Methods for Proving Termination

Termination

1. Termination

Software Correctness

- Outputs Are Correct
- Terminates (or Doesn't)
- Resource issues
- Accuracy Issues
- Timing Issues

Termination

- Algorithm Halts for All (Specified) Inputs
  - Iterative Loops
  - Nested Loops
  - Recursive Loops
  - Symbolic Computation

Microsoft: Liveness

- A matter of practical importance:
  - Is every call to AcquireLock() is followed by a call to ReleaseLock()?
  - Does SerialPnpDispatch(…..) always return control back to its caller?

Plan

- Termination is Undecidable
- The Easy Cases
- The Hard Cases

Requirements

- Attendance and participation
- Readings and discussions
- Try to solve assignments
- final exam or term paper or system (tbd)

Readings

- Turing, 1936
- Strachey, 1965
- Katz & Manna, 1975

History

- Euclid
- Alan Turing
- Bob Floyd
- Zohar Manna

Euclid (c. -300)

Euclid's GCD algorithm appeared in his *Elements*.

Formulated geometrically: Find common measure for 2 lines.

Used repeated subtraction of the shorter segment from the longer.

ANTIQUE ALGORITHM

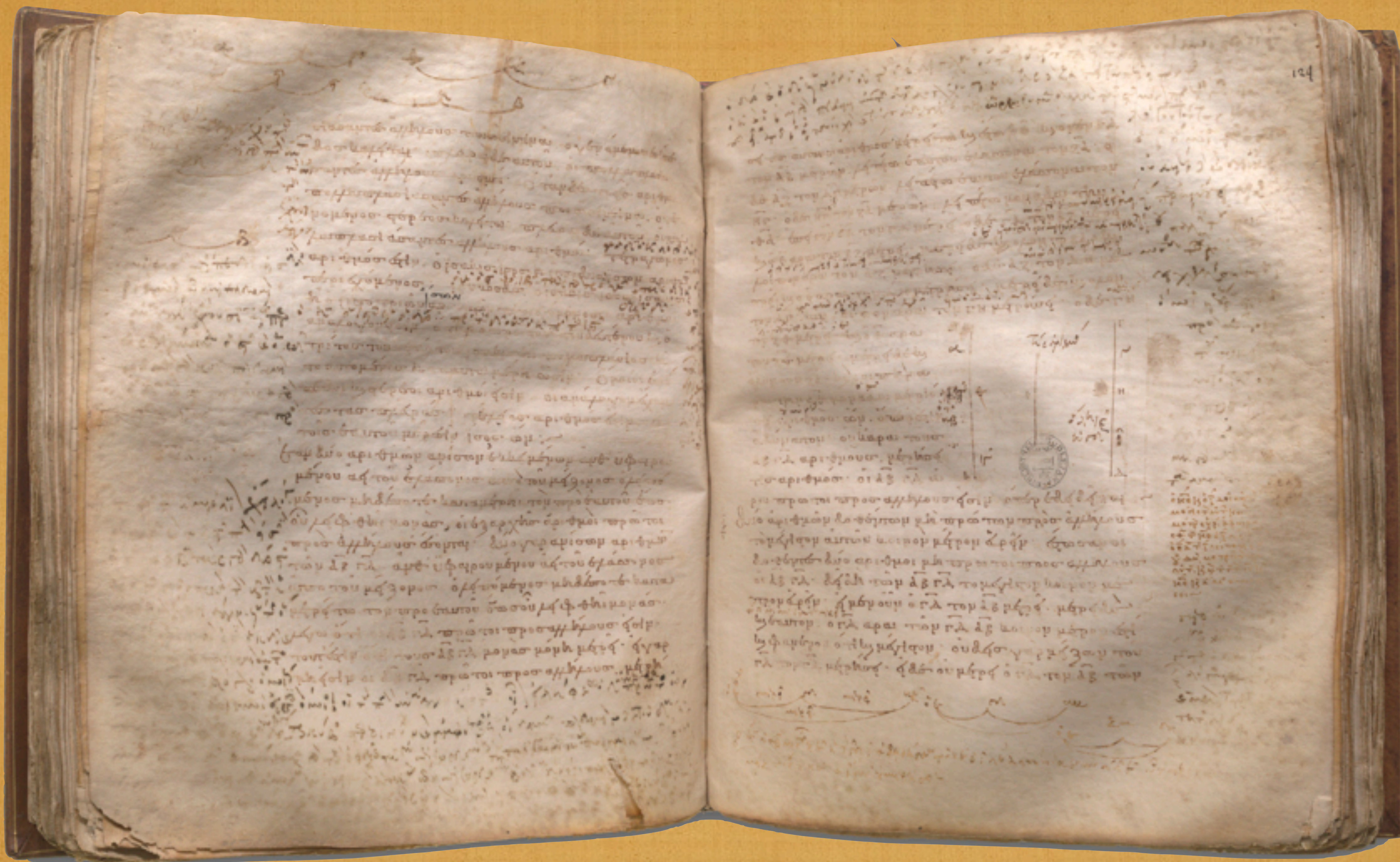


Image courtesy of the Clay Mathematics Institute

Antenaresis

Δύο ἀριθμῶν ἀνίσων ἐκκειμένων, ἀνθυφαιρουμένου δὲ ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος, ἐὰν ὁ λειπόμενος μηδέποτε καταμετρῇ τὸν πρὸ ἑαυτοῦ, ἕως οὗ λειφθῇ μονάς, οἱ ἐξ ἀρχῆς ἀριθμοὶ πρῶτοι πρὸς ἀλλήλους ἔσονται

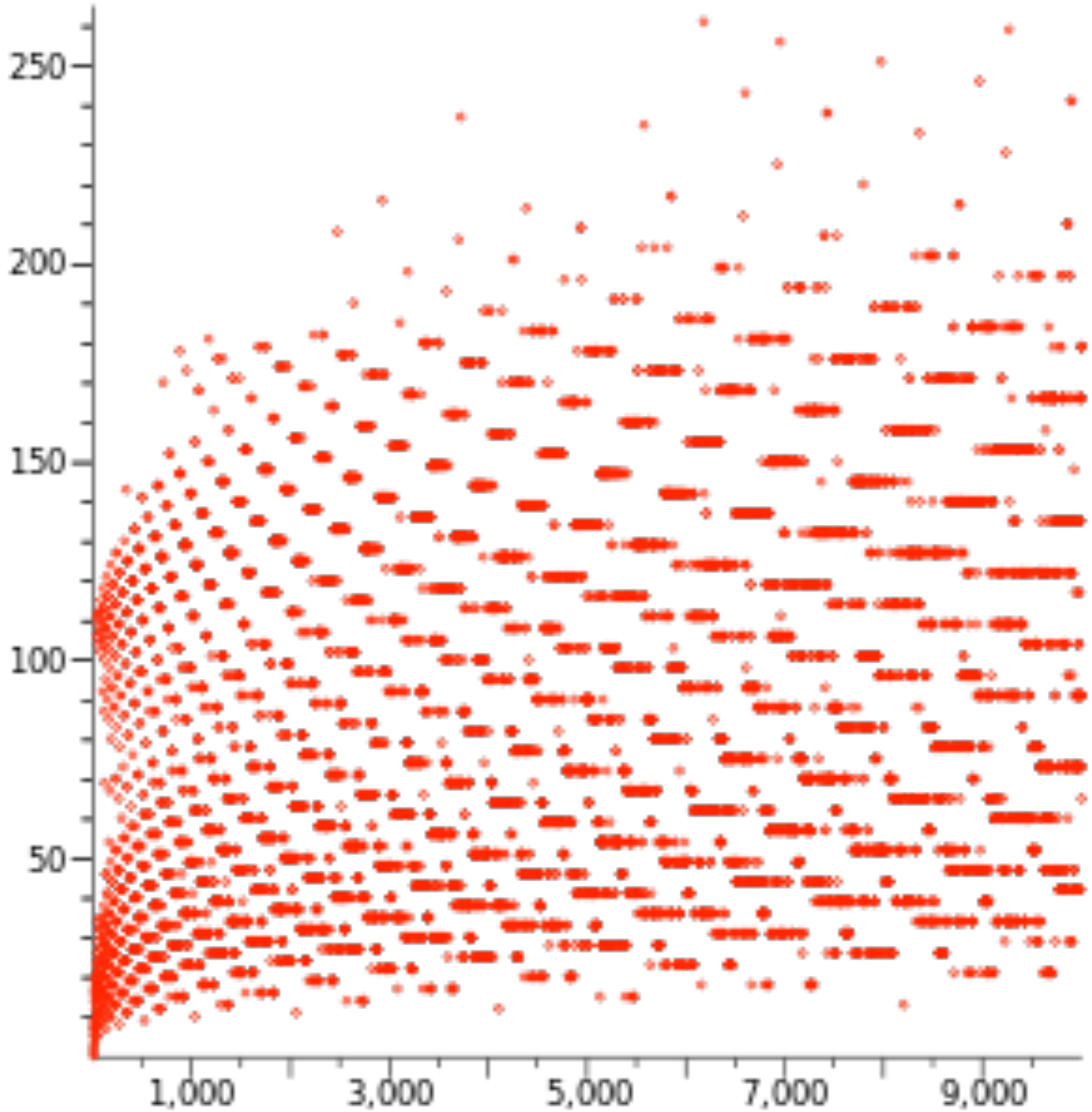
When two unequal numbers are set out, and the less is continually subtracted in turn from the greater, if the number which is left never measures the one before it until a unit is left, then the original numbers are relatively prime.

Greatest Common

```
repeat
  if m=n then return n
  if m<n then n := n-m
  if m>n then m := m-n
```

Hailstones

```
Loop until x=1
  if 2|x
    then x := x/2
    else x := 3x+1
```



250
200
150
100
50
1,000
3,000
5,000
7,000
9,000

A 2-MINUTE PROOF
OF THE
2nd-MOST IMPORTANT THEOREM
OF THE
2nd MILLENNIUM

by Doron Zeilberger

Written: Oct. 4, 1998

To the Editor,
The Computer Journal.

An impossible program

Sir,

A well-known piece of folk-lore among programmers holds that it is impossible to write a program which can examine any other program and tell, in every case, if it will terminate or get into a closed loop when it is run. I have never actually seen a proof of this in print, and though Alan Turing once gave me a verbal proof (in a railway carriage on the way to a Conference at the NPL in 1953), I unfortunately and promptly forgot the details. This left me with an uneasy feeling that the proof must be long or complicated, but in fact it is so short and simple that it may be of interest to casual readers. The version below uses CPL, but not in any essential way.

Suppose $T[$
(or program)
argument and
if run and tha
Consider the r

rec r

If $T[P] =$ **T**
only terminate
exactly the wr
that the functi

Churchill Coll
Cambridge.

Correspondence

g programmers
gram which can
every case, if it
when it is run.
is in print, and
bal proof (in a
nference at the
ptly forgot the
eeling that the
in fact it is so
erest to casual
out not in any

Suppose $T[R]$ is a Boolean function taking a routine (or program) R with no formal or free variables as its argument and that for all R, $T[R] =$ **True** if R terminates if run and that $T[R] =$ **False** if R does not terminate. Consider the routine P defined as follows

rec routine P

§ L : **if** $T[P]$ **go to** L

Return §

If $T[P] =$ **True** the routine P will loop, and it will only terminate if $T[P] =$ **False**. In each case $T[P]$ has exactly the wrong value, and this contradiction shows that the function T cannot exist.

Yours faithfully,
C. STRACHEY.

Churchill College,
Cambridge.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The fallacy in this argument lies in the assumption that β is computable. It would be true if we could enumerate the computable sequences by finite means, but the problem of enumerating computable sequences is equivalent to the problem of finding out whether a given number is the D.N of a circle-free machine, and we have no general process for doing this in a finite number of steps. In fact, by applying the diagonal process argument correctly, we can show that there cannot be any such general process.

Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.

- Construct the program

alan(p) = if halt(p,p) says "yes"
then "do nothing" forever
otherwise answer "yes"

- Consider the question halt(alan,alan).

Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
- Consider halt(alan,alan)

alan(alan) = if halt(alan,alan) says "yes"
        then "do nothing" forever
        otherwise answer "yes"

- halt(alan,alan)?

Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
- Consider halt(alan,alan)

alan(alan) = if halt(alan,alan) says "yes"
        then "do nothing" forever
        otherwise answer "yes"

- No answer: BAD

Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
- Consider halt(alan,alan)

alan(alan) = if halt(alan,alan) says "yes"
    then "do nothing" forever
    otherwise answer "yes"

- Yes: alan(alan) = do nothing forever: BAD

Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
- Consider halt(alan,alan)

alan(alan) = if halt(alan,alan) says "yes"
then "do nothing" forever
otherwise answer "yes"

- No: alan(alan) = yes: BAD

Size Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise -- provided |p|,|x| $\leq$ n
- Consider halt(alan,alan)

alan(x) = if halt(x,x) says "yes"
then "do nothing" forever
otherwise answer "yes"

- |alan| = |halt|+c > n

Size Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise -- provided |p|,|x| $\leq$ n
- Consider halt(alan,alan)

alan(x) = if halt(x,x) says "yes"
then "do nothing" forever
otherwise answer "yes"

- |halt| > n-c

Size Proof

- Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise -- provided |p|,|x| $\leq$ n
- Consider halt(alan,alan)

alan(x) = if halt(x,x) says "yes"
then "do nothing" forever
otherwise answer "yes"

- assuming almost nothing

Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

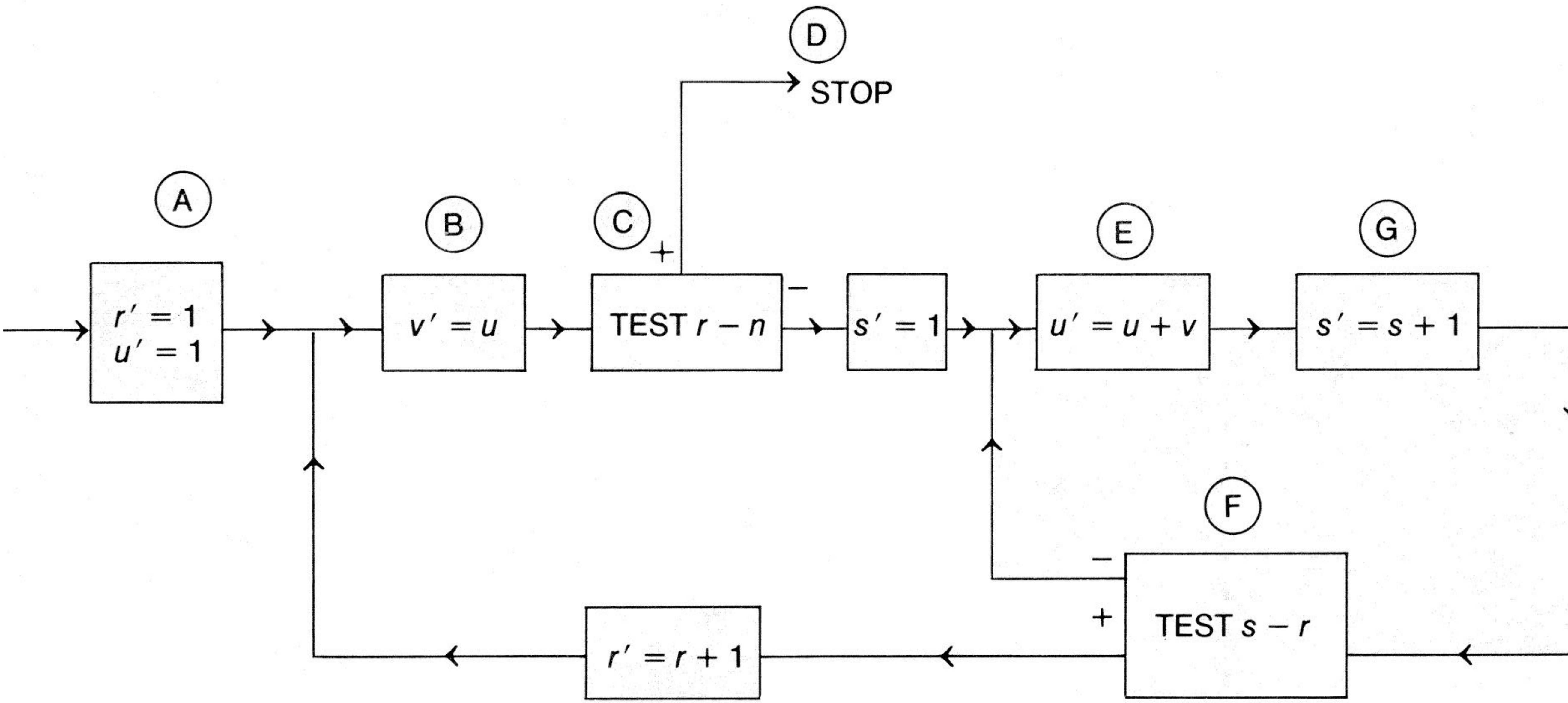
Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
 ----
26104
```

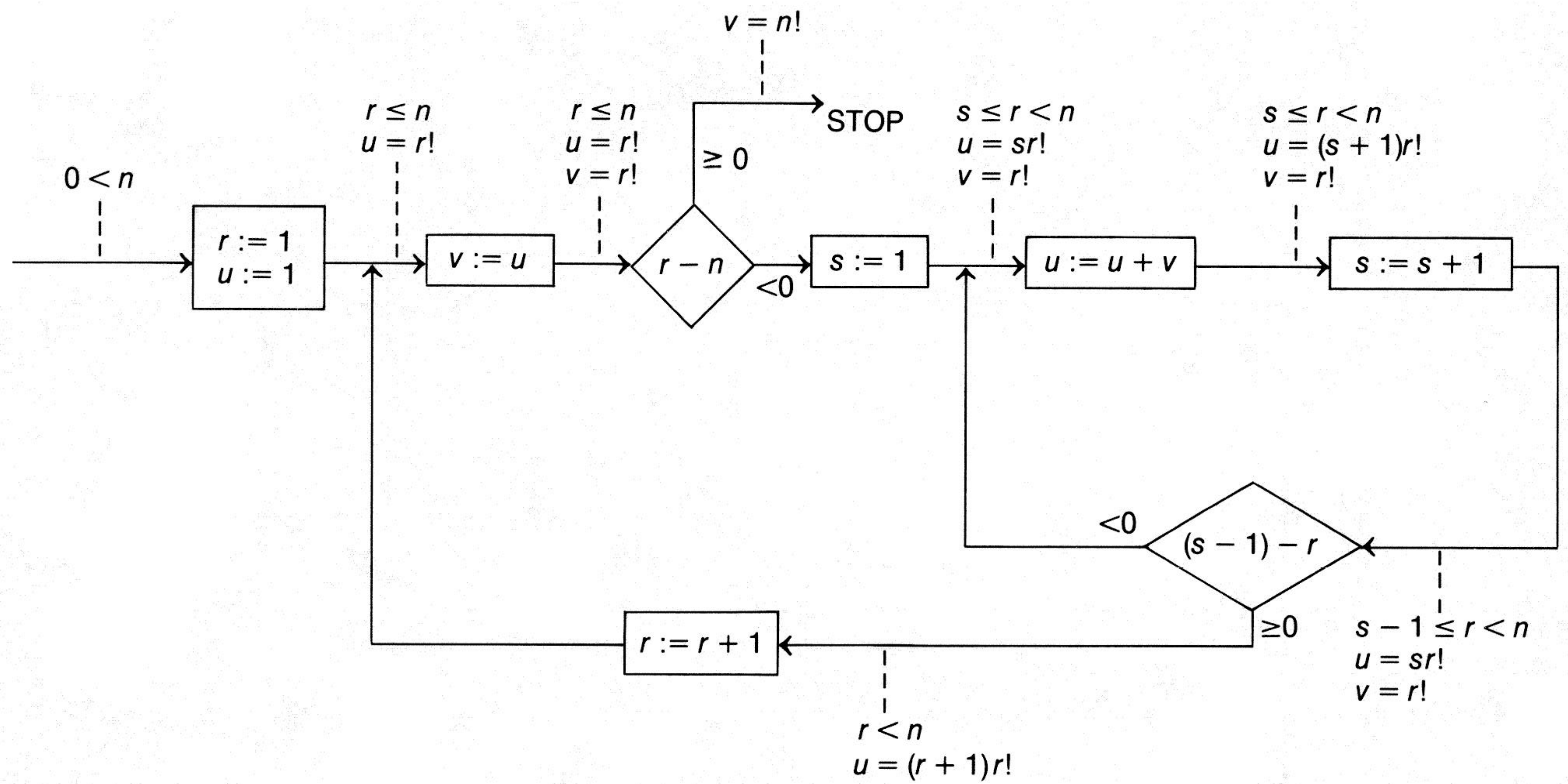
one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
1374
5906
6719
```


