Efficient Proof–Programming

Aleksey Nogin

February 7, 2000

Overview

- What I mean by proof–programming
- Overview of the NuPRL Automata Library
- General principles of efficient proof–programming
- Sources of computational inefficiencies in the old Automata Library
- Applying the general principles to make NuPRL Automata Library efficient

Overview of the Automata Library

- `http://nuprlauto-b.nogin.org/`, `http://nuprlauto.nogin.org/`
- Finite Sets theory
  - $Fin(s) \ == \ \exists n:\mathbb{N}.\ \exists f:\mathbb{N}n \to s.\ Bij(\mathbb{N}n; s; f)$
  - Basic properties of finite types and lists of elements of finite types. $Fin(S) \Rightarrow Fin(\{x : S\,\text{List} \mid \|x\| = n\})$
  - Pigeon-hole principle
- Language theory
  - $L(A) \ == \ A\,\text{List} \to \mathbb{P}$
  - Language operations — intersection, union, product and complement
- Action Sets theory
  - $ActionSet(T) \ == \ car : \mathbb{U} \times (T \to car \to car)$
  - $(S : L \leftarrow s) \ ==_r \ \texttt{if}\ null(L)\ \texttt{then}\ s$
    $\texttt{else}\ (S.act\, hd(L)\, (S : tl(L) \leftarrow s))\ \texttt{fi}$
  - Pumping lemma

Overview of the Automata Library (cont.)

- Deterministic Automata theory
  - $Automata(Alph; States) \ ==$
    $act : (States \to Alph \to States) \times init : States \times (States \to \mathbb{B})$
  - $DA(l) \ ==_r \ \texttt{if}\ null(l)\ \texttt{then}\ I(DA)$
    $\texttt{else}\ ((\delta DA)\ DA(tl(l))\ hd(l))\ \texttt{fi}$
    $DA(l)\downarrow \ == \ F(DA)\ DA(l)$
  - reach_dec theorem
- Myhill–Nerode Theorem theory
  - Rl relation: $x\, Rl\, y$ $\texttt{iff}$ $\forall z : A\,\text{List}.\ L(z@x) \Leftrightarrow L(z@y)$
  - Rg relation — same as Rl for $L(x) =\uparrow g(x)$
  - mn_23_lem_1 lemma: for any equivalence relation R such that $\forall x, y, z : Alph\,\text{List}.\ x\,R\,y \Rightarrow (z@x)R(z@y)$ and $Fin(x, y : (Alph\,\text{List})//(x\,R\,y))$, for any $g \in (x, y : (Alph\,\text{List})//(x\,R\,y)) \to \mathbb{B}$, Rg is decidable.
  - The rest of the Myhill–Nerode theorem.

General Principles of Efficient Proof–Programming

- Use "expensive" statements non-computationally
- Induction steps are the main source of computational complexity
- Turn loop invariants into induction statements
- Use existential quantifiers as memory
- Use lists as memory

Sources of Computational Inefficiencies in the Old Automata Library

- Pigeon-Hole Principle
- Decidability of State Reachablity
- MN_23_LEM_1 Lemma

Pigeon-Hole Principle: Inefficient Proof

$\forall n : \{1...\}.\ \forall f : \mathbb{N}(n+1) \to \mathbb{N}n.$
$\quad \exists i : \mathbb{N}(n+1).\ \exists j : \{(i+1)..(n+1)^-\}.\ fi = fj$

- *Base of induction*: $n = 1$, $f(1) = f(0) = 0$
- *Induction step*: If $\exists k : \mathbb{N}n. f(n) = f(k)$, then we are done. If not, take $g = \lambda x\,.\,\texttt{if}\ (f(x) = n-1)\ \texttt{then}\ f(n)\ \texttt{else}\ f(x)\ \texttt{fi}$ and apply induction hypothesis.

