CS345A Notes for Lecture 5/4/94

The NAIL System

To get a temporary account suitable to run NAIL:

1. Fill out an account form at the desk of MJH. Ask for an account on theory, for which I will sign.

2. Once on theory, look at

   `/u/morisita/nail/README`

   which will explain how to set up your directory to run NAIL.

- Please do not abuse theory account; use it only to experiment with NAIL.

- Manual will be posted to the newsgroup.

Structure of NAIL

1. GLUE: logical rules plus conventional flow-of-control, procedures, etc.

2. NAIL: conventional logic front end. Translated via magic-sets and well-founded negation to GLUE procedures.

3. IGLUE: back end. Adaptive data structures (e.g., indexes).

- Principal credits: Geoff Phipps (GLUE), Marcia Derr (IGLUE), Shinichi Morishita (current NAIL), Kate Morris (old NAIL).

GLUE Assignments

- Assignment statements look like rules, but *they are executed whenever, and only when, they are reached during program execution.*

- Four types of assignment: Clear-and-assign, insert, delete, update.

- Clear-and-assign represented by `:=`

Example :

```
p(X,Y) := q(X,Z) & r(Z,Y).
```

- Note period at end. Semicolon is an OK substitute.
- Means $p := \pi_{1,3}(q \bowtie r)$.
- Executed as written whenever control flow reaches this statement.

Other Assignment Operators

- Insert represented by `+=`

Example :

```
p(X,Y) += q(X,Z) & r(Z,Y).
```

means $p := p \cup \pi_{1,3}(q \bowtie r)$.

- Delete represented by `-=`

Example :

```
p(X,Y) -= q(X,Y).
```

means $p := p - q$.

- Modify represented by an assignment operator `+=[Z]` for some set of variables Z. See manual.

Three Types of Subgoals

GLUE rules can have three types of subgoals.

1. EDB: stored physically in a subdirectory called `edb`.
2. IDB: defined by NAIL rules included with the GLUE program.
3. Procedural. These subgoals have their value defined by a GLUE procedure. GLUE procedures have input and output arguments, and the input arguments must be bound when this subgoal is reached in a left-to-right evaluation of the GLUE statement.

Control Flow Statements

While-statement:

while <condition> **do** <statements> **end**;

- Condition is a rule body.
- Its value is "true" iff the body is nonempty.
- Similar idea: repeat, loop, if-then-else statements.

Example : See lines (12)–(17) of Fig. 1. This program performs seminaive evaluation for the rules

```
path(X,Y) :- arc(X,Y)
path(X,Y) :- path(X,Z) & arc(Z,Y)
```

- It is not necessary to write this program in GLUE; the NAIL compiler produces it automatically.
- Line (12): On round 1, the new paths are the arcs.
- Line (13): Initially, *path* is empty.
- Line (14): Go around the loop until there are no new paths discovered on some round.
- Line (15): Copy the new paths into the full set of paths.
- Line (16): Compute "new new" paths by extending only the new paths by one more arc.
- Line (17): Throw away from the new new paths those paths that are actually old.

Module Structure

module <name>;
export, import, EDB declarations
GLUE procedures and NAIL declarations
end

Procedure Structure

> **proc** <name>(input args : output args)
> local declarations
> statements
> **end**

- **prog** for **proc** in distinguished procedure that executes when module is invoked.

Example : In Fig. 1:

- Lines (1) and (20): module–end.
- Line (2): export/import information.
- Line (3): EDB information.
- Lines (4)–(19): program `main`. This procedure has no input or output arguments.
- Line (5): declaration of local relation $newPath$.
- Lines (6)–(11): initialize arc relation.
- Lines (12)–(18): seminaive evaluation.

Rule Processing in GLUE

Left-to-right evaluation; uses supplementary relations at each subgoal. Three kinds of subgoals:

1. EDB: join with supplementary relation.
2. IDB: In the compiling phase NAIL determines the binding pattern for this subgoal and generates a GLUE procedure that implements magic-sets and seminaive evaluation.
3. GLUE procedure:

- Procedure call must "fit," i.e., take as input arguments the bound arguments of the subgoal and return a relation over all the arguments, including input arguments.
- Special relations used in procedure declaration:

  in(proc-name(input args))

to receive input and

return(input args : output args)

to return a relation.

Example : See Fig. 2: a nim-playing module.

- Nim rules: Start with hash marks in rows of 7, 5, 3. In turn, take any number ≥ 1 from one of the three rows. You win by taking the last mark, i.e., "stalemating" your opponent.
- Boards are represented by triples (i, j, k), where $0 \leq i \leq 7$, $0 \leq j \leq 5$, and $0 \leq k \leq 3$.
- EDB relation lt was created by another module and contains "less-than" facts for integers up to 10. It is needed for safety of the rules.
- Lines (4)–(6): the entire "win" program.
- Line (7): IDB predicate $interesting$, true of those triples that represent nim boards.
- Lines (8)–(10): legal moves. Any of the three rows can be decreased by one or more, but not below 0.
- Line (11): The usual "win" rule, with w as the predicate.
- Note ! is the negation symbol.
- Line (5) is executed by first invoking a compiled procedure for $interesting^{fff}$. The result is the entire $interesting$ relation.
- Then, a compiled procedure for w^{bbb} is called; it computes the well-founded model of w, intersected with the interesting boards (those that are legal nim positions).
- win is computed by intersecting the two returned relations.