DPL seminar notes and problems

Handout 1

Konstantine Arkoudas

November 6, 2002
Athena syntax

\[ E ::= I \mid S \mid \epsilon \mid (\text{var } I) \mid (\text{meta-id } I) \mid 'I' \mid (\text{cell } F) \mid (\text{ref } E) \mid (\text{set! } E \ F) \]

(function (I*) E) \mid (E \ F^*) \mid [F^*] \mid (\text{check } (F \ E)^*)

(match F \ (\pi E)^*) \mid (\text{let } (I \ F)^* \ E) \mid (\text{letrec } (I \ F)^* \ E) \mid (\text{begin } F^+)

(while F_1 \ F_2) \mid (\& F^+) \mid (\| F^+)

D ::= (\text{apply-method } E \ F^*) \mid (\text{assume } F \ D) \mid (\text{assume-let } (I \ F)^* \ D)

(suppose-absurd F \ D) \mid (\text{suppose-absurd-let } (I \ F)^* \ D) \mid (\text{dcheck } (F \ D)^*)

(dm match F \ (\pi D)^*) \mid (\text{dlet } (I \ F)^* \ D) \mid (\text{dletrec } (I \ F)^* \ D)

(dm match F \ (\pi D)^*) \mid (\text{try } D^+) \mid (\text{dbegin } D^+) \mid (E \ \text{BY } D)

(generalize-over E \ D) \mid (\text{pick-any } I \ D) \mid (\text{with-witness } E \ F \ D)

(pick-witness \ I \ F \ D) \mid (\text{by-induction-on } E \ F \ (\pi \ D)^*)

F ::= E \mid D

\[ \pi ::= \epsilon \mid [I \ S \ [\pi^*]] \mid (\text{var } \pi) \mid (\text{meta-id } \pi) \mid 'I' \mid (\text{val-of } \pi)
\]

(list-of \ \pi_1 \ \pi_2) \mid (\text{split } \pi_1 \ \pi_2) \mid (\pi \ \pi^+) \mid (\text{cell-of } \pi) \mid (\text{bind } \pi \ \pi)

(some-list \ D) \mid (\text{some-cell } D) \mid (\text{some-var } D) \mid (\text{some-quant } D)

(some-prop-con \ D) \mid (\text{some-term } D) \mid (\text{some-atom } D) \mid (\text{some-prop } D)

(some-function \ D) \mid (\text{some-method } D) \mid (\text{some-symbol } D) \mid (\text{some-sub } D)

We write V* for zero or more occurrences of V, and V+ for one or more occurrences of it. In addition, 'I can be written as an abbreviation for (meta-id I), ?I can be written as an abbreviation for (var I), and (!E F*) can be written as an abbreviation for (apply-method E F*). The non-
terminals I and S range over identifiers and strings, respectively. A string is normally input between quotation marks, as in

"This is the first line.\nThis is the second line."

and is just an abbreviation for a list of characters. For instance, "foo" is understood as the list

['f 'o 'o'].

1
Problems

1. Give an Athena proof for each of the following arguments:
   - Premises: \(((\forall x) [(P(x) \land Q(x)) \Rightarrow R(x)], (\exists x) [Q(x) \land \neg R(x)]\)
     Conclusion: \((\exists x) \neg P(x)\)
   - Premises: \(((\forall x) [P(x) \Rightarrow Q(x)], (\exists x) [R(x) \land \neg Q(x)], (\forall x) [R(x) \Rightarrow (P(x) \lor S(x))])\)
     Conclusion: \((\exists x) R(x) \land S(x).\)
   - Premises: \(((\exists y) (\forall x) R(x, y))\)
     Conclusion: \((\forall x) (\exists y) R(x, y).\)
   - Premises: \(((\forall x) (\forall y) [(P(x) \land Q(y)) \Rightarrow R(x, y)], (\forall x) (\forall y) [(P(x) \land R(x, y)) \Rightarrow S(y)]\)
     Conclusion: \((\exists x) [P(x) \Rightarrow (\forall x) (Q(x) \Rightarrow S(x))]\)
   - Premises: \(((\exists x) [P(x) \land (\forall y) (Q(y) \Rightarrow R(x, y))], (\forall x) (\forall y) [(P(x) \land R(x, y)) \Rightarrow S(y)]\)
     Conclusion: \((\forall x) (Q(x) \Rightarrow S(x))\)
   - Premises: \(((\exists x) [P(x) \land (\forall y) [(P(x) \land Q(y)) \Rightarrow R(x, y)]], (\forall x) (\forall y) [(P(x) \land R(x, y)) \Rightarrow S(y)]\)
     Conclusion: \(((\exists x) (\exists y) [P(x) \land P(y) \land \neg R(x, y)]\)

2. Write a function \texttt{remove-duplicates} that takes an arbitrary list of strings \(L\) and removes all duplicate occurrences of the same string in \(L\). For instance, your function should behave as follows:

   \[
   \text{(remove-duplicates ["foo" "boo" "foo" "ab123"])} \rightarrow ["foo" "boo" "ab123"]
   \]

   \[
   \text{(remove-duplicates ["foo"])} \rightarrow ["foo"]
   \]

   \[
   \text{(remove-duplicates ["foo" "a" "b" "a" "a" "b"])} \rightarrow ["foo" "a" "b"]
   \]

   Your function should have exactly one recursive call, and that call should be a tail recursion. It should not use a \texttt{while} loop either, nor should you define any other auxiliary functions that do any kind of looping, either by recursion or via \texttt{while}. (Hint: use a \texttt{split} pattern.)

3. Athena has a built-in nullary function \texttt{fresh-var} that returns a uniquely fresh Athena variable every time it is invoked. Use \texttt{fresh-var} to write a function \texttt{rename-prop} that takes an arbitrary proposition \(P\) and produces a proposition that is obtained from \(P\) by consistently replacing the bound variables of \(P\) by fresh variables.

4. Write a deduction \(D\) that always fails (i.e., such that an error message is generated when you evaluate \(D\) in any assumption base, environment, and store).