A
$r' = 1$
$u' = 1$
B
$v' = u$
C
TEST $r - n$
+
−
D
STOP
$s' = 1$
E
$u' = u + v$
G
$s' = s + 1$
F
TEST $s - r$
−
+
$r' = r + 1$

Invariants


$v = n!$
STOP
≥ 0
$0 < n$
$r := 1$
$u := 1$
$r \leq n$
$u = r!$
$v := u$
$r \leq n$
$u = r!$
$v = r!$
$r - n$
<0
$s := 1$
$s \leq r < n$
$u = sr!$
$v = r!$
$u := u + v$
$s \leq r < n$
$u = (s+1)r!$
$v = r!$
$s := s + 1$
<0
$(s-1) - r$
≥ 0
$s - 1 \leq r < n$
$u = sr!$
$v = r!$
$r := r + 1$
$r < n$
$u = (r+1)r!$

Invariants

```
r := 1
u := 1
loop    v := u                          1≤r≤n
        until r≥n
        s := 1
        loop   u := u+v        1≤s≤r+1
               s := s+1
               while s≤r
               repeat
```

He has also to verify that each of the assertions in the lower half of the table is correct. In doing this the columns may be taken in any order and quite independently. Thus for column B the checker would argue. "From the flow diagram we see that after B the box $v' = u$ applies. From the upper part of the column for B we have $u = r$. Hence $v' = r$ i.e. the entry for v i.e. for line 31 in C should be r. The other entries are the same as in B".

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops. To the pure mathematician it is natural to give an ordinal number. In this problem the ordinal might be $(n - r)\,\omega^2 + (r - s)\,\omega + k$. A less highbrow form of the same thing would be to give the integer $2^{80}(n - r) + 2^{40}(r - s) + k$. Taking the latter case and the step from B to C there would be a decrease from $2^{80}(n - r) + 2^{40}(r - s) + 5$ to $2^{80}(n - v) + 2^{40}(r - s) + 4$. In the step from F to B there is a decrease from $2^{80}(n - r) + 2^{40}(r - s) + 1$ to $2^{80}(n - r\ 1) + 2^{40}(r + 1 - s) + 5$.

In the course of checking that the process comes to an end the time involved may also be estimated by arranging that the decreasing quantity represents an upper bound to the time till the machine stops.

Turing's Proof

- The checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops. To the pure mathematician it is natural to give an ordinal number. In this problem the ordinal might be $(n - r)\omega^2 + (r - s)\omega + k$. A less highbrow form of the same thing would be to give the integer $2^{80}(n - r) + 2^{40}(r - s) + k$.

Acta Informatica 5, 333—352 (1975)

A Closer Look at Termination

Shmuel Katz and Zohar Manna

Received October 16, 1974

Summary. Several methods for proving that computer programs terminate are presented and illustrated. The methods considered involve (a) using the "no-infinitely-descending-chain" property of well-founded sets (Floyd's approach), (b) bounding a counter associated with each loop (*loop* approach), (c) showing that some exit of each loop must be taken (*exit* approach), or (d) inducting on the structure of the data domain (Burstall's approach). We indicate the relative merit of each method for proving termination or non-termination as an integral part of an automatic verification system.

Greatest Common

```
repeat
  if m=n then return n
  if m<n then n := n-m
  if m>n then m := m-n
```

Method

- Find a measure that decreases with each iteration
- And cannot decrease forever

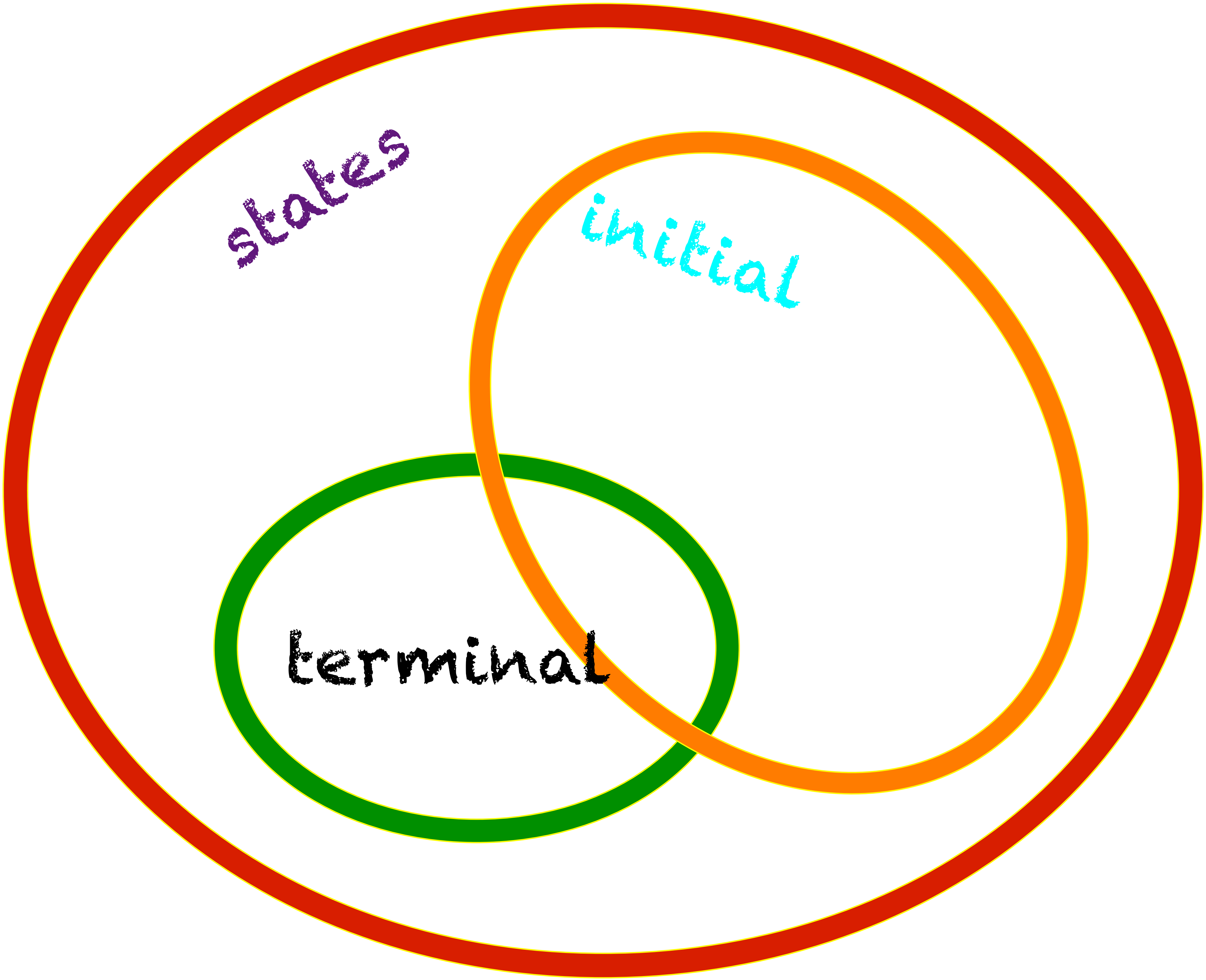
Loop Invariants

- Need to know that m and n are nonnegative

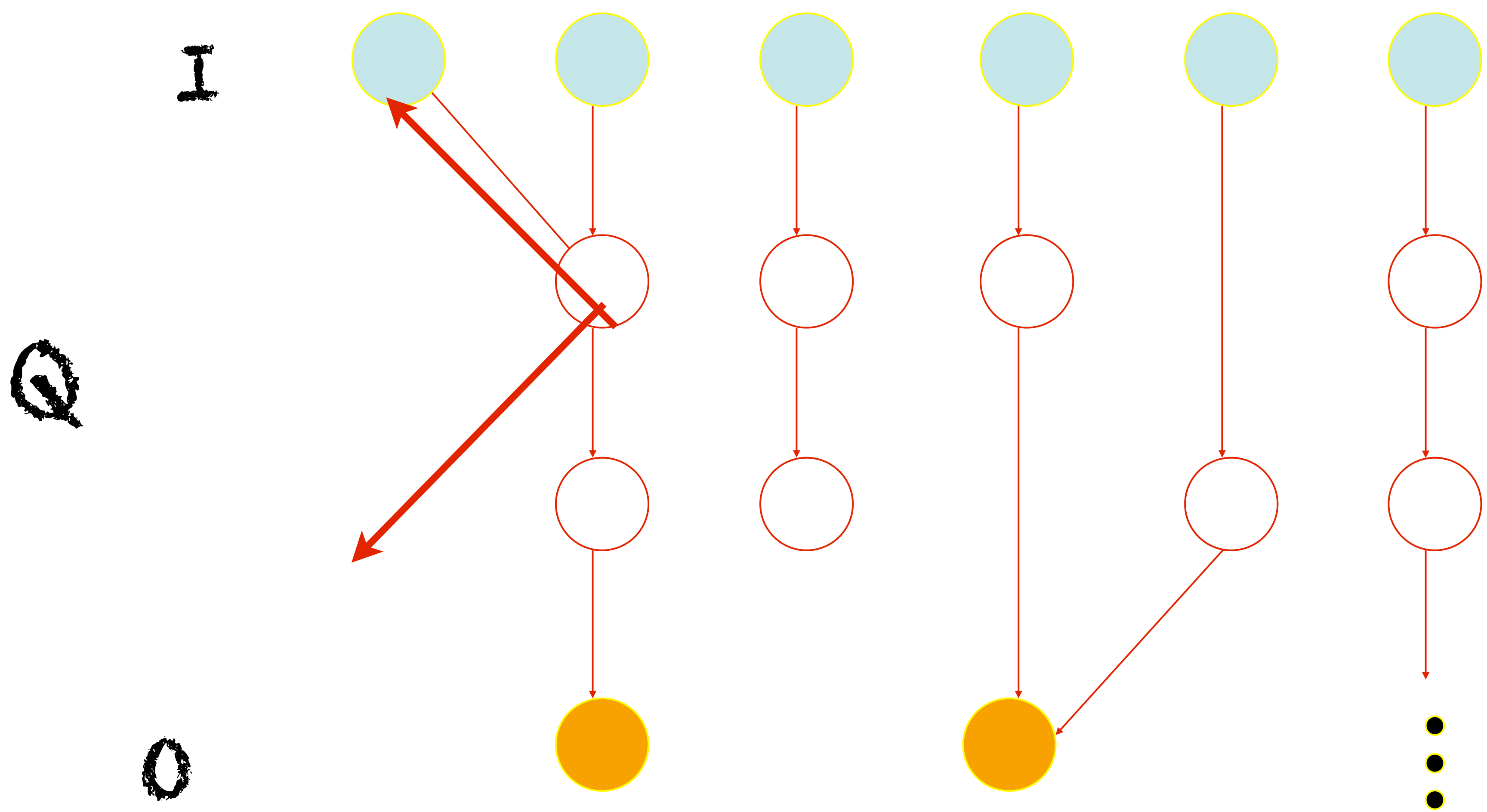
Knuth (1966)

A computational
method
comprises
a set of states…

In this way we can divorce abstract algorithms from particular programs that represent them.


states
initial
terminal

Transition System



Discrete Steps

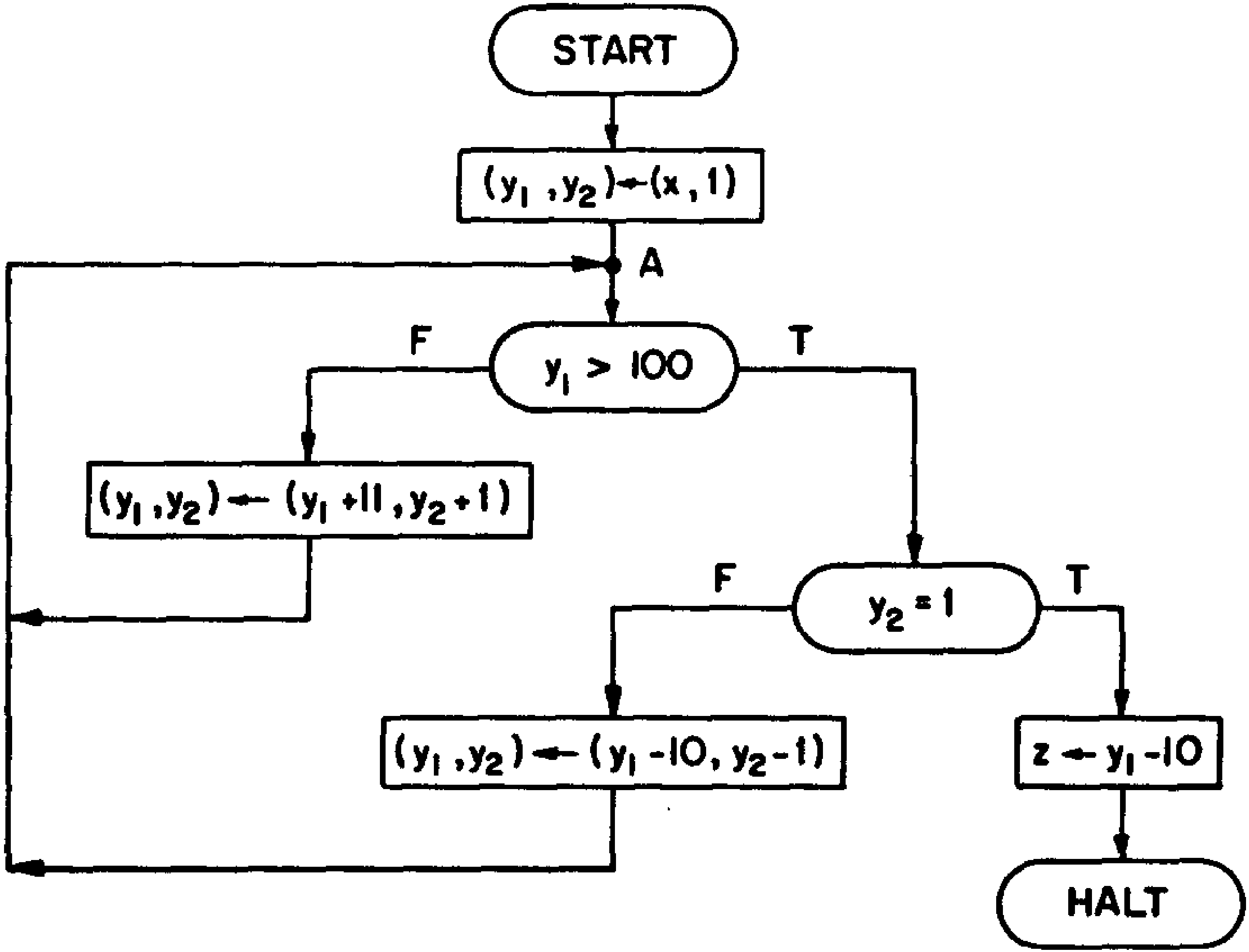
- An algorithm is a discrete state-transition system.
- Its transitions are a binary relation on states.

