Logic Programming: The Query Evaluator

Johan Fabry - jfabry @dcc

Acknowledgments

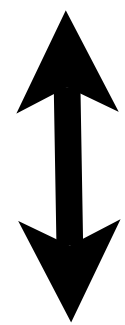
- Theo D'Hondt (Dutch version slides online) http://prog.vub.ac.be/~tjdhondt
- Structure and Interpretation of Computer Programs Chapter 4.4 (Fulltext online) http://mitpress.mit.edu/sicp

Logic Programming

Use

- Assertions
- Rules
- Queries

Versus



Implementation

- Streams of Frames
- Pattern Recognition
- Unification

Assertions

- An assertion is an n-tuple of symbols and/or assertions
- A symbol is an atomic value with an unique identity
- The first element of an assertion (usually) is a symbol that qualifies (=identifies) the assertion
- In the context of the assertion the used symbols acquire a meaning

Assertions Example

```
(is-planet Mars)
(has-distance-to-sun Mars 1.5237)
(has-mass Mars 0.109)
(has-diameter Mars 6790)
(has-roundtriptime Mars 1.881)
(has-rotationtime Mars + 1.03)
(is-moon-of Phobos Mars)
(is-moon-of Deimos Mars)
(has-diameter Phobos 22)
(has-diameter Deimos 8)
(is-discovered Phobos 1877 Hall)
(is-discovered Deimos 1877 Hall)
```

Queries

- A query is an n-tuple of variables, symbols and/or queries
- A variable is a symbolic value starting with ?
- Evaluating a query within a set of assertions gives rise to the subset of assertions of which parts correspond with the elements of the query, after substitution of variables by symbols or assertions

Query example

```
(is-planet Phobos)
 => { }

(is-moon-of Phobos ?planet)
 => { (is-moon-of phobos mars) }

(is-moon-of ?moon Mars)
 => { (is-moon-of deimos mars) ,
      (is-moon-of phobos mars) }
```

Test Database

```
(is-planeet Mercurius)
(is-planeet Venus    )
(is-planeet Aarde    )
(is-planeet Mars     )
(is-planeet Jupiter  )
(is-planeet Saturnus )
(is-planeet Uranus   )
(is-planeet Neptunus )
(is-planeet Pluto    )

(gebruikt-eenheid heeft-afstand-tot-zon AE)

(heeft-afstand-tot-zon Mercurius 0.3871)
(heeft-afstand-tot-zon Venus     0.7233)
(heeft-afstand-tot-zon Aarde     1.0000)
(heeft-afstand-tot-zon Mars      1.5237)
(heeft-afstand-tot-zon Jupiter   5.2028)
(heeft-afstand-tot-zon Saturnus  9.5388)
(heeft-afstand-tot-zon Uranus   19.1819)
(heeft-afstand-tot-zon Neptunus 30.0578)
(heeft-afstand-tot-zon Pluto    39.2975)
```

(is-planet <planet>)

(has-distance-to-sun <planet><dist>)

Test Database

```
(gebruikt-eenheid heeft-massa Aard-massa)

(heeft-massa Mercurius   0.053)
(heeft-massa Venus       0.815)
(heeft-massa Aarde       1.000)
(heeft-massa Mars        0.109)
(heeft-massa Jupiter   317.900)
(heeft-massa Saturnus   95.100)
(heeft-massa Uranus     14.500)
(heeft-massa Neptunus   17.500)
(heeft-massa Pluto       1.000)

(gebruikt-eenheid heeft-middellijn KM)

(heeft-middellijn Mercurius   4840)
(heeft-middellijn Venus      12200)
(heeft-middellijn Aarde      12756)
(heeft-middellijn Mars        6790)
(heeft-middellijn Jupiter   142800)
(heeft-middellijn Saturnus  119300)
(heeft-middellijn Uranus     47100)
(heeft-middellijn Neptunus   44800)
(heeft-middellijn Pluto       5000)
```

(has-mass <planet> <mass>)

(has-diameter <planet> <diameter>)

Test Database

```
(gebruikt-eenheid heeft-omlooptijd Aard-jaar)

(heeft-omlooptijd Mercurius   0.241)
(heeft-omlooptijd Venus       0.615)
(heeft-omlooptijd Aarde       1.000)
(heeft-omlooptijd Mars        1.881)
(heeft-omlooptijd Jupiter    11.862)
(heeft-omlooptijd Saturnus   29.458)
(heeft-omlooptijd Uranus     84.013)
(heeft-omlooptijd Neptunus  164.793)
(heeft-omlooptijd Pluto     248.430)

(gebruikt-eenheid heeft-rotatietijd Aard-dag)

(heeft-rotatietijd Mercurius   +  58.79)
(heeft-rotatietijd Venus       - 243.68)
(heeft-rotatietijd Aarde       +   1.00)
(heeft-rotatietijd Mars        +   1.03)
(heeft-rotatietijd Jupiter     +   0.41)
(heeft-rotatietijd Saturnus    +   0.43)
(heeft-rotatietijd Uranus      -   0.45)
(heeft-rotatietijd Neptunus    +   0.63)
(heeft-rotatietijd Pluto       +   0.26)
```

(has-roundtriptime <planet> <time>)

(has-rotationtime <planet><dir><time>)

Test Database

```
(is-maan-van Maan        Aarde   )
(is-maan-van Phobos      Mars    )
(is-maan-van Deimos      Mars    )
(is-maan-van Io          Jupiter )
(is-maan-van Europa      Jupiter )
(is-maan-van Ganymedes   Jupiter )
(is-maan-van Callisto    Jupiter )
(is-maan-van Mimas       Saturnus)
(is-maan-van Enceladus   Saturnus)
(is-maan-van Tethys      Saturnus)
(is-maan-van Dione       Saturnus)
(is-maan-van Rhea        Saturnus)
(is-maan-van Titan       Saturnus)
(is-maan-van Hyperion    Saturnus)
(is-maan-van Japetus     Saturnus)
(is-maan-van Phoebe      Saturnus)
(is-maan-van Janus       Saturnus)
(is-maan-van Ariel       Uranus  )
(is-maan-van Umbriel     Uranus  )
(is-maan-van Titania     Uranus  )
(is-maan-van Oberon      Uranus  )
(is-maan-van Miranda     Uranus  )
(is-maan-van Triton      Neptunus)
(is-maan-van Nereide     Neptunus)
```

(is-moon-of <moon><plan>)

Test Database

```
(heeft-middellijn Maan      3476)
(heeft-middellijn Phobos      22)
(heeft-middellijn Deimos       8)
(heeft-middellijn Io        3550)
(heeft-middellijn Europa    3100)
(heeft-middellijn Ganymedes 5600)
(heeft-middellijn Callisto  5050)
(heeft-middellijn Mimas      520)
(heeft-middellijn Enceladus  600)
(heeft-middellijn Tethys    1200)
(heeft-middellijn Dione     1300)
(heeft-middellijn Rhea      1300)
(heeft-middellijn Titan     4950)
(heeft-middellijn Hyperion   400)
(heeft-middellijn Japetus   1200)
(heeft-middellijn Phoebe     300)
(heeft-middellijn Janus      350)
(heeft-middellijn Ariel      600)
(heeft-middellijn Umbriel    400)
(heeft-middellijn Titania   1000)
(heeft-middellijn Oberon     800)
(heeft-middellijn Miranda    100)
(heeft-middellijn Triton    4000)
(heeft-middellijn Nereide    300)
```

(has-diameter <moon> <diameter>)

Test Database

```
(is-ontdekt Phobos    1877 Hall     )
(is-ontdekt Deimos    1877 Hall     )
(is-ontdekt Io        1610 Galilei  )
(is-ontdekt Europa    1610 Galilei  )
(is-ontdekt Ganymedes 1610 Galilei  )
(is-ontdekt Callisto  1610 Galilei  )
(is-ontdekt Mimas     1789 Herschel )
(is-ontdekt Enceladus 1789 Herschel )
(is-ontdekt Tethys    1684 Cassini  )
(is-ontdekt Dione     1684 Cassini  )
(is-ontdekt Rhea      1672 Cassini  )
(is-ontdekt Titan     1655 Huygens  )
(is-ontdekt Hyperion  1848 Bond     )
(is-ontdekt Japetus   1671 Cassini  )
(is-ontdekt Phoebe    1898 Pickering)
(is-ontdekt Janus     1966 Dolfus   )
(is-ontdekt Ariel     1851 Lassell  )
(is-ontdekt Umbriel   1851 Lassell  )
(is-ontdekt Titania   1787 Herschel )
(is-ontdekt Oberon    1787 Herschel )
(is-ontdekt Miranda   1948 Kuiper   )
(is-ontdekt Triton    1846 Lassell  )
(is-ontdekt Nereide   1949 Kuiper   )
```

(is-discovered <moon> <year> <name>)

Test Query

```
(and (is-planeet ?planeet)
     (is-maan-van ?maan ?planeet)
     (is-ontdekt ?maan ?jaar Galilei))
=>
{ (and (is-planeet jupiter)
       (is-maan-van callisto jupiter)
       (is-ontdekt callisto 1610 galilei)) ,
  (and (is-planeet jupiter)
       (is-maan-van ganymedes jupiter)
       (is-ontdekt ganymedes 1610 galilei)) ,
  (and (is-planeet jupiter)
       (is-maan-van europa jupiter)
       (is-ontdekt europa 1610 galilei)) ,
  (and (is-planeet jupiter)
       (is-maan-van io jupiter)
       (is-ontdekt io 1610 galilei)) }
```

Galilei?

Antwoord: Jupiter

Filters

```
lisp-value =
lisp-value >
lisp-value <
lisp-value >=
lisp-value <=
```

- Filters are applied via predicates that compare and/or manipulate elements of assertions
- Filters are inherited from the meta-level (i.e. Scheme)

Filters Example

```
(and (is-planeet ?planeet)
     (heeft-massa ?planeet ?massa)
     (lisp-value > ?massa 5))
=>

{ (and (is-planeet neptunus)
       (heeft-massa neptunus 17.5)
       (lisp-value > 17.5 5)) ,
  (and (is-planeet uranus)
       (heeft-massa uranus 14.5)
       (lisp-value > 14.5 5)) ,
  (and (is-planeet saturnus)
       (heeft-massa saturnus 95.1)
       (lisp-value > 95.1 5)) ,
  (and (is-planeet jupiter)
       (heeft-massa jupiter 317.9)
       (lisp-value > 317.9 5)) }
```

Antwoord: Neptunus, Uranus, Saturnus, Jupiter

Filters Example

```
(and (is-planeet ?planeet)
     (is-maan-van ?maan ?x)
     (heeft-middellijn ?planeet ?middellijn-p)
     (heeft-middellijn ?maan ?middellijn-m)
     (lisp-value > ?middellijn-m ?middellijn-p))
=>
{ (and (is-planeet pluto)
       (is-maan-van callisto jupiter)
       (heeft-middellijn pluto 5000)
       (heeft-middellijn callisto 5050)
       (lisp-value > 5050 5000)) ,
  (and (is-planeet pluto)
       (is-maan-van ganymedes jupiter)
       (heeft-middellijn pluto 5000)

       (heeft-middellijn mercurius 4840)
       (heeft-middellijn callisto 5050)
       (lisp-value > 5050 4840)) ,
  (and (is-planeet mercurius)
       (is-maan-van ganymedes jupiter)
       (heeft-middellijn mercurius 4840)
       (heeft-middellijn ganymedes 5600)
       (lisp-value > 5600 4840)) }
```

middellijn dan minstens een maan?

Antwoord: Pluto en Mercurius

Rules

- A rule determines a logic inference of the form: conclusion results from assumption
- An assumption has the form of a query
- A conclusion is a pattern that has the form of a query with only variable components, except for the query (usually)

Rule Example

```
(rule (ml=  ?planeet ?middellijn ?e)
   (and (is-planeet ?planeet)
      (heeft-middellijn ?planeet ?middellijn)
      (gebruikt-eenheid heeft-middellijn ?e))

(ml= Jupiter ?ml ?e)
=>
{ (ml= jupiter 142800 km) }
```

Rules II

- Variables from the conclusion usually also are present in the assumption
- Other variables may be present in the assumption
- The rule represents all possible values of variables from the conclusion for which assertions or rules can be found or constructed that satisfy the assumption
- If the assumption is empty, the conclusion is always true

Rules Example

```
(rule (ml=  ?planeet ?middellijn ?e)
   (and (is-planeet ?planeet)
      (heeft-middellijn ?planeet ?middellijn)
      (gebruikt-eenheid heeft-middellijn ?e)))

(ml= Jupiter ?ml ?e)
=>
{ (ml= jupiter 142800 km) }

(ml= ?pl ?ml ?e)
=>
{ (ml= pluto 5000 km) ,
  (ml= neptunus 44800 km) ,
  (ml= uranus 47100 km) ,
  (ml= saturnus 119300 km) ,
  (ml= jupiter 142800 km) ,
  (ml= mars 6790 km) ,
  (ml= aarde 12756 km) ,
  (ml= venus 12200 km) ,
  (ml= mercurius 4840 km) }

(ml= ?pl 5000 ?e)
=>
{ (ml= pluto 5000 km) }
```

Grouping existing assertions into 1

Rules Example

```
(rule (alles-over ?planeet
                  ?massa
                  ?middellijn
                  ?omlooptijd
                  ?rotatiezin
                  ?rotatietijd
                  ?afstand-tot-zon)

(and (is-planeet ?planeet)
     (heeft-massa ?planeet
                  ?massa)
     (heeft-middellijn ?planeet
                       ?middellijn)
     (heeft-omlooptijd ?planeet
                       ?omlooptijd)
     (heeft-rotatietijd ?planeet
                        ?rotatiezin
                        ?rotatietijd)
     (heeft-afstand-tot-zon ?planeet
                            ?afstand-tot-zon))))

(alles-over Jupiter ?ms ?ml ?ot ?rz ?rt ?az)
=>
{ (alles-over jupiter 317.9 142800 11.862 + .41 5.2028) }
```