Pigeon-Hole Principle: Efficient Proof

$\forall n : \{1...\}.\ \forall f : \mathbb{N}(n+1) \to \mathbb{N}n.\ \exists i : \mathbb{N}(n+1).\ \exists j : \mathbb{N}i.\ fi = fj$

- *Base of induction*: $n = 1$, $f(1) = f(0) = 0$
- *Induction step*: **Algorithm**: check i's from n down to 1 and for each i, the j's from $i-1$ down to 0. **Invariant** (loop over i): we have not found the correct i, j yet and such a pair still exists ahead of us:

  $\forall iii : \mathbb{N}(n+1).\ \forall ii : \{(iii+1)..(n+1)^-\}.\ \forall jj : \mathbb{N}ii.\ \neg(fii = fjj)) \Rightarrow$
  $\quad (\exists i : \mathbb{N}(iii+1).\ \exists j : \mathbb{N}i.\ fi = fj)$
- *Prooving it by induction - base case*: $iii = 0$ — we searched all i and failed
- *Induction step* (using *reverce engeneering*): Check whether $\exists jj : \{0..iii^-\}.(f\,ii) = (f\,jjj)$. If not, use IH.

Efficient Proof–Programming

Part II

Aleksey Nogin

February 14, 2000

Pigeon-Hole Principle — again

$$\begin{array}{l}\forall n : \{1 ...\}.\ \forall f : \mathbb{N}(n+1) \to \mathbb{N}n. \\ \quad \exists i : \mathbb{N}(n+1).\ \exists j : \{(i+1)..(n+1)^{-}\}.\ f i = f j\end{array}$$

Better(?) way:

$$\begin{array}{l}\forall n : \mathbb{N}. \forall P : \mathbb{Z} \to \mathbb{P}.\ \forall f : \mathbb{N}(n+1) \to \{x : \mathbb{Z} | P\, x\}.\ \#(\{x : \mathbb{Z} | P\, x\}) = n \Rightarrow \\ \quad (\exists i : \mathbb{N}(n+1).\ \exists j : \mathbb{N}i.\ f i = f j)\end{array}$$

State Reachability

$$\forall Alph, St : \mathbb{U}.\ \forall Auto : Automata(Alph; St).$$
$$Fin(Alph) \Rightarrow Fin(St) \Rightarrow \forall s : St.\ Dec(\exists w : Alph\,\textsf{List}.\ Auto(w) = s)$$

State Reachability: Efficient Proof

- Instead of proving for automata, we'll prove for arbitrary action sets
- Will prove existence of the list of all reachable states:

$$\forall Alph : \mathbb{U}.\ \forall S : ActionSet(Alph).\ \forall si : S.car.$$
$$Fin(S.car) \Rightarrow Fin(Alph) \Rightarrow (\exists RL : S.car\,\textsf{List}\ \forall s : S.car.$$
$$(\exists w : Alph\,\textsf{List}.\ (S : w \leftarrow si) = s) \Leftrightarrow mem\_f(S.car; s; RL))$$

- Algorithm ideas:
  - Keep all the states we already know to be reachable in a list RL
  - For each state s from RL to go over all the letters of the alphabet to get all the states immediately reachable from s
  - For efficiency, only compute $S.act\ s\ \alpha$ once for any pair s, α

State Reachability: LOOP INVARIANT

After adding n states to RL (putting the initial state si into RL does not count as a step) at least one of the following two statements is true:

1. RL consists exactly of all reachable (from si) elements of $S.car$
2. RL consists of $n+1$ distinct reachable elements of $S.car$ and if we go over the first k letters of the alphabet, we can have the RLa with the following three properties:
   (a) all elements of $S.car$ immediately reachable from the elements of RL (other than its head, which was just added) are either in RL or in RLa (possibly both)
   (b) for any letter a out of the first k letters of the alphabet, the element $(S.act\,a\,hd(RL))$ should appear in either RL or RLa (or both)
   (c) all elements of RLa are reachable (from si)

