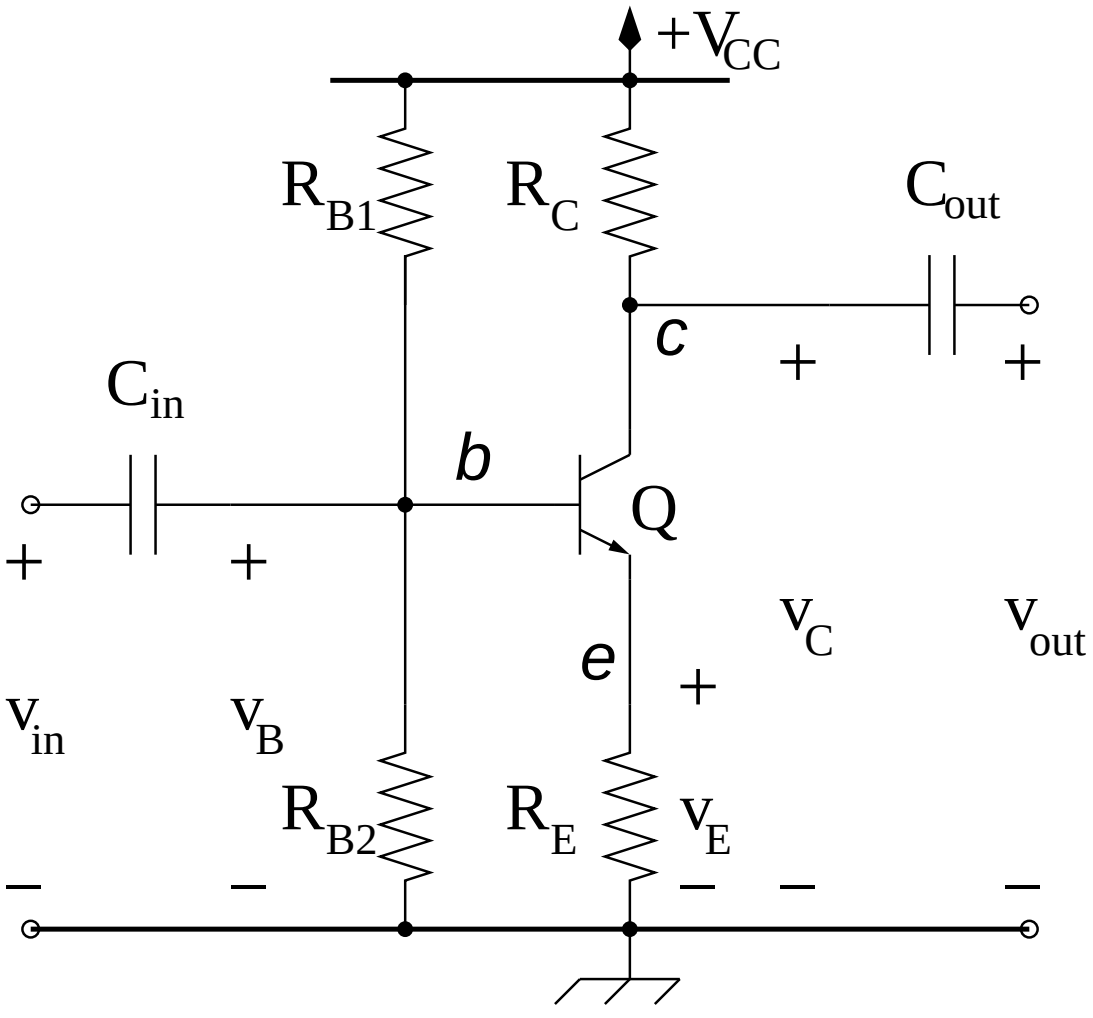
The Art of the Propagator

Alexey Radul and Gerald Jay Sussman

CSAIL and EECS

MIT


+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

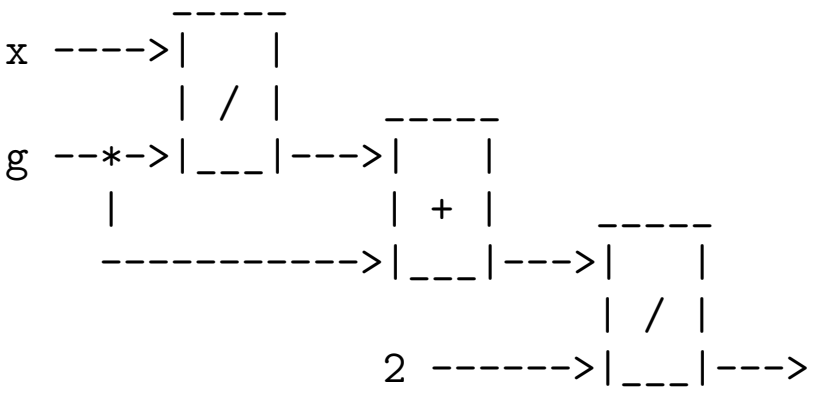
Independent Stateless Machines

Connecting Stateful Cells

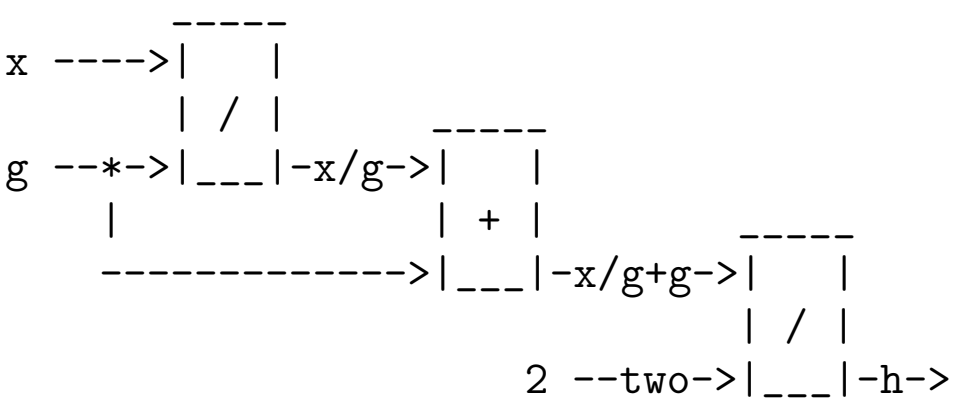
Expressions to Propagators

