"I think you should be more explicit here in step two."
Why Theorem Prooving

- We want to make sure algorithms (and their implementations) are correct.
Why Theorem Proooving

- We want to make sure algorithms (and their implementations) are correct.
- Ideally we develop the algorithm and the proof of its correctness concurrently.

Nice example about regular expression matching:

'Proof-directed debugging' revisited for a first-order version by Kwangkeun Yi.

This is based on an earlier paper by Robert Harper.
Languages

\textbf{definition}

\texttt{lang\_seq} :: "\texttt{string set} \Rightarrow \texttt{string set} \Rightarrow \texttt{string set}" ("\_ ; \_")
\texttt{where}

"\texttt{L1 \ ; \ L2} = \{\texttt{s1} @ \texttt{s2} \mid \texttt{s1 \ s2. s1} \in \texttt{L1} \land \texttt{s2} \in \texttt{L2}\}"

\textbf{fun}

\texttt{lang\_pow} :: "\texttt{string set} \Rightarrow \texttt{nat} \Rightarrow \texttt{string set}" ("\_ \uparrow \_")
\texttt{where}

"\texttt{L} \uparrow 0 = \{[]\}"
\texttt{\mid "L} \uparrow (\texttt{Suc i}) = \texttt{L} \; (\texttt{L} \uparrow \texttt{i})"

\textbf{definition}

\texttt{lang\_star} :: "\texttt{string set} \Rightarrow \texttt{string set}" ("\_⋆")
\texttt{where}

"\texttt{L⋆} \equiv \bigcup \texttt{i. (L} \uparrow \texttt{i)}"
Regular Expressions

datatype rexp =
  EMPTY
  | CHAR char
  | SEQ rexp rexp
  | ALT rexp rexp
  | STAR rexp

fun
  L :: "rexp ⇒ string set"
where
  "L(EMPTY) = {[[]]}"
  | "L(CHAR c) = {[c]}"
  | "L(SEQ r1 r2) = (L r1) ; (L r2)"
  | "L(ALT r1 r2) = (L r1) ∪ (L r2)"
  | "L(STAR r) = (L r)⋆"
function

dagger :: "rexp ⇒ char ⇒ rexp set" ("_ † _")

where

  r1: "(EMPTY) † c = {}"
  r2: "(CHAR c') † c = (if c = c' then {EMPTY} else {})"
  r3: "(ALT r1 r2) † c = r1 † c ∪ r2 † c"
  r4: "(SEQ EMPTY r2) † c = r2 † c"
  r5: "(SEQ (CHAR c') r2) † c = (if c = c' then {r2} else {})"
  r6: "(SEQ (SEQ r11 r12) r2) † c = (SEQ r11 (SEQ r12 r2)) † c"
  r7: "(SEQ (ALT r11 r12) r2) † c = (SEQ r11 r2) † c ∪ (SEQ r12 r2) † c"
  r8: "(SEQ (STAR r1) r2) † c = r2 † c ∪ {SEQ (SEQ r' (STAR r1)) r2 | r'. r' ∈ r1 † c}"
  r9: "(STAR r) † c = {SEQ r' (STAR r) | r'. r' ∈ r † c}"
Matcher

function matcher :: "rexp ⇒ string ⇒ bool" ("_ ! _")
where
  s01: "EMPTY ! s = (s =[])"
  s02: "CHAR c ! s = (s = [c])"
  s03: "ALT r1 r2 ! s = (r1 ! s ∨ r2 ! s)"
  s04: "STAR r ! [] = True"
  s05: "STAR r ! c#s =
       (False ∨ OR {SEQ (r') (STAR r)!s | r'. r' ∈ r † c})"
  s06: "SEQ r1 r2 ! [] = (r1 ! [] ∧ r2 ! [])
  s07: "SEQ EMPTY r2 ! (c#s) = (r2 ! c#s)"
  s08: "SEQ (CHAR c') r2 ! (c#s) = (if c'=c then r2 ! s else False)"
  s09: "SEQ (SEQ r11 r12) r2 ! (c#s) = (SEQ r11 (SEQ r12 r2) ! c#s)"
  s10: "SEQ (ALT r11 r12) r2 ! (c#s) =
                     ((SEQ r11 r2) ! (c#s) ∨ (SEQ r12 r2) ! (c#s))"
  s11: "SEQ (STAR r1) r2 ! (c#s) =
       (r2 ! (c#s) ∨ OR {SEQ r' (SEQ (STAR r1) r2) ! s | r'. r' ∈ r1 † c})"
Correctness of the matcher:

\( r \not\rightarrow s \implies s \in L_r \)

\( \neg r \not\rightarrow s \implies s \notin L_r \)