

Computer Science I

Professor Tom Ellman
Lecture 23

Party Planner

- Given:
 - A data base of people, properties and relationships.
 - A DAY on which to hold a party.
 - A number N of party guests.
- Find:
 - A set of people.
 - That will make a good party.

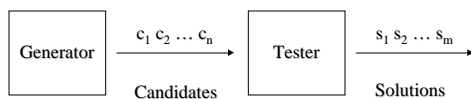
Party Rules

- The guest set must have exactly N members.
- Each guest must be a person.
- Each guest must be available on DAY.
- At least one interesting person is a guest.
- At least one funny person is a guest.
- Numbers of men and women are equal.
- Everyone knows someone else.
- No one dislikes anyone else.
- No mixing Republicans and Democrats.

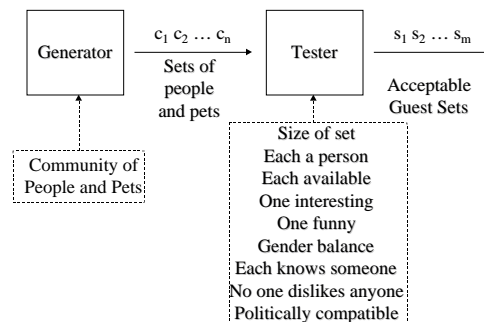
Generate and Test

- A *weak method* for solving problems.
- Identify a universe of candidate solutions.
- Generate candidates one at a time.
- Test each candidate against the constraints.
- Return one or all of the candidates that satisfy the constraints.

Generate and Test



Using Generate and Test to Solve the Party Planner Problem



```

(define party
  (lambda (size date)
    (set-comprehension (power-set *community*)
      (lambda (set)
        (and (= (cardinal set) size)
              (each-a-person? set)
              (each-available? set date)
              (one-funny? set)
              (one-interesting? set)
              (gender-balanced? set)
              (each-knows-someone? set)
              (no-enemies? set)
              (politically-compatible? set)))))))

```

```

(define gender-balanced?
  (lambda (set)
    (letrec ((helper (lambda (s n)
                       (if (empty-set? s)
                           (<= (abs n) 1)
                           (let ((e (pick s)))
                             (if (male? e)
                                 (helper ((residue e) s) (+ n 1))
                                 (helper ((residue e) s) (- n 1))))))))
      (helper set 0))))

```

```

(define each-a-person?
  (lambda (set) ((for-all (lambda (x) (person? x)) set))))

(define each-available?
  (lambda (set date) ((for-all (lambda (x) (available? x date)) set))))

```

```

(define no-enemies?
  (lambda (set)
    ((for-all (lambda (x) (not ((there-exists (lambda (y) (dislikes? x y))
                                              set))))
      set)))

(define each-knows-someone?
  (lambda (set)
    ((for-all (lambda (x) ((there-exists (lambda (y) (knows? x y))
                                          set))))
      set)))

```

```

(define politically-compatible?
  (lambda (set)
    (not (and ((there-exists democratic?) set)
              ((there-exists republican?) set)))))

```

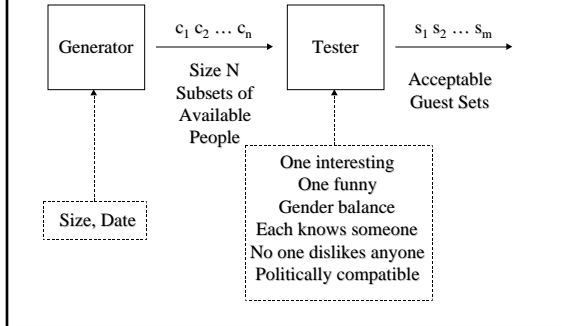
```

(define one-interesting?
  (lambda (set) ((there-exists interesting?) set)))

(define one-funny?
  (lambda (set) ((there-exists funny?) set)))

```

Improving Generate & Test



```

(define party
  (lambda (size date)
    (let ((temp (set-comprehension *people*
                                   (lambda (x) (available? x date))))
          (set-comprehension (subsets-of-size temp size)
                             (lambda (set)
                               (and (one-funny? set)
                                    (one-interesting? set)
                                    (gender-balanced? set)
                                    (politically-compatible? set)
                                    (each-knows-someone? set)
                                    (no-enemies? set)
                                    ))))))))
  
