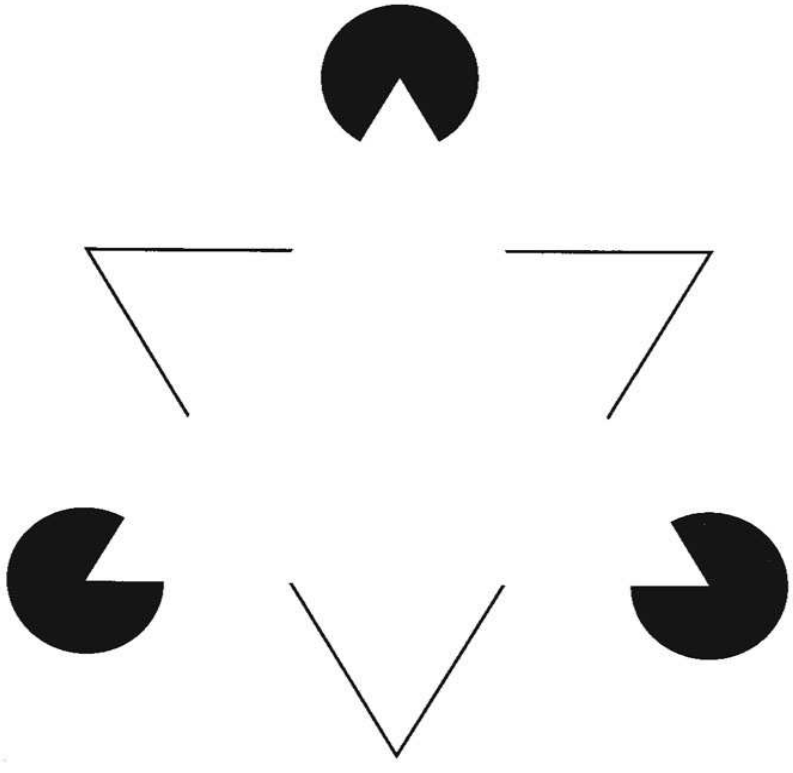
We Really Don't Know How to Compute!

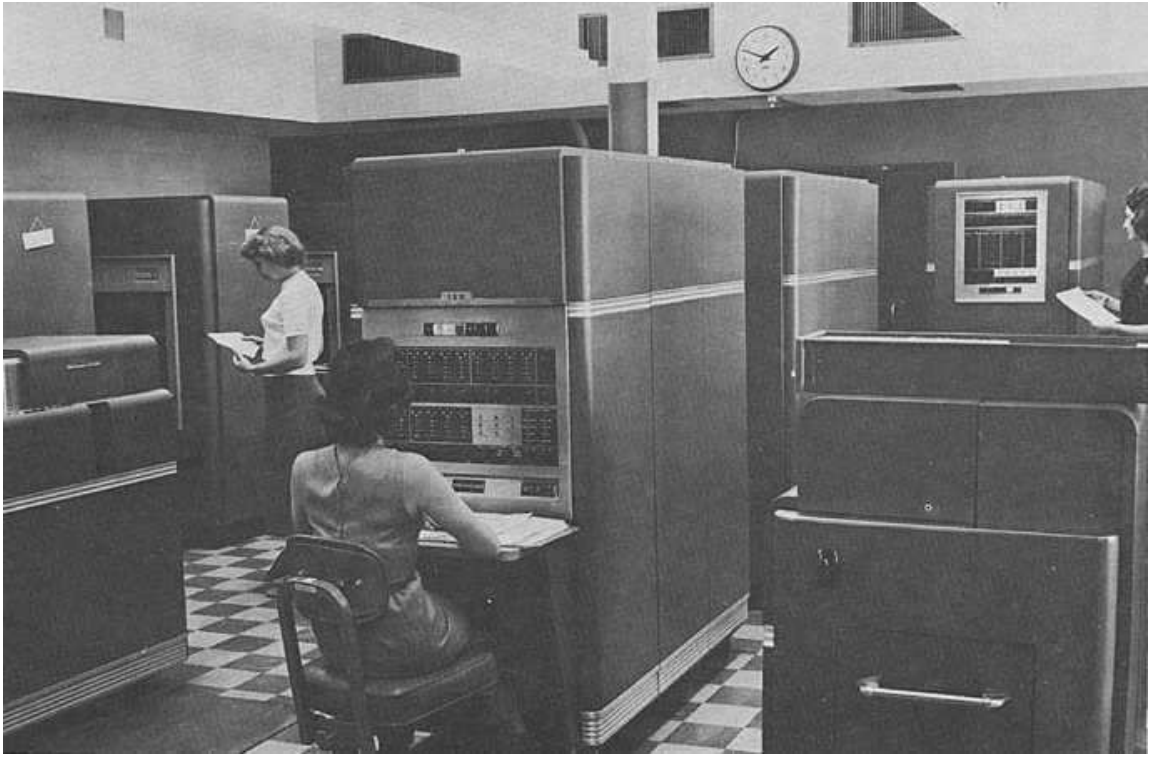
Gerald Jay Sussman
Massachusetts Institute of Technology