An expression has anonymous connections

```
(/ (+ (/ x g) g) 2)
```



In propagators, we make these explicit:




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

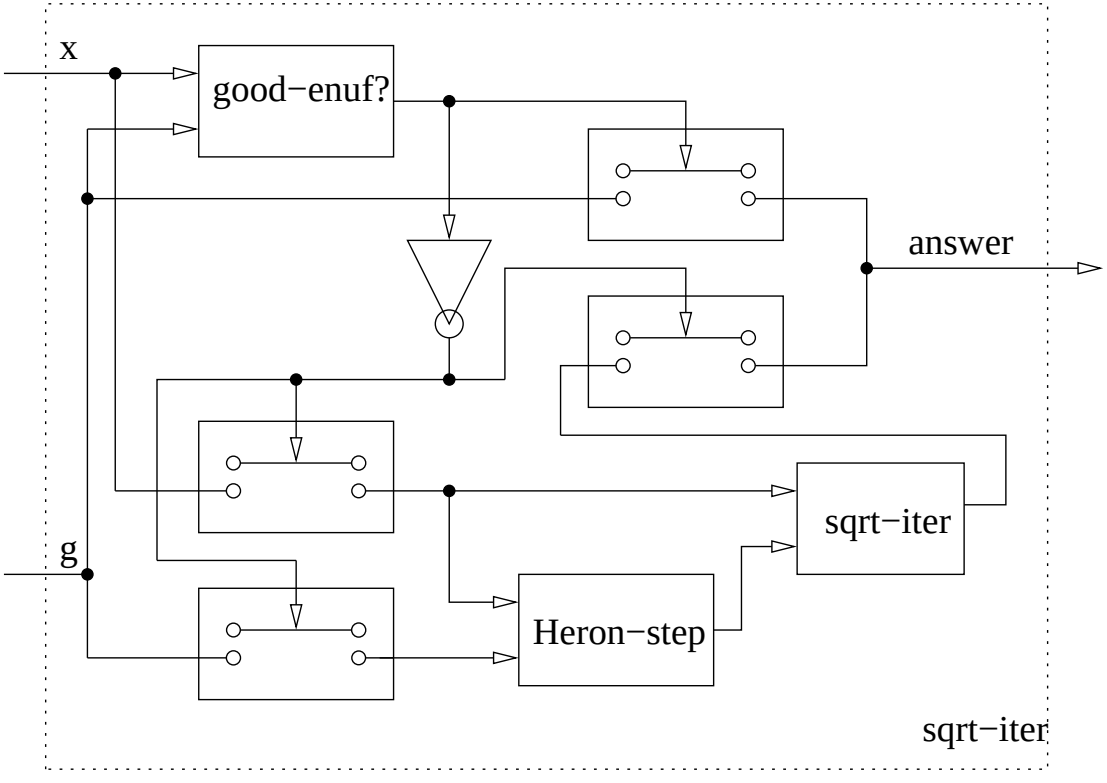
```
(initialize-scheduler)

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

Iteration: a Recursive Machine



Building is delayed until change on boundary.