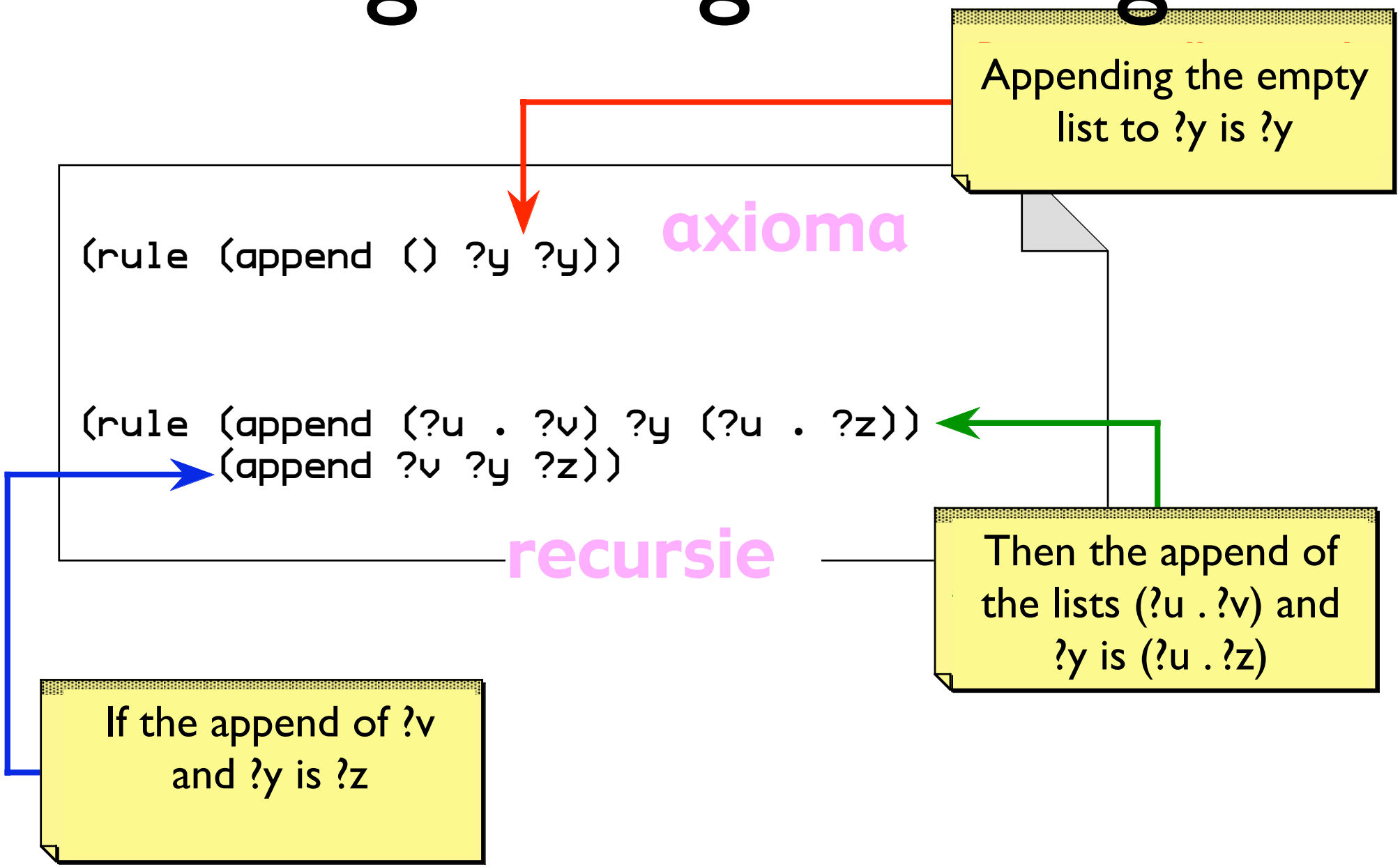
Rapporting all about something

Rules Example

```
(rule (ontdekt-in-de-17e-eeuw ?maan ?pe
      (and (is-ontdekt ?maan ?jaar ?persoon)
           (lisp-value > ?jaar 1599)
           (lisp-value < ?jaar 1700))))

(ontdekt-in-de-17e-eeuw ?maan ?persoon)
=>
{ (ontdekt-in-de-17e-eeuw japetus cassini) ,
  (ontdekt-in-de-17e-eeuw titan huygens) ,
  (ontdekt-in-de-17e-eeuw rhea cassini) ,
  (ontdekt-in-de-17e-eeuw dione cassini) ,
  (ontdekt-in-de-17e-eeuw tethys cassini) ,
  (ontdekt-in-de-17e-eeuw callisto galilei) ,
  (ontdekt-in-de-17e-eeuw ganymedes galilei) ,
  (ontdekt-in-de-17e-eeuw europa galilei) ,
  (ontdekt-in-de-17e-eeuw io galilei) }
```

Procedure-like fixing of possible queries

Logic Programming

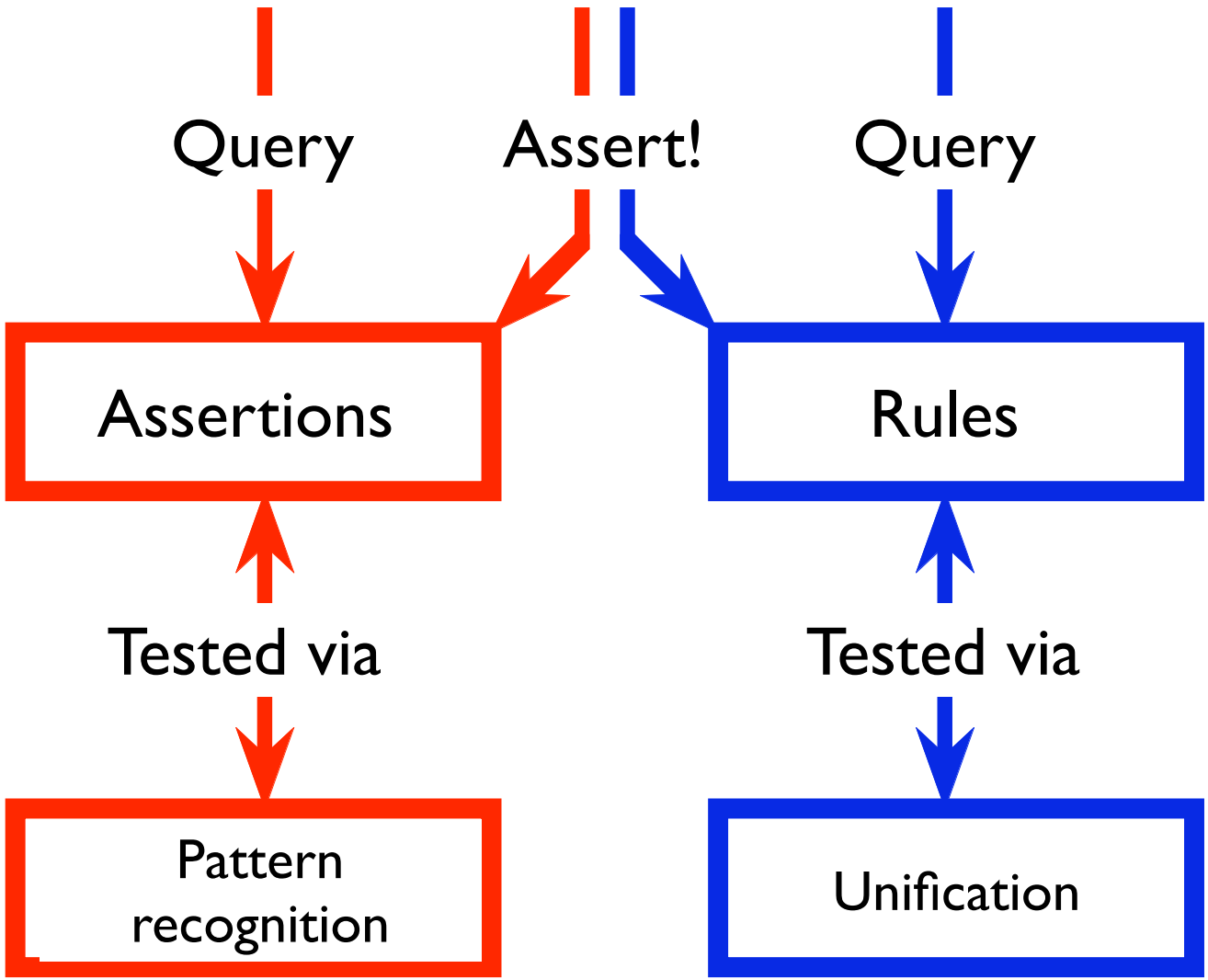


Logic Programming

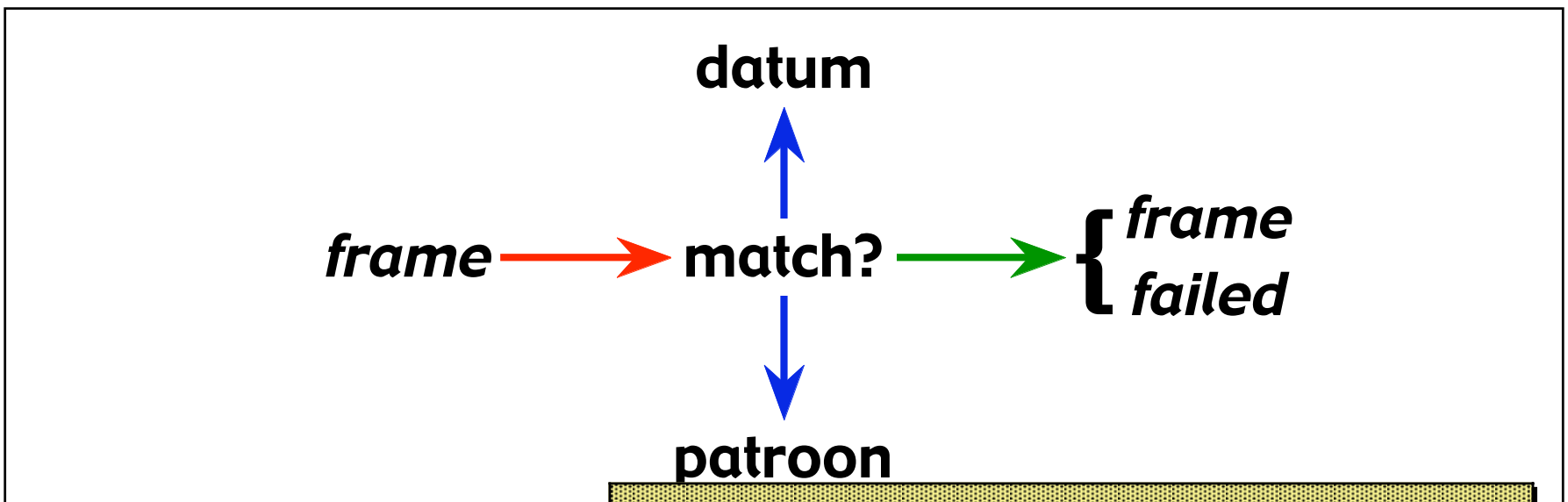
```
(append (a b) (c d) ?z)
=>
{ (append (a b) (c d) (a b c d)) }

(append (a b) ?y (a b c d))
=>
{ (append (a b) (c d) (a b c d)) }

(append ?x ?y (a b c d))
=>
{ (append (a b c d) () (a b c d)) ,
  (append (a b c) (d) (a b c d)) ,
  (append (a b) (c d) (a b c d)) ,
  (append (a) (b c d) (a b c d)) ,
  (append () (a b c d) (a b c d)) }
```

Works left-to-right, right-to-left and in both directions!

A Query System

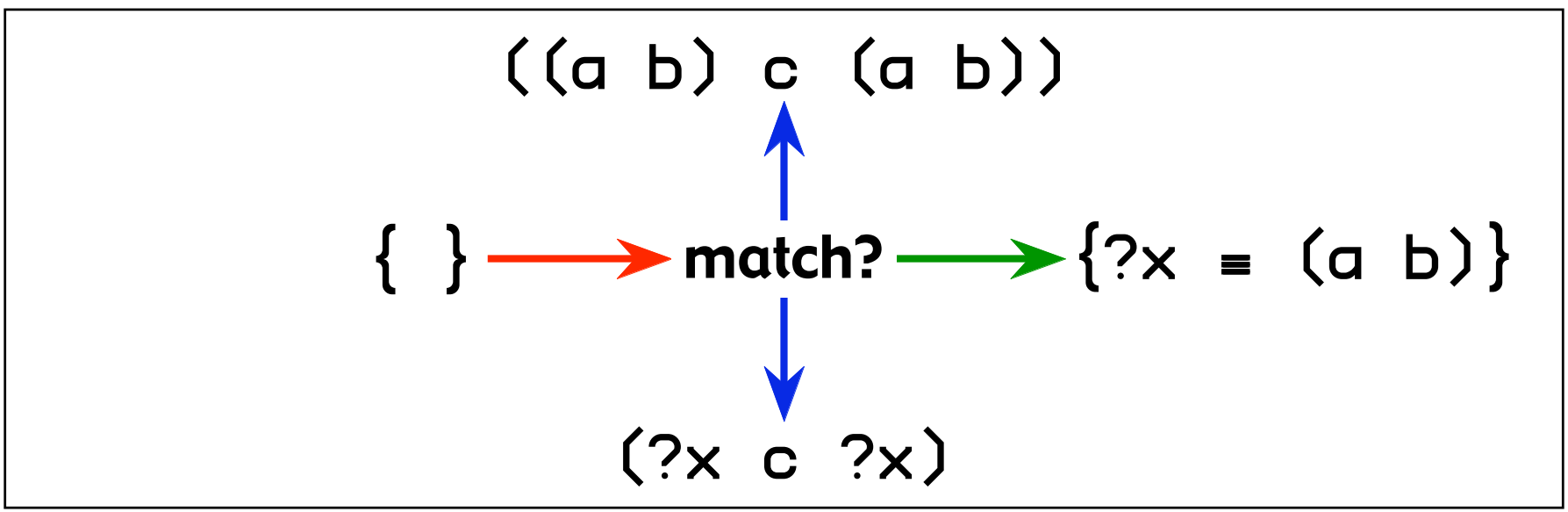


Pattern Matching



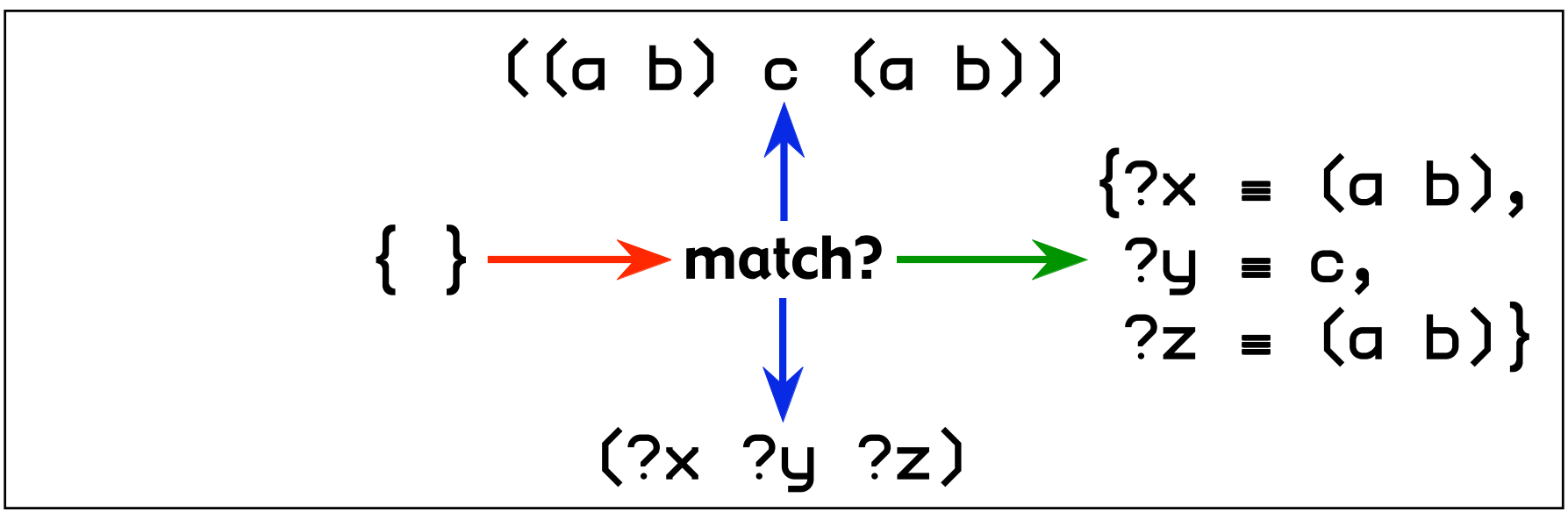
an atomic pattern recognizer that tests an elementary query to a elementary assertion, taking a *frame* of existing bindings into account