CSAIL and EECS

A clue? Kanizsa's Triangle Illusion



Gaetano Kanizsa (1955)



The IBM 650 Data-Processing System

Photo: U.S. Army Anniston Ordnance Depot,
from BRL Report 1115, 1961

Evolving and maintaining computer systems is expensive. This cost can be anywhere from 50% of total programming effort to 75% of total available effort. In addition, the proportion of effort devoted to maintenance has been increasing: from 35-40% in the 1970s, through 40-60% in the 1980s up to 70-80% in the 1990s.

Huw Evans
Department of Computing Science,
The University of Glasgow,
Glasgow, Scotland, UK, G12 8RZ

Traditional generic arithmetic in Scheme

```
(+ 1 2 3)          ; Exact Integer
#| 6 |#

(+ 1 (/ 2 3))      ; Exact Rational
#| 5/3 |#

(+ 1 (sqrt 2))     ; Inexact Real
#| 2.414213562373095 |#

(+ 1 (sqrt -2))    ; Inexact Complex
#| 1+1.4142135623730951i |#

(+ 1 (sqrt -4))    ; Exact Complex
#| 1+2i |#
```

Arithmetic on Functions

```
(sin 2.5)
#| .5984721441039565 |#

(square 2.5)
#| 6.25 |#

((+ sin square) 2.5)
#| 6.848472144103956 |#


((+ (square sin) (square cos)) 2.5)
#| 1. |#
```

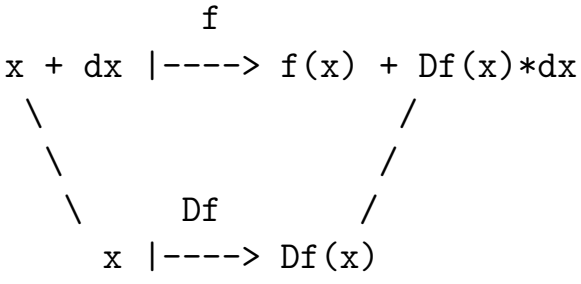
Symbolic Arithmetic

```
(let ((r1 (literal-number 'r_1))
      (r2 (literal-number 'r_2)))
  (/ 1 (+ (/ 1 r1) (/ 1 r2))))

#| (/ (* r_1 r_2) (+ r_1 r_2)) |#
```

Automatic Differentiation

```
((D (+ cos square)) 'x)
#|
(+ (* 2 x) (* -1 (sin x)))
|#


(define (r x y)
  (sqrt (+ (square x) (square y))))

((D r) 'x 'y)
#|
(down (/ x (sqrt (+ (expt x 2) (expt y 2))))
      (/ y (sqrt (+ (expt y 2) (expt x 2)))))
|#
```