```
(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)
(run)
(content answer)
  => 1.4142135623730951
```

How Propagators Work

Simulating in SCHEME

A First Cut at Implementation

```
(define (make-cell)
  (let ((neighbors '()) (content nothing))
    (define (new-neighbor! new-neighbor)
      (if (not (memq new-neighbor neighbors))
          (begin (set! neighbors
                       (cons new-neighbor neighbors))
                 (alert-propagators new-neighbor))))
    (define (add-content increment)
      (cond ((nothing? increment) 'ok)
            ((nothing? content)
             (set! content increment)
             (alert-propagators neighbors))
            (else (if (equal? content increment)
                      'ok
                      (error "Inconsistency!")))))
    (define (me message)
      (cond ((eq? message 'new-neighbor!) new-neighbor!)
            ((eq? message 'add-content) add-content)
            ((eq? message 'content) content)
            (else (error "Unknown message"))))
    me))
```

For Convenience...

```
(define (new-neighbor! cell neighbor)
  ((cell 'new-neighbor!) neighbor))

(define (add-content cell increment)
  ((cell 'add-content) increment))

(define (content cell)
  (cell 'content))
```

No information

```
(define nothing #(*the-nothing*))

(define (nothing? thing)
  (eq? thing nothing))
```

```
(define (propagator neighbors to-do)
  (for-each (lambda (cell)
              (new-neighbor! cell to-do))
            (listify neighbors))
  (alert-propagators to-do))

(define (handling-nothings f)
  (lambda args
    (if (any nothing? args)
        nothing
        (apply f args))))

(define (function->propagator-constructor f)
  (lambda cells
    (let ((output (car (last-pair cells)))
          (inputs (except-last-pair cells)))
      (propagator inputs
        (lambda ()
          (add-content output
            (apply f (map content inputs))))))))
```

Some primitive Propagators

```
(define p:+
  (function->propagator-constructor
   (handling-nothings +)))
(define p:-
  (function->propagator-constructor
   (handling-nothings -)))
(define p:*
  (function->propagator-constructor
   (handling-nothings *)))

;;; ... many more ...