Hartley Rogers, Jr.

For any given input, the computation is carried out in a discrete stepwise fashion, without use of continuous methods or analogue devices.

Computation is carried forward deterministically, without resort to random methods or devices, e.g., dice.

Non-Example



START
$(y_1, y_2) \leftarrow (x, 1)$
A
$y_1 > 100$
F
T
$(y_1, y_2) \leftarrow (y_1 + 11, y_2 + 1)$
$y_2 = 1$
F
T
$(y_1, y_2) \leftarrow (y_1 - 10, y_2 - 1)$
$z \leftarrow y_1 - 10$
HALT

91

```
(a) int mccarthy (int n)
(b) {int c;
(c)     for (c = 1; c != 0; ) {
(d)         if (n > 100) {
(e)             n = n - 10;
(f)             c--;
(g)         } else {
(h)             n = n + 11;
(i)             c++;
(j)         }
(k)     }
(l)     return n;
(m) }
```

Solve for Decrease

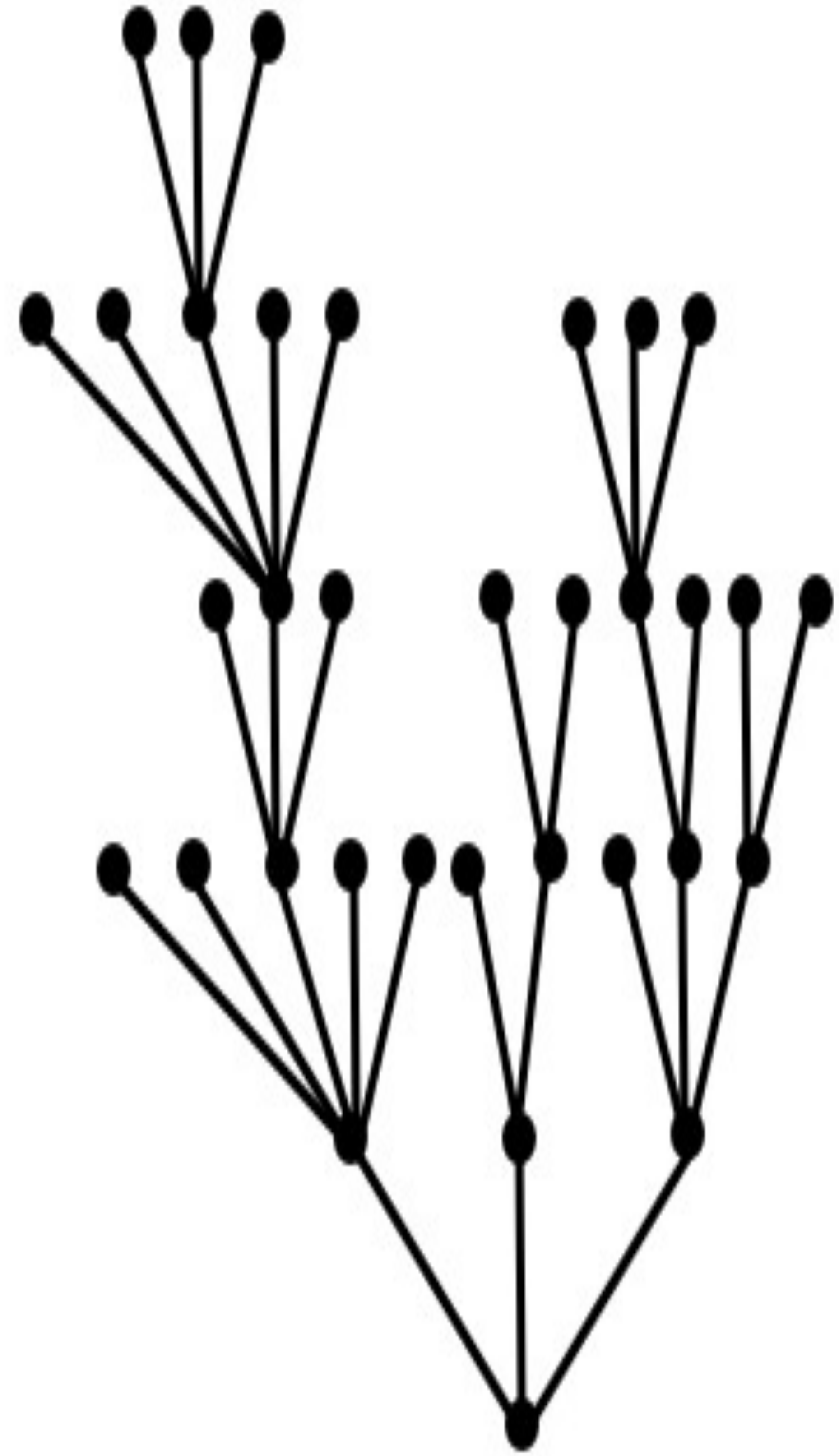
- Suppose measure is a linear combination of the variables
- $n > 100$: $an+bc > a(n-10)+b(c-1)$
- $n < 99$: $an+bc > a(n+11)+b(c+1)$
  - $11a+b < 0 < 10a+b$

Artificial Variables

```
(a) int mccarthy (int n)
(b) {int c; i=0; j=0;
(c)     for (c = 1; c != 0; ) {
(d)         if (n > 100) {
(e)             n = n - 10;
(f)             c--; i++;
(g)         } else {
(h)             n = n + 11;
(i)             c++; j++;
(j)         }
(k)     }
(l)     return n;
(m) }
```

Infer Invariants

- $c = j-i+1$
- ...



König's Lemma

- A tree is finite (has finitely many edges)

  if and only if

  - all nodes have finite degree

  and

  - all branches (simple paths) have finite length.

Binary Search

- $l := a$
- $r := b$
- loop until $l=r$
  - $m := [(l+r) \div 2]$
  - if $y[m] \geq x$
    - then $r :=$??
- else $l :=$???
- given:
  - $a \leq b$
  - $y[j] \leq y[j+1]$
  - $x=y[i]$, $a \leq i \leq b$
- unbounded integers

Binary Search

- $l := a$
- $r := b$
- loop until $l = r$
  - $m := [(l+r) \div 2]$
  - if $y[m] \geq x$
    - then $r := \underline{m}$
    - else $l := \underline{m+1}$

- given:
  - $a \leq b$
  - $y[j] \leq y[j+1]$
  - $x = y[i]$, $a \leq i \leq b$
- unbounded integers
- invariants:
  - $a \leq l \leq r \leq b$
  - $y[l] \leq x \leq y[r]$

Binary Search is Hard

- Don Knuth: the idea is comparatively straightforward; the details can be surprisingly tricky.
- Jon Bentley assigned it as a problem in a course for professional programmers. 90% failed even after several hours.
- accurate code is only found in 5 out of 20 textbooks.
- Bentley's own implementation (in his Programming Pearls) contains an error that went undetected for over 20 years.

Termination

2. Games

Readings

- Floyd, "Assigning Meaning to Programs"
- "Proving Termination with Multiset Orderings"

Robert W. Floyd

ASSIGNING MEANINGS TO PROGRAMS[1]

Introduction. This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion of an interpretation of a program: that is, an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken. To prevent an interpretation from being chosen arbitrarily, a condition is imposed on each command of the program. This condition guarantees that whenever a command is reached by way of a connection whose associated proposition

Invariants

```
r := 1
u := 1
loop    v := u                              1≤r≤n
        until r≥n
        s := 1
        loop    u := u+v                    1≤s≤r+1
                s := s+1
                while s≤r
                repeat
        r := r+1
        repeat
```

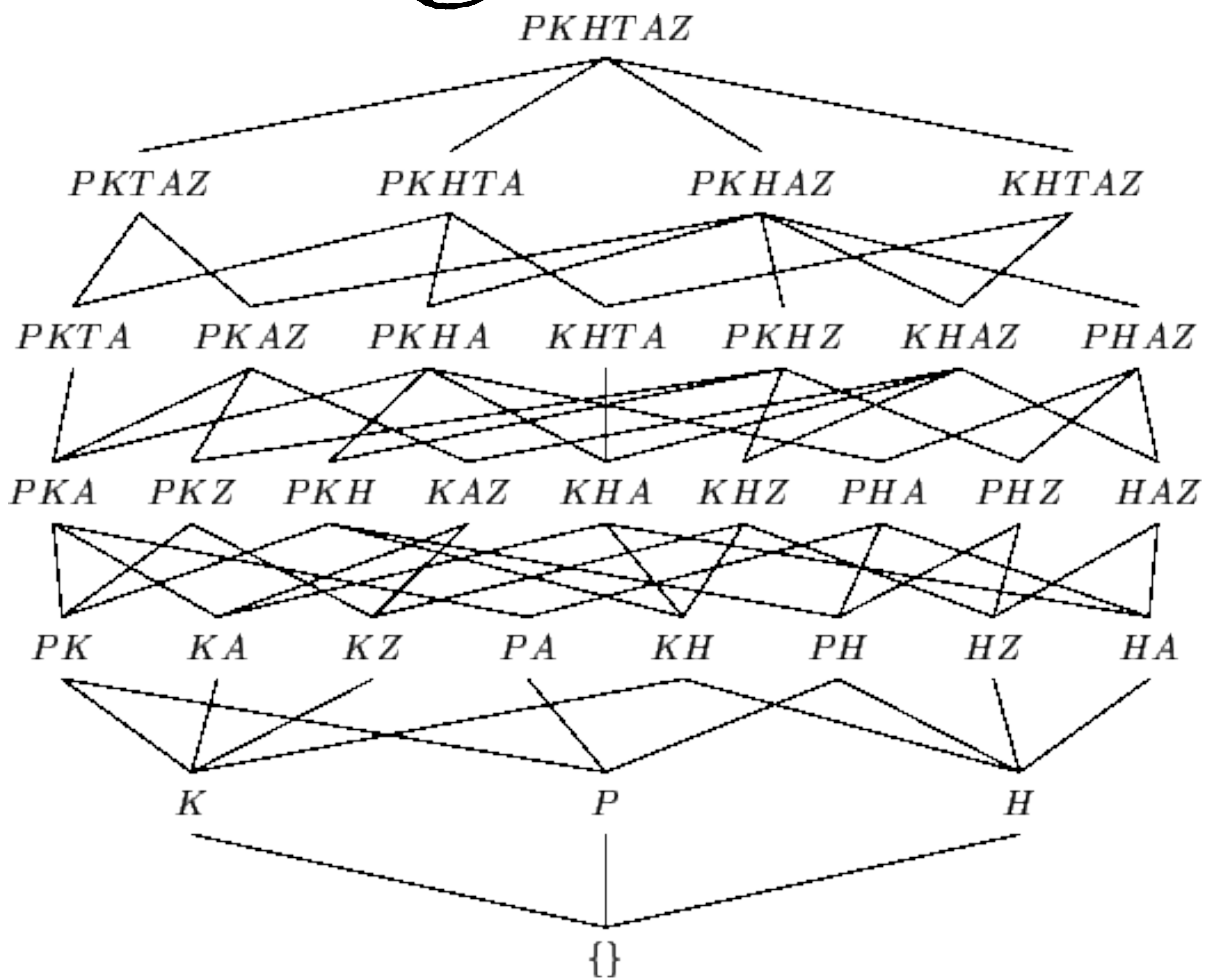
Double Induction

- Inner loop
- Outer loop

Orderings

- Partial ordering
  - Irreflexive
  - Transitive
  - Asymmetric

Hasse Diagram



Orderings (Well-founded)

- Partial ordering
  - Irreflexive
  - Transitive
  - Asymmetric
- Well-founded
  - No infinite decreasing chains

Well-Founded

- N, $>$
- Z^-, $<$
- Z, ???
- Finite trees, subtree
- NxN, lexicographic
- Σ^*, subword
- Σ^*, lexicographic ???

Couples

$(a,b) > (a',b')$

- Component-wise: $a>a'$ & $b\geq b'$ or $a\geq a'$ & $b>b'$
- Lexicographic: $a>a'$ or $a=a'$ & $b>b'$
- Reverse lexicographic: $a>a'$ & $b=b'$ or $b>b'$
- Pairs of pairs: $(1,0) > (0,(1,0)) > \ldots$

Mixed Couples

If V and W are well-founded, then their pairs VxW are well-founded lexicographically.

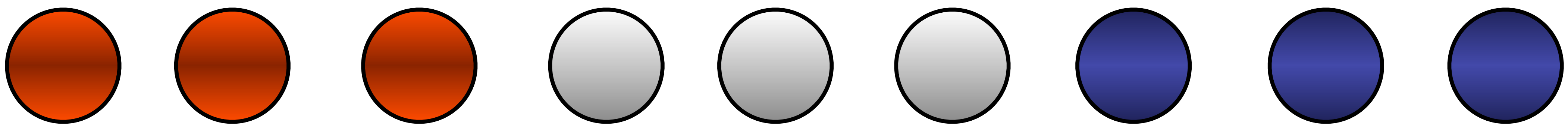
Ackermann

- Termination of recursion
  - Induction on (m,n)

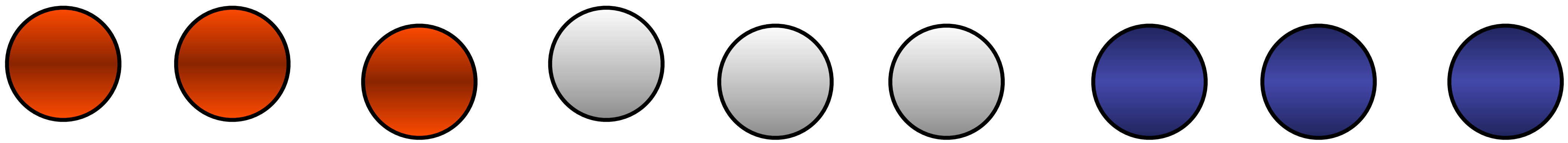
Turing's Program

```
r := 1
u := 1
loop   v := u
       until r≥n                    (n-r,r-s)
       s := 1
       loop  u := u+v
             s := s+1
             while s≤r
             repeat
       r := r+1
       repeat
```

