

Take Home Exam for Integrated Logic Systems: Advanced WAM Module

May 26, 2008

The rules for this take-home test are as follows. Although the slides should be sufficient to answer all questions, you can consult any written or web-based material you like. However you may not ask questions of anyone except Prof. Swift, and you may not consult with each other. Failure to observe these rules will unconditionally result in a grade of 0 for this module.

Exams must be emailed to `tswift@cs.sunysb.edu` no later than June 1 2008, 24:00. PDF files are preferred, but not required.

Updated slides are at <http://www.cs.sunysb.edu: tswift/webtalks/ils0.pdf>

MT 2 points In the multi-threaded server on page 184 explain the use of `catch/3` in `attend_client/1`.

Theory 2 points Show a simple program containing negation that is Predicate Stratified. Show a simple program that is Locally Stratified but not Predicate Stratified.

Tabling 2 points Present a list of terms such that if the terms were inserted into an answer trie, the trie would contain the table instructions `trie_retry_structure`, and `trie_do_value` (among other instructions).

Tabling 2 points Suppose a program has a single clause for a predicate, `p/n`. If `p/n` is not tabled (i.e. is executed with Prolog) no choice point instruction will be needed. However, in the SLGWAM, if `p/n` is tabled, the initial tabling instruction, such as `tabletrysingle` must create a choice point. Explain why.

Tabling 3 points Write a program and query

- (a) which does not terminate unless tabling is used; and
- (b) for which call subsumption gives a speedup in asymptotic complexity over call variance

Indicate a program for which the behavior of call variance is more useful than for call subsumption (Hint: consider a meta-interpreter).

Simple experiments indicate that in XSB call subsumption imposes an overhead about 25-30% for a tabled evaluation that encounters no subsuming calls (i.e. one for which call subsumption is not helpful). Discuss the advantages and disadvantages of making variance the "default" tabling strategy.

MT 2 points As discussed, many open-source Prologs, including XSB, YAP, SWI, and Ciao implement multi-threading. Another language, Erlang, is based partly on Prolog but has a different approach to concurrency and parallelism. An Erlang instance can create other Erlang instances, but each Erlang is a different process, and may be on a different machine. Erlang processes communicate through message queues, but these queues are based on sockets rather than on memory shared by threads.

Discuss advantages and disadvantages of a multi-process approach such as Erlang's as compared to that of Prolog multi-threading.

Tabling 3 points As discussed, XSB, YAP, Ciao and other Prologs implement tabling based on tries of some sort. Binary Decision Diagrams, BDDs, are often used for model-checking. See, e.g.

http://en.wikipedia.org/wiki/Binary_decision_diagram

There's been a lot of work done on BDDs, but we need to only consider a couple of their properties. First, BDDs can be generalized to have an arbitrary number of leaf-nodes rather than just 0 and 1 leaf nodes. From our perspective, what is interesting about BDDs is that they store elements using a DAG rather than using a tree (as tries do).

What about using a BDD-like structure to store tables? Discuss advantages and disadvantages of a BDD-like structure compared to tries in terms of the tabling operations discussed in class, space utilization and any other factors.

[harder] For full credit, discuss the applicability, if any, of variable ordering in BDDs for storing terms in tables.

Tabling 4 points Prove the following theorem using the definitions below and any definitions in the slides. (2.5 points)

Theorem 0.1 *Let $\mathcal{E} = \mathcal{F}_0, \dots, \mathcal{F}_n$ be a Local SLG Evaluation of an atomic query Q to a definite program P where n is finite. Prove that for any forest $\mathcal{F}_i, 0 \leq i \leq n$, $SDG(\mathcal{F}_i)$ will have a single maximal independent SCC.*

Hint: use induction on \mathcal{E} and consider cases for each SLG operation.

(1.5 points) Note that even if you could not prove the theorem you should be able to use the theorem to address the following.

Give a clear and convincing argument (not a formal proof) that for a local evaluation of a definite program, the algorithm for completion on pages 91 and following will have no trapped SCCs on the stacks – in other words, at a state \mathcal{F}_i of \mathcal{E} , when the engine backtracks into a leader choice point and determines that all ANSWER RESOLUTION, PROGRAM CLAUSE RESOLUTION or NEW SUBGOAL operations have been performed for subgoals in the ASCC, then the ASCC will exactly match a SCC of $SDG(F)$.

Here are some definitions for the proof. Most are in the slides, but some of the ones below are updated. In particular, make sure to use the (restricted) definition of completely evaluated given below.