(define (switch-function control input)
  (if control input nothing))

(define switch
  (function->propagator-constructor
   (handling-nothings switch-function)))
```

Compound Propagators

```
(define (delayed-propagator builder)
  (lambda cells
    (one-shot-prop cells
      (lambda ()
        (apply builder cells)))))

(define (one-shot-prop neighbors action)
  (let ((done? #f)
        (neighbors (listify neighbors)))
    (define (test)
      (if done?
          'ok
          (if (every nothing?
                     (map content neighbors))
              'ok
              (begin (set! done? #t)
                     (action)))))
    (propagator neighbors test)))
```

Propagators to 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))))
```

Monotonic Information

The Power of Merge

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))
(run)
(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))
(run)
(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 (make-interval 45 45))
(run)
(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)
```

All Operators are Extensible Generics

Cells **Merge** Information **Monitonically**

make-cell with generic merge

```
(define (make-cell)
  (let ((neighbors '()) (content nothing))
    (define (new-neighbor! new)
      (if (not (memq new neighbors))
          (begin (set! neighbors (cons new neighbors))
                 (alert-propagators new))))
    (define (add-content increment)  ; ***
      (let ((answer (merge content increment)))
        (cond ((eq? answer content) 'ok)
              ((contradictory? answer)
               (error "Ack! Inconsistency!"))
              (else (set! content answer)
                    (alert-propagators neighbors)))))
    (define (me message)
      (cond ((eq? message 'new-neighbor!) new-neighbor!)
            ((eq? message 'add-content) add-content)
            ((eq? message 'content) content)
            (else (error "Unknown message"))))
    me))
```

```
(define merge
  (make-generic-operator 2 'merge
   (lambda (content increment)
     (if (equal? content increment)
         content
         the-contradiction))))

(define the-contradiction
         (list 'contradiction))

(define contradictory?
  (make-generic-operator 1 'contradictory?
   (lambda (x) (eq? x the-contradiction))))
```

```
(defhandler merge
 (lambda (content increment) content)
 any? nothing?)

(defhandler merge
 (lambda (content increment) increment)
 nothing? any?)

;;; and we can add intervals
(defhandler merge
 (lambda (content increment)
   (let ((new-range
          (intersect-intervals content increment)))
     (cond ((interval-equal? new-range content)
            content)
           ((interval-equal? new-range increment)
            increment)
           (else new-range))))
 interval? interval?)
```

Intervals interoperate with numbers

```
(defhandler merge
 (lambda (content increment)
   (ensure-inside increment content))
 number? interval?)

(defhandler merge
 (lambda (content increment)
   (ensure-inside content increment))
 interval? number?)

(define (ensure-inside interval number)
  (if (<= (interval-low interval)
          number
          (interval-high interval))
      number
      the-contradiction))
```

Dependencies and Supported Values

Tracking of Provenance

```
(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
 (supported (make-interval 54.9 55.1) '(shadows)))
(add-content baro-height
 (supported (make-interval 0.3 0.32) '(shadows)))
(add-content baro-shadow
 (supported (make-interval 0.36 0.37) '(shadows)))

(run)

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

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

(fall-duration fall-time bldg-height)

(add-content fall-time
 (supported (make-interval 2.9 3.3) '(lousy-fall-time)))

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


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

(content bldg-height)
=> #(supported #(interval 44.514 47.243)
            (better-fall-time shadows))
```

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

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

;;; is reflected back into our measurements.
(content baro-height)
=> #(supported #(interval .3 .30328)
        (super better-fall-time shadows))

(content baro-shadow)
=> #(supported #(interval .366 .37)
        (better-fall-time super shadows))

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

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

Merging Supported Values

```
;;; Merging supported values

(define (v&s-merge v&s1 v&s2)
  (let* ((v&s1-value (v&s-value v&s1))
         (v&s2-value (v&s-value v&s2))
         (value-merge (merge v&s1-value v&s2-value)))
    (cond ((eq? value-merge v&s1-value)
           (if (implies? v&s2-value value-merge)
               (if (more-informative-support? v&s2 v&s1)
                   v&s2
                   v&s1)
               v&s1))
          ((eq? value-merge v&s2-value) v&s2)
          (else
           (supported value-merge
               (merge-supports v&s1 v&s2))))))

(defhandler merge v&s-merge v&s? v&s?)
```

```
;;; Where

(define (implies? v1 v2)
  (eq? v1 (merge v1 v2)))