Pattern Matching



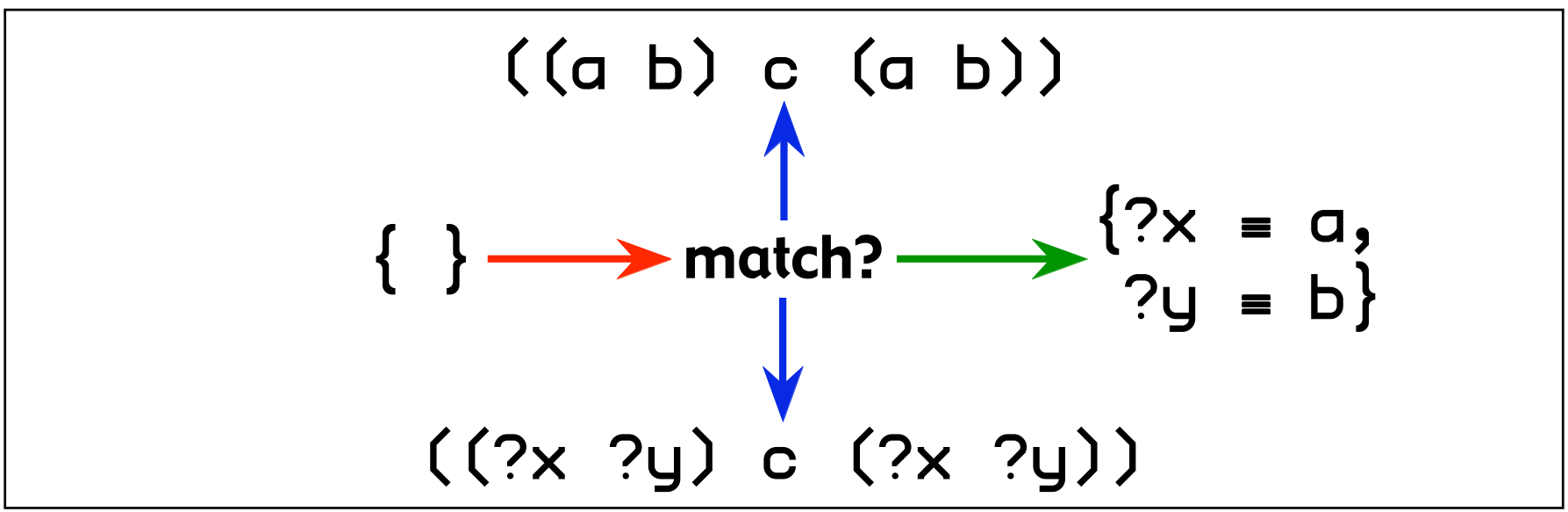
from zero to one binding

Pattern Matching



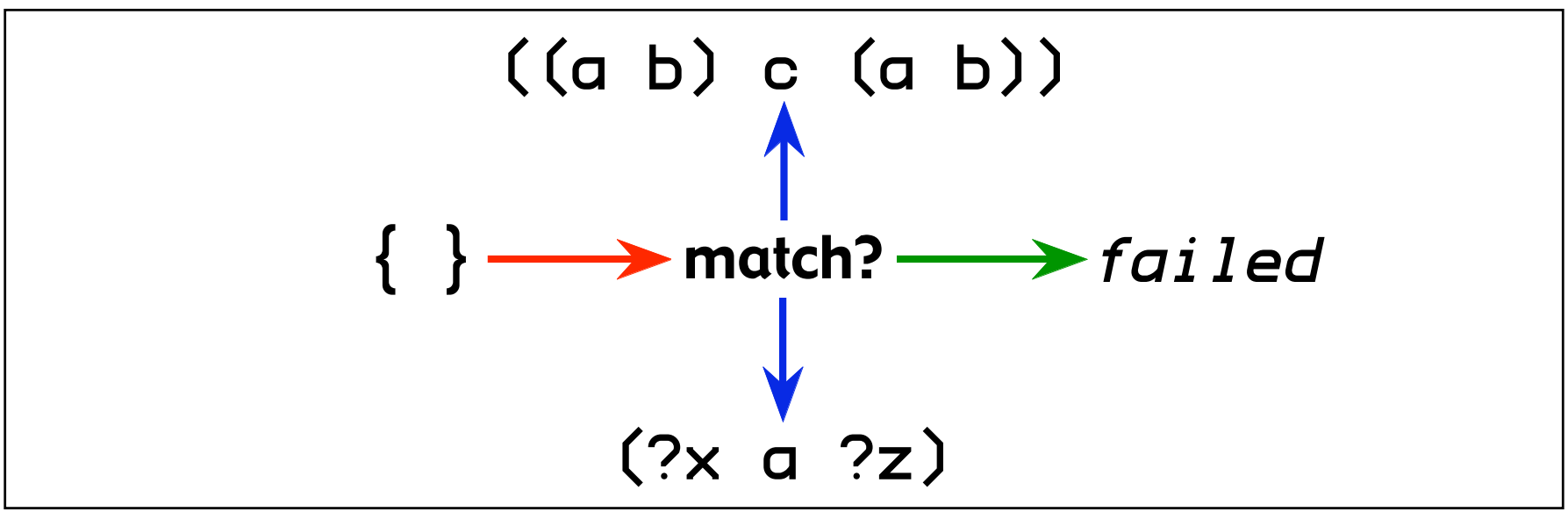
from zero to three bindings

Pattern Matching



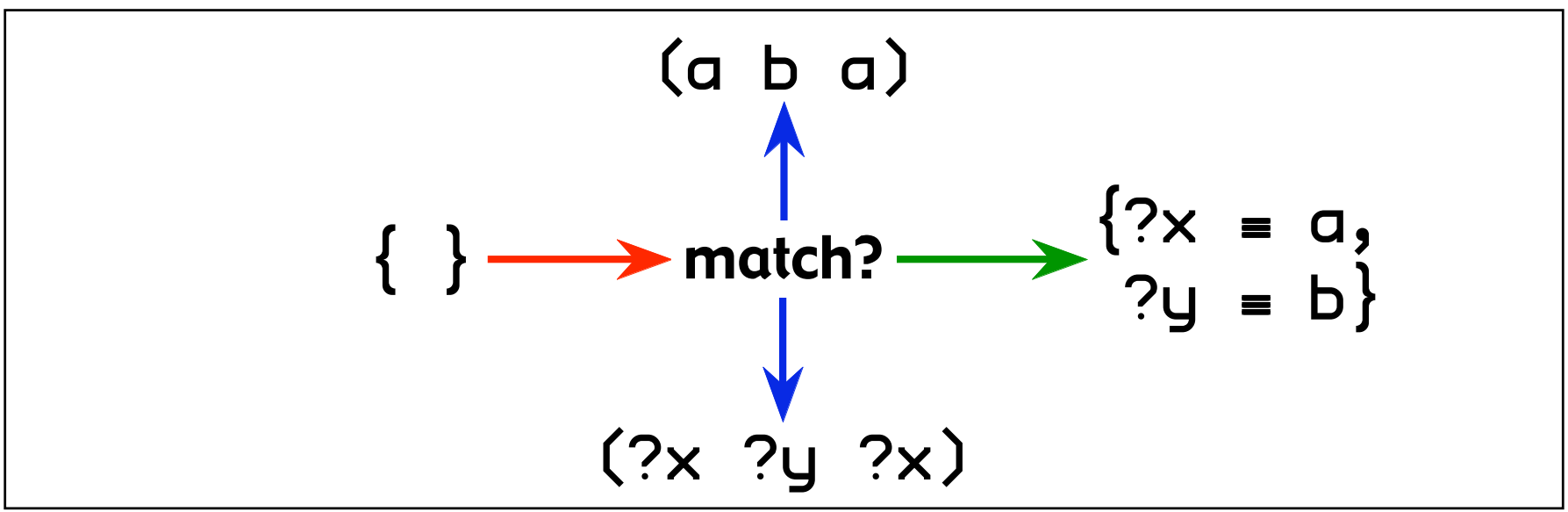
from zero to two bindings

Pattern Matching



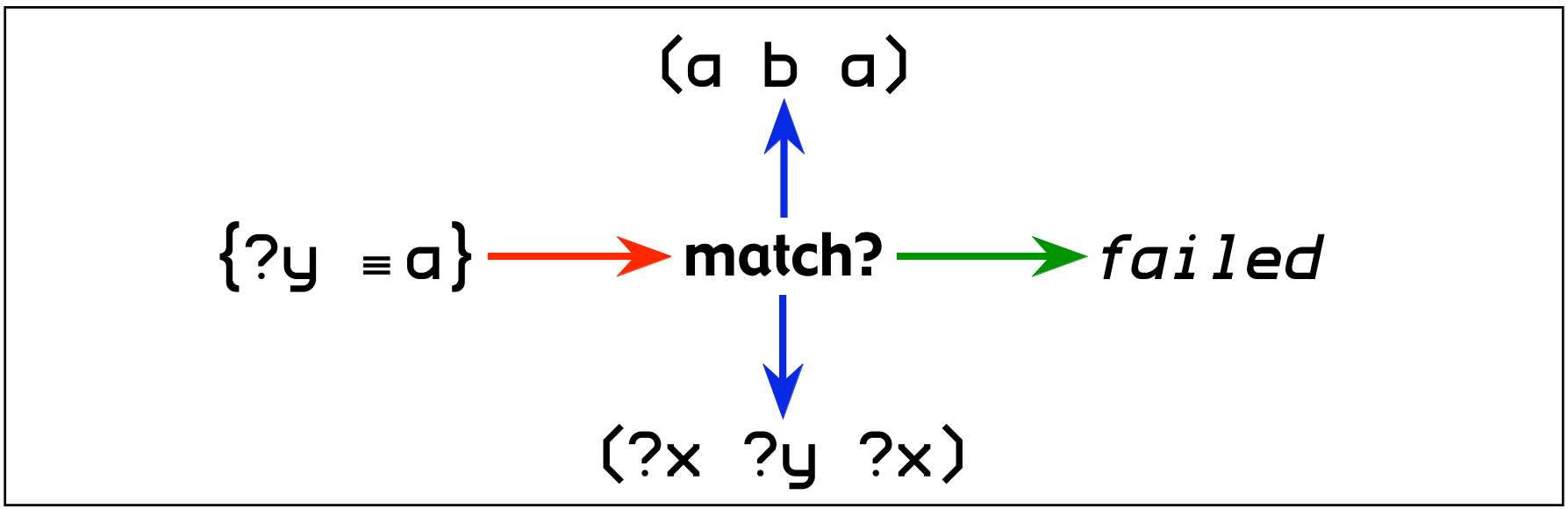
no valid bindings

Pattern Matching



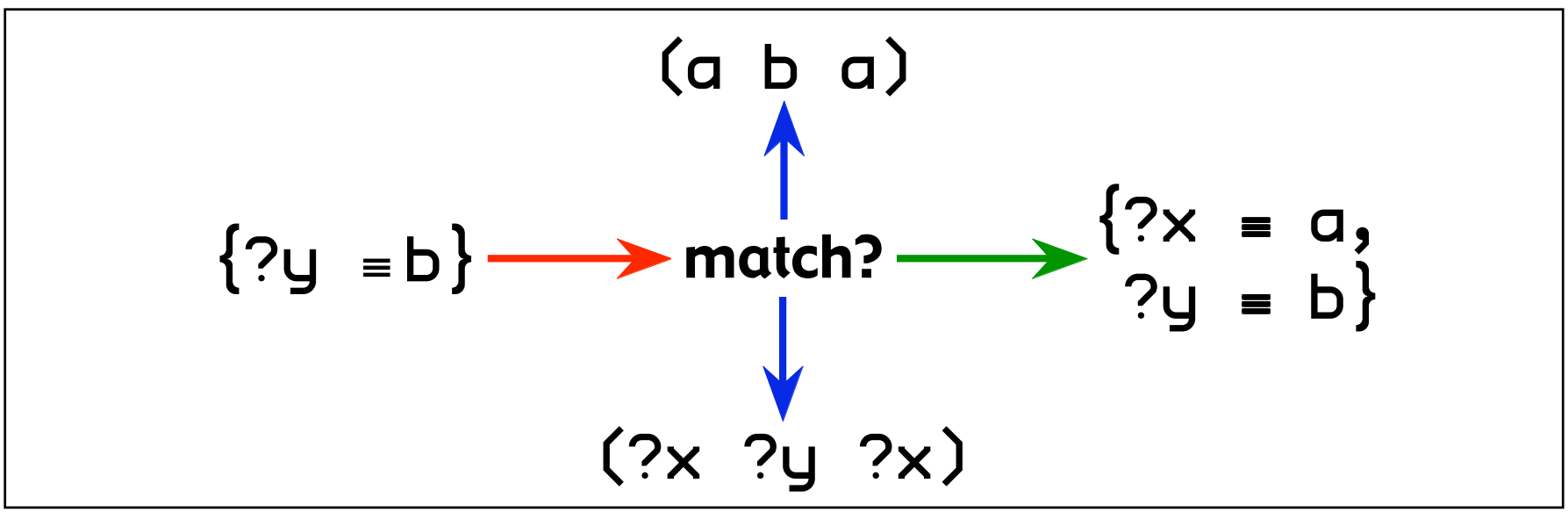
with valid bindings

Pattern Matching



with existing binding

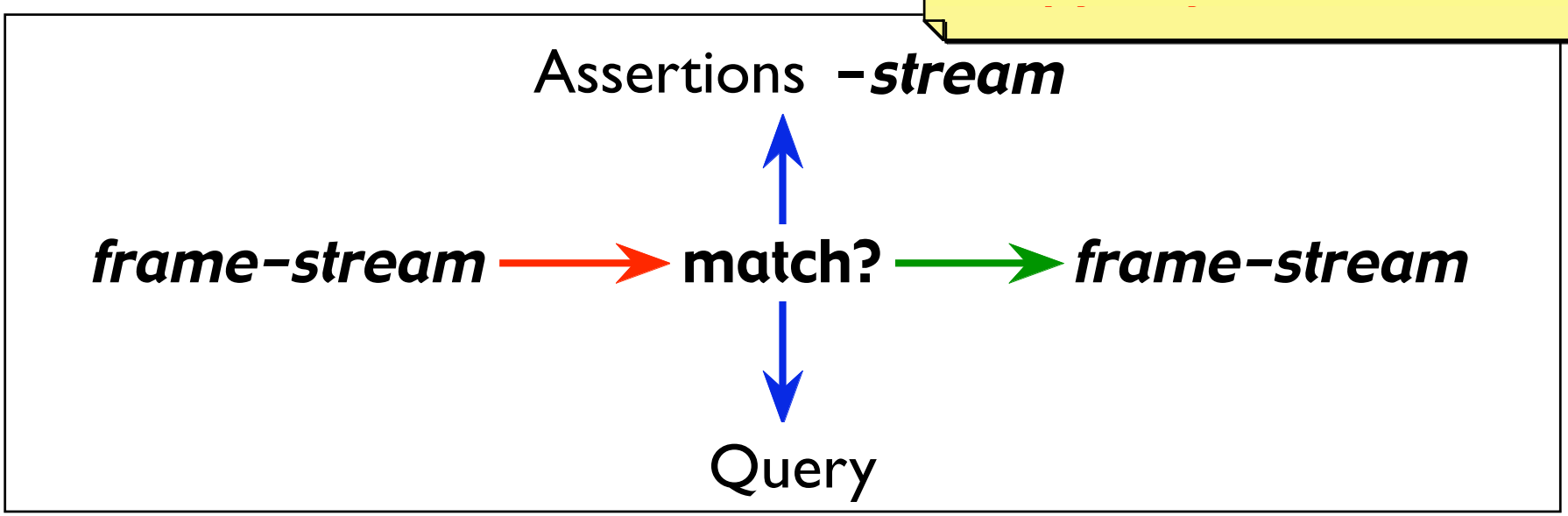
Pattern Matching



but now with valid extension

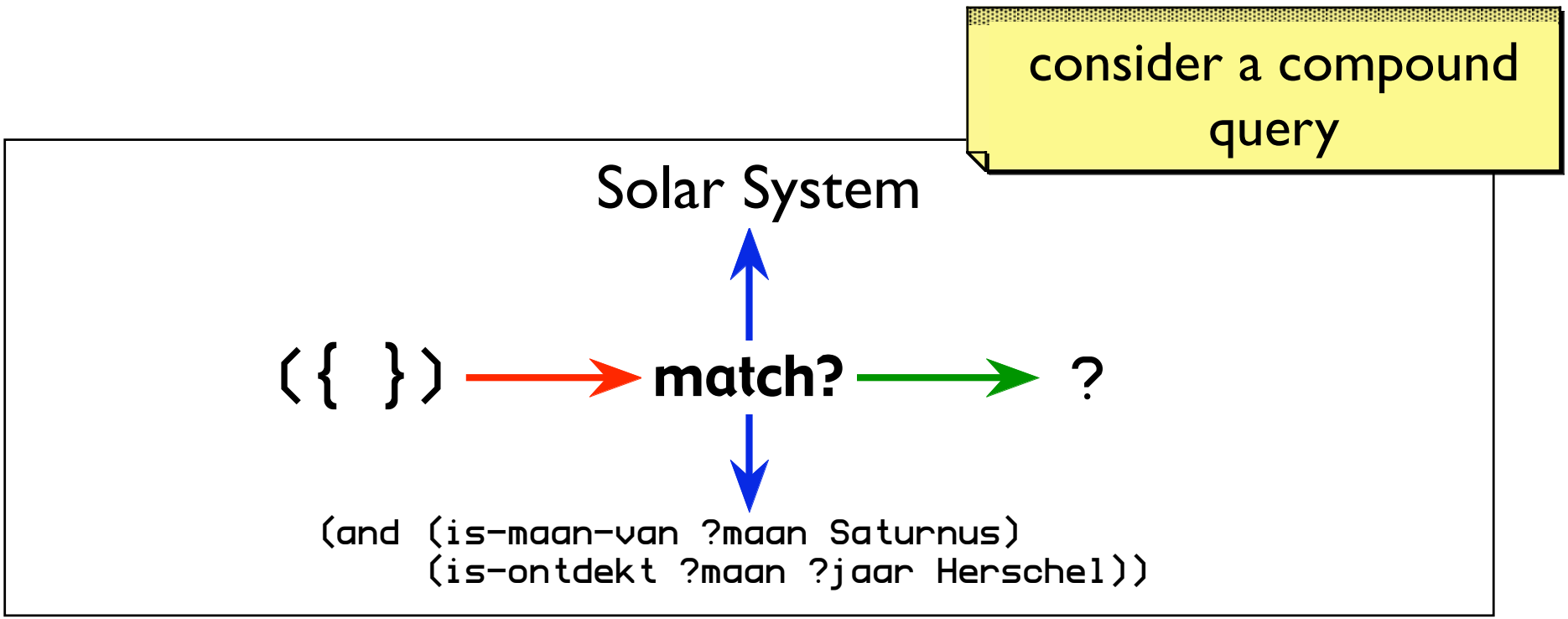
Pattern Recogniton

we consider a set of assertions, stored in a *stream*

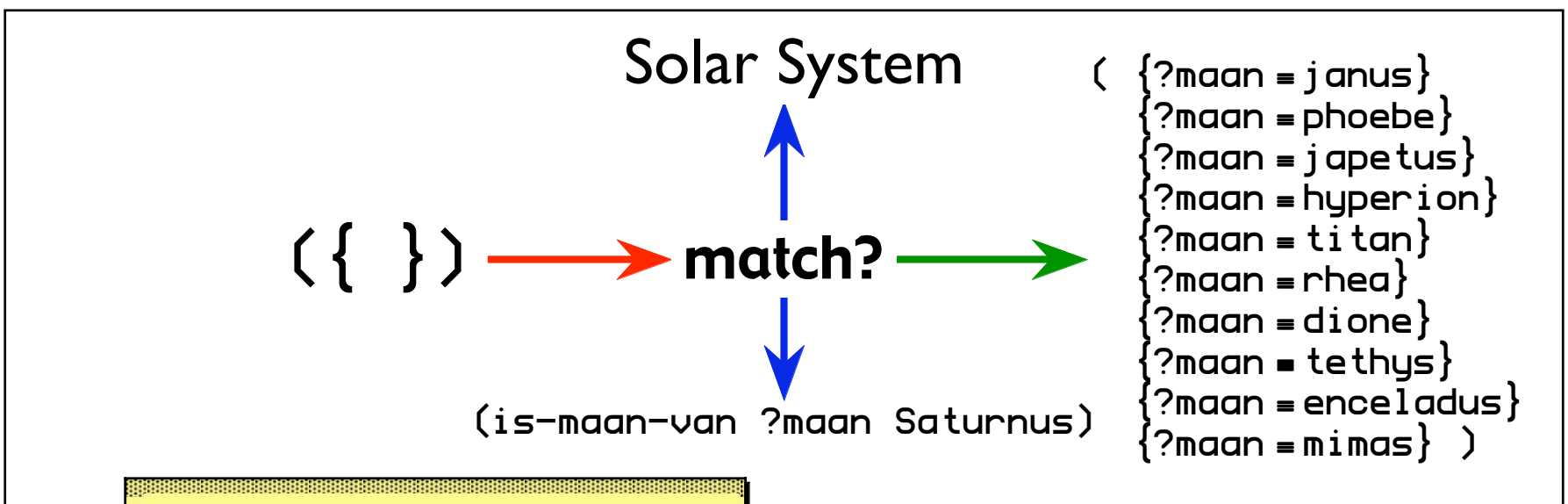


we consider a set of existing *frames*, stored in a *stream*

Pattern Recognition

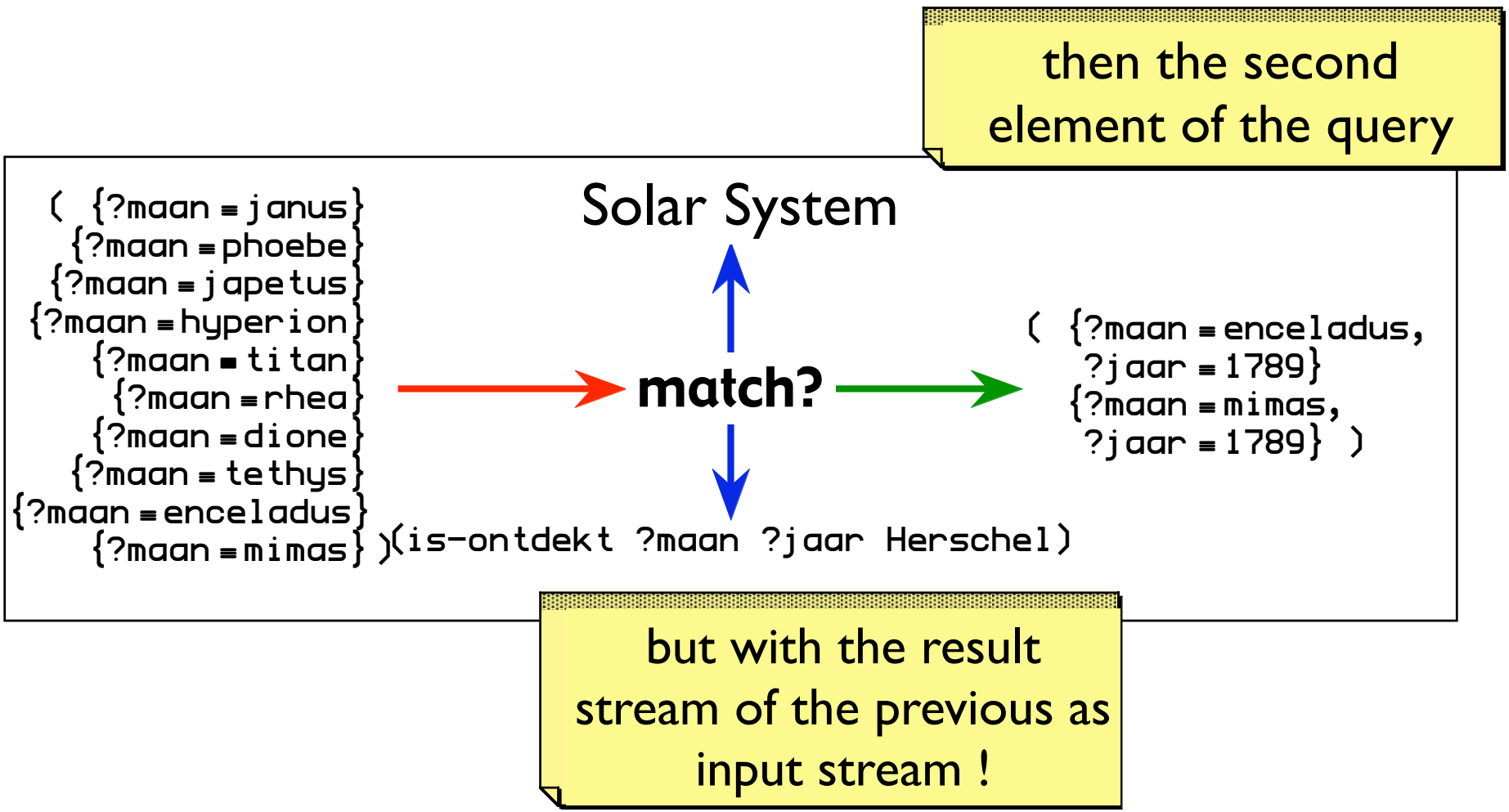


Pattern Recognition



take the first element of the query

Pattern Recognition



Query Evaluator

```
(define (query-driver-loop)
  (prompt-for-input input-prompt)
  (let ((q (query-syntax-process (read))))
    (cond
     ((assertion-to-be-added? q)
       (add-assertion! (add-assertion-body q))
       (newline)
       (display "Assertion added to data base.")
       (query-driver-loop))
      (else
       (newline)
       (display output-prompt)
       (display-stream
       (stream-map
        (lambda (frame)
          (instantiate q
                       frame
                       (lambda (v f)
                         (contract-question-mark v))))
        (qeval q NIL)))
       (query-driver-loop)))))
```

```
(define input-prompt
  ";;; Query input:")
(define output-prompt
  ";;; Query results:")
```

```
(define NIL (singleton-stream '()))
```

```
(define (prompt-for-input string)
  (newline) (newline)
  (display string) (newline))
```

Query Evaluator

```
(define (query-driver-loop)
  (prompt-for-input input-prompt)
  (let ((q (query-syntax-process (read))))
    (cond
     ((assertion-to-be-added? q)
       (add-assertion! (add-assertion-body q))
       (newline)
       (display "Assertion added to data
       (query-driver-loop))
      (else
       (newline)
       (display output-prompt)
       (display-stream
       (stream-map
        (lambda (frame)
          (instantiate q
                       frame
                       (lambda (v f)
                         (contract-question-mark v))))
        (qeval q (singleton-stream '())))))
       (query-driver-loop)))))
```

parsing the query

adding an assertion

printing found assertions

