most ILP algorithms are interested in query satisfiability, not in the answers: query evaluation stops as soon as an answer is found. This is usually implemented by **pruning** at the Prolog level.
  - Pruning over tabled computations results in **incomplete tables**: we may have found several answers but not the complete set.
  - Thus, when a repeated call appears we cannot simply trust the answers from an incomplete table because we may loose part of the computation.
  - A common approach is to **throw away** the already found answers and restart the evaluation from the beginning when a repeated call appears.

Incomplete Tables: Our Approach

- **Basic Idea**
  - By default, we keep incomplete tables for pruned subgoals.
  - Then, when a repeated call appears, we start by consuming the available answers from its incomplete table.
  - If the table is exhausted, then we restart the evaluation from the beginning.

Incomplete Tables: Our Approach

➤ Basic Idea

- ♦ By default, we keep incomplete tables for pruned subgoals.
- ♦ Then, when a repeated call appears, we start by consuming the available answers from its incomplete table.
- ♦ If the table is exhausted, then we restart the evaluation from the beginning.

➤ Main Goal

- ♦ **Avoid re-computation** when the answers in an incomplete table are enough to evaluate repeated calls.

Tabling and ILP: Complete Tables

- When we use tabling for applications that build **very many or very large tables**, we can quickly **run out of memory**.
- A common approach is to have a set of primitives that the programmer can use to dynamically abolish some of the tables.

Tabling and ILP: Complete Tables

- When we use tabling for applications that build **very many or very large tables**, we can quickly **run out of memory**.
- A common approach is to have a set of primitives that the programmer can use to dynamically abolish some of the tables.
  - However, this can be hard to use and very difficult to decide what are the **potentially useless tables** that should be deleted.

Complete Tables: Our Approach

➤ **Basic Idea**

- A memory management strategy based on a least recently used algorithm, that **dynamically recovers space** from the **least recently used tables** when the system runs out of memory.

Complete Tables: Our Approach

➤ **Basic Idea**

♦ A memory management strategy based on a least recently used algorithm, that **dynamically recovers space** from the **least recently used tables** when the system runs out of memory.

➤ **Main Goal**

♦ The programmer can still force the deletion of particular tables, but can also rely on the effectiveness of our memory management algorithm to completely avoid the problem of deciding what potentially useless tables should be deleted.

Concluding Remarks

- Our proposals have been implemented in the **YapTab tabling system** with minor changes to the original design.
- Preliminaries results using the April ILP system showed very substantial performance gains and a substantial increase of the size of the problems that can be solved by combining ILP with tabling.
- The problems and proposals presented in this work are not restricted to ILP applications and can be generalised and applied to any other application.