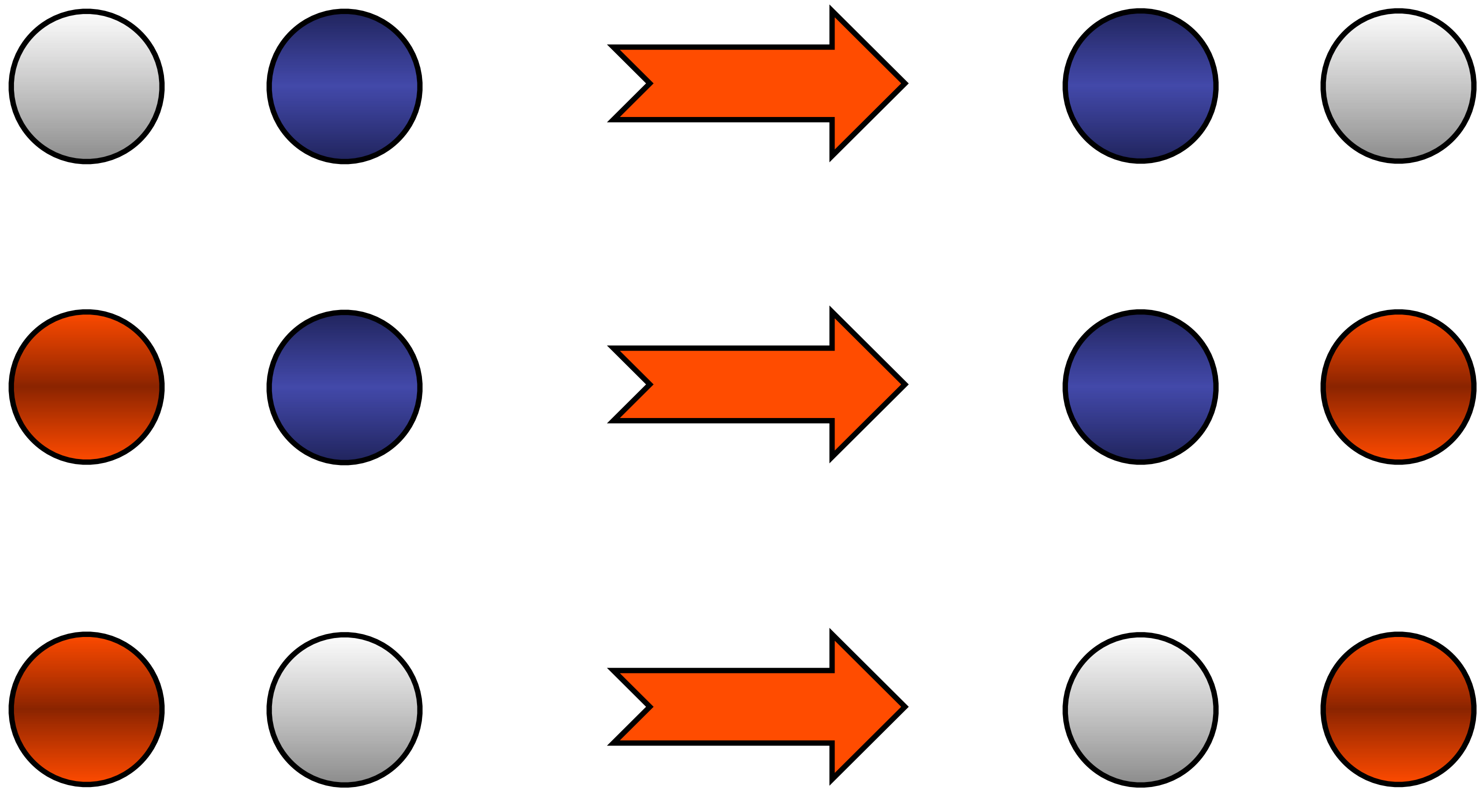
Dutch National Flag



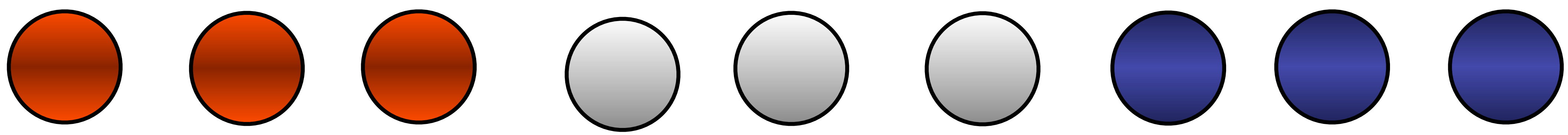
Dutch National Flag



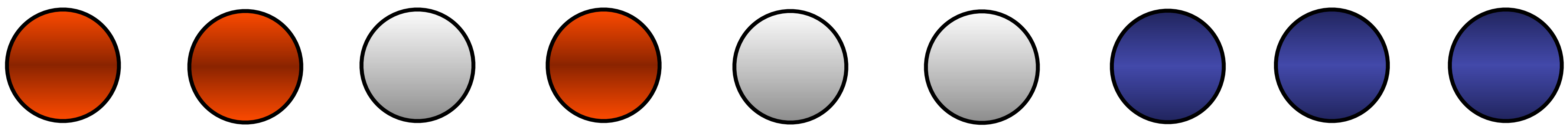
Flag Problem



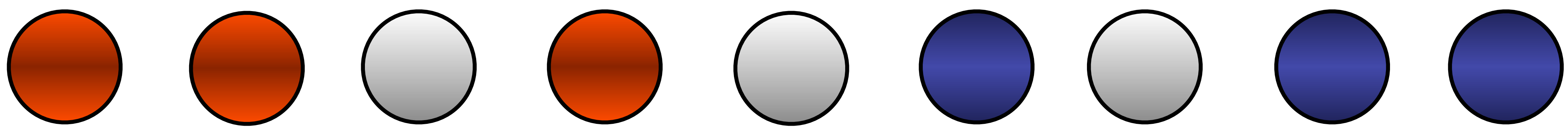
Dutch National Flag



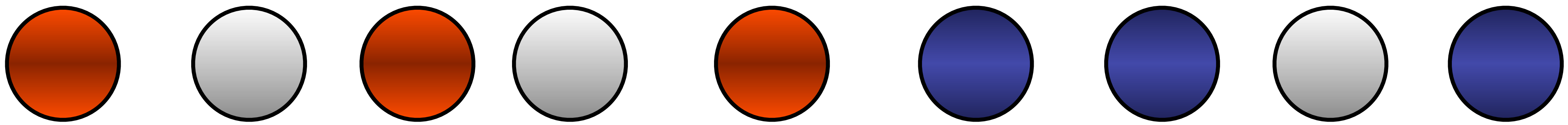
Dutch National Flag



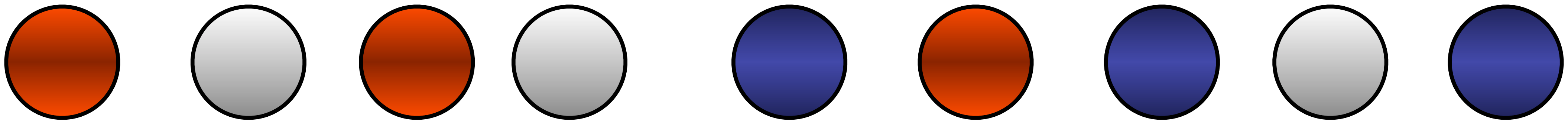
Dutch National Flag



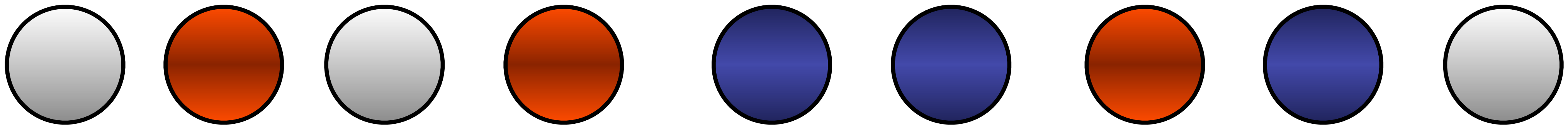
Dutch National Flag



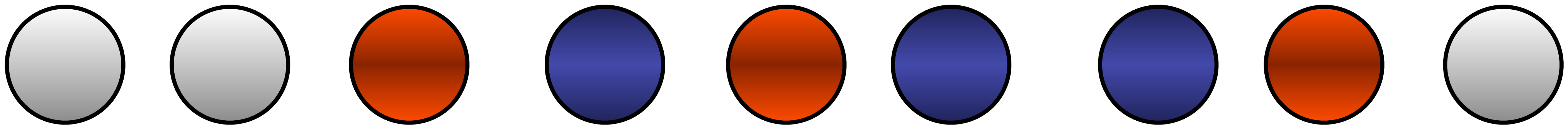
Dutch National Flag



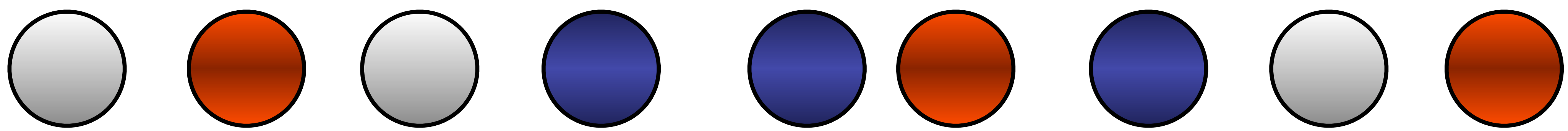
Dutch National Flag



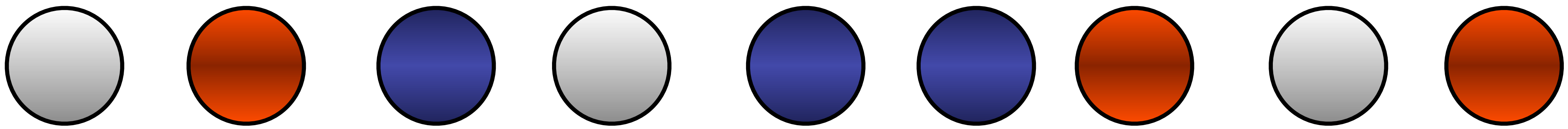
Dutch National Flag



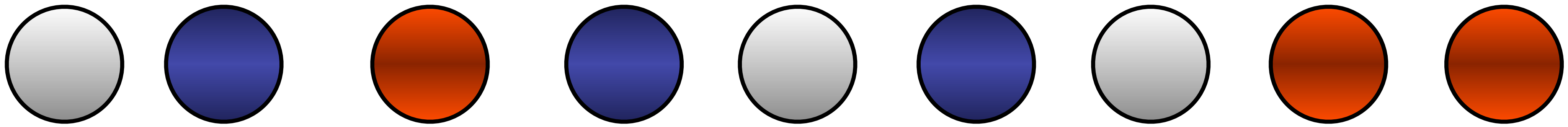
Dutch National Flag



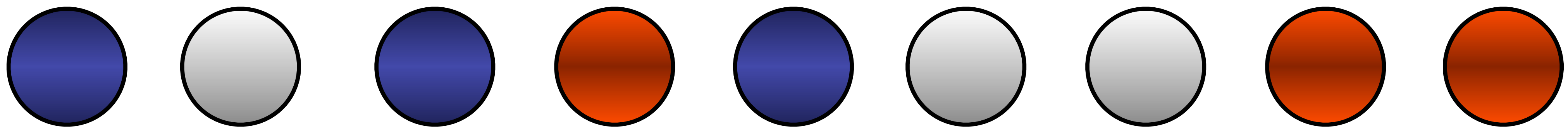
Dutch National Flag



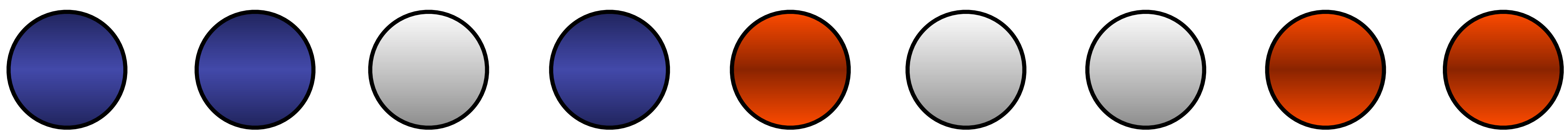
Dutch National Flag



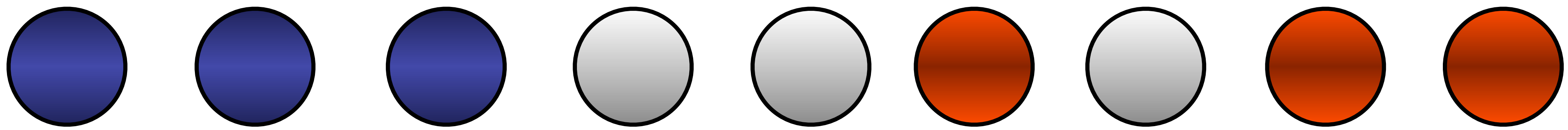
Dutch National Flag



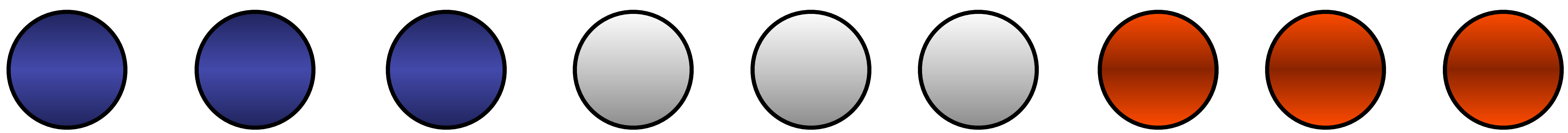
Dutch National Flag



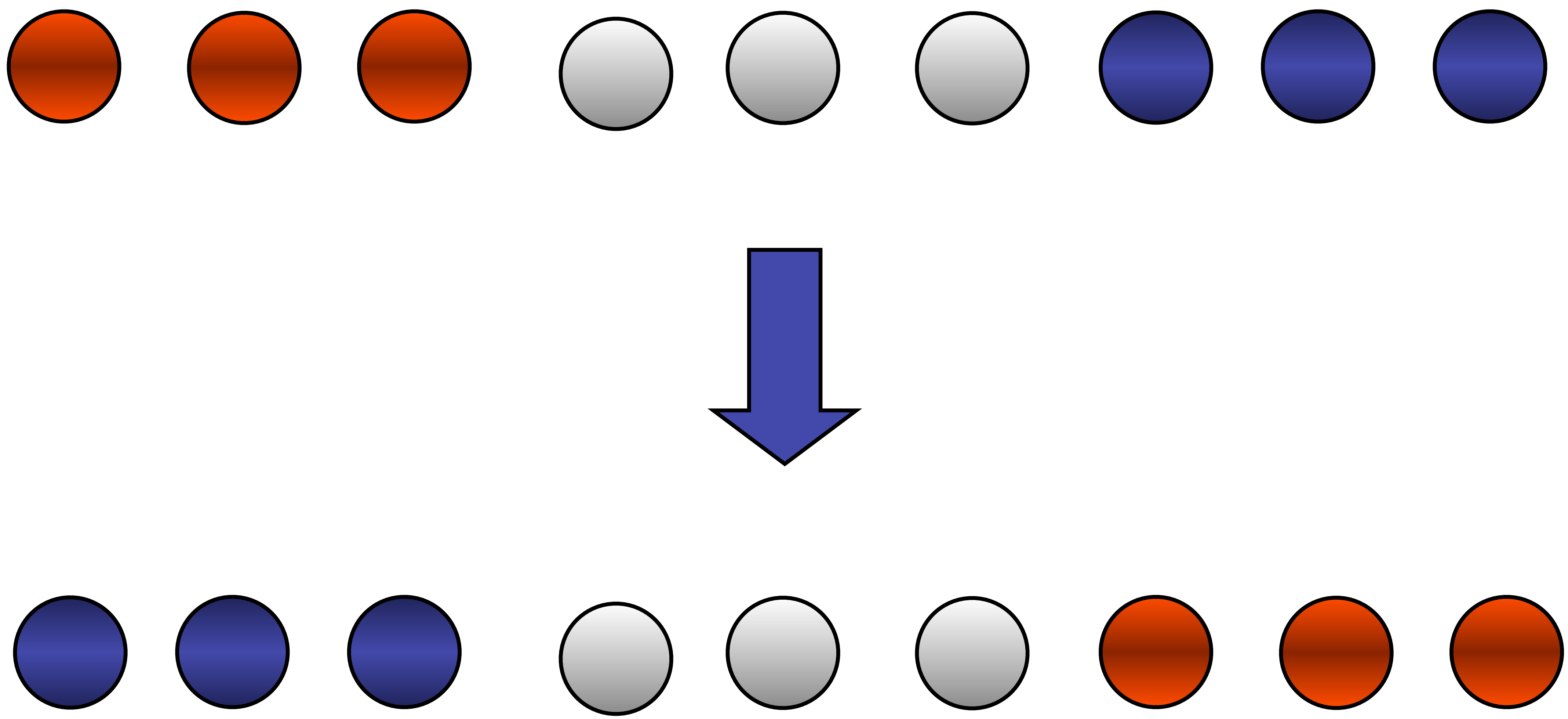
Dutch National Flag



Dutch National Flag



Dutch National Flag



Ackermann's

$$A(0,n) = n+1$$

$$A(m+1,0) = a(m,1)$$

$$a(m+1,n+1) = a(m,a(m+1,n))$$

Ackermann

```
      INTEGER FUNCTION  ACKER(M, N)
C COMPUTE ACKERMANN FUNCTION, DEFINED BY
C                  ACKER(0, N)  =  N+1 ,
C                  ACKER(M+1, 0) = ACKER(M, 1)  ,
C                  ACKER(M+1, N+1) = ACKER(M, ACKER(M+1, N)) .
C
C  SIZE OF VALUE AND PLACE TABLES IS ONE MORE THAN LARGEST  M  EXPECTED.
      INTEGER  VALUE(6), PLACE(6)
C  TEST FOR ZERO  M .
      IF (M .NE. 0)  GO TO 1
      ACKER = N+1
      RETURN
C  NON-ZERO  M . INITIALIZE FOR ITERATION.
1     VALUE = 1
      PLACE = 0
C  ITERATION LOOP. GET NEW VALUE.
2     VALUE = VALUE+1
      PLACE = PLACE+1
C   PROPAGATE VALUE.
      DO 4  I=1,M
       IF (PLACE(I) .NE. 1)  GO TO 3
C    INITIATE NEW LEVEL.
       VALUE(I+1) = VALUE
       PLACE(I+1) = 0
       IF (I .EQ. M)  GO TO 5
       GO TO 2
 3     IF (PLACE(I) .NE. VALUE(I+1))  GO TO 2
       VALUE(I+1) = VALUE
 4     PLACE(I+1) = PLACE(I+1)+1
C   CHECK FOR END OF ITERATION.
5     IF (PLACE(M+1) .NE. N)  GO TO 2
      ACKER = VALUE
      RETURN
         E N D
```

Ackermann

- $a(4,4) \approx 2\uparrow 7 - 3$
- Computation is much longer
- Fact: $a(m,n) > m+n \geq m,n$

Double Induction

- Call by value termination

- Assume terminating for smaller m
    - Assume terminating for same m and smaller n

Primitive Recursion

- 0
- +1
- projections
- composition
- f(x,n) := if n=0 then g(x) else h(f(x,n-1),x,n-1)

Ackermann's Function

- $A(0,n) = n+1$
- $A(m+1,0) = A(m,1)$
- $A(m+1,n+1) = A(m,A(m+1,n))$

$A(m,n) > m+n$

- Induction on (m,n)
  - $A(0,n) = n+1 > n$
  - $A(m+1,0) = A(m,1) > m+1$
  - $A(m+1,n+1) = A(m,A(m+1,n))$
    $> m+A(m+1,n) \geq m+n+2$

$x > y \Rightarrow A(m,x) > A(m,y)$

- Induction on (m,x)
  - $A(0,x) = x+1 > y+1 = A(0,y)$
  - $A(m+1,x+1) = A(m,A(m+1,x)) > A(m,A(m+1,y)) = A(m+1,y+1)$

$x>y \Rightarrow A(x,n) > A(y,n)$

- Induction on (x,n)
  - $A(x,n) > x+n > n = A(0,n)$
  - $A(x+1,0) = A(x,1) > A(y,1) = A(y+1,0)$
  - $A(x+1,n+1) = A(x,A(x+1,n)) > A(y,A(x+1,n)) > A(y,A(y+1,n)) = A(y+1,n+1)$

$A(m+n+2,x) >$

- Induction $(m+n,x)$
  - $A(n+2,x) > A(n+1,x) \geq A(n,x)+1 = A(0,A(n,x))$
  - $A(m+n+2,0) = A(m+n+1,1) > A(m,A(n-1,1)) = A(m,A(n,0))$
  - $A(m+n+2,x+1) = A(m+n+1,A(n+m+2,x)) > A(m,A(n,A(m,x))) > A(m,A(n,x+m)) \geq A(m,A(n,x+1))$

A isn't Primitive

- Denote $x = x_1,\ldots,x_k$ and $x_m = \max x_j$
- Say $A_i > g$ if $A(i,x_m) > g(x)$ for all x
- Easy: $A_0 > 0$; $A_1 > +1$; $A_0 > proj_i$
- Suppose $f(x) = h(g_1x,\ldots,g_kx)$, $A_s > g_1,\ldots,g_k,h$
  - $A_{2s+2} > f$: $A(2s+2,x) > A(s,A(s,x))$

A isn't Primitive

- Suppose $A_s > g,h$ and $f(x,n) = \text{if } n=0 \text{ then } g(x) \text{ else } h(f(x,n-1),x,n-1)$
- $A(r,n+x_m) > f(x,n)$, $r = 2s+1$, by induction on n:
  - $f(x,0) = g(x) < A(s,x_m) < A(r,0+x_m)$
  - $f(x,n+1) = h(f(x,n),x,n) < A(s,\max\{f(x,n),n,x_m\}) < A(s,A(r,n+x_m)) < A(2s,A(r,n+x_m)) = A(r,n+1+x_m)$
- $f(x,n) < A(r,n+x_m) < A(r,2N+3) = A(r,A(2,N)) < A(r+4,N)$ where $N = \max\{n,x_m\}$

Basic A(m,n)

```
   DIM s(tsize + 1)

    t = 1: s(t) = m
    DO
       c = c + 1
       m = s(t): t = t - 1
       IF m = 0 THEN
          n = n + 1
       ELSEIF n = 0 THEN
          t = t + 1: s(t) = m - 1
          n = 1
       ELSE
          t = t + 1: s(t) = m - 1
          t = t + 1: s(t) = m
          n = n - 1
       END IF
       IF t > d THEN
          d = t
          IF d > tsize THEN
             PRINT "failure": END
          END IF
       END IF
    LOOP UNTIL t = 0

A = n
END FUNCTION
```

Basic A(m,n)

```
   DIM s(tsize + 1)

    t = 1: s(t) = m
    DO
       c = c + 1
       m = s(t): t = t - 1
       IF m = 0 THEN
          n = n + 1
       ELSEIF n = 0 THEN
          t = t + 1: s(t) = m - 1
          n = 1
       ELSE
          t = t + 1: s(t) = m - 1
          t = t + 1: s(t) = m
          n = n - 1
       END IF
       IF t > d THEN
          d = t
          IF d > tsize THEN
             PRINT "failure": END
          END IF
       END IF
    LOOP UNTIL t = 0

A = n
END FUNCTION
```

Basic A(m,n)

```
   DIM s(tsize + 1)

    t = 1: s(t) = m
    DO
       c = c + 1
       m = s(t): t = t - 1
       IF m = 0 THEN
          n = n + 1
       ELSEIF n = 0 THEN
          t = t + 1: s(t) = m - 1
          n = 1
       ELSE
          t = t + 1: s(t) = m - 1
          t = t + 1: s(t) = m
          n = n - 1
       END IF
       IF t > d THEN
          d = t
          IF d > tsize THEN
             PRINT "failure": END
          END IF
       END IF
    LOOP UNTIL t = 0

A = n
END FUNCTION
```

s(1:tsize)
lexicographically

Sequences

(a,b,c,...) > (a',b',c',d',...)