pattern recognition

Query Grammar

```
(define (query-syntax-process exp)
  (map-over-symbols expand-question-mark exp))

(define (map-over-symbols proc exp)
  (cond ((pair? exp)
         (cons (map-over-symbols proc (car exp))
               (map-over-symbols proc (cdr exp))))
        ((symbol? exp) (proc exp))
        (else exp)))
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (query-syntax-process '(a ?x b (c ?y)))
(a (? x) b (c (? y)))
>
```

Query Grammar

```
(define (expand-question-mark symbol)
  (let ((chars (symbol->string symbol)))
    (if (string=? (substring chars 0 1) "?")
        (list '?
              (string->symbol
               (substring chars
                          1
                          (string-length chars))))
        symbol)))
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (expand-question-mark 'xyz)
xyz
> (expand-question-mark '?uvw)
(? uvw)
>
```

Query Grammar

```
(define (var? exp)
  (tagged-list? exp '?))

(define (constant-symbol? exp) (symbol? exp))

(define (contract-question-mark variable)
  (string->symbol
   (string-append "?"
     (if (number? (cadr variable))
         (string-append
           (symbol->string (caddr variable))
           "_"
           (number->string (cadr variable)))
         (symbol->string (cadr variable))))))
```

```
(define (tagged-list? exp tag)
  (if (pair? exp)
      (eq? (car exp) tag)
      false))
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (contract-question-mark '(? abc))
?abc
> (contract-question-mark '(? 123 xyz))
?xyz-123
>
```

Query Grammar

```
(define (type exp)
  (if (pair? exp)
      (car exp)
      (error "Unknown expression TYPE" exp)))

(define (contents exp)
  (if (pair? exp)
      (cdr exp)
      (error "Unknown expression CONTENTS" exp)))

(define (assertion-to-be-added? exp)
  (eq? (type exp) 'assert!))

(define (add-assertion-body exp)
  (car (contents exp)))
```

```
Language: Textual Full Scheme (MzScheme).
done
> (type '(abc (1 2 3)))
abc
> (add-assertion-body '(assert! (abc (1 2 3))))
(abc (1 2 3))
>
```

Database

```
(define THE-ASSERTIONS the-empty-stream)

(define (fetch-assertions pattern frame)
  (if (use-index? pattern)
      (get-indexed-assertions pattern)
      (get-all-assertions)))

(define (add-assertion! assertion)
  (store-assertion-in-index assertion)
  (let ((old-assertions THE-ASSERTIONS))
    (set! THE-ASSERTIONS
          (cons-stream assertion old-assertions))
    'ok))

(define (get-all-assertions) THE-ASSERTIONS)
```