Definition 0.1 (SLG Trees and Forest) An SLG forest consists of a forest (set) of SLG trees. Nodes of SLG trees have the form:

$$\text{Answer_Template} \text{ :- Goal_List}$$

The *Answer_Template* is an atom, and *Goal_List* is a sequence of literals. The selected atoms in a forest \mathcal{F} are the set of atoms A such that A is a selected literal of some node of a tree in \mathcal{F} . An SLG tree may be marked with the token *complete*,

A node N is an answer when it is a leaf node for which *Goal_List* is empty.

Definition 0.2 (Completely Evaluated) A set \mathcal{S} of subgoals in a forest \mathcal{F} is completely evaluated if at least one of the conditions holds for each $S \in \mathcal{S}$

- (a) For each node N in the tree for S :
 - i. The selected literal L_S of N is completed or in \mathcal{S} ; or
 - ii. There are no applicable `NEW SUBGOAL`, `PROGRAM CLAUSE RESOLUTION` or `ANSWER_RETURN` operations for N .

Definition 0.3 (SLG Operations) Given a forest \mathcal{F}_n of a SLG evaluation of program P and query Q , \mathcal{F}_{n+1} may be produced by one of the following operations.

- (a) `NEW SUBGOAL`: Let \mathcal{F}_n contain a non-root node

$$N = \text{Ans} \text{ :- } S, \text{Goal_List}$$

Assume \mathcal{F}_n contains no tree with root subgoal S . Then add the tree $S \text{ :- } |S$ to \mathcal{F}_n .

- (b) `PROGRAM CLAUSE RESOLUTION`: Let \mathcal{F}_n contain a root node $N = S \text{ :- } |S$ and C be a program clause $\text{Head} \text{ :- Body}$ such that Head unifies with S with mgu θ . Assume that in \mathcal{F}_n , N does not have a child $N_{\text{child}} = (S \text{ :- } |Body)\theta$. Then add N_{child} as a child of N .
- (c) `ANSWER_RETURN`: Let \mathcal{F}_n contain a non-root node N whose selected literal S is positive. Let Ans be an answer node for S in \mathcal{F}_n and N_{child} be the resolvent of N and Ans on S . Assume that in \mathcal{F}_n , N does not have a child N_{child} . Then add N_{child} as a child of N .
- (d) `COMPLETION`: Given a completely evaluated set \mathcal{S} of subgoals (Definition 0.2), mark the trees for all subgoals in \mathcal{S} as completed.

Definition 0.4 (SLG Evaluation) Given a program P , and an atomic query Q an SLG evaluation \mathcal{E} is a sequence of SLG forests $\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_\beta$, such that:

- \mathcal{F}_0 is the forest containing a single tree $Q \text{ :- } |Q$
- For each successor ordinal, $n + 1 \leq \beta$, \mathcal{F}_{n+1} is obtained from \mathcal{F}_n by an application of an SLG operation.

If no operation is applicable to \mathcal{F}_β , \mathcal{F}_β is called a final forest of \mathcal{E} . If \mathcal{F}_β contains a leaf node with a non-ground selected negative literal, it is floundered.

Definition 0.5 (Subgoal Dependency Graph) Let \mathcal{F} be a forest, and let $S_1 :- |S_1$ root of a non-completed tree in \mathcal{F} . S_1 directly depends on a non-completed subgoal S_2 in \mathcal{F} iff either S_2 or $\text{not}(S_2)$ is the selected literal of some node in the tree for S_1 in \mathcal{F} . If S_2 is the selected literal of some node in the tree for S_1 in \mathcal{F} , then S_1 directly depends on S_2 . The Subgoal Dependency Graph of \mathcal{F} $\text{SDG}(\mathcal{F}) = (V, E)$ of \mathcal{F} is a directed graph in which $(S_i, S_j) \in E$ iff subgoal S_i directly depends on subgoal S_j , and V is the underlying set of E .

Definition 0.6 (Independent SCC) A strongly connected component SCC is independent if $\forall S \in \text{SCC}$, if S depends on some S' , then $S' \in \text{SCC}$.

Definition 0.7 (Locality property) Let \mathcal{E} be a finitely terminating SLG Evaluation of a query Q against a definite program P , with forests $\mathcal{F}_0, \dots, \mathcal{F}_n$. Let \mathcal{F}_i be a forest in \mathcal{E} in which an ANSWER RESOLUTION , $\text{PROGRAM CLAUSE RESOLUTION}$ or NEW SUBGOAL operation is applied to a node N in \mathcal{F}_i to produce \mathcal{F}_{i+1} . If N is in an independent SCC of $\text{SDG}(\mathcal{F}_i)$ for any such \mathcal{F}_i in \mathcal{E} , then \mathcal{E} has the locality property.