```

```

(define subsets-of-size
  (lambda (set n)
    (cond ((= n 0) (make-set the-empty-set))
          ((empty-set? set) the-empty-set)
          (else (let ((e (pick set)))
                  (let ((new1 (subsets-of-size ((residue e) set) n))
                      (new2 (subsets-of-size ((residue e) set) (- n 1))))
                    (union new1
                           (set-map (lambda (s) (adjoin e s))
                                    new2))))))))
  
```

```

(define available?
  (lambda (person date)
    ((element (make-op person date)) *available*)))

(define knows?
  (lambda (p1 p2) ((element (make-op p1 p2)) *know*)))

(define dislikes?
  (lambda (p1 p2) ((element (make-op p1 p2)) *dislike*)))
  
```

```

(define male?
  (lambda (x) ((element x) *men*)))

(define female?
  (lambda (x) ((element x) *women*)))

(define democratic?
  (lambda (x) ((element (make-op x 'democratic)) *registered*)))

(define republican?
  (lambda (x) ((element (make-op x 'republican)) *registered*)))

(define independent?
  (lambda (x) ((element (make-op x 'independent)) *registered*)))
  
```

```

(define funny?
  (lambda (x) ((element x) *funny-people*)))

(define interesting?
  (lambda (x) ((element x) *interesting-people*)))

(define person?
  (lambda (x) ((element x) *people*)))
  
```

```

(define *men* (make-set 'tom 'fred 'billy 'tim 'frank 'barry))

(define *women* (make-set 'sue 'jane 'betty 'ellen 'joan 'betsy))

(define *people* (set-union *men* *women*))

(define *pets* (make-set 'fido 'rover 'wiskers 'garfield))

(define *community* (set-union *people* *pets*))

```

```

(define *available*
  (make-set
    (make-op 'tom 'saturday)
    (make-op 'fred 'saturday)
    (make-op 'sue 'saturday)
    (make-op 'jane 'saturday)
    (make-op 'tom 'friday)
    (make-op 'billy 'friday)
    (make-op 'sue 'friday)
    (make-op 'betty 'friday)
    (make-op 'tim 'saturday)
    (make-op 'frank 'saturday)
    (make-op 'ellen 'saturday)
    (make-op 'joan 'saturday)
    (make-op 'tim 'friday)
    (make-op 'barry 'friday)
    (make-op 'ellen 'friday)
    (make-op 'betsy 'friday)))

```

```

(define *registered* (make-set
  (make-op 'tom 'democratic)
  (make-op 'fred 'republican)
  (make-op 'sue 'democratic)
  (make-op 'jane 'republican)
  (make-op 'billy 'independent)
  (make-op 'betty 'independent)
  (make-op 'tim 'democratic)
  (make-op 'frank 'republican)
  (make-op 'ellen 'democratic)
  (make-op 'joan 'republican)
  (make-op 'barry 'independent)
  (make-op 'betsy 'independent)
))

```

```

(define *know*
  (make-set
    (make-op 'tom 'fred) (make-op 'fred 'tom)
    (make-op 'fred 'billy) (make-op 'billy 'fred)
    (make-op 'billy 'betty) (make-op 'betty 'billy)
    (make-op 'betty 'sue) (make-op 'tom 'sue)
    (make-op 'sue 'tom) (make-op 'sue 'betty)
    (make-op 'sue 'jane) (make-op 'jane 'sue)
    (make-op 'tim 'frank) (make-op 'frank 'tim)
    (make-op 'frank 'barry) (make-op 'barry 'frank)
    (make-op 'barry 'betsy) (make-op 'betsy 'barry)
    (make-op 'betsy 'ellen) (make-op 'tim 'ellen)
    (make-op 'ellen 'tim) (make-op 'ellen 'betsy)
    (make-op 'ellen 'joan) (make-op 'joan 'ellen)))

```

```

(define *dislike* (make-set
  (make-op 'billy 'fred)
  (make-op 'sue 'jane)
  (make-op 'barry 'frank)
  (make-op 'ellen 'joan)
))

(define *funny-people* (make-set 'tom 'sue 'tim 'ellen))

(define *interesting-people* (make-set 'fred 'betty 'frank 'betsy))

```