(define (more-informative-support? v&s1 v&s2)
  (proper-subset? (v&s-support v&s1)
                  (v&s-support v&s2)))



;;; And contradiction detection

(defhandler contradictory?
 (lambda (v&s)
   (contradictory? (v&s-value v&s)))
 v&s?)
```

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 (supported (make-interval 54.9 55.1) '(shadows))))

(add-content baro-height
 (make-tms (supported (make-interval 0.3 0.32) '(shadows))))

(add-content baro-shadow
 (make-tms (supported (make-interval 0.36 0.37) '(shadows))))

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

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

(fall-duration fall-time bldg-height)

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

(content bldg-height)
=> #(tms (#(supported #(interval 44.514 47.243)
                      (shadows fall-time))
          #(supported #(interval 44.514 48.978)
                      (shadows))))

(tms-query (content bldg-height))
=> #(supported #(interval 44.514 47.243)
                (shadows fall-time))
```

```
(kick-out! 'fall-time)

(tms-query (content bldg-height))
=> #(supported #(interval 44.514 48.978) (shadows))

(bring-in! 'fall-time)
(kick-out! 'shadows)

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

(content bldg-height)
=> #(tms (#(supported #(interval 41.163 47.243)
                      (fall-time))
          #(supported #(interval 44.514 47.243)
                      (shadows fall-time))
          #(supported #(interval 44.514 48.978)
                      (shadows))))
```

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

(content bldg-height)
=> #(tms (#(supported 45 (super))
          #(supported #(interval 41.163 47.243)
                      (fall-time))
          #(supported #(interval 44.514 47.243)
                      (shadows fall-time))
          #(supported #(interval 44.514 48.978)
                      (shadows))))

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

```
(bring-in! 'shadows)

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

(content baro-height)
=> #(tms (#(supported #(interval .3 .30328)
                      (fall-time super shadows))
          #(supported #(interval .29401 .30328)
                      (super shadows))
          #(supported #(interval .3 .31839)
                      (fall-time shadows))
          #(supported #(interval .3 .32)
                      (shadows))))

(tms-query (content baro-height))
=> #(supported #(interval .3 .30328)
                (fall-time super shadows))
```

```
(kick-out! 'fall-time)

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

(bring-in! 'fall-time)

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

(content baro-height)
=> #(tms (#(supported #(interval .3 .30328)
                      (super shadows))
          #(supported #(interval .3 .31839)
                      (fall-time shadows))
          #(supported #(interval .3 .32)
                      (shadows))))
```

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

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

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

```
(kick-out! 'super)

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

(tms-query (content baro-height))
=> #(supported #(interval .30054 .31839)
               (pressure fall-time shadows))

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

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

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

Truth Maintenance System In Every Cell

Implementation With Generic Merge

```
(define (tms-merge tms1 tms2)
  (let ((candidate (tms-assimilate tms1 tms2)))
    (let ((consequence
           (strongest-consequence candidate)))
      (check-consistent! consequence)
      (tms-assimilate candidate consequence))))

(defhandler merge tms-merge tms? tms?)

(define (tms-assimilate tms stuff)
  (cond ((nothing? stuff) tms)
        ((v&s? stuff)
         (tms-assimilate-one tms stuff))
        ((tms? stuff)
         (fold-left tms-assimilate-one
                    tms
                    (tms-values stuff)))
        (else (error "This should never happen"))))
```

```
(define (subsumes? v&s1 v&s2)
  (and (implies? (v&s-value v&s1) (v&s-value v&s2))
       (subset? (v&s-support v&s1) (v&s-support v&s2))))

(define (tms-assimilate-one tms v&s)
  (if (any (lambda (old-v&s) (subsumes? old-v&s v&s))
           (tms-values tms))
      tms
      (let ((subsumed
             (filter (lambda (old-v&s)
                       (subsumes? v&s old-v&s))
                     (tms-values tms))))
        (make-tms
         (set-adjoin
          (set-difference (tms-values tms) subsumed)
          v&s)))))
```