Derivative Computation
a Generic Extension

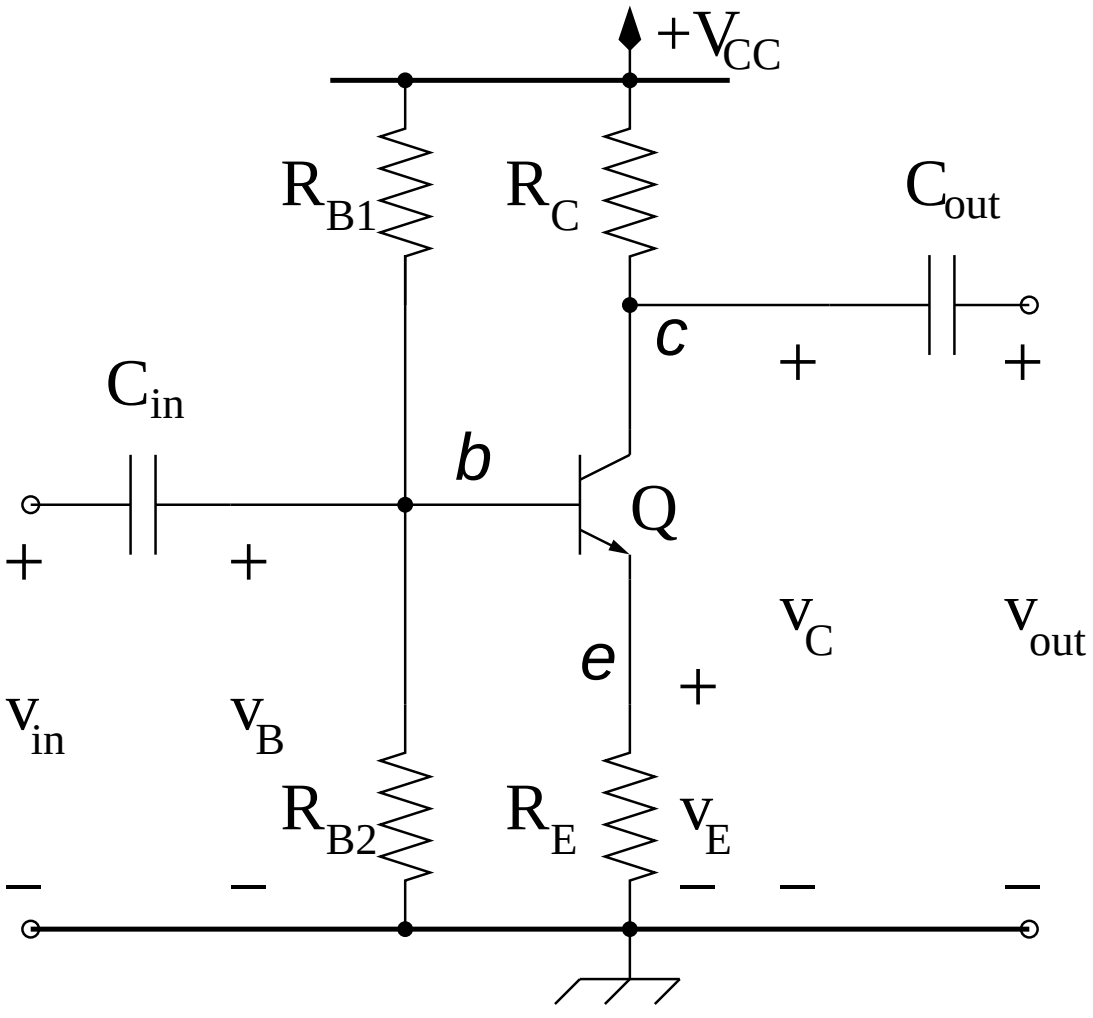


f g

x + dx |----> f(x) + Df(x)*dx |----> g(f(x)) + Dg(f(x))*Df(x)*dx

Advanced Use of Generics.

```
(define ((Lagrange-equations Lagrangian) config-space-path)
  (let ((state-path (Gamma config-space-path)))
    (- (D (compose ((partial 2) Lagrangian) state-path))
       (compose ((partial 1) Lagrangian) state-path))))

(define ((Gamma w) t)
  (up t (w t) ((D w) t)))

(define ((L-harmonic m k) state)
  (- (* 1/2 m (square (velocity state)))
     (* 1/2 k (square (coordinate state)))))

(((Lagrange-equations (L-harmonic 'm 'k))
  (literal-function 'x))
 't)
#| (+ (* k (x t)) (* m (((expt D 2) x) t))) |#
```


+V_CC
R_B1
R_C
C_out
c
C_in
b
Q
e
v_C
v_out
v_in
v_B
R_B2
R_E
v_E

```
(define-propagator (ce-amplifier)
  (let-cells ((Rb1 (resistor))  (Rb2 (resistor))
              (Rc  (resistor))  (Re  (resistor))
              (Cin (capacitor)) (Cout (capacitor))
              (Q (infinite-beta-bjt))
              (+rail-w (short-circuit)) (-rail-w (short-circuit)))
    (let-cells ((+rail-node
                 (node (the t1 Rb1) (the t1 Rc) (the t2 +rail-w)))
                (-rail-node
                 (node (the t2 Rb2) (the t2 Re) (the t2 -rail-w)))
                (e (node (the t1 Re) (the emitter Q)))
                (c (node (the t2 Rc) (the t2 Cout)
                         (the collector Q)))
                (b (node (the t2 Rb1) (the t1 Rb2)
                         (the base Q) (the t2 Cin))))
      (let-cells ((+rail (the t1 +rail-w))
                  (-rail (the t1 -rail-w))
                  (sigin (the t1 Cin))
                  (sigout (the t1 Cout)))
                  (e:inspectable-object
                   +rail -rail sigin sigout ...)))))
```

```
(let-cells ((power (bias-voltage-source))
            (vin   (signal-voltage-source))
            (vout  (open-circuit))
            (amp   (ce-amplifier)))
  (let-cells ((gnd
               (node (the t2 power)
                     (the t2 vin) (the t2 vout)
                     (the -rail amp)))
              (+V (node (the t1 power) (the +rail amp)))
              (in (node (the t1 vin) (the sigin amp)))
              (out (node (the t1 vout) (the sigout amp))))
    ((constant 0) (the potential gnd))))

(assume! (the strength power amp) (& 15. volt))
(assume! (the resistance rc amp) (& 5000. ohm))
(assume! (the resistance re amp) (& 1000. ohm))
(assume! (the resistance rb1 amp) (& 51000. ohm))
(assume! (the resistance rb2 amp) (& 10000. ohm))

(assume! (the vthreshold q amp bias) (& +0.7 volt))
(assume! (the I0 q amp) (& 1e-15 ampere))
(assume! (the beta q amp) 100)
```

```
(presume! (the amplifying operation q amp bias)))
(presume! (the beta-infinite q amp bias)))
(presume! (the emitter-follows q amp bias)))

(value (the potential b amp bias))
; (plunk (potential b amp bias) e12)
; CEP97   0=(+ -2.94e-4 (* 1.20e-4 e12))
; CEP109  (> (+ -.0007 (* .001 e12)) 0)
; CEP122  (> (+ 20.67 (* -6.60 e12)) .2)
; ((potential b amp bias) = 2.46)
;   (from (resistance rb2 amp)
;         (beta-infinite q amp bias)
;         (resistance rb1 amp)
;         (strength power))
;Value: (& 2.459016393442623 volt)

(value (the potential e amp bias))
; ((potential e bias) = 1.76)
;   (set by (-rhs:kvl-be q amp bias))
;     (because (potential b amp bias)
;              (vbe q amp bias))
;Value: (& 1.759016393442623 volt)
```