Query-Evaluator

```
(define (qeval query frame-stream)
  (let ((qproc (get (type query) 'qeval)))
    (if qproc
        (qproc (contents query) frame-stream)
        (simple-query query frame-stream))))

(define (simple-query query-pattern frame-stream)
  (stream-flatmap
   (lambda (frame)
     (stream-append-delayed
      (find-assertions query-pattern frame)
      (delay (apply-rules query-pattern frame))))
   frame-stream))
```

```
Language: Textual Full Scheme (MzScheme).
done
> (display-stream (simple-query
   (query-syntax-process '(is-maan-van ?x Jupiter))
   NIL))
(((? x) . callisto))
(((? x) . ganymedes))
(((? x) . europa))
(((? x) . io))
>
```

Query-Evaluator

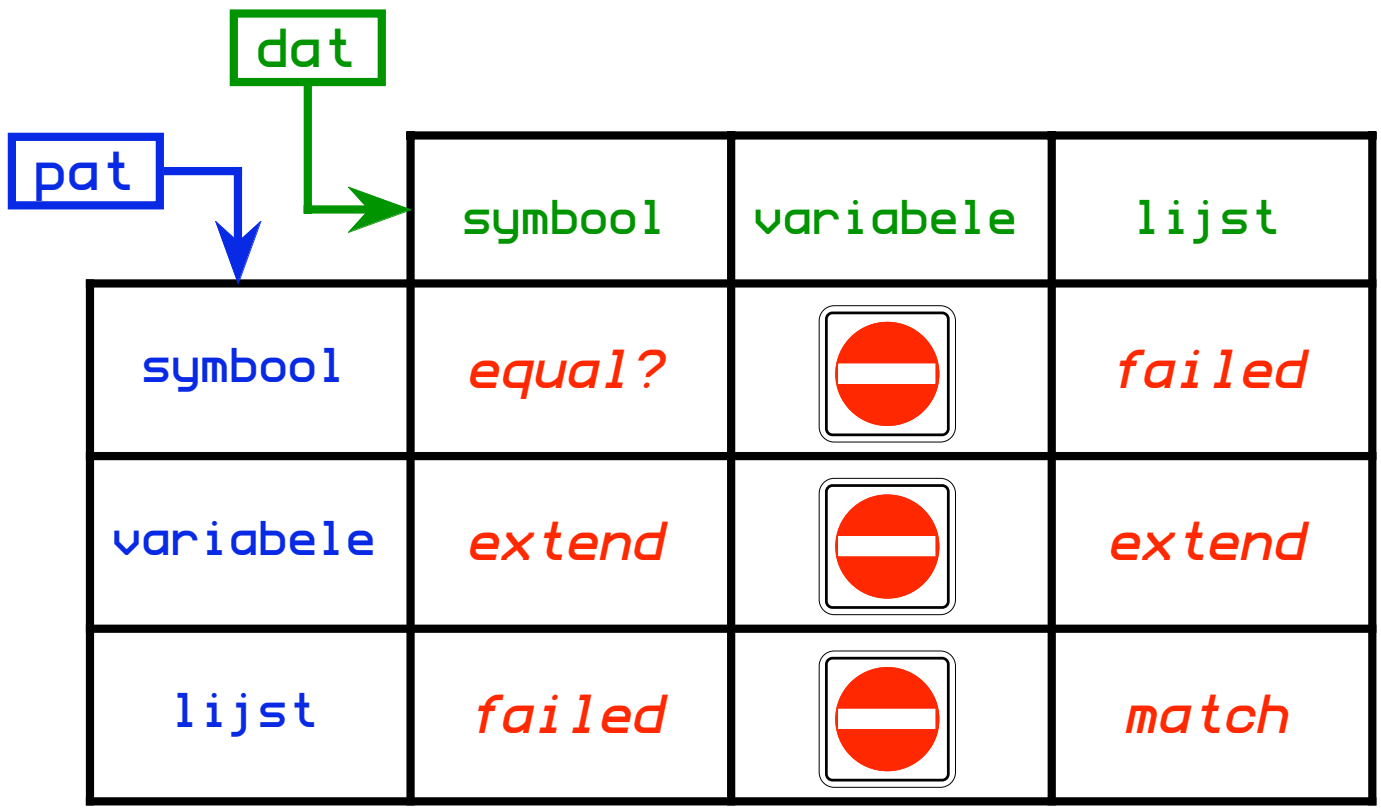
```
(define (find-assertions pattern frame)
  (stream-flatmap
    (lambda (datum)
       (check-an-assertion datum pattern frame))
    (fetch-assertions pattern frame)))

(define (check-an-assertion assertion query-pat query-frame)
  (let ((match-result
         (pattern-match query-pat
                        assertion query-frame)))
    (if (eq? match-result 'failed)
        the-empty-stream
        (singleton-stream match-result))))
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (check-an-assertion
           '(is-maan-van Ganymedes Jupiter)
           '(is-maan-van (? x) Jupiter)
           '())
((((? x) . ganymedes)) . #<promise>)
>
```

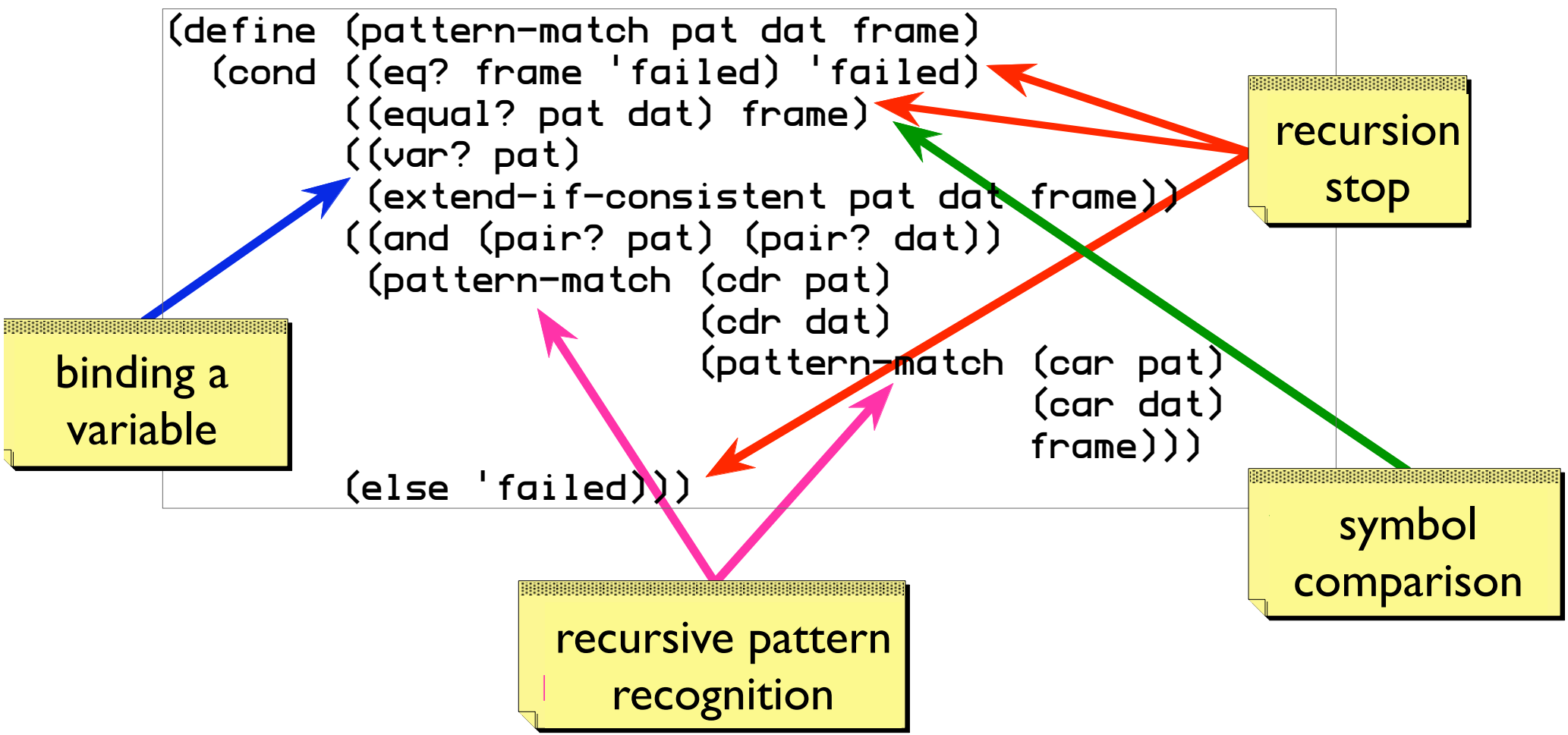
Pattern Recognition

```
(define (pattern-match pat dat frame)
  (cond ((eq? frame 'failed) 'failed)
        ((equal? pat dat) frame)
        ((var? pat)
         (extend-if-consistent pat dat frame))
        ((and (pair? pat) (pair? dat))
         (pattern-match (cdr pat)
                        (cdr dat)
                        (pattern-match (car pat)
                                       (car dat)
                                       frame)))
        (else 'failed)))
```

Pattern Recognition



Pattern Recognition



Pattern Recognition

```
(define (extend-if-consistent var dat frame)
  (let ((binding (binding-in-frame var frame)))
    (if binding
        (pattern-match (binding-value binding)
                       dat frame)
        (extend var dat frame))))
```

if not free, check binding is the same

if free, make the binding

Pattern Recognition

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (pattern-match '((? x) c (? x)) '((a b) c (a b)) '())
(((? x) a b))
> (pattern-match '((? x) (? y) (? z)) '((a b) c (a b)) '())
(((? z) a b) ((? y) . c) ((? x) a b))
> (pattern-match '(((? x) (? y)) c ((? x) (? y))) '((a b) c (a b)) '())
(((? y) . b) ((? x) . a))
> (pattern-match '((? x) a (? x)) '((a b) c (a b)) '())
failed
> (pattern-match '((? x) (? y) (? x)) '(a b a) '())
(((? y) . b) ((? x) . a))
> (pattern-match '((? x) (? y) (? x)) '(a b a) '(((? y) . a)))
failed
> (pattern-match '((? x) (? y) (? x)) '(a b a) '(((? y) . b)))
(((? x) . a) ((? y) . b))
>
```

Frames and Bindings

```
(define (make-binding variable value)
  (cons variable value))

(define (binding-variable binding)
  (car binding))

(define (binding-value binding)
  (cdr binding))

(define (binding-in-frame variable frame)
  (assoc variable frame))

(define (extend variable value frame)
  (cons (make-binding variable value) frame))
```

Printing

```
(define (instantiate exp frame unbound-var-handler)
  (define (copy exp)
    (cond ((var? exp)
           (let ((binding (binding-in-frame exp frame)))
             (if binding
                 (copy (binding-value binding))
                 (unbound-var-handler exp frame))))
          ((pair? exp)
           (cons (copy (car exp)) (copy (cdr exp))))
          (else exp)))
  (copy exp))
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (instantiate '(is-maan-van (? maan) (? planeet))
    '(((? maan) . Callisto) ((? planeet) . Jupiter))
     (lambda (v f) (contract-question-mark v)))
(is-maan-van callisto jupiter)
>
```

Streams

```
(define the-empty-stream '())

(define (stream-null? stream)
  (null? stream))

(define-macro cons-stream
  (lambda (car cdr)
    `(cons ,car (delay ,cdr))))

(define (stream-car stream)
  (car stream))

(define (stream-cdr stream)
  (force (cdr stream)))

(define (list->stream list)
  (if (pair? list)
      (cons-stream (car list) (list->stream (cdr list)))
      the-empty-stream))
```

Streams

```
(define (stream-append s1 s2)
  (if (stream-null? s1)
      s2
      (cons-stream (stream-car s1)
                   (stream-append (stream-cdr s1) s2))))

(define (stream-map proc s)
  (if (stream-null? s)
      the-empty-stream
      (cons-stream (proc (stream-car s))
                   (stream-map proc (stream-cdr s)))))

(define (stream-for-each proc s)
  (if (stream-null? s)
      'done
      (begin (proc (stream-car s))
             (stream-for-each proc (stream-cdr s)))))
```

Streams

```
(define (display-stream s)
  (stream-for-each display-line s))

(define (display-line x)
  (newline)
  (display x))

(define (singleton-stream x)
  (cons-stream x the-empty-stream))
```

Streams

```
(define (stream-append-delayed s1 delayed-s2)
  (if (stream-null? s1)
      (force delayed-s2)
      (cons-stream
       (stream-car s1)
       (stream-append-delayed (stream-cdr s1) delayed-s2))))

(define (interleave-delayed s1 delayed-s2)
  (if (stream-null? s1)
      (force delayed-s2)
      (cons-stream
       (stream-car s1)
       (interleave-delayed (force delayed-s2)
                           (delay (stream-cdr s1))))))
```

Streams

```
(define (stream-flatmap proc s)
  (flatten-stream (stream-map proc s)))

(define (flatten-stream stream)
  (if (stream-null? stream)
      the-empty-stream
      (interleave-delayed
       (stream-car stream)
       (delay (flatten-stream (stream-cdr stream))))))
```

Property lists

```
(define PROPERTY-LIST '())

(define (put symb qual obj)
  (define entry (assoc symb PROPERTY-LIST))
  (if entry
      (let ((qual-obj (assoc qual (cdr entry))))
        (if qual-obj
            (set-cdr! qual-obj bj)
            (set-cdr! entry
                      (cons (cons qual obj) (cdr entry)))))
      (set! PROPERTY-LIST
            (cons (cons symb list (cons qual obj)))
                  PROPERTY-LIST))
  PROPERTY-LIST)
```

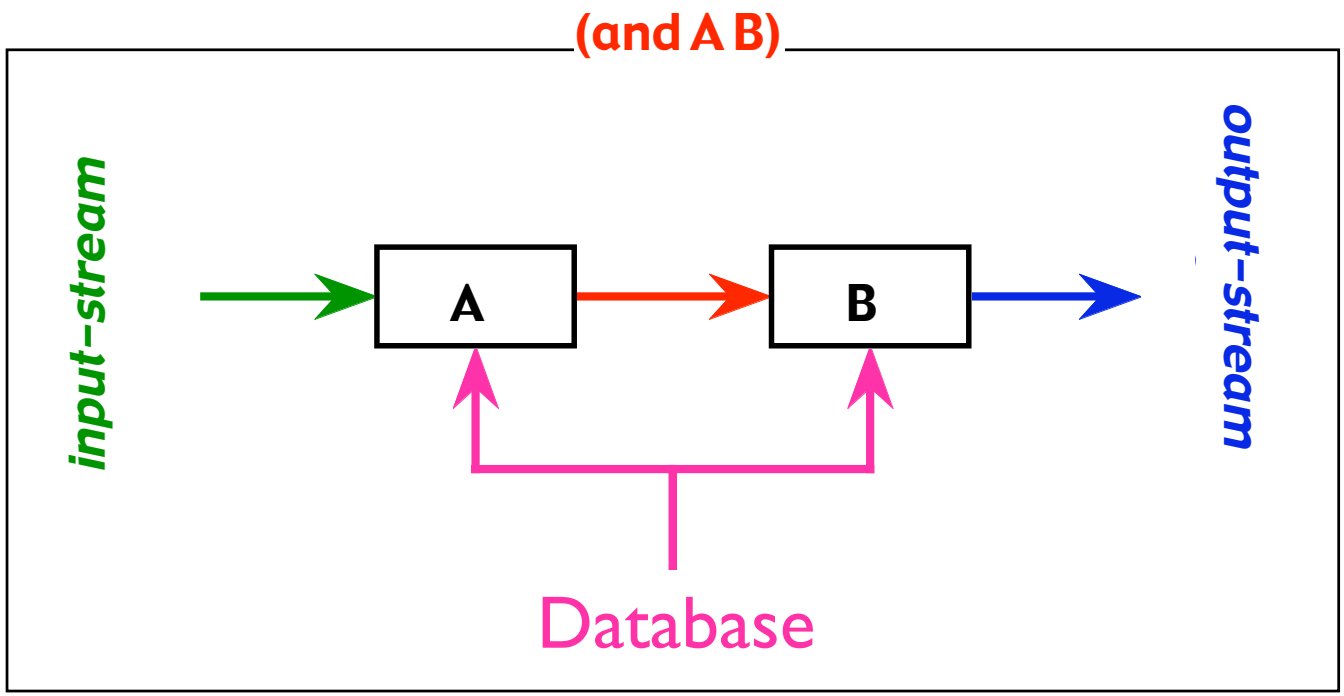
Property lists