- Lex is bad : 10 > 010 > 0010 > ...
- Length-lex: 0010 > 010 > 001 > 10 > 01

Unbounded

- Sorted-lex: 221 > 211110000 > 2111000000 > …
- Sorted-lex: ∞∞21 > ∞88880 > 9998888000 > …

Sorted Sequences

- $s11 \geq s12 \geq s13 \geq \ldots \geq s1j \geq \ldots$
- $s21 \geq s22 \geq s23 \geq \ldots \geq s2j \geq \ldots$
- etc. …
- Let j be first unstable column, changing at i
- $s\_1,1 = s\_i,1 \geq s\_i,j > s\_i{+}1,j$
- Consider rest: $s[i+1..\infty, j..\infty]$ and continue

Harder A(m,n)

```
t := 1
s[t] := m
loop
   c := c + 1
   m := s[t]
   t := t - 1
   if m = 0
   then
      n := n + 1
   elseif n = 0
   then
      t := t + 1
      s[t] := m - 1
      n := 1
   else
      t := t + 2
      s[t-1] := m - 1
      s[t] := m
      n := n - 1
   until t = 0
```

s can grow and grow

(sorted) lex doesn’t work

Harder A(m,n)

```
t := 1
s[t] := m
loop
   c := c + 1
   m := s[t]
   t := t - 1
   if m = 0
   then
      n := n + 1
   elseif n = 0
   then
      t := t + 1
      s[t] := m - 1
      n := 1
   else
      t := t + 2
      s[t-1] := m - 1
      s[t] := m
      n := n - 1
   until t = 0
```

$N := a(m,n)$

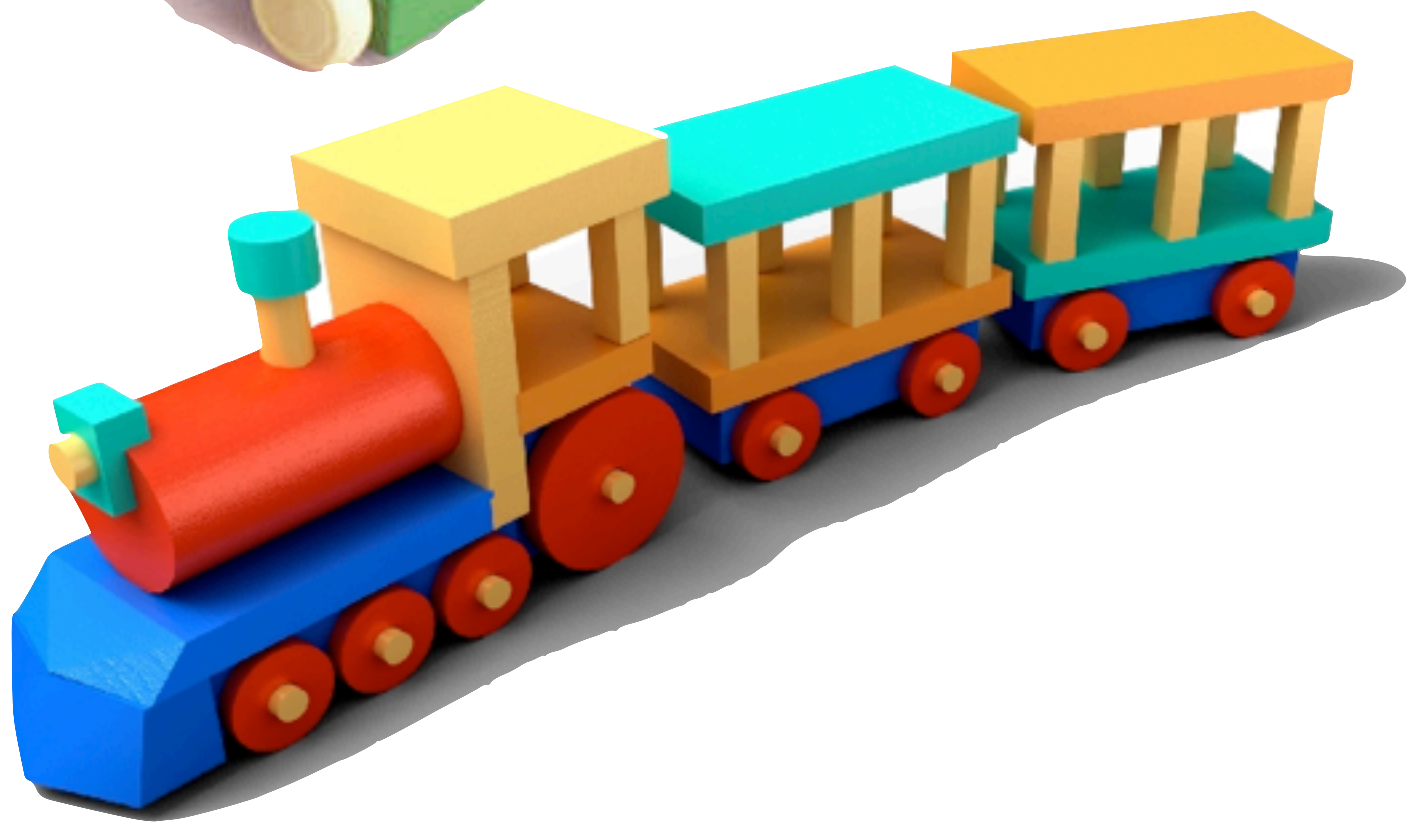
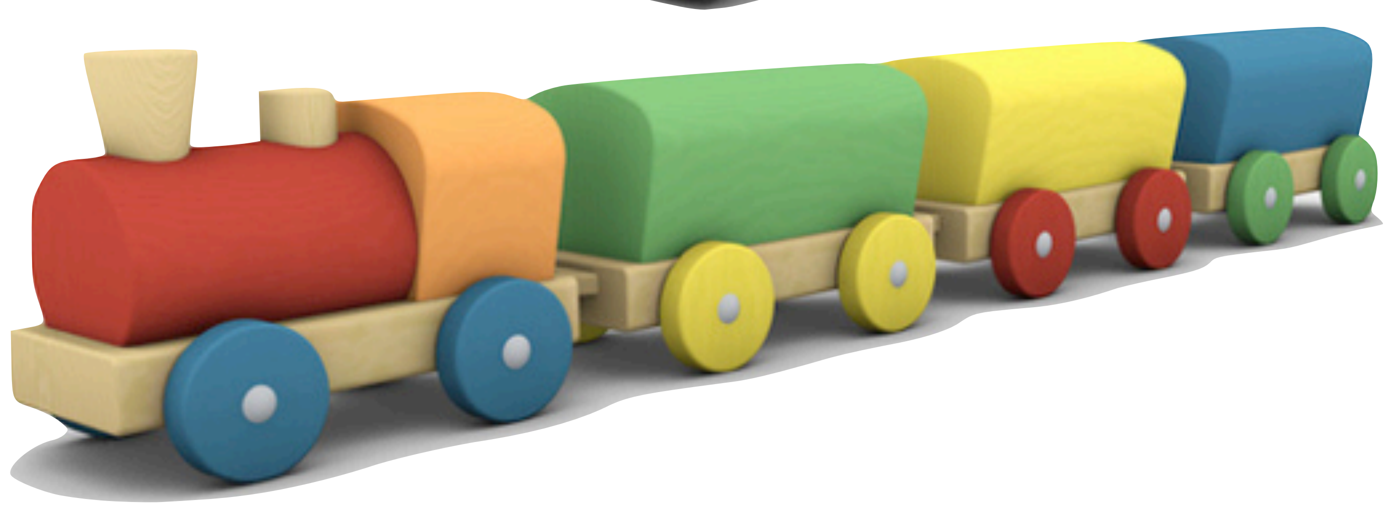
$$\sum 3^{Ns[j] + \left\{ \begin{matrix} N \\ n \end{matrix} \right.}$$

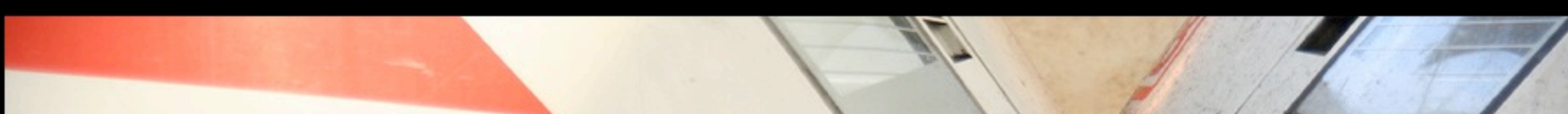
Well-Orderings

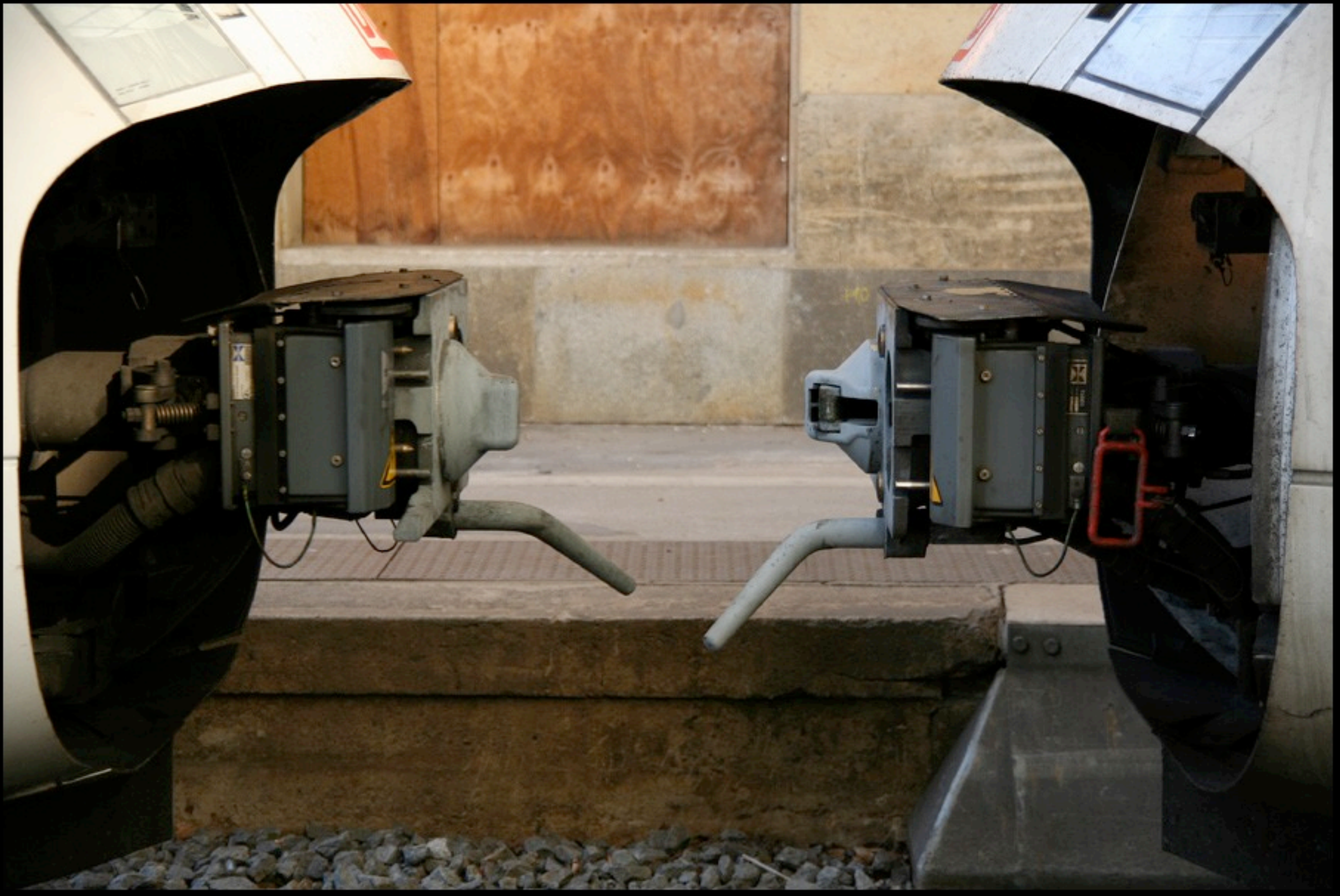
- a b c ...
- a b c ... ∞
- a b c ... 0 1 2 ...
- a0 a1 a2 ... b0 b1 b2 ... c0 c1 c2 ... ...
- 000 001 002 ... 010 011 ... 020 ... 100

Chocolate Bar

- Yumm (click here)





Before & After

- $n \rightarrow \lfloor n/2 \rfloor$, $\lceil n/2 \rceil$ $(n>1)$

Before & After

- $1 \rightarrow$
- $n \rightarrow \lfloor n/2 \rfloor , \lceil n/2 \rceil \quad (n>1)$

Before & After

- $1 \to$
- $n \to 1, n-1 \quad (n>1)$

Before & After

- $1 \rightarrow$
- $n \rightarrow \; i \, , \, n-i \quad (n>1, \, i>0)$

Before & After

- m →
- n → n-1 , n-1   (n>1, i>0)

Proof by Cases

A[x]

A[true], A[false]

Before & After

- $1 \rightarrow$
- $n \rightarrow i\ ,\ j \quad (0<i,j<n)$

Before & After

- $1 \rightarrow$
- $n \rightarrow i, j, k \quad (0<i,j,k<n)$

Before & After

- 1 →
- n → n1 , n2 , ..., nk   (0<ni<n)