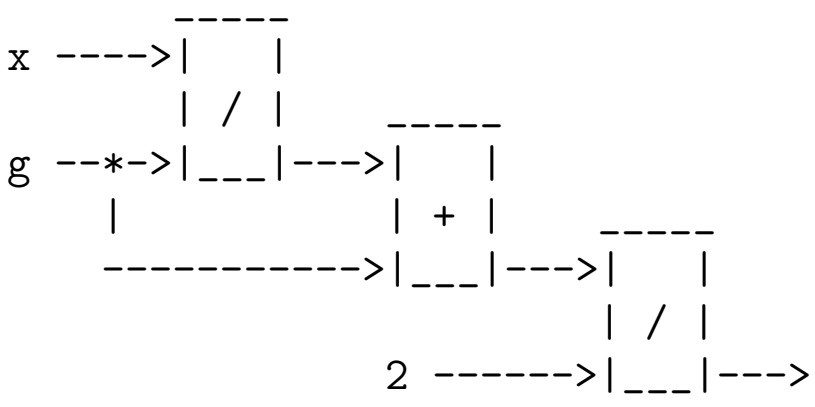
```
(why? (the potential e amp bias))
;(CEP132 (potential e bias) = 1.76
;   set by (-rhs:kvl-be q amp bias) (CEP130 CEP71))
;(CEP130 (potential b amp bias) = 2.46
;   because CEP60 CEP15 CEP58 CEP52)
;(CEP71  (vbe q amp bias) = .7
;   set by (emitter-follows q bias) (CEP62 CEP18))
;(CEP60  (resistance rb2 amp) = 10000.
;   set by assumption (CEP61))
;(CEP15  beta-infinite PREMISE)
;(CEP58  (resistance rb1 amp) = 51000.
;   set by assumption (CEP59))
;(CEP52  (strength power) = 15.
;   set by assumption (CEP53))
;(CEP62  (vthreshold q amp bias) = .7
;   set by assumption (CEP63))
;(CEP18  emitter-follows PREMISE)
;Value: QED
```

The Propagator Idea

Independent Stateless Machines

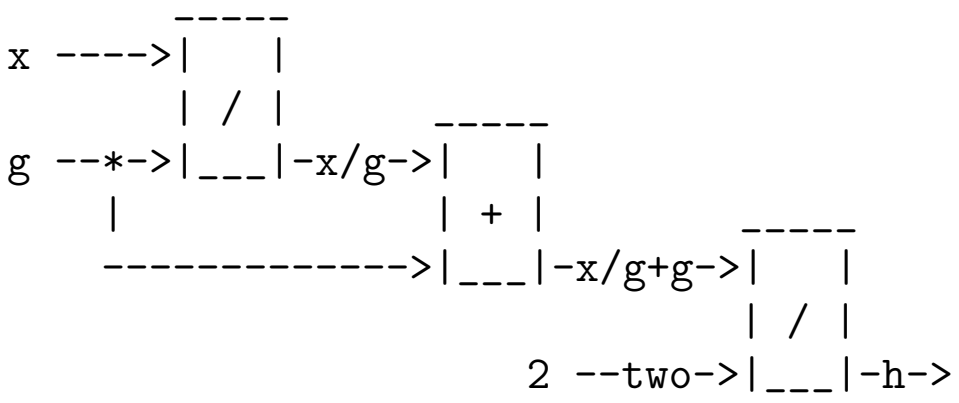
Connecting Stateful Cells

Expressions have Anonymous Connections




```
(define (heron-step x g)
  (/ (+ (/ x g) g) 2))
```