```
(define (strongest-consequence tms)
  (let ((relevant-v&ss
         (filter v&s-believed? (tms-values tms))))
    (fold-left merge nothing relevant-v&ss)))

(define (v&s-believed? v&s)
   (every premise-in? (v&s-support v&s)))

(define (check-consistent! v&s)
  (if (contradictory? v&s)
      (process-nogood! (v&s-support v&s))))

(define (tms-query tms)
  (let ((answer (strongest-consequence tms)))
    (let ((better-tms (tms-assimilate tms answer)))
      (if (not (eq? tms better-tms))
          (set-tms-values! tms (tms-values better-tms)))
      (check-consistent! answer)
      answer)))
```

```
(define (kick-out! premise)
  (if (premise-in? premise)
      (alert-all-propagators!))
  (mark-premise-out! premise))

(define (bring-in! premise)
  (if (not (premise-in? premise))
      (alert-all-propagators!))
  (mark-premise-in! premise))

(define (process-nogood! nogood)
  (abort-process `(contradiction ,nogood)))
```

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
```

```
(define *number-of-calls-to-fail* 0)

(define (process-nogood! nogood)
  (set! *number-of-calls-to-fail*
        (+ *number-of-calls-to-fail* 1))
  (process-one-contradiction nogood))

(define (process-one-contradiction nogood)
  (let ((hyps (filter hypothetical? nogood)))
    (if (null? hyps)
        (abort-process `(contradiction ,nogood))
        (begin (kick-out! (car hyps))
          (for-each
           (lambda (premise)
             (assimilate-nogood! premise nogood))
           nogood)))))
```

```
(define (assimilate-nogood! premise new-nogood)
  (let ((item (delq premise new-nogood))
        (set (premise-nogoods premise)))
    (if (any (lambda (old) (subset? old item)) set)
        #f
        (let ((subsumed
               (filter (lambda (old)
                         (subset? item old))
                       set)))
          (set-premise-nogoods! premise
            (set-adjoin
              (set-difference set subsumed)
              item))))))
```

```
(define (require cell)
  ((constant #t) cell))

(define (forbid cell)
  ((constant #f) cell))

(define (require-distinct cells)
  (for-each-distinct-pair
   (lambda (c1 c2)
     (let ((p (make-cell)))
       (=? c1 c2 p)
       (forbid p)))
   cells))
```

```
(define (one-of values output-cell)
  (let ((cells
         (map (lambda (value)
                (let ((cell (make-cell)))
                  ((constant value) cell)
                  cell))
              values)))
    (one-of-the-cells cells output-cell)))

(define (one-of-the-cells input-cells output-cell)
  (cond ((= (length input-cells) 2)
         (let ((p (make-cell)))
           (conditional p
                        (car input-cells)
                        (cadr input-cells)
                        output-cell)
           (binary-amb p)))
        ((> (length input-cells) 2)
         ;...)
        (else
         (error "Inadequate choices"
```

```
(define (binary-amb cell)
  (let ((true-premise  (make-hypothetical))
        (false-premise (make-hypothetical)))
    (define (amb-choose)
      (let ((reasons-against-true
             (filter all-premises-in? (premise-nogoods true-premise)))
            (reasons-against-false
             (filter all-premises-in? (premise-nogoods false-premise))))
        (cond ((null? reasons-against-true)
               (kick-out! false-premise) (bring-in! true-premise))
              ((null? reasons-against-false)
               (kick-out! true-premise) (bring-in! false-premise))
              (else        ; this amb must fail.
               (kick-out! true-premise) (kick-out! false-premise)
               (process-contradictions
                (pairwise-union reasons-against-true
                                reasons-against-false))))))
    ((constant
      (make-tms (list (supported #t (list true-premise))
                      (supported #f (list false-premise)))))
     cell)
    (propagator cell amb-choose)))
```

```
(define (process-contradictions nogoods)
  (process-one-contradiction
   (car (sort-by nogoods
          (lambda (nogood)
            (length
             (filter hypothetical? nogood)))))))
```