**CS571**

- Notes 16
- Denotational Semantics of a Simple Calculator

**The Calculator**

- Two functions: + and *
- Unbounded natural numbers (no negatives)
- Conditional: if-then-else
- Parentheses
- One memory register

**The Button Layout**

- Display (unlimited)
- Scroll buttons
- Arithmetic and conditional
- Separates expressions in conditional
- Evaluates expression

**Abstract syntax of correct button push sequences**

- \( P \in \text{Program} \)
- \( S \in \text{Expression-sequence} \)
- \( E \in \text{Expression} \)
- \( N \in \text{Numeral} \)
- \( D \in \text{Digit} \)
- \( P ::= \text{ON} \ S \)
- \( S ::= E \ \text{TOTAL} \ S | E \ \text{TOTAL} \ OFF \)
- \( E ::= E_1 \ + \ E_2 | E_1 \ * \ E_2 | N | ( E ) | \text{IF} \ E_1 , E_2 , E_3 | \text{LASTANSWER} \)
- \( N ::= N \ D | D \)
- \( D ::= 0 \ | 1 \ | 2 \ | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 \)
Example program
- ON 1 + 2 TOTAL 3 * LASTANSWER TOTAL OFF
- Result is a sequence of values: (3, 9)
- The denotation of a program will be this sequence, represented as a list in the semantic algebras

Semantic algebras
- Natural numbers, with constants, addition, multiplication and equality:
  - zero, one etc.
  - plus, times, equals
- Booleans with constants, and the conditional form:
  - true, false
  - _→_ ☐☐

Algebra for a List
- Domain: List = A* (A is any domain)
- Operations:
  - nil: List
  - hd: List → A
  - tl: List → List
  - cons: A → List → List

Valuation functions - Functionality
- P: Program → Nat*
- S: Expression-sequence → Nat → Nat
- E: Expression → Nat → Nat
- N: Numeral → Nat
- D: Digit → Nat
Valuation functions - numbers

\[ D[0] = \text{zero} \]

\[ \cdots \]

\[ D[9] = \text{nine} \]

\[ N[N,D] = \text{(ten times N[N]) plus D[D]} \]

\[ N[D] = D[D] \]

Valuation functions – simple expressions

\[ E[N](n) = N[N] \]

\[ E[(E)](n) = E[E](n) \]

\[ E[\text{LASTANSWER}](n) = n \]

\[ E[E_1 + E_2](n) = E[E_1](n) \text{ plus } E[E_2](n) \]

\[ E[E_1 \times E_2](n) = E[E_1](n) \text{ times } E[E_2](n) \]

Valuation functions – the conditional form

\[ E[\text{IF } E_1, E_2, E_3](n) = \]

\[ \left( E[E_1](n) \text{ equals zero} \right) \rightarrow E[E_2](n) \cup E[E_3](n) \]

Each expression produces its value when the TOTAL button is pushed – as in the expression sequence

Valuation functions – the expression sequence

\[ S[E \text{ TOTAL } S](n) = \text{let } n' = E[E](n) \text{ in } n' \text{ cons } S[S](n') \]

\[ S[E \text{ TOTAL OFF}](n) = E[E](n) \text{ cons nil} \]

\[ n' \] is the value put into memory for the evaluation of the rest of the sequence
Valuation functions – the program

\[ P[\text{ON } S] = S[S](\text{zero}) \]

- \text{zero} is the initial value of the memory passed to the first expression

A Sample derivation

- Program is:
  \[
  \text{ON } 1 + 2 \text{ TOTAL } 3 * \text{LASTANSWER TOTAL OFF}
  \]

\[
P[\text{ON } 1 + 2 \text{ TOTAL } 3 * \text{LASTANSWER TOTAL OFF}] = S[1 + 2 \text{ TOTAL } 3 * \text{LASTANSWER TOTAL OFF}](\text{zero})
\]

\[
= \text{let } n' = E[1 + 2](\text{zero}) \text{ in}
\]

\[
n' \text{ cons } S[3 * \text{LASTANSWER TOTAL OFF}](n')
\]

Derivation continued

- Work on the first expression:

\[
E[1 + 2](\text{zero})
\]

\[
= E[1](\text{zero}) \text{ plus } E[2](\text{zero})
\]

\[
= N[1] \text{ plus } N[1]
\]

\[
= D[1] \text{ plus } D[1]
\]

\[
= \text{one plus two}
\]

Derivation continued

- Reduce \textit{two plus one} to \textit{three}. The derivation then continues:

\[
\text{three cons } S[3 * \text{LASTANSWER TOTAL OFF}](\text{three})
\]

\[
= \text{three cons } (E[3 * \text{LASTANSWER}](\text{three})) \text{ cons nil}
\]
**Derivation continued**

- Again work on the expression separately:
  
  \[ E[3 \ast \text{LASTANSWER}](three) \]
  
  \[ = E[3](three) \text{ times } E[\text{LASTANSWER}](three) \]
  
  \[ = N[3] \text{ times } three \]
  
  \[ = D[3] \text{ times } three \]
  
  \[ = \text{three times } three \]
  
  \[ = \text{nine} \]

**The Derivation Finished**

- The final result is (the program’s denotation) is:
  
  \[ \text{three cons nine cons nil} \]
  
  - Without reducing the algebraic expressions it is the ‘compiled’ version:
    
    \[ (\text{one plus two}) \text{ cons (three times (one plus two)) cons nil} \]

**Lambda forms (alternative form for valuation functions)**

- Can push extra arguments to right-hand side as lambda parameters:
  
  \[ S[E \text{ TOTAL } S] = \lambda n.\text{let } n’ = E[E](n) \text{ in } n’ \text{ cons } S[S](n’) \]
  
  \[ S[E \text{ TOTAL OFF}] = \lambda n.E[E](n) \text{ cons nil} \]
  
  - Can sometimes omit same argument on both sides:
    
    \[ E[(E)] = E[E] \]