Lecture 3: Halting Problem & LISP

CSC 131
Fall, 2014

Kim Bruce

Decidable Problems

- A yes/no question about all values x is *decidable* iff there is an algorithm A that given any input x, always halts after a finite amount of time and gives the correct answer.
  - Has to eventually stop for all inputs x
  - Has to always give the right answer

Undecidable Problems

- Can we come up with a general program so that for any program P and any input x, it will tell us whether on not P halts on x?
- Called the Halting Problem
  - Proved undecidable by Turing (*no algorithm to decide)*
  - Holds for any sufficiently complex language
    - Clearly solvable if language has no control constructs (e.g., loops, goto, recursion, etc.)

Solving Halting?

- Suppose H is a function that solves the halting problem (in your favorite PL):
  - H takes two string arguments P and x, and H(P,x) returns true iff P is a legal program and when P is run with input x, then P(x) halts.
  - Returns false otherwise (P not halt, P not legal, etc.)
  - Note H must stop in a finite amount of time with the correct answer.
  - Show by contradiction that there can't be such an H.

Diagonalizing

- Given H(P,x), define D(P) to be the following program:
  - Run H(P,P). It is guaranteed to stop with an answer.
  - If it returns true, loop forever, otherwise halt.
    - Thus, if P(P) halts, then D(P) runs forever
    - If P(P) doesn't halt then D(P) stops
    - *Note we aren't executing P(P), just asking H(P,P).*

Diagonalizing

- Consider D(D):
  - Run H(D,D). It is guaranteed to stop with an answer.
  - If it returns true, loop forever, otherwise halt.
    - Thus, if D(D) halts, then D(D) runs forever
    - If D(D) doesn't halt then D(D) stops
    - *Contradiction!!!*
  - Hence there cannot be such an H.
  - Halting problem is undecidable.

Everything Interesting is Undecidable

- Examples:
  - Will program eventually divide by 0?
  - Will program eventually dereference a null pointer?
  - Will program touch a particular piece of memory?
  - Will program ever print out 0?

Proving Problems Undecidable

- Reduction Proofs:
  - Assume there is an algorithm A' to solve new problem.
  - Show that can use A' to solve halting problem.
  - Contradiction shows A' can't exist.

Example

- Show no algorithm to determine if program Q halts for any input (at least one).
  - Suppose A'(Q) always halts and returns yes iff Q halts for some input.
  - Build algorithm to solve halting problem
    - Given input P, w for halting, build new program P_w that ignores its input and then simulates P on input w.
      - I.e., replace read statement by assigning value w to vble.
    - Then A'(P_w) returns yes iff P halts on w
      - because all runs of P_w are the same!! So if any accepted, so are all.
    - *Solves halting problem: contradiction!!!*
      Hence there cannot be such an A'

LISP and Scheme (Racket)

Language Design

- For each language you learn, consider:
  - Motivating applications
  - Abstract machine
  - Theoretical foundations

LISP & Scheme

- Developed in late 50's by John McCarthy
  - McCarthy won Turing award in 1971
  - See McCarthy "History of Lisp" on web.
- Support for AI programming
  - Symbolic differentiation, language processing
  - Emacs written in LISP
  - Almost cult-like fervor, even though most AI programming now in other languages.

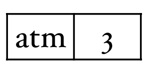
LISP Features

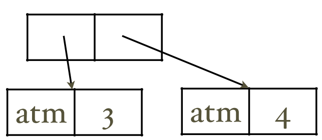
- Compute w/symbolic expressions instead of numbers
- Representation of expressions as (nested) lists
- Small set of selector and constructor ops expressed as functions. Composition to compose functions.
- Use of conditional *expressions* for branching
- Recursive use of conditional expressions for building computable functions.
- Representation of LISP programs as LISP data.
- LISP eval function as formal def. & interpreter
- Garbage collection

LISP Diaspora

- MacLISP, FranzLISP, … -- incompatible
- Scheme -- relatively compact dialect
  - developed at MIT by Gerry Sussman & Guy Steele
- Common LISP
  - Steele wrote first language description
  - # of pages in index of CL report exceeds total pages in Scheme report.
- *Programs in text are sort-of LISP*
- Racket grew out of PLT Scheme, aimed at education.
  - Includes dialects, macros, etc.

LISP/Scheme Data Rep

- Data: <sexp> :: = <atom> | (<sexp> . <sexp>)
- atom: 3 

- (3 . 4) 

- Lists are abbreviations:
  - (4 5 6) = (4 . (5 . (6 . NIL)))

LISP/Scheme List Ops

- cons, car, cdr:
  - car (a . b) = a, cdr (a . b) = b
  - (cons 1 ‘(2 3 4)) ⇒ (1 2 3 4)
  - (car ‘(3 4 5)) ⇒ 3
  - (cdr ‘(3 4 5)) ⇒ (4 5)
  - (cond ((= n 0) 0) ((> n 0) 1) (true -1))
  - (if (= n 0) 0 1)
- *All ops are prefix*

Defining functions

- (lambda (x) (* x x)) *anonymous function*
- (define z 22) *naming exp*
- (define square (lambda (x) (* x x))) *or*
- (define (square x) (* x x))

Recursive Functions

- ```
  (define (append l1 l2)
    (if (null? l1)
        l2
        (cons (car l1) (append (cdr l1) l2)))
    )
  ```
- (append ‘(1 2 3) ‘(4 5 6))

More functions

- Predefined list functions:
  - (map f ‘(a b c d)) ⇒ ((f a) (f b) (f c) (f d))
  - (member 1 ‘(3 2 1 0)) ⇒ (1 0)
- Local variables:
  - ```
    (define (roots a b c)
       (let ((disc (- (* b b) (* 4 a c))))
          (if (>= disc 0) (list (/ (+ (- 0 b) (sqrt disc)) (* 2 a))
                                (/ (- (- 0 b) (sqrt disc)) (* 2 a)))
                          ‘(0 0)
          )
       )
    )
    ```

Dynamically Typed

- Types associated w/ values instead of variables.
- Values have tag w/type
- (* a b) -- actual operation depends on whether both ints, both doubles, or one something else
- Requires run-time check for type safety

Evaluation

- Very successful in AI & elsewhere
- Good for experimental programming
- Blur boundaries between data and program
- Simple abstract machine:
  - Atoms and cons cells -- simple representation
  - expression, continuation, association list (environment), and heap.
- See more modern functional languages soon