Konig's Lemma

- A tree is finite (has finitely many edges)

if and only if

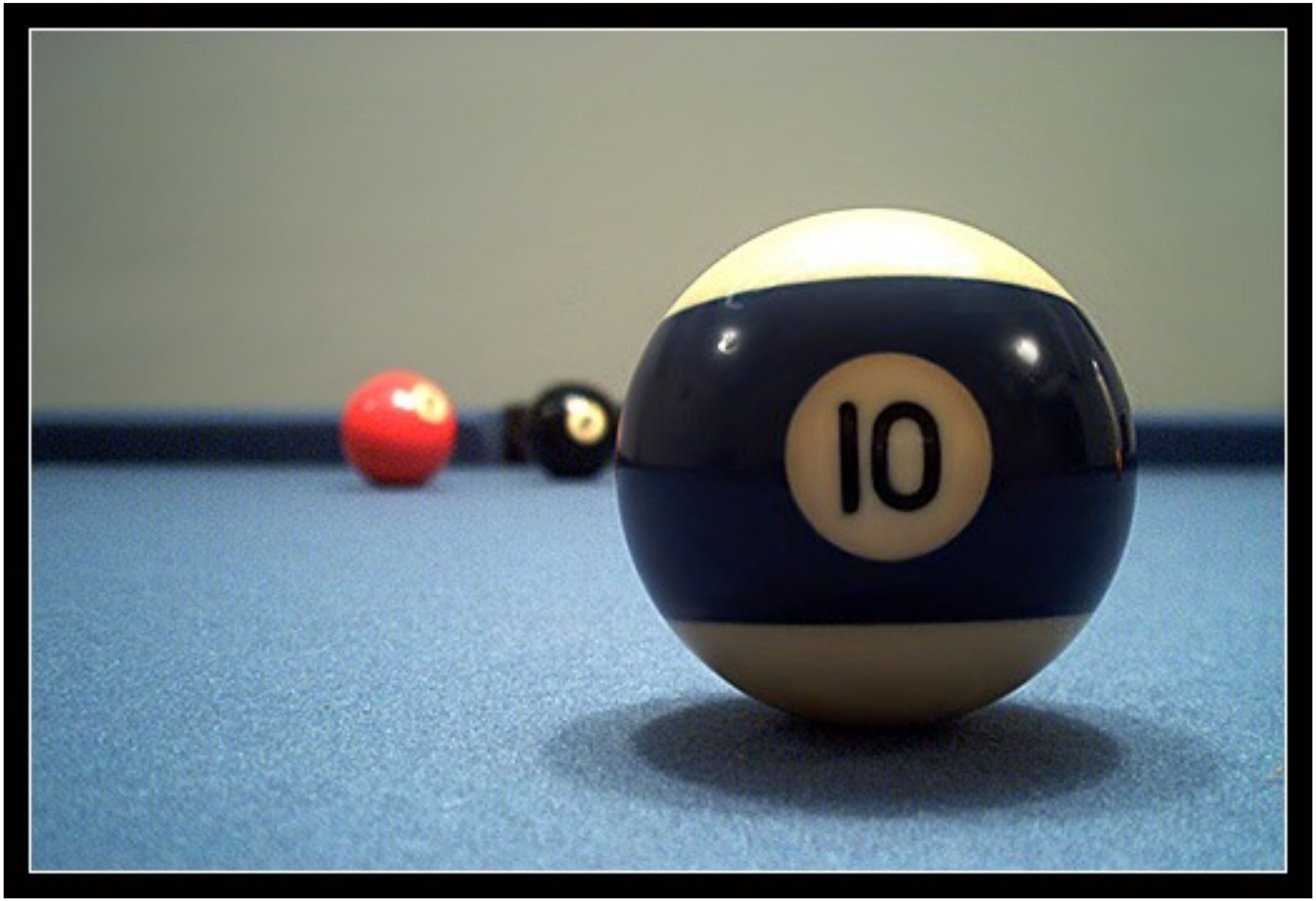
  - all nodes have finite degree

  and

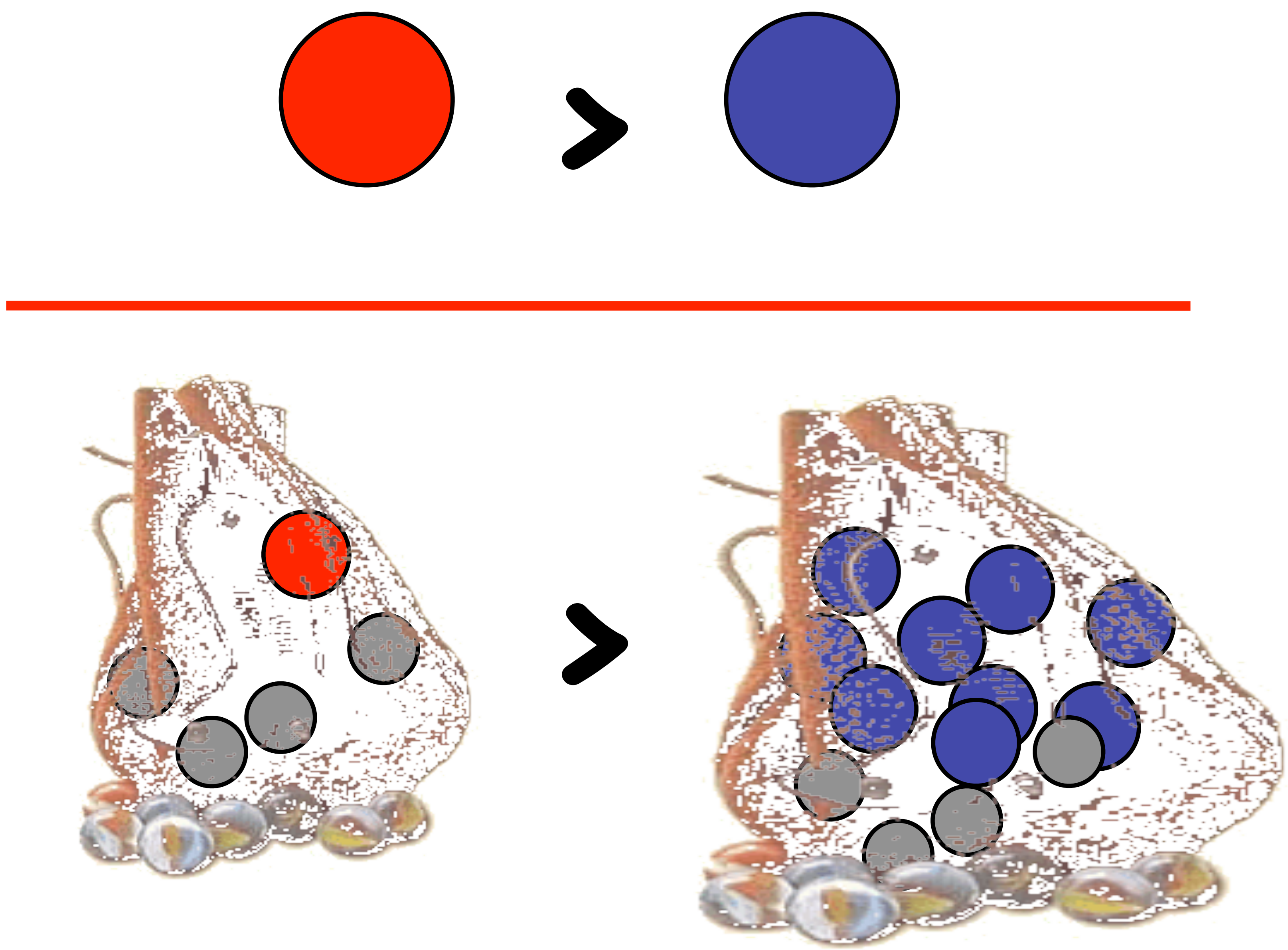
  - all branches (simple paths) have

Billiards

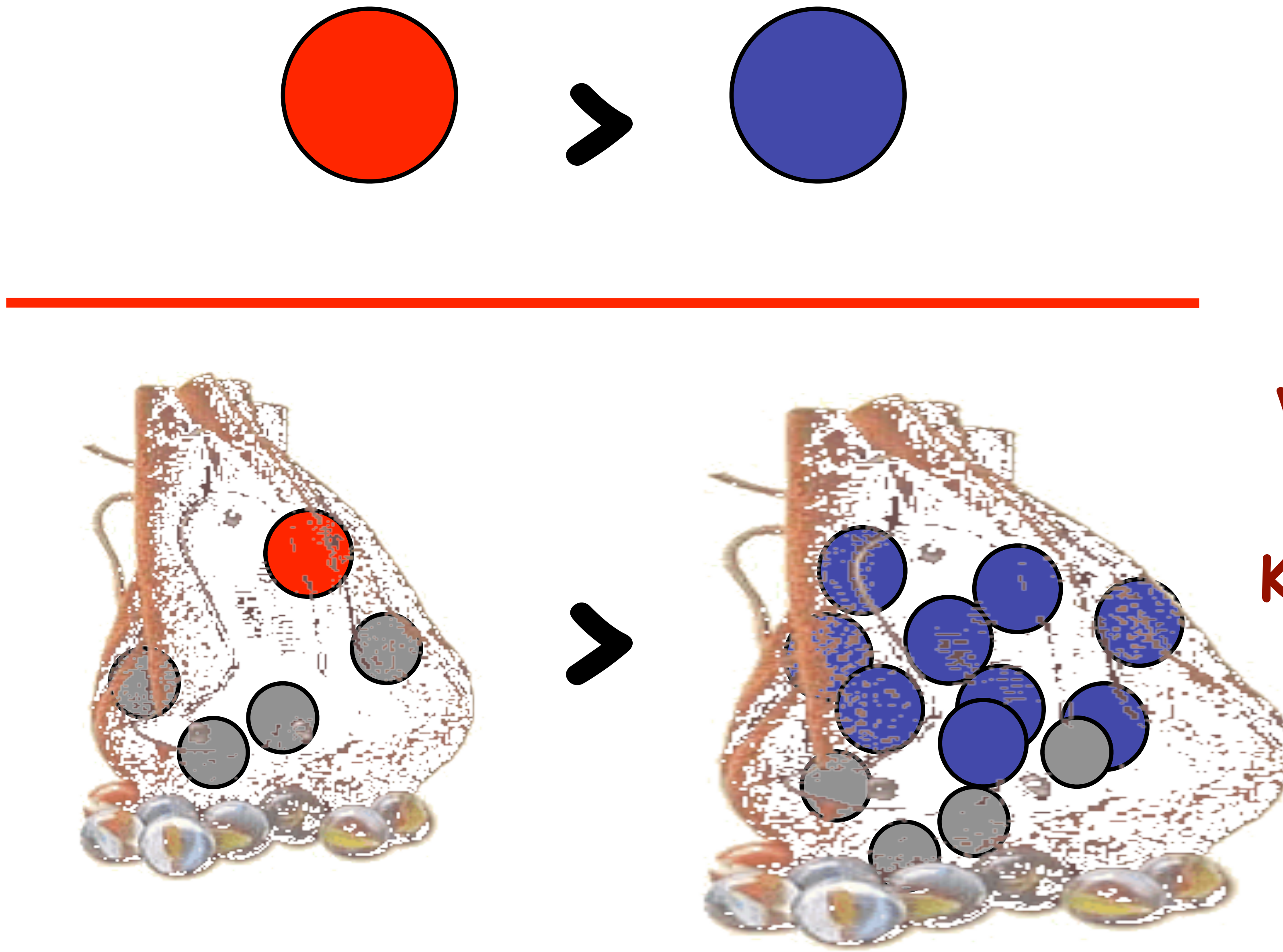
Smullyan's Billiards



Multiset (Bag)



Multiset (Bag)



Well-founded
by
König's Lemma

Harder A(m,n)

```
t := 1
s[t] := m
loop
   c := c + 1
   m := s[t]
   t := t - 1
   if m = 0
   then
      n := n + 1
   elseif n = 0
   then
      t := t + 1
      s[t] := m - 1
      n := 1
   else
      t := t + 2
      s[t-1] := m - 1
      s[t] := m
      n := n - 1
   until t = 0
```

Bag of pairs

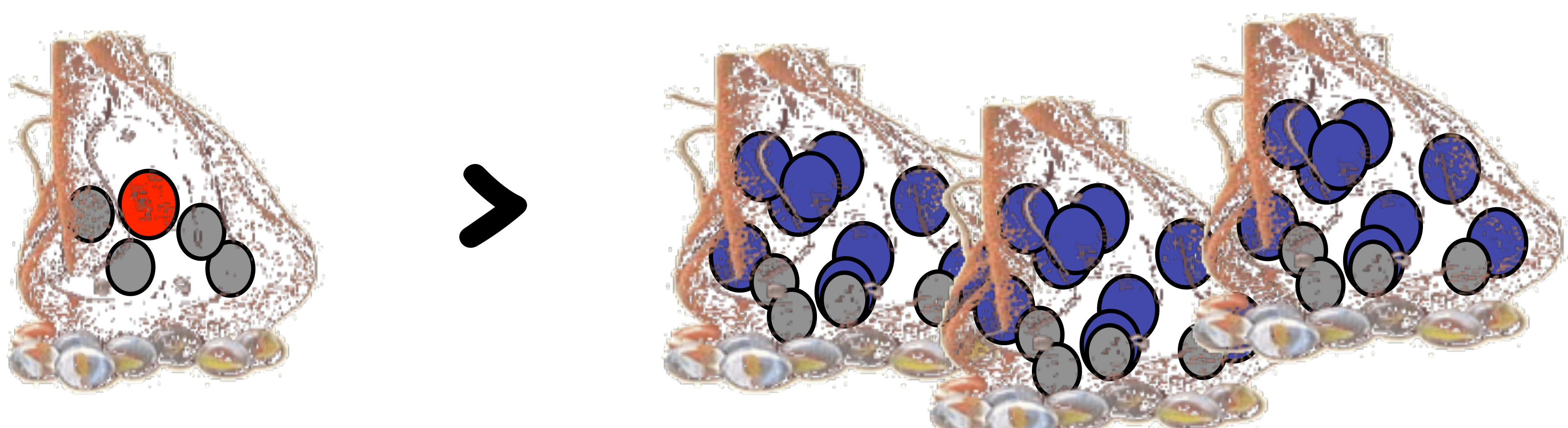
(s[i],∞)   i<t

(s[t],n)