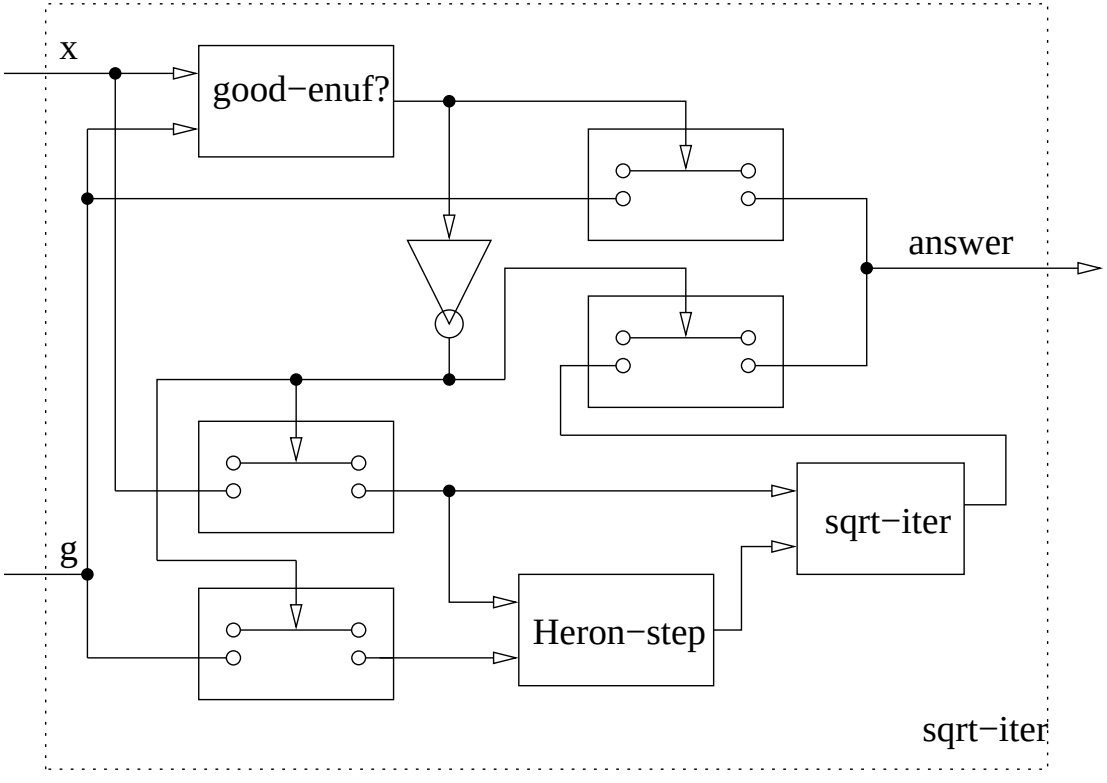
Propagators: All Connections Explicit




```
(define (heron-step x g h)
  (let ((x/g   (make-cell))
        (g+x/g (make-cell))
        (two   (make-cell)))
    (divider x g x/g)
    (adder g x/g g+x/g)
    ((constant 2) two)
    (divider g+x/g two h)))
```

```
(define x (make-cell))
(define guess (make-cell))
(define better-guess (make-cell))

(heron-step x guess better-guess)

(add-content x 2)
(add-content guess 1.4)

(content better-guess)
    => 1.4142857142857141
```

Iteration: a Recursive Machine



```
(define sqrt-iter
  (delayed-propagator
   (lambda (x g answer)
     (let ((done (make-cell))
           (not-done (make-cell))
           (x-again (make-cell))
           (g-again (make-cell))
           (new-g (make-cell))
           (new-answer (make-cell)))
       (good-enuf? g x done)
       (switch done g answer)
       (inverter done not-done)
       (switch not-done new-answer answer)
       (switch not-done x x-again)
       (switch not-done g g-again)
       (heron-step x-again g-again new-g)
       (sqrt-iter x-again new-g new-answer)))))
```

It works.

```
(define (sqrt-network x answer)
  (let ((one (make-cell)))
    ((constant 1.) one)
    (sqrt-iter x one answer)))


(define x (make-cell))
(define answer (make-cell))

(sqrt-network x answer)

(add-content x 2)

(content answer)
  => 1.4142135623730951
```