```
(define (get symb qual)
  (define entry (assoc symb PROPERTY-LIST))
  (if entry
      (let ((qual-obj (assoc qual (cdr entry))))
        (if qual-obj
            (cdr qual-obj)
            false))
      false))
```

Property lists

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (put 'a 'xyz 123)
((a (xyz . 123)))
> (put 'a 'uvw 345)
((a (uvw . 345) (xyz . 123)))
> (put 'b 'xyz 333)
((b (xyz . 333)) (a (uvw . 345) (xyz . 123)))
> (put 'b 'uvw 555)
((b (uvw . 555) (xyz . 333)) (a (uvw . 345) (xyz . 123)))
> (put 'c 'klm 999)
((c (klm . 999)) (b (uvw . 555) (xyz . 333)) (a (uvw . 345) (xyz . 123)))
> (get 'b 'xyz)
333
> (get 'c 'uvw)
#f
>
```

Conjunction



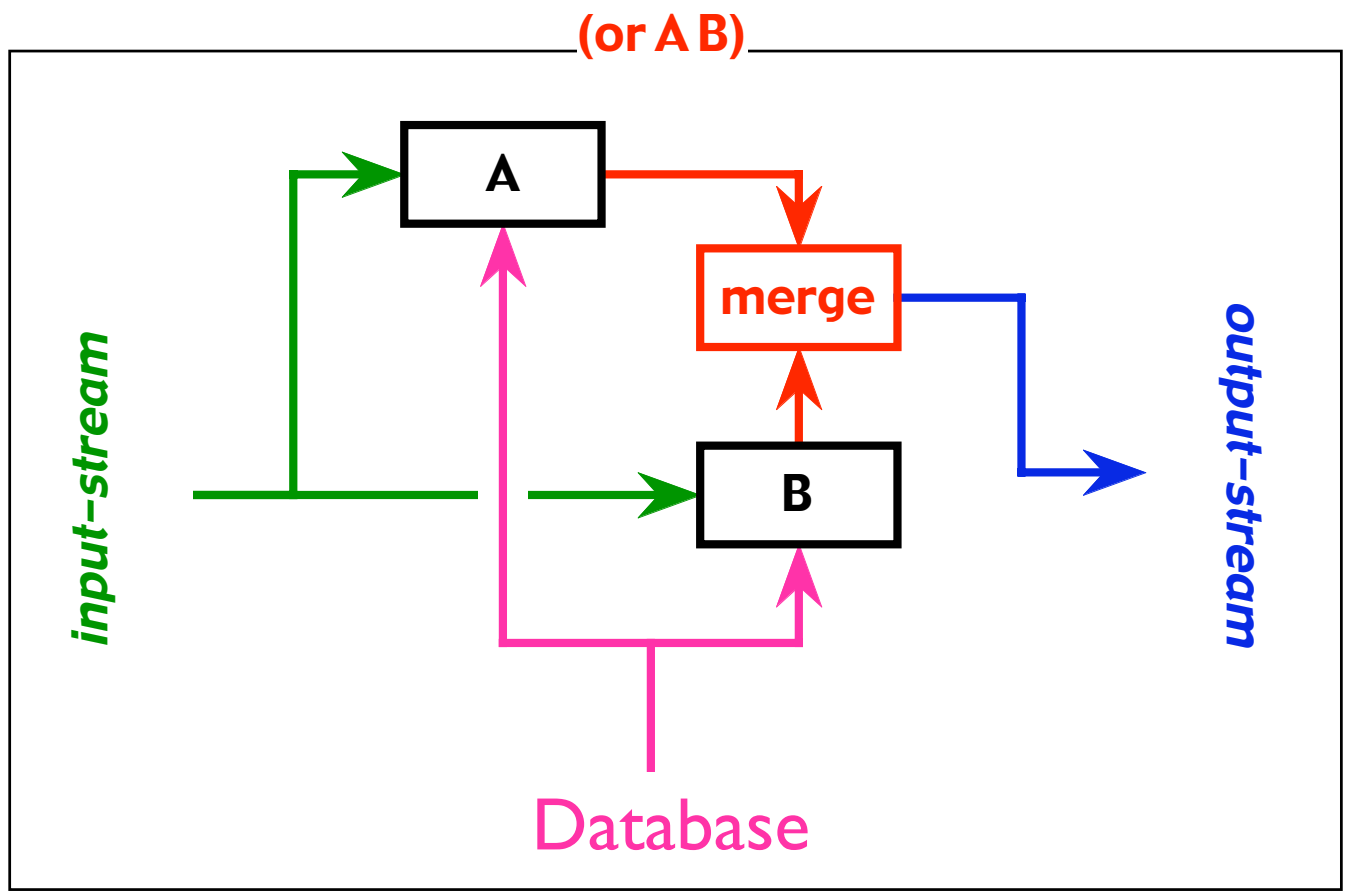
Conjunction

```
(define (conjoin conjuncts frame-stream)
  (if (empty-conjunction? conjuncts)
      frame-stream
      (conjoin (rest-conjuncts conjuncts)
               (qeval (first-conjunct conjuncts)
                      frame-stream))))
```

```
(define empty-conjunction? null?)
(define first-conjunct car)
(define rest-conjuncts cdr)
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (display-stream (qeval '(is-maan-van (? maan) Saturnus) NIL))
(((? maan) . janus))
(((? maan) . phoebe))
(((? maan) . japetus))
(((? maan) . hyperion))
(((? maan) . titan))
(((? maan) . rhea))
(((? maan) . dione))
(((? maan) . tethys))
(((? maan) . enceladus))
(((? maan) . mimas))
done
> (display-stream (qeval '(is-ontdekt (? maan) (? jaar) Herschel) NIL))
(((? jaar) . 1787) ((? maan) . oberon))
(((? jaar) . 1787) ((? maan) . titania))
(((? jaar) . 1789) ((? maan) . enceladus))
(((? jaar) . 1789) ((? maan) . mimas))
done
> (display-stream (conjoin '((is-maan-van (? maan) Saturnus)
                             (is-ontdekt (? maan) (? jaar) Herschel)) NIL))
(((? jaar) . 1789) ((? maan) . enceladus))
(((? jaar) . 1789) ((? maan) . mimas))
done
>
```

Disjunction



Disjunction

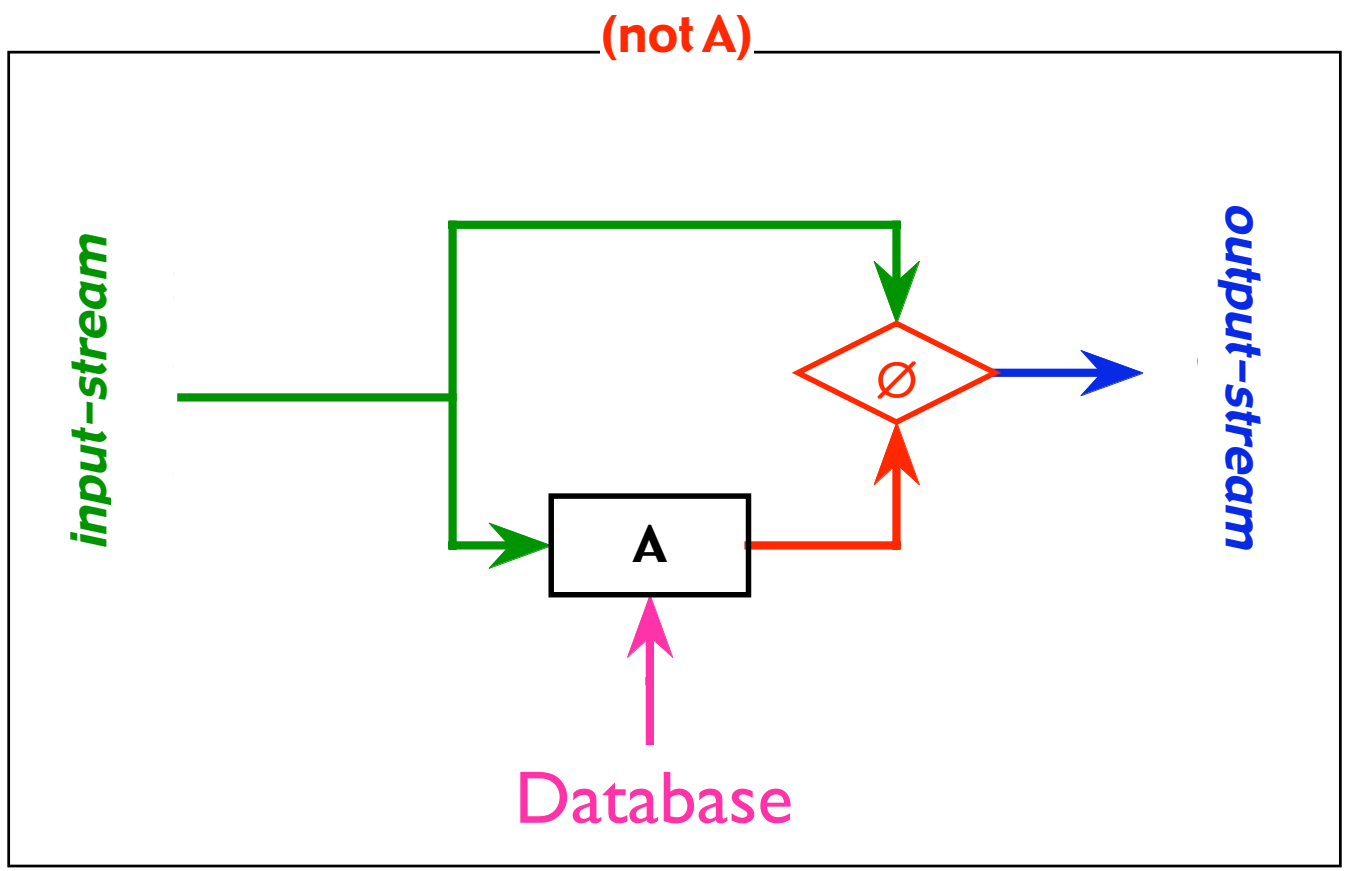
```
(define (disjoin disjuncts frame-stream)
  (if (empty-disjunction? disjuncts)
      the-empty-stream
      (interleave-delayed
       (qeval (first-disjunct disjuncts) frame-stream)
       (delay (disjoin (rest-disjuncts disjuncts)
                       frame-stream)))))
```

```
(define (empty-disjunction? exps) (null? exps))
(define (first-disjunct exps) (car exps))
(define (rest-disjuncts exps) (cdr exps))
```

Disjunction

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (display-stream (qeval
   '(is-ontdekt (? maan) 1948 (? persoon)) NIL))

(((? persoon) . kuiper) ((? maan) . miranda))
done
> (display-stream (qeval
   '(is-ontdekt (? maan) 1949 (? persoon)) NIL))

(((? persoon) . kuiper) ((? maan) . nereide))
done
> (display-stream (disjoin
   '((is-ontdekt (? maan) 1948 (? persoon))
     (is-ontdekt (? maan) 1949 (? persoon))) NIL))

(((? persoon) . kuiper) ((? maan) . miranda))
(((? persoon) . kuiper) ((? maan) . nereide))
done
>
```

Negation



Negation

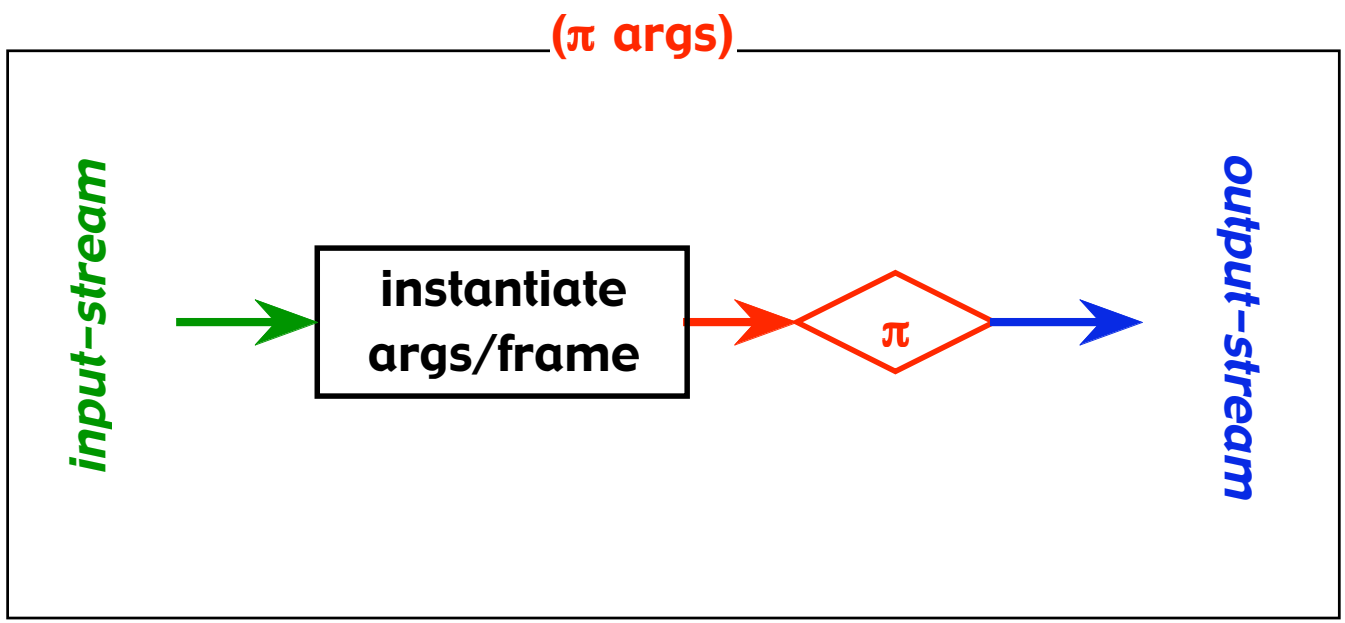
```
(define (negate operands frame-stream)
  (stream-flatmap
   (lambda (frame)
     (if (stream-null? (qeval (negated-query operands)
                              (singleton-stream frame)))
         (singleton-stream frame)
         the-empty-stream))
   frame-stream))
```