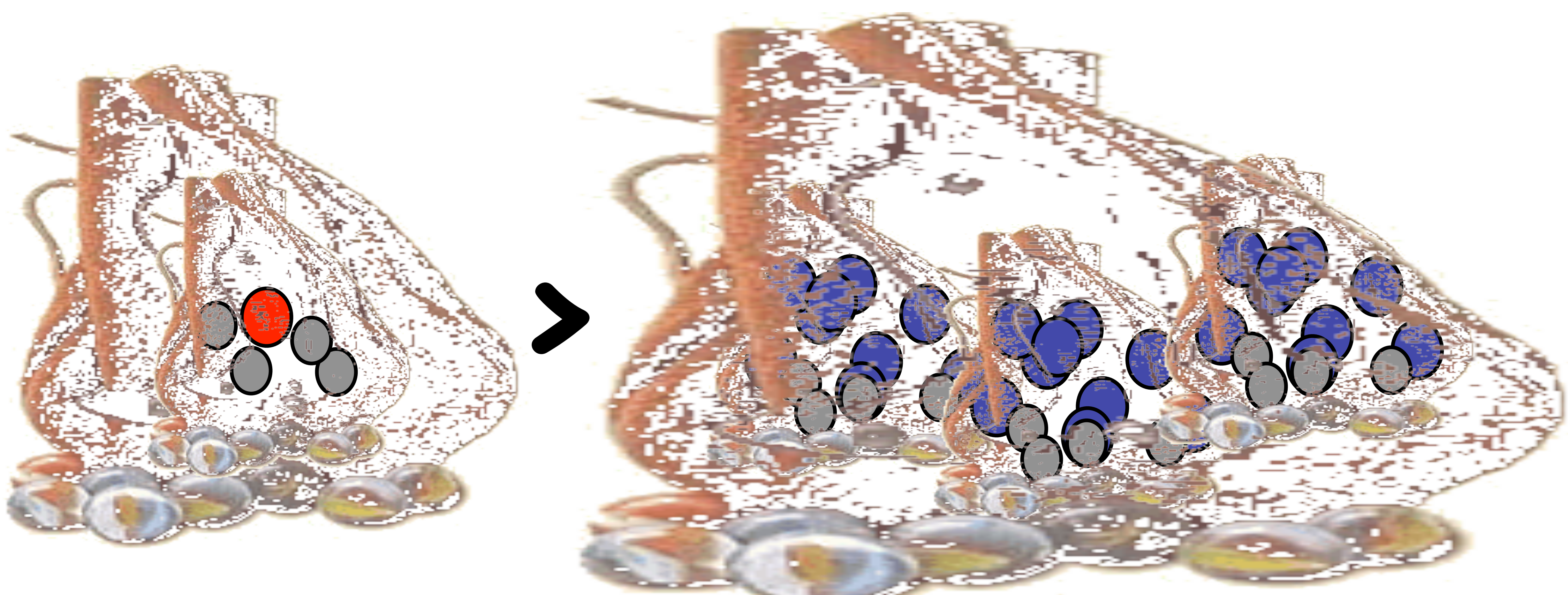
Nested Matryoshka

Nested Bags

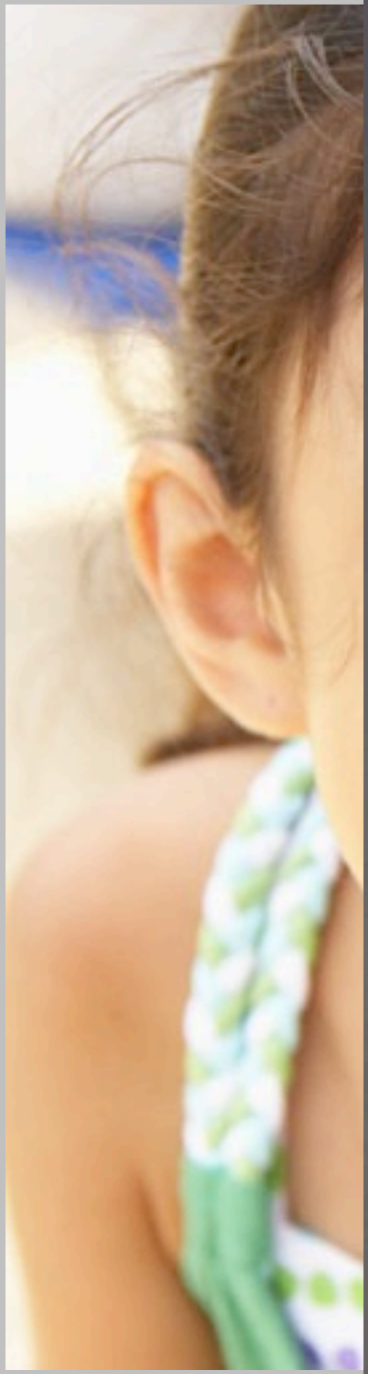
Nested Ordering

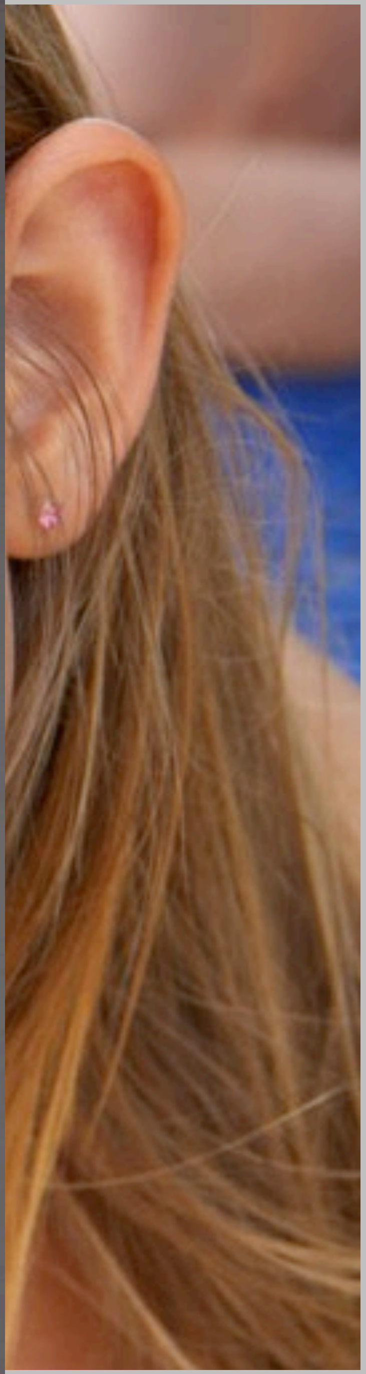


Nested Ordering



Hydra



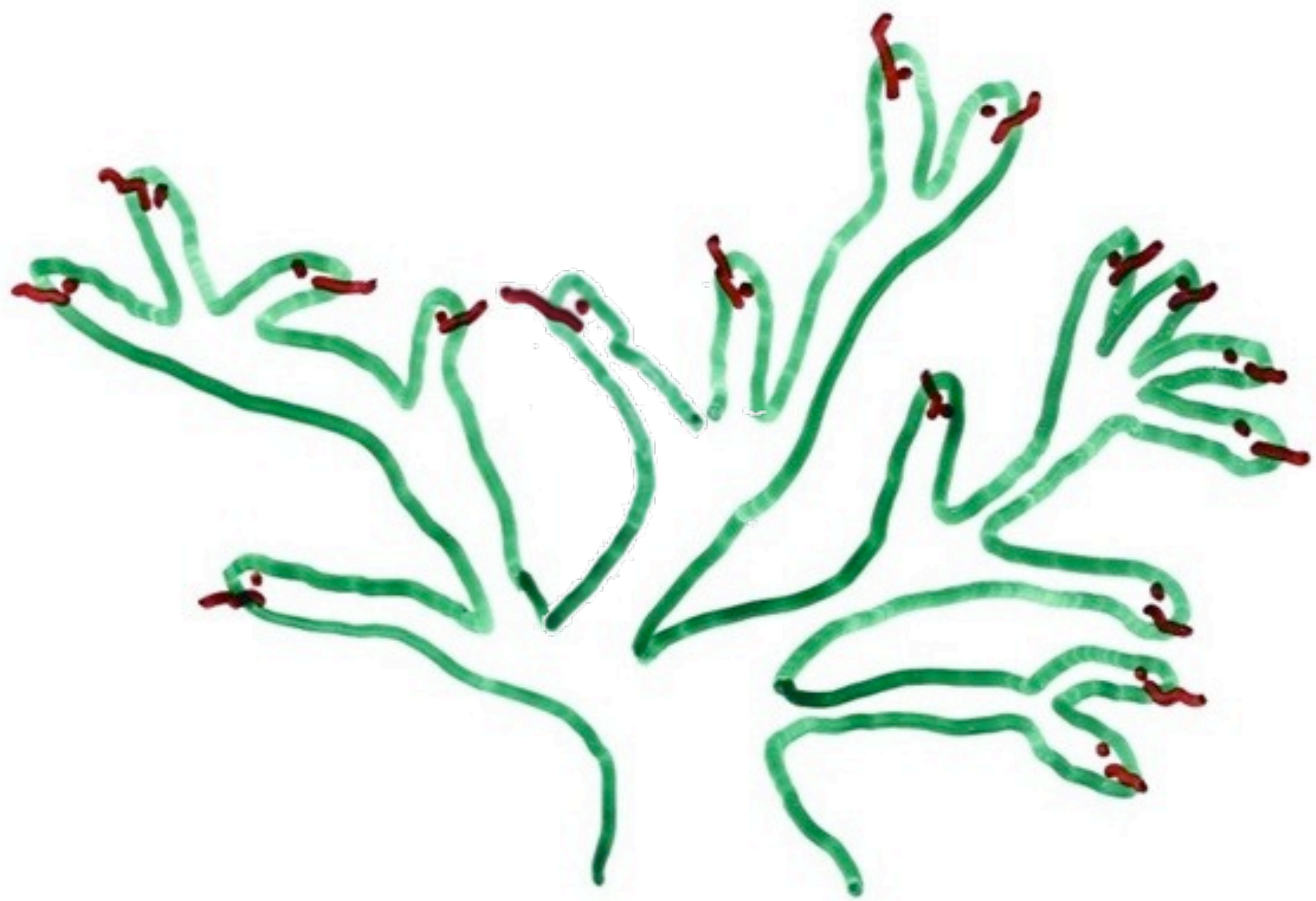


Hercules' Second Labor

Each time Hercules bashed one of Hydra's heads, Iolaus held a torch to the headless neck.

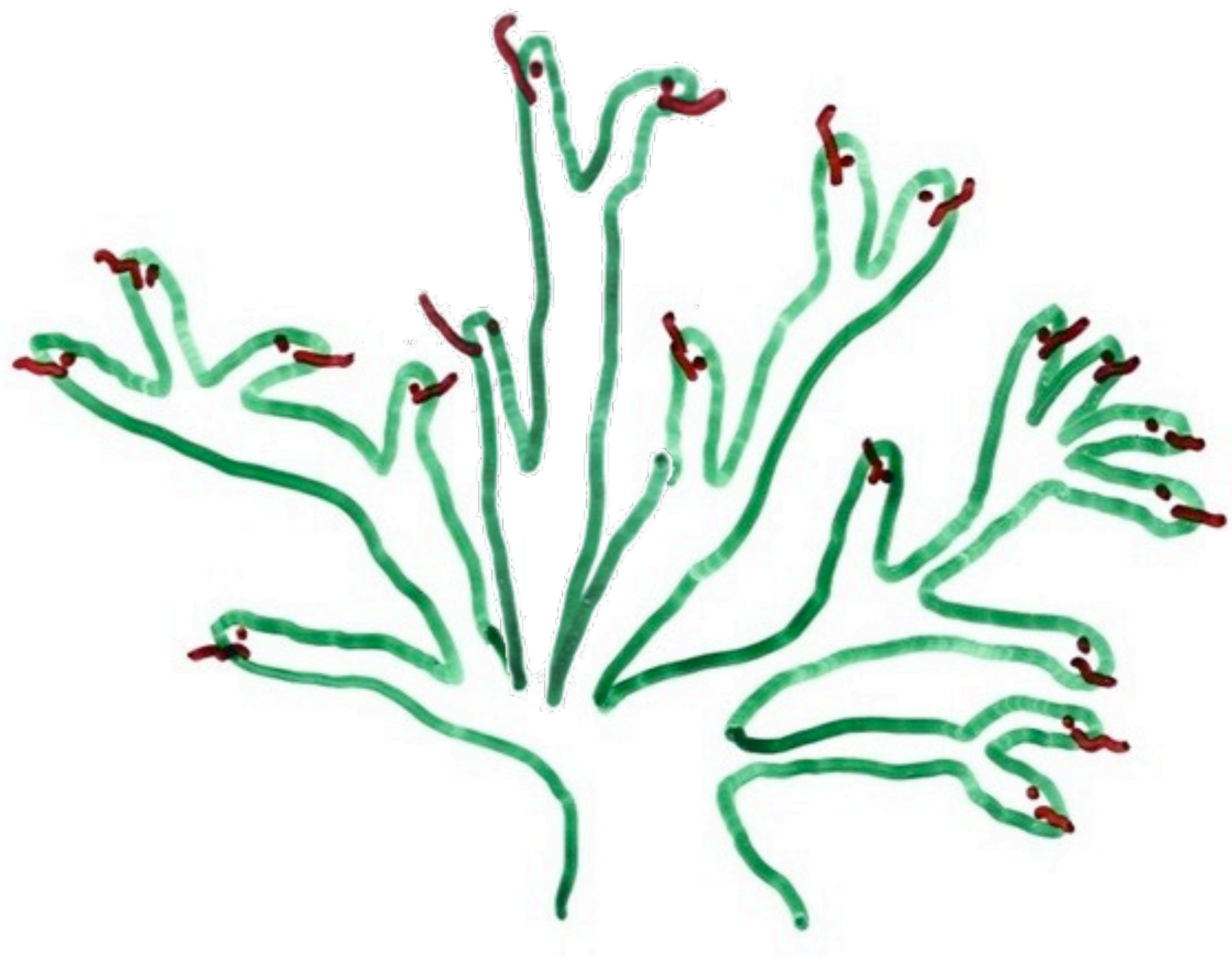
After destroying eight mortal heads, Hercules chopped off the ninth, immortal head, which he buried at the side of the road from Lerna to Elaeus, and covered with a heavy rock.