Multidirectional Constraints

```
(define (c:+ a b c)
  (p:+ a b c)
  (p:- c a b)
  (p:- c b a))

(define (c:* a b c)
  (p:* a b c)
  (p:/ c a b)
  (p:/ c b a))

(define pass-through
  (function->propagator-constructor
   (lambda (x) x)))

(define (identity-constraint a b)
  (pass-through a b)
  (pass-through b a))
```

Back to electricity!

```
(define ((2-terminal-device vic) t1 t2)
  (let ((i1 (current t1)) (e1 (potential t1))
        (i2 (current t2)) (e2 (potential t2)))
    (let ((v (make-cell)) (Power (make-cell))
          (zero (make-cell)))
      ((constant 0) zero)
      (c:+ v e2 e1)
      (c:+ i1 i2 zero)
      (c:* i1 v Power)
      (vic v i1)
      P)))

;;; For example
(define (linear-resistor R)
  (2-terminal-device
   (lambda (v i)
     (c:* i R v))))
```

All Operators are Extensible Generics

Cells **Merge** Information **Monotonically**

How to use a barometer, a stopwatch, and a ruler to measure the height of a building.

Sunlight and Similar Triangles: $\frac{H_{bldg}}{H_{bar}} = \frac{S_{bldg}}{S_{bar}}$

```
(define (similar-triangles s-bar h-bar s-bld h-bld)
  (let ((ratio (make-cell)))
    (c:* s-bar ratio s-bld)
    (c:* h-bar ratio h-bld)))

(define baro-height (make-cell))
(define baro-shadow (make-cell))
(define bldg-height (make-cell))
(define bldg-shadow (make-cell))
(similar-triangles baro-shadow baro-height
                   bldg-shadow bldg-height)

(add-content bldg-shadow (make-interval 54.9 55.1))
(add-content baro-height (make-interval 0.3 0.32))
(add-content baro-shadow (make-interval 0.36 0.37))

(content bldg-height)
=> #(interval 44.514 48.978)
```

Drop the Barometer! $s = \frac{g}{2}t^2$

```
(define (fall-duration t h)
  (let ((g (make-cell)) (one-half (make-cell))
        (t^2 (make-cell)) (gt^2 (make-cell)))
    ((constant (make-interval 9.789 9.832)) g)
    ((constant (make-interval 1/2 1/2)) one-half)
    (c:square t t^2)
    (c:* g t^2 gt^2)
    (c:* one-half gt^2 h)))


(define fall-time (make-cell))
(define bldg-height (make-cell))
(fall-duration fall-time bldg-height)

(add-content fall-time (make-interval 2.9 3.1))

(content bldg-height)
=> #(interval 41.163 47.243)
```

Two Measurements Are Better Than One.

```
(add-content bldg-shadow (make-interval 54.9 55.1))
(add-content baro-height (make-interval 0.3 0.32))
(add-content baro-shadow (make-interval 0.36 0.37))
(content bldg-height)
=> #(interval 44.514 48.978)

(add-content fall-time (make-interval 2.9 3.1))
(content bldg-height)
=> #(interval 44.514 47.243)


;;; But even better!

(content baro-height)
=> #(interval .3 .31839)  ; Was (.3 .32)
(content fall-time)
=> #(interval 3.0091 3.1) ; Was (2.9 3.1)
```

Bribery gets another answer!

```
(add-content bldg-height 45)

(content baro-height)
=> #(interval .3 .30328)

(content baro-shadow)
=> #(interval .366 .37)

(content bldg-shadow)
=> #(interval 54.9 55.1)

(content fall-time)
=> #(interval 3.0255 3.0322)
```

Dependencies and Contingent Values

Tracking of Provenance

Who ordered that?

```
(define baro-height (make-cell))
(define baro-shadow (make-cell))
(define bldg-height (make-cell))
(define bldg-shadow (make-cell))

(similar-triangles baro-shadow baro-height
                   bldg-shadow bldg-height)

(add-content bldg-shadow
 (contingent (make-interval 54.9 55.1) '(shadow)))
(add-content baro-height
 (contingent (make-interval 0.3 0.32) '(shadow)))
(add-content baro-shadow
 (contingent (make-interval 0.36 0.37) '(shadow)))

(content bldg-height)
=> #(contingent #(interval 44.514 48.978) (shadow))
```

```
(define fall-time (make-cell))

(fall-duration fall-time bldg-height)

(add-content fall-time
 (contingent (make-interval 2.9 3.1) '(time)))

(content bldg-height)
=> #(contingent #(interval 44.514 47.243)
                (time shadow))
```

```
;;; An authoritative result
(add-content bldg-height (contingent 45 '(super)))

(content bldg-height)
=> #(contingent 45 (super))

;;; is reflected back into our measurements.
(content baro-height)
=> #(contingent #(interval .3 .30328)
                (super time shadow))

(content bldg-shadow)
=> #(contingent #(interval 54.9 55.1) (shadow))

(content fall-time)
=> #(contingent #(interval 3.0255 3.0322)
                (shadow super))
```