```
(define (negated-query exps) (car exps))
```

Negation

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (define p (qeval '(is-planeet (? planeet)) NIL))
> (display-stream p)
(((? planeet) . pluto))
(((? planeet) . neptunus))
(((? planeet) . uranus))
(((? planeet) . saturnus))
(((? planeet) . jupiter))
(((? planeet) . mars))
(((? planeet) . aarde))
(((? planeet) . venus))
(((? planeet) . mercurius))
done
> (display-stream (negate
   '((is-maan-van (? maan) (? planeet)))
   p))
(((? planeet) . pluto))
(((? planeet) . venus))
(((? planeet) . mercurius))
done
>
```

Lisp filters



Lisp filters

```
(define (lisp-value call frame-stream)
  (stream-flatmap
   (lambda (frame)
     (if (execute
          (instantiate
           call
           frame
           (lambda (v f)
             (error "Unknown pat var -- LISP-VALUE" v))))
         (singleton-stream frame)
         the-empty-stream))
   frame-stream))

(define (execute exp)
  (apply (eval (predicate exp))
         (args exp)))
```

```
(define (predicate exps) (car exps))
(define (args exps) (cdr exps))
```

changed to conform to DrScheme

Lisp filters

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (define p (qeval '(is-planeet (? planeet)) NIL))
done
> (define q
   (conjoin
    '((heeft-middellijn (? planeet) (? ml))) p))
> (display-stream q)
(((? ml) . 5000) ((? planeet) . pluto))
(((? ml) . 44800) ((? planeet) . neptunus))
(((? ml) . 47100) ((? planeet) . uranus))
(((? ml) . 119300) ((? planeet) . saturnus))
(((? ml) . 142800) ((? planeet) . jupiter))
(((? ml) . 6790) ((? planeet) . mars))
(((? ml) . 12756) ((? planeet) . aarde))
(((? ml) . 12200) ((? planeet) . venus))
(((? ml) . 4840) ((? planeet) . mercurius))
done
> (display-stream (lisp-value '>  (? ml) 30000) q))
(((? ml) . 44800) ((? planeet) . neptunus))
(((? ml) . 47100) ((? planeet) . uranus))
(((? ml) . 119300) ((? planeet) . saturnus))
(((? ml) . 142800) ((? planeet) . jupiter))
done
>
```

Driver loop init

```
(define (initialize-data-base rules-and-assertions)
  (define (deal-out r-and-a rules assertions)
    (cond ((null? r-and-a)
           (set! THE-ASSERTIONS (list->stream assertions))
           (set! THE-RULES (list->stream rules))
           'done)
          (else
           (let ((s (query-syntax-process (car r-and-a))))
             (cond
               ((rule? s)
                (store-rule-in-index s)
                (deal-out (cdr r-and-a) (cons s rules) assertions))
               (else
                (store-assertion-in-index s)
                (deal-out (cdr r-and-a) rules (cons s assertions)))))))))
  (set! PROPERTY-LIST '())
  (put 'and 'qeval conjoin)
  (put 'or 'qeval disjoin)
  (put 'not 'qeval negate)
  (put 'lisp-value 'qeval lisp-value)
  (put 'always-true 'qeval always-true)
  (deal-out rules-and-assertions '() '()))
```

```
(define (always-true ignore frame-stream)
   frame-stream)
```

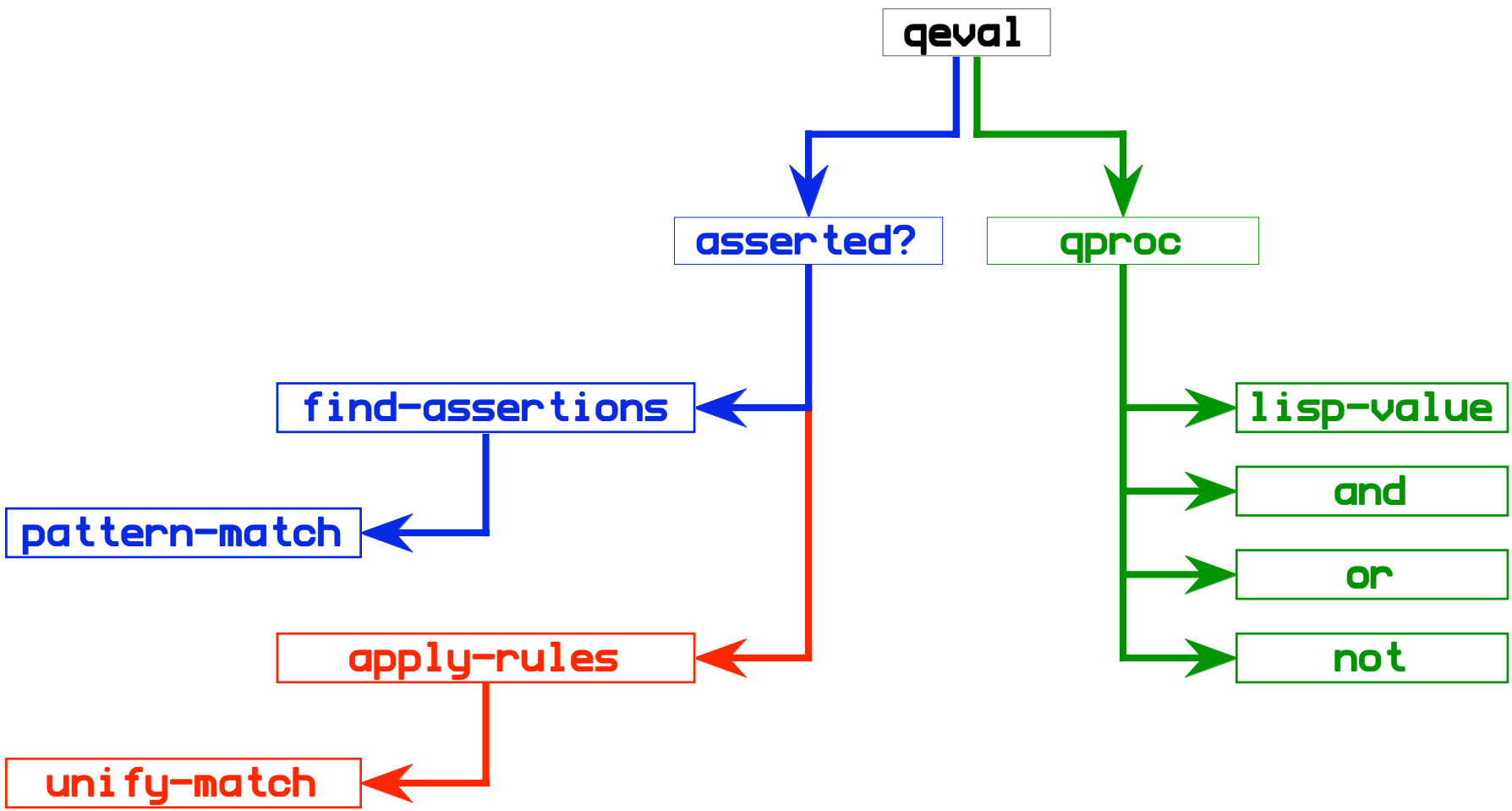
Driver loop init

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
> (initialize-data-base
  '((is-planeet Mercurius)
    (is-planeet Venus    )
    (is-planeet Aarde    )



    (is-ontdekt Hyperion  1848 Bond     )
    (is-ontdekt Japetus   1671 Cassini  )
    (is-ontdekt Phoebe    1898 Pickering)
    (is-ontdekt Janus     1966 Dolfus   )
    (is-ontdekt Ariel     1851 Lassell  )
    (is-ontdekt Umbriel   1851 Lassell  )
    (is-ontdekt Titania   1787 Herschel )
    (is-ontdekt Oberon    1787 Herschel )
    (is-ontdekt Miranda   1948 Kuiper   )
    (is-ontdekt Triton    1846 Lassell  )
    (is-ontdekt Nereide   1949 Kuiper   )))
done
> (query-driver-loop)
;;; Query input:
```

Adding rules support

```
(define (qeval query frame-stream)
  (let ((qproc (get (type query) 'qeval)))
    (if qproc
        (qproc (contents query) frame-stream)
        (simple-query query frame-stream))))
```

Logic Rules


qeval
asserted?
qproc
find-assertions
pattern-match
apply-rules
unify-match
lisp-value
and
or
not

Adding Rules

```
(define THE-RULES the-empty-stream)

(define (fetch-rules pattern frame)
  (if (use-index? pattern)
      (get-indexed-rules pattern)
      (get-all-rules)))

(define (get-all-rules) THE-RULES)

(define (add-rule! rule)
  (store-rule-in-index rule)
  (let ((old-rules THE-RULES))
    (set! THE-RULES (cons-stream rule old-rules))
    'ok))

(define (add-rule-or-assertion! assertion)
  (if (rule? assertion)
      (add-rule! assertion)
      (add-assertion! assertion)))
```

```
(define (rule? statement)
  (tagged-list? statement 'rule))
```

Applying Rules

```
(define (simple-query query-pattern frame-stream)
  (stream-flatmap
   (lambda (frame)
     (stream-append-delayed
      (find-assertions query-pattern frame)
      (delay (apply-rules query-pattern frame))))
   frame-stream))
```

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (query-driver-loop))
;;; Query input:
(assert! (rule (append () ?y ?y)))
Assertion added to data base.
;;; Query input:
(assert! (rule (append (?u . ?v) ?y (?u . ?z)) (append ?v ?y ?z)))
Assertion added to data base.
;;; Query input:
(append (a b) (c d) ?z)
;;; Query results:
(append (a b) (c d) (a b c d))
;;; Query input:
(append (a b) ?y (a b c d))
;;; Query results:
(append (a b) (c d) (a b c d))
;;; Query input:
(append ?x ?y (a b c d))
;;; Query results:
(append (a b c d) () (a b c d))
(append () (a b c d) (a b c d))
(append (a) (b c d) (a b c d))
(append (a b) (c d) (a b c d))
(append (a b c) (d) (a b c d))
;;; Query input:
```

Applying Rules

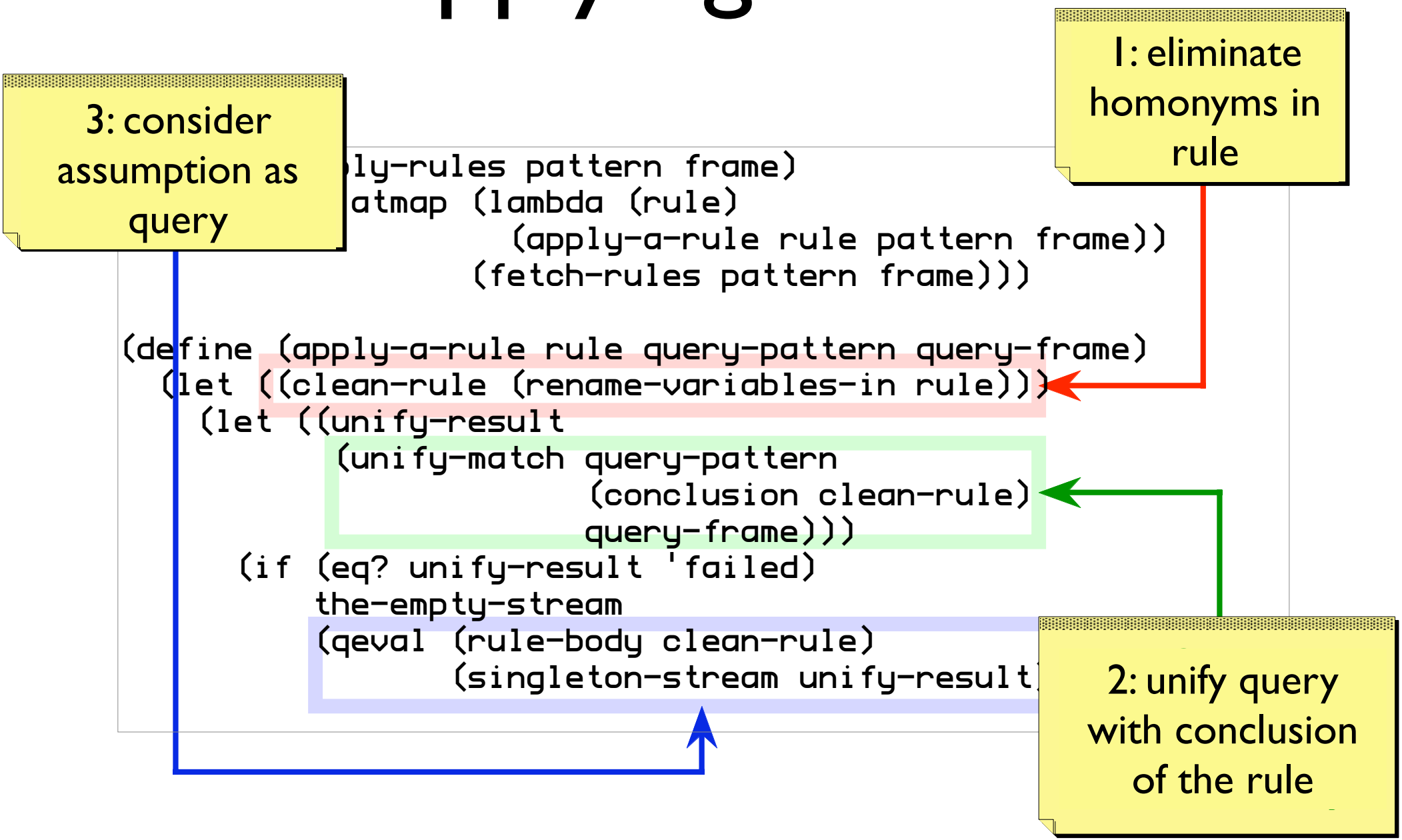
```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (add-rule!
   '(rule (zelfde-planeet (? x) (? y))
          (and (is-maan-van (? x) (? p))
               (is-maan-van (? y) (? p))
               (not (lisp-value eq? (? x ) (? y))))))
ok
> (display-stream (apply-rules
   '(zelfde-planeet phobos (? m)) '()))

(((? 1 y) . deimos) ((? 1 p) . mars) ((? m) ? 1 y) ((? 1 x) . phobos))
done
>
```

Applying Rules

```
(define (conclusion rule)
  (cadr rule))

(define (rule-body rule)
  (if (null? (cddr rule))
      '(always-true)
      (caddr rule)))
```