State Reachability: FORMAL LOOP INVARIANT

$$
\begin{array}{l}
\forall Alph : \mathbb{U}.\ \forall S : ActionSet(Alph).\ \forall si : S.car. \\
\forall nn : \mathbb{N}.\ \forall f : \mathbb{N}nn \to Alph.\ \forall g : Alph \to \mathbb{N}nn. \\
\ \ Fin(S.car) \Rightarrow InvFuns(\mathbb{N}nn; Alph; f; g) \Rightarrow (\forall n : \mathbb{N} \\
\ \ \ \ \exists RL : \{y : \{x : S.car\ \mathsf{List} \mid 0 < ||x|| \wedge ||x|| \le n+1\} \mid y[(||y||-1)] = si\} \\
\ \ \ \ \ \ (\forall s : S.car.\ (\exists w : Alph\ \mathsf{List}.\ (S : w \leftarrow si) = s) \Leftrightarrow mem\_f(S.car; s; RL)) \\
\ \ \ \ \vee (||RL|| = n+1 \wedge (\forall i : \mathbb{N}||RL||.\ \forall j : \mathbb{N}i.\ \neg(RL[i] = RL[j])) \\
\ \ \ \ \ \wedge (\forall s : S.car.\ mem\_f(S.car; s; RL) \Rightarrow (\exists w : Alph\ \mathsf{List}.\ (S : w \leftarrow si) = s)) \\
\ \ \ \ \ \wedge (\forall k : \mathbb{N}.\ k \le nn \Rightarrow (\exists RLa : S.car\ \mathsf{List} \\
\ \ \ \ \ \ \ \ (\forall i : \{1..||RL||^{-}\}.\ \forall a : Alph. \\
\ \ \ \ \ \ \ \ \ \ mem\_f(S.car; S.act\,a\,RL[i]; RL) \vee mem\_f(S.car; S.act\,a\,RL[i]; RLa)) \\
\ \ \ \ \ \ \ \ \ \ \wedge (\forall a : Alph.\ ga < k \Rightarrow mem\_f(S.car; S.act\,a\,hd(RL); RL) \\
\ \ \ \ \ \ \ \ \ \ \ \ \vee mem\_f(S.car; S.act\,a\,hd(RL); RLa)) \\
\ \ \ \ \ \ \ \ \ \ \wedge (\forall s : S.car.\ mem\_f(S.car; s; RLa) \Rightarrow \\
\ \ \ \ \ \ \ \ \ \ \ \ (\exists w : Alph\ \mathsf{List}.\ (S : w \leftarrow si) = s))))))
\end{array}
$$

State Reachability: Reverse Engeneering the Proof

- We start (base case, $n = 0$) with si as the only element of RL.
  - We take empty RLa (base case, $k = 0$)
  - Induction step (from $k-1$ to k): add $S.act\,(f\,(k-1))\,si$ to RLa
- Induction step:
  - Take elements from RLa (list induction) and check whether it is already in RL until either we've found some element s in RLa but not in RL or RLa becomes empty.
    * If RLa becomes empty, then we can prove (by list induction on w) that statement (1) holds, so we are done.
    * If we've found such an s, then we add it to the top of RL and then we take the rest of RLa as a new RLa for $k = 0$ and then start a cycle (induction) for k from 1 up to nn adding $S.act\,(f\,(k-1))\,s$ to RLa on each step.

Decidability of the Automata Relation

$$
\begin{aligned}
&\forall Alph : \mathbb{U}.\ \forall R : Alph\,\mathsf{List} \to Alph\,\mathsf{List} \to \mathbb{P} \\
&\quad Fin(Alph) \Rightarrow EquivRel(Alph\,\mathsf{List}; x, y. x\,R\,y) \\
&\quad \Rightarrow\ Fin(x, y : (Alph\,\mathsf{List})//(x\,R\,y)) \\
&\quad \Rightarrow\ (\forall x, y, z : Alph\,\mathsf{List}.\ x\,R\,y\ \Rightarrow\ (z\,@\,x)\,R\,(z\,@\,y)) \\
&\quad \Rightarrow\ (\forall g : x, y : (Alph\,\mathsf{List})//(x\,R\,y) \to \mathbb{B}.\ \forall x, y : x, y : (Alph\,\mathsf{List})//(x\,R\,y) \\
&\qquad Dec(x\,\mathsf{R}g\,y))
\end{aligned}
$$