Truth Maintenance Systems

Locally-Consistent Worldviews

```
(define baro-height (make-cell))
(define baro-shadow (make-cell))
(define bldg-height (make-cell))
(define bldg-shadow (make-cell))

(similar-triangles baro-shadow baro-height
                   bldg-shadow bldg-height)

(add-content bldg-shadow
 (make-tms (contingent (make-interval 54.9 55.1)
                       '(shadow))))

(add-content baro-height
 (make-tms (contingent (make-interval 0.3 0.32)
                       '(shadow))))

(add-content baro-shadow
 (make-tms (contingent (make-interval 0.36 0.37)
                       '(shadow))))
```

```
(define fall-time (make-cell))

(fall-duration fall-time bldg-height)

(add-content fall-time
 (make-tms (contingent (make-interval 2.9 3.1)
                       '(time))))

(tms-query (content bldg-height))
=> #(contingent #(interval 44.514 47.243)
                (shadow time))
```

```
(kick-out! 'time)

(tms-query (content bldg-height))
=> #(contingent #(interval 44.514 48.978) (shadow))

(bring-in! 'time)
(kick-out! 'shadow)

(tms-query (content bldg-height))
=> #(contingent #(interval 41.163 47.243) (time))

(content bldg-height)
=> #(tms (#(contingent #(interval 41.163 47.243)
                       (time))
          #(contingent #(interval 44.514 47.243)
                       (shadow time))
          #(contingent #(interval 44.514 48.978)
                       (shadow))))
```

```
(add-content bldg-height (contingent 45 '(super)))

(bring-in! 'shadow)

(tms-query (content baro-height))
=> #(contingent #(interval .3 .30328)
                (time super shadow))

(kick-out! 'time)

(tms-query (content baro-height))
=> #(contingent #(interval .3 .30328) (super shadow))

(bring-in! 'time)

(tms-query (content baro-height))
=> #(contingent #(interval .3 .30328) (super shadow))
```

```
(add-content bldg-height
  (contingent (make-interval 46. 50.) '(pressure)))
=> (contradiction (super pressure))

(tms-query (content bldg-height))
=> #(contingent (contradiction) (super pressure))

(tms-query (content baro-height))
=> #(contingent #(interval .3 .30328) (super shadow))

(kick-out! 'super)

(tms-query (content bldg-height))
=> #(contingent #(interval 46. 47.243)
                (time pressure))

(tms-query (content baro-height))
=> #(contingent #(interval .30054 .31839)
                (pressure time shadow))
```

```
(bring-in! 'super)
(kick-out! 'pressure)

(tms-query (content bldg-height))
=> #(contingent 45 (super))

(tms-query (content baro-height))
=> #(contingent #(interval .3 .30328) (super shadow))
```

Truth Maintenance System In Every Cell

Implementation With Generic Merge

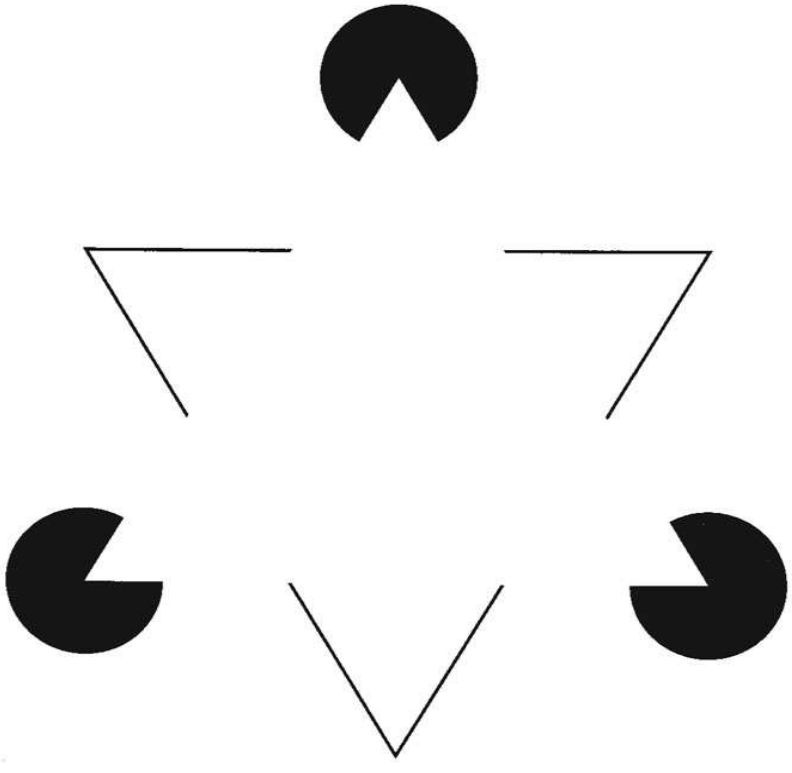
Don't Crash; Chuckle!