```
(define (apply-rules pattern frame)
  (stream-flatmap (lambda (rule)
                    (apply-a-rule rule pattern frame))
                  (fetch-rules pattern frame)))

(define (apply-a-rule rule query-pattern query-frame)
  (let ((clean-rule (rename-variables-in rule)))
    (let ((unify-result
           (unify-match query-pattern
                        (conclusion clean-rule)
                        query-frame)))
      (if (eq? unify-result 'failed)
          the-empty-stream
          (qeval (rule-body clean-rule)
                 (singleton-stream unify-result))))))
```

Applying Rules



Homonym elimination

```
(define rule-counter 0)

(define (new-rule-application-id)
  (set! rule-counter (+ 1 rule-counter))
  rule-counter)

(define (make-new-variable var rule-application-id)
  (cons '? (cons rule-application-id (cdr var))))

(define (rename-variables-in rule)
  (let ((rule-application-id (new-rule-application-id)))
    (define (tree-walk exp)
      (cond ((var? exp)
             (make-new-variable exp rule-application-id))
            ((pair? exp)
             (cons (tree-walk (car exp))
                   (tree-walk (cdr exp))))
            (else exp)))
    (tree-walk rule)))
```

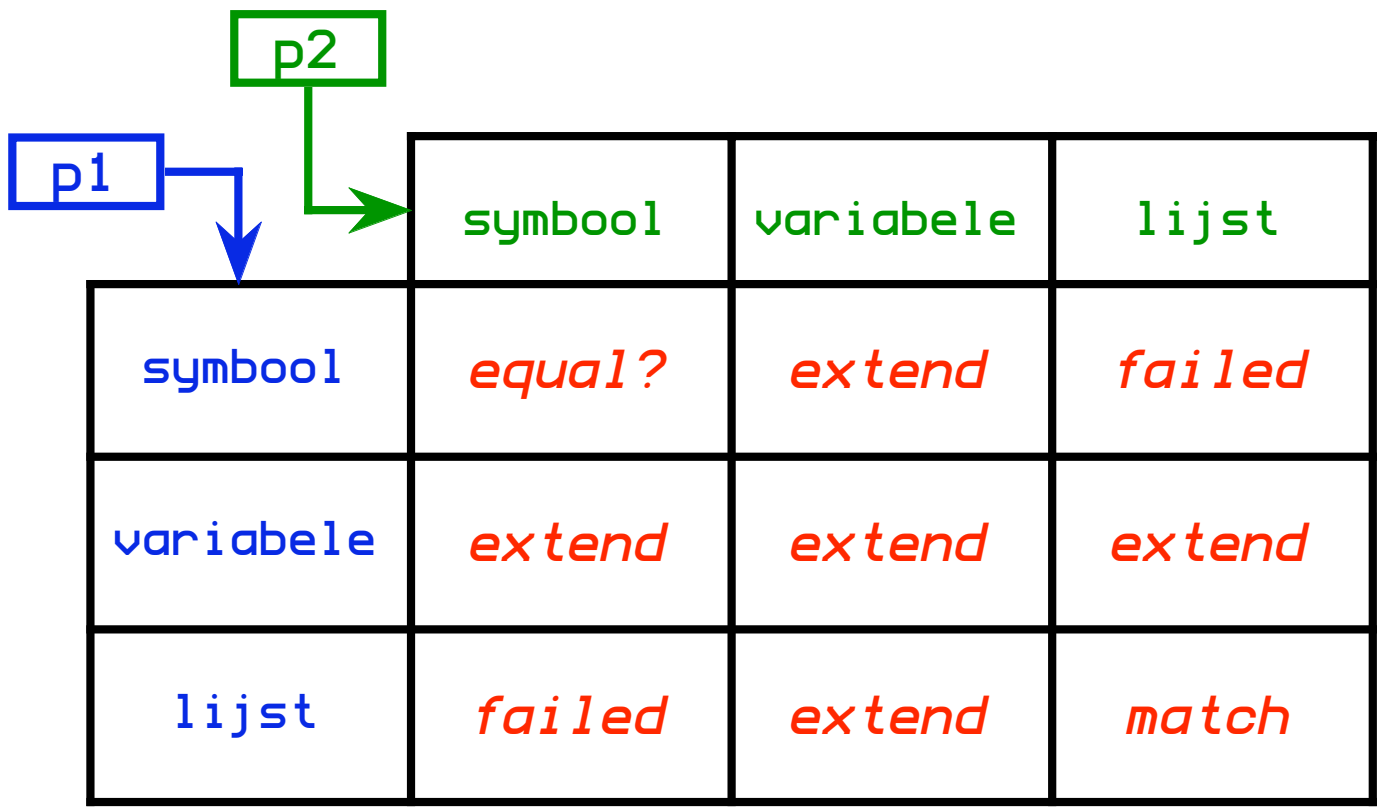
Homonym Elimination

```
Welcome to DrScheme, version 102.
Language: Textual Full Scheme (MzScheme).
done
> (set! rule-counter 0)
> (new-rule-application-id)
1
> (make-new-variable '(? xyz) 99)
(? 99 xyz)
> (rename-variables-in
   '(rule (zelfde-planeet (? x) (? y))
      (and (is-maan-van (? x) (? p))
           (is-maan-van (? y) (? p)))))
(rule
 (zelfde-planeet (? 2 x) (? 2 y))
 (and (is-maan-van (? 2 x) (? 2 p))
      (is-maan-van (? 2 y) (? 2 p))))
>
```

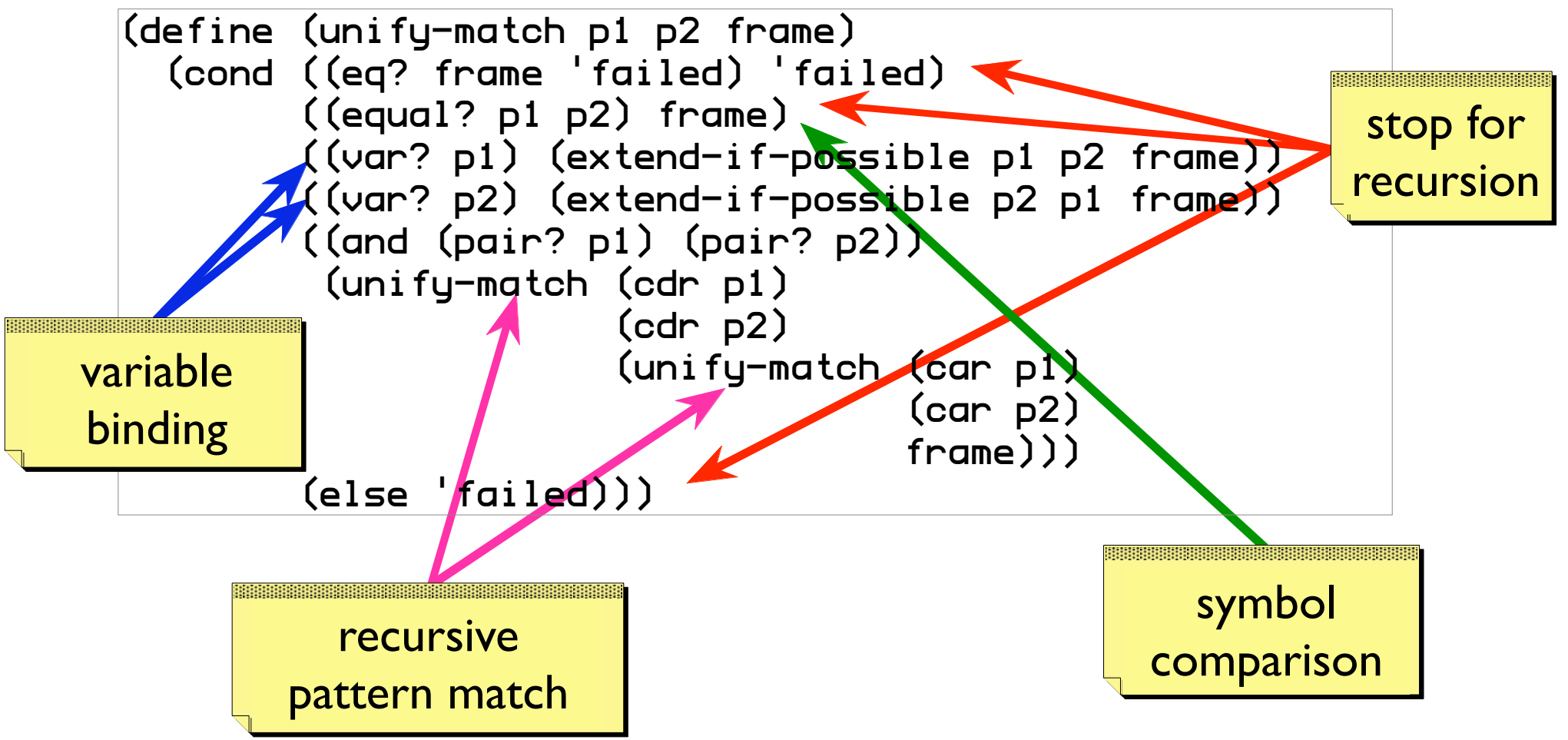
Unification

```
(define (unify-match p1 p2 frame)
  (cond ((eq? frame 'failed) 'failed)
        ((equal? p1 p2) frame)
        ((var? p1) (extend-if-possible p1 p2 frame))
        ((var? p2) (extend-if-possible p2 p1 frame))
        ((and (pair? p1) (pair? p2))
         (unify-match (cdr p1)
                      (cdr p2)
                      (unify-match (car p1)
                                   (car p2)
                                   frame)))
        (else 'failed)))
```

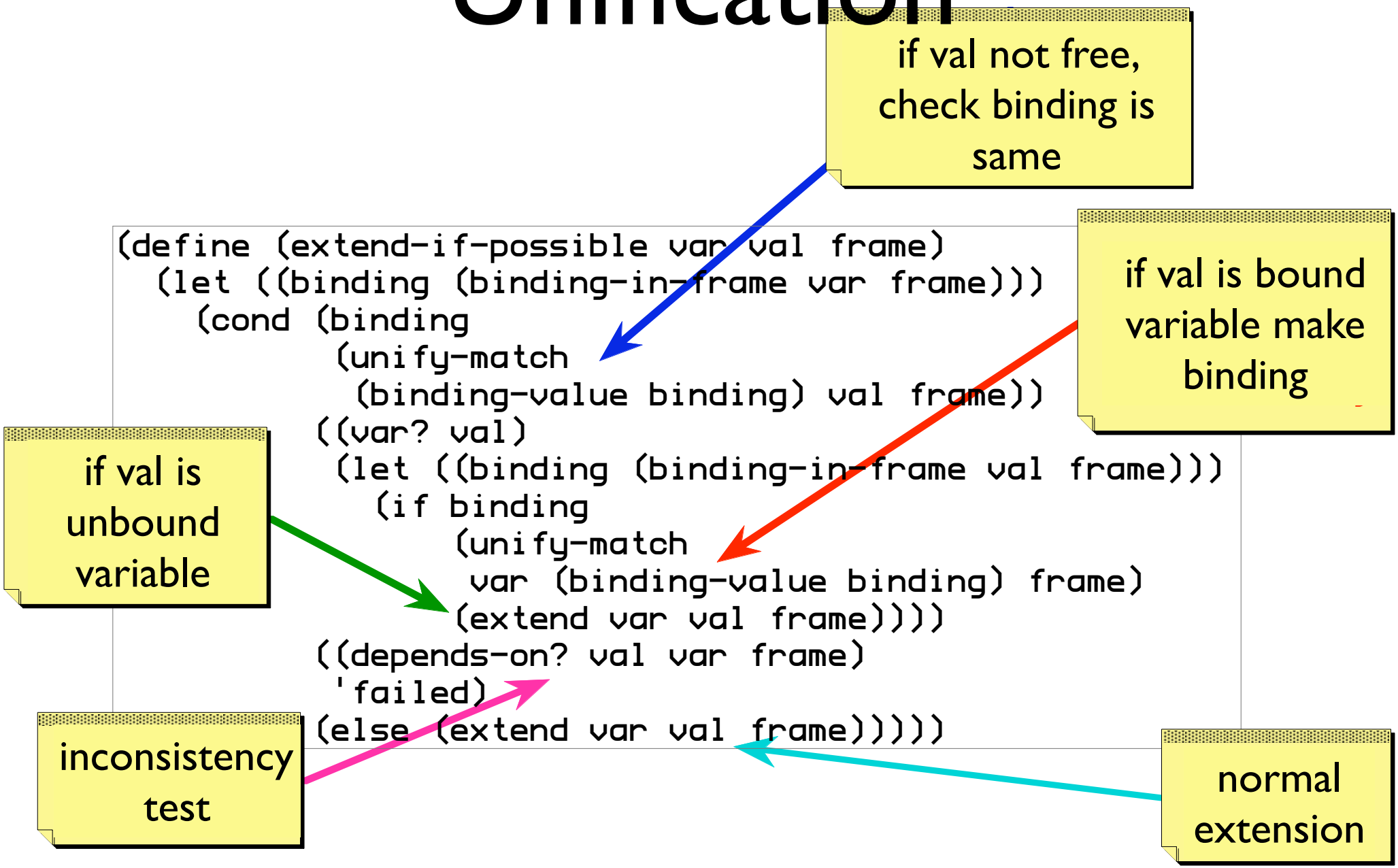
Unification



Unification



Unification



Unification

check that exp does not have var in transitive closure

```
(define (depends-on? exp var frame)
  (define (tree-walk e)
    (cond ((var? e)
           (if (equal? var e)
               true
               (let ((b (binding-in-frame e frame)))
                 (if b
                     (tree-walk (binding-value b))
                     false))))
          ((pair? e)
           (or (tree-walk (car e))
               (tree-walk (cdr e))))
          (else false)))
  (tree-walk exp))
```

Unification

controleer dat exp niet var in de transitieve afsluiting heeft

```
(define (depends-on? exp var frame)
  (define (tree-walk e)
    (cond ((var? e)
           (if (equal? var e)
               true
               (let ((b (binding-in-frame e frame)))
                 (if b
                     (tree-walk (binding-value b))
                     false))))
          ((pair? e)
           (or (tree-walk (car e))
               (tree-walk (cdr e))))
          (else false)))
  (tree-walk exp))
```

Initialising driver loop

```
(define (initialize-data-base rules-and-assertions)
  (define (deal-out r-and-a rules assertions)
    (cond ((null? r-and-a)
           (set! THE-ASSERTIONS (list->stream assertions))
           (set! THE-RULES (list->stream rules))
           'done)
          (else
           (let ((s (query-syntax-process (car r-and-a))))
             (cond
               ((rule? s)
                (store-rule-in-index s)
                (deal-out (cdr r-and-a) (cons s rules) assertions))
               (else
                (store-assertion-in-index s)
                (deal-out (cdr r-and-a) rules (cons s assertions))))))))
  (set! PROPERTY-LIST '())
  (put 'and 'qeval conjoin)
  (put 'or 'qeval disjoin)
  (put 'not 'qeval negate)
  (put 'lisp-value 'qeval lisp-value)
  (put 'always-true 'qeval always-true)
  (deal-out rules-and-assertions '() '()))
```