Dependency-Directed Backtracking

```
(define (multiple-dwelling)
  (let ((baker    (one-of 1 2 3 4 5))
        (cooper   (one-of 1 2 3 4 5))
        (fletcher (one-of 1 2 3 4 5))
        (miller   (one-of 1 2 3 4 5))
        (smith    (one-of 1 2 3 4 5)))
    (require-distinct
     (list baker cooper fletcher miller smith))
    (forbid (= baker 5))
    (forbid (= cooper 1))
    (forbid (= fletcher 5))
    (forbid (= fletcher 1))
    (require (> miller cooper))
    (forbid (= 1 (abs (- smith fletcher))))
    (forbid (= 1 (abs (- fletcher cooper))))
    (list baker cooper fletcher miller smith)))
```

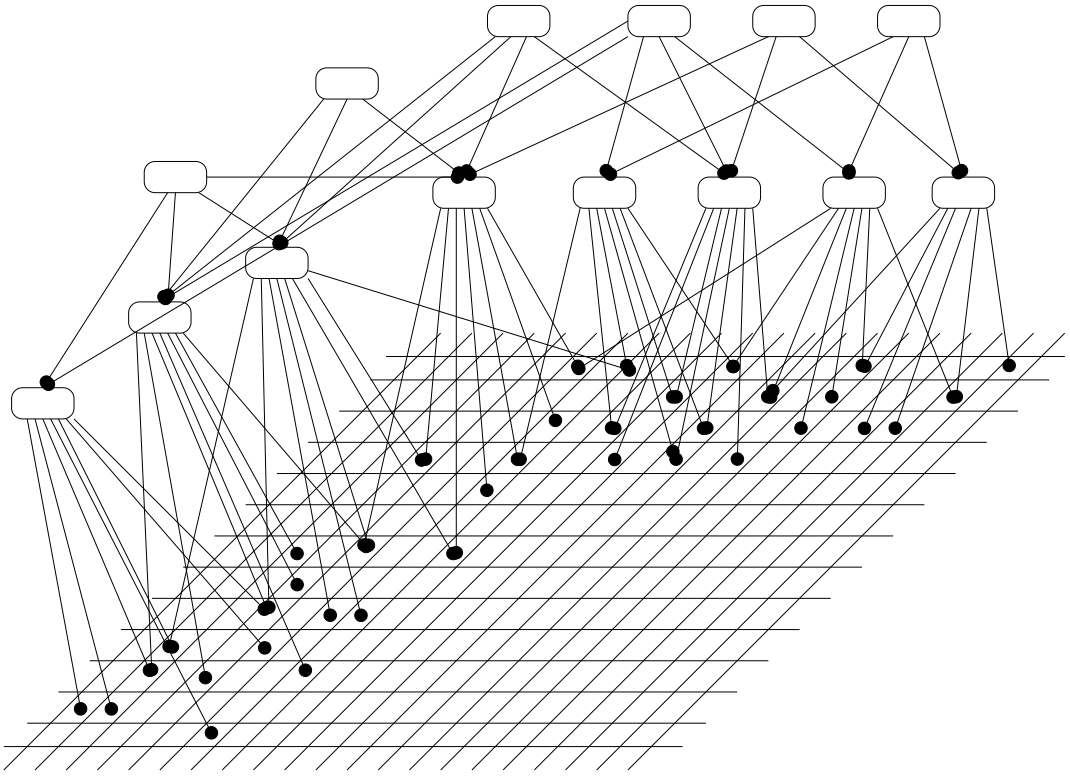
```
(define answers (multiple-dwelling))

(map v&s-value
     (map tms-query (map content answers)))
=> (3 2 4 5 1)

*number-of-calls-to-fail*
=> 63
```

Kanizsa's Triangle Illusion



Gaetano Kanizsa (1955)

?Propagator Constraint Architecture?



Summary

- The Problem is Evolvability not Correctness
- Extensible Generic Operations may help
  - Extension of function without modification
  - Makes Proofs very difficult
- Propagator Architecture unlocks freedoms
  - Essentially Concurrent and Parallel
  - Redundancy and Degeneracy
  - Maintain Provenance
  - Dependency-Directed Backtracking