Hydra vs. Hercules

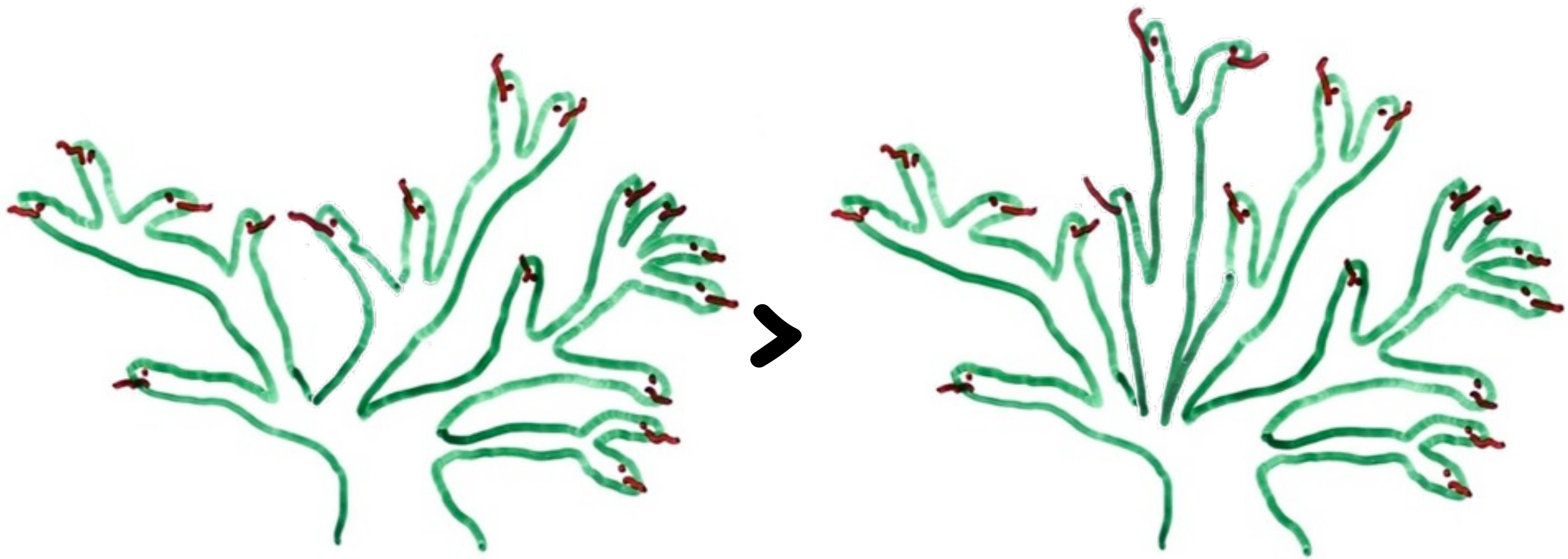


Hydra vs. Hercules

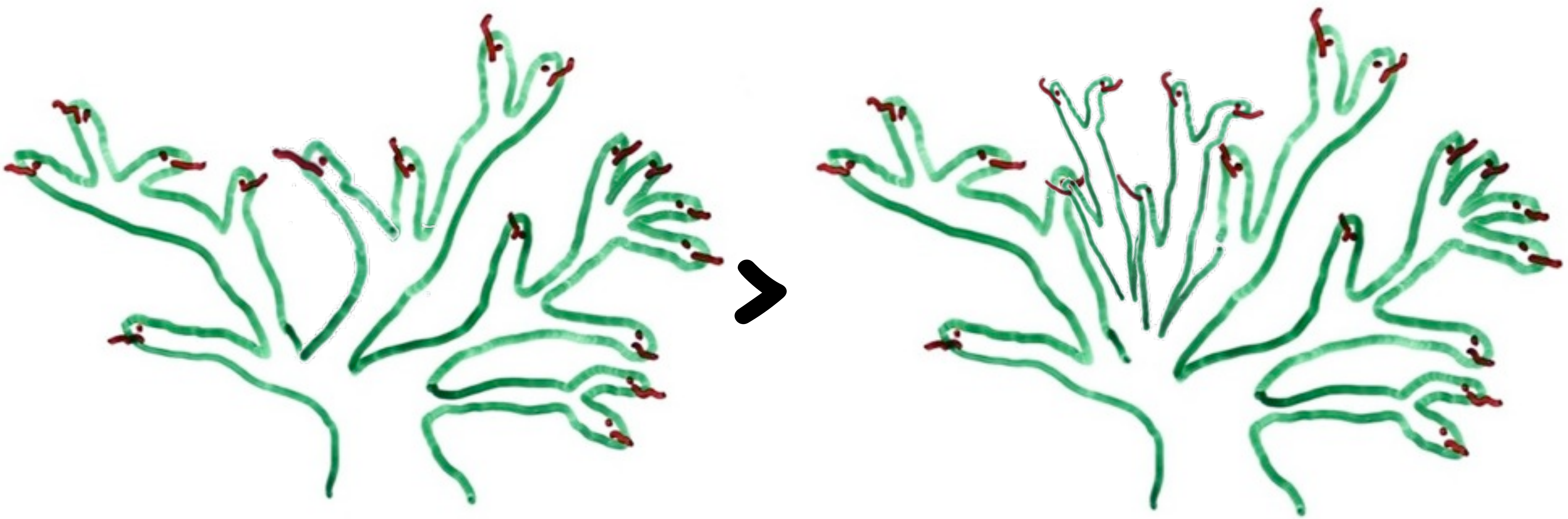
Hydra vs. Hercules



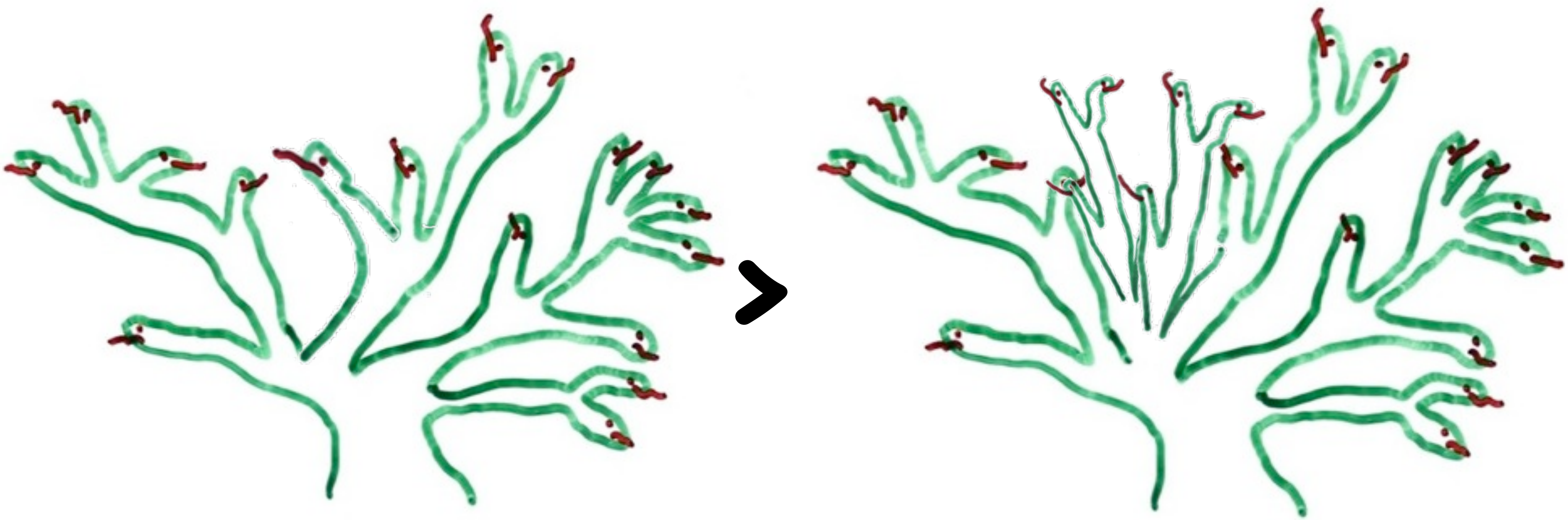
Hercules > Hydra



Hercules > Hydra



Hercules > Hydra



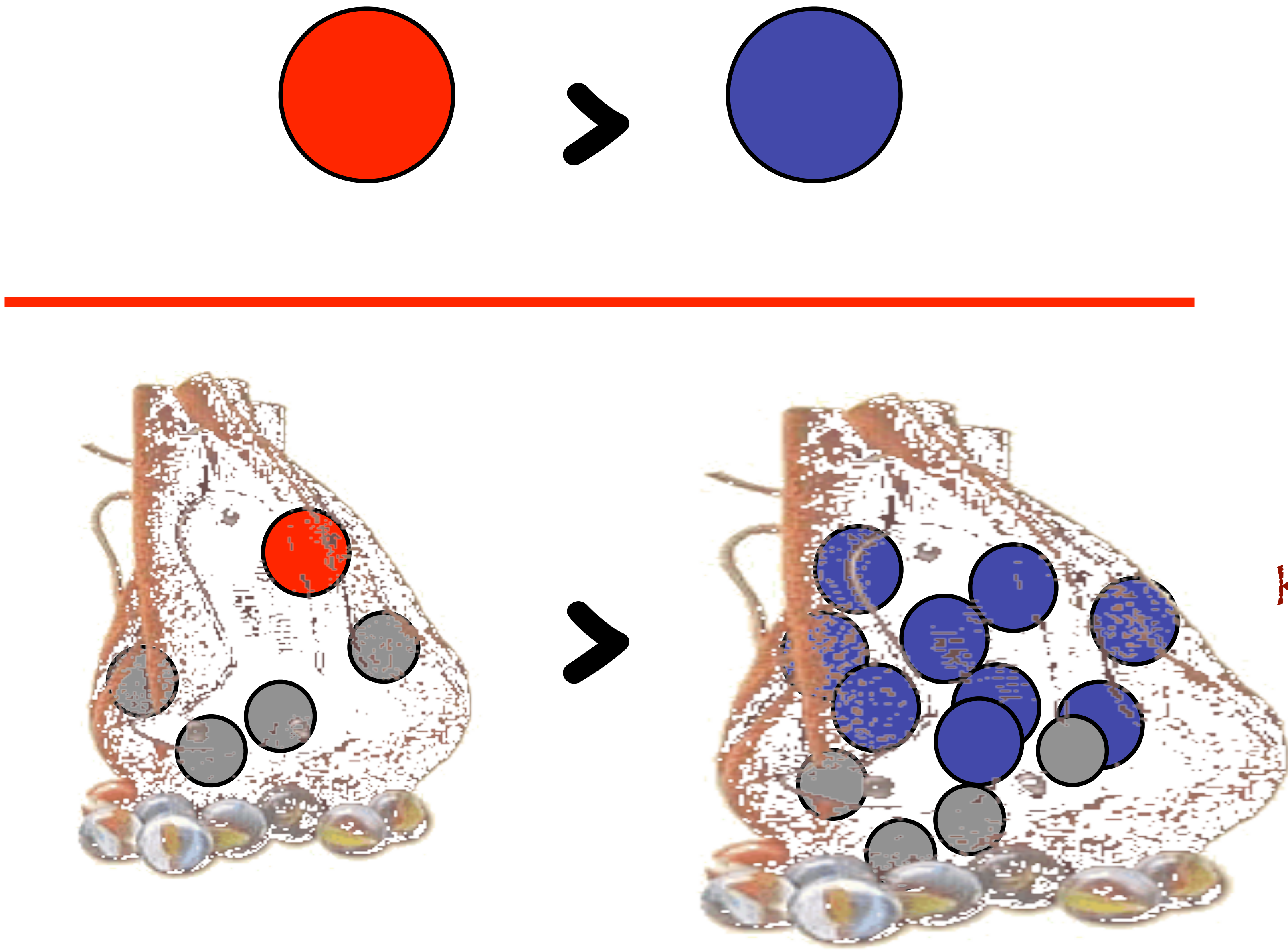
{{o{{ooo}o} {oo{oo}} {o{oooo}o} {oo}} > {{o{{ooo}o} {o{oo}} {o{oo}} {o{oo}} {o{oooo}o} {oo}}

Hercules Defeats Hydra

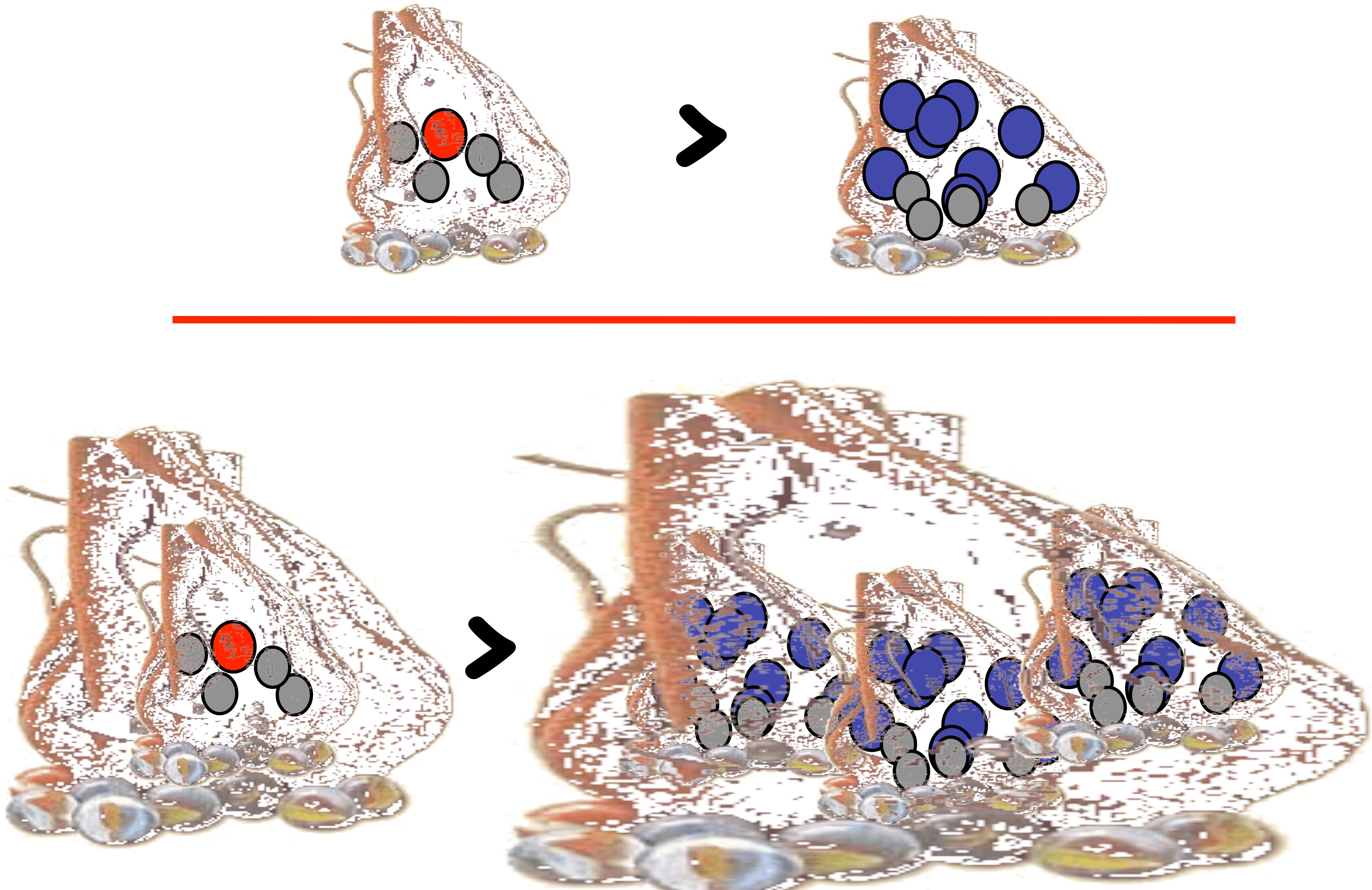
- Cannot be proved in Peano Arithmetic [Paris & Kirby]
- Requires induction up to ε_0
  - Natural numbers do not suffice
  - Sophisticated variants require more

Termination

3. Bigger & Bigger



Well-founded
by
König's Lemma



Well-Founded

$$\frac{\forall x.\ [\forall y{<}x.\ P(y)] \Rightarrow P(x)}{\forall x.\ P(x)}$$

Ordinals

$$0 < 1 < 2 < \cdots$$

$$< \omega < \omega+1 < \omega+2 < \cdots$$

$$< \omega 2 < \omega 2+1 < \cdots < \omega 3 < \cdots < \omega 4 < \cdots$$

$$< \omega^2 < \omega^2+1 < \cdots < \omega^2+\omega < \omega^2+\omega+1 < \cdots$$

$$< \omega^3 < \omega^3+1 < \cdots < \omega^4 < \cdots < \omega^5 < \cdots$$

Bags of Bags

- An empty bag is worth 0
- A bag containing bags worth α_i, is worth $\Sigma\omega^{\alpha_i}$

Goodstein Step

- Increment base & decrement number
  - $4 : 2^2$
  - $26 : 3^3 -1 = 27-1 = 26 = 3^2 + 3^2 + 3 + 3 + 2$
  - $41 : 4^2 + 4^2 + 4 + 4 + 1$

Goodstein 4

4, 26, 41, 60, 83, 109, 139, 173, 211, 253, 299, 348, 401, 458, 519, 584, 653, 726, 803, 884, 969, 1058, 1151, 1222, 1295, 1370, 1447, 1526, 1607, 1690, 1775, 1862, 1951, 2042, 2135, 2230, 2327, 2426, 2527, 2630, 2735, 2842, 2951, 3062, 3175, 3290, 3407,..., 11115, 11327,..., 40492,40895,..., 154349,

1621295857800314 89, 1621295865853 37855,

$3 \cdot 2^{402653210} - 1$, ..., 2, 1, 0

Goodstein 19

- 19, 7625597484990, ~1.3x10^{154}, ...
-

Goodstein Step

- Increment base & decrement number
  - $4 : 2^2$
  - $26 : 3^3 - 1 = 27 - 1 = 26 = 3^2 + 3^2 + 3 + 3 + 2$
  - $41 : 4^2 + 4^2 + 4 + 4 + 1$

Goodstein Step

- Base is a bag (and the whole thing is in a bag)
  - 2^2 is {{{}}}
  - $3^2 + 3^2 + 3 + 3 + 2$ is {{2},{2},{},{},2}
  - $4^2 + 4^2 + 4 + 4 + 1$ is {{2},{2},{},{},1}

Goodstein 16

$g_{16}(2) = 16 = 2^{2^\wedge 2}$

$g_{16}(3) = 3^{3^\wedge 3}-1 = 2\cdot 3^{2\cdot 3^\wedge 2+2\cdot 3+2} + 2\cdot 3^{2\cdot 3^\wedge 2+2\cdot 3+1}$
$+ 2\cdot 3^{2\cdot 3^\wedge 2+2\cdot 3} + 2\cdot 3^{2\cdot 3^\wedge 2+1\cdot 3+2} + 2\cdot 3^{2\cdot 3^\wedge 2+1\cdot 3+1}$
$+ 2\cdot 3^{2\cdot 3^\wedge 2+1\cdot 3} + 2\cdot 3^{2\cdot 3^\wedge 2+2} + 2\cdot 3^{2\cdot 3^\wedge 2+1} + 2\cdot 3^{2\cdot 3^\wedge 2}$
$+ 2\cdot 3^{3^\wedge 2+2\cdot 3+2} + 2\cdot 3^{3^\wedge 2+2\cdot 3+1} + 2\cdot 3^{3^\wedge 2+2\cdot 3} + 2\cdot 3^{3^\wedge 2+1\cdot 3+2}$
$+ 2\cdot 3^{3^\wedge 2+1\cdot 3+1} + 2\cdot 3^{3^\wedge 2+1\cdot 3} + 2\cdot 3^{3^\wedge 2+2} + 2\cdot 3^{3^\wedge 2+1} + 2\cdot 3^{3^\wedge 2}$
$+ 2\cdot 3^{2\cdot 3+2} + 2\cdot 3^{2\cdot 3+1} + 2\cdot 3^{2\cdot 3} + 2\cdot 3^{1\cdot 3+2} + 2\cdot 3^{1\cdot 3+1} + 2\cdot 3^{1\cdot 3}$
$+ 2\cdot 3^{2} + 2\cdot 3^{1} + 2 = 7625597484986$

$a(4) = 50973998591214355139406377.$

Goodstein 16

$g_{16}(2) = 16 = 2^{2^\wedge 2}$

$g_{16}(3) = 3^{[1000]}-1 = 2\cdot3^{[222]} + 2\cdot3^{[221]} + 2\cdot3^{[220]} +$
$2\cdot3^{[212]} + 2\cdot3^{[211]} + 2\cdot3^{[210]} + 2\cdot3^{[202]} + 2\cdot3^{[201]} +$
$2\cdot3^{[200]} + 2\cdot3^{[122]} + 2\cdot3^{[121]} + 2\cdot3^{[120]} + 2\cdot3^{[112]} + 2\cdot3^{[111]} +$
$2\cdot3^{[110]} + 2\cdot3^{[102]} + 2\cdot3^{[101]} + 2\cdot3^{[100]} + 2\cdot3^{[022]} + 2\cdot3^{[021]}$
$+ 2\cdot3^{[020]} + 2\cdot3^{[012]} + 2\cdot3^{[011]} + 2\cdot3^{[010]} + 2\cdot3^{[002]} +$
$2\cdot3^{[001]} + 2\cdot3^{[000]} = 7625597484986$

where [abc] is the base 3 representation

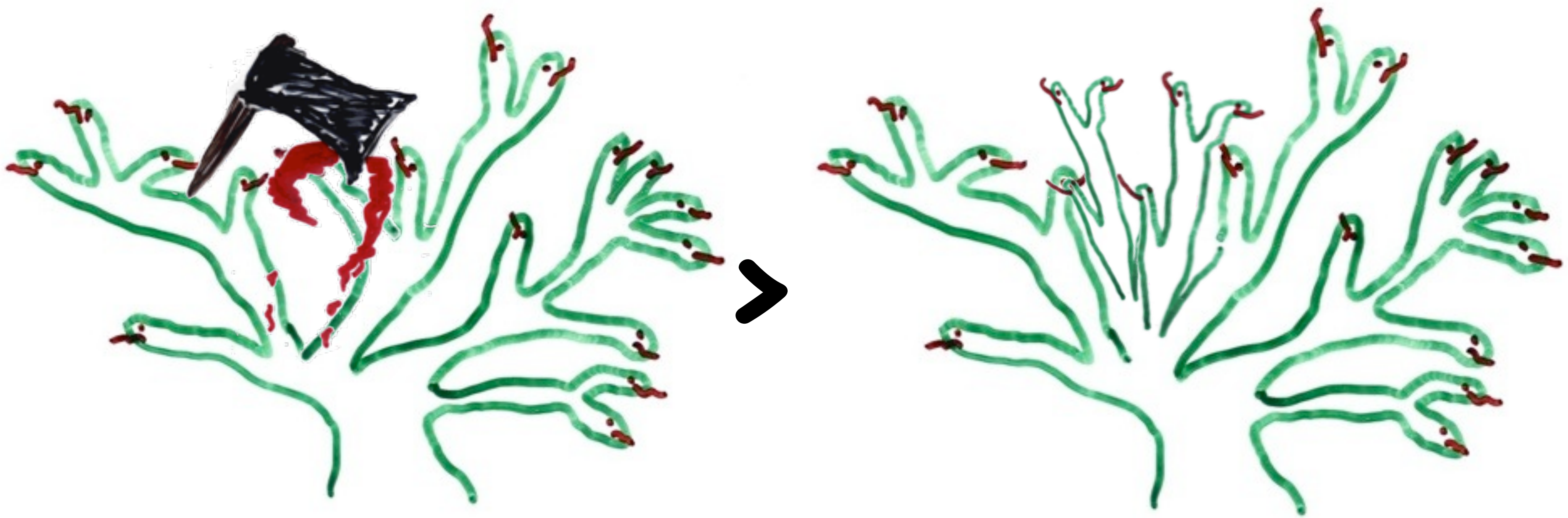
Goodstein 16

$g_{16}(2) = \omega^{\omega\hat{}\omega}$

$g_{16}(3) = 2\cdot\omega^{2\cdot\omega\hat{}2 + 2\cdot\omega + 2} + 2\cdot\omega^{2\cdot\omega\hat{}2 + 2\cdot\omega + 1} + 2\cdot\omega^{2\cdot\omega\hat{}2 + 2\cdot\omega}$
$+ 2\cdot\omega^{2\cdot\omega\hat{}2 + 1\cdot\omega + 2} + 2\cdot\omega^{2\cdot\omega\hat{}2 + 1\cdot\omega + 1} + 2\cdot\omega^{2\cdot\omega\hat{}2 + 1\cdot\omega} +$
$2\cdot\omega^{2\cdot\omega\hat{}2 + 2} + 2\cdot\omega^{2\cdot\omega\hat{}2 + 1} + 2\cdot\omega^{2\cdot\omega\hat{}2} + 2\cdot\omega^{\omega\hat{}2 + 2\cdot\omega + 2} +$
$2\cdot\omega^{\omega\hat{}2 + 2\cdot\omega + 1} + 2\cdot\omega^{\omega\hat{}2 + 2\cdot\omega} + 2\cdot\omega^{\omega\hat{}2 + 1\cdot\omega + 2} +$
$2\cdot\omega^{\omega\hat{}2 + 1\cdot\omega + 1} + 2\cdot\omega^{\omega\hat{}2 + 1\cdot\omega} + 2\cdot\omega^{\omega\hat{}2 + 2} + 2\cdot\omega^{\omega\hat{}2 + 1} +$
$2\cdot\omega^{\omega\hat{}2} + 2\cdot\omega^{2\cdot\omega + 2} + 2\cdot\omega^{2\cdot\omega + 1} + 2\cdot\omega^{2\cdot\omega} + 2\cdot\omega^{1\cdot\omega + 2} +$
$2\cdot\omega^{1\cdot\omega + 1} + 2\cdot\omega^{1\cdot\omega} + 2\cdot\omega^{2} + 2\cdot\omega^{1} + 2$

Goodstein

- Cannot be proved terminating in Peano Arithmetic



Hercules Defeats

- Cannot be proved in Peano Arithmetic [Paris & Kirby]
- Requires induction up to ε_0
- Natural numbers do not suffice
- Sophisticated variants require more powerful systems [Friedman]

Hydra Step

- Every head is an empty bag
- Every node (including the ground) is a bag of its children
- Each step replaces some internal bag with some number of smaller bags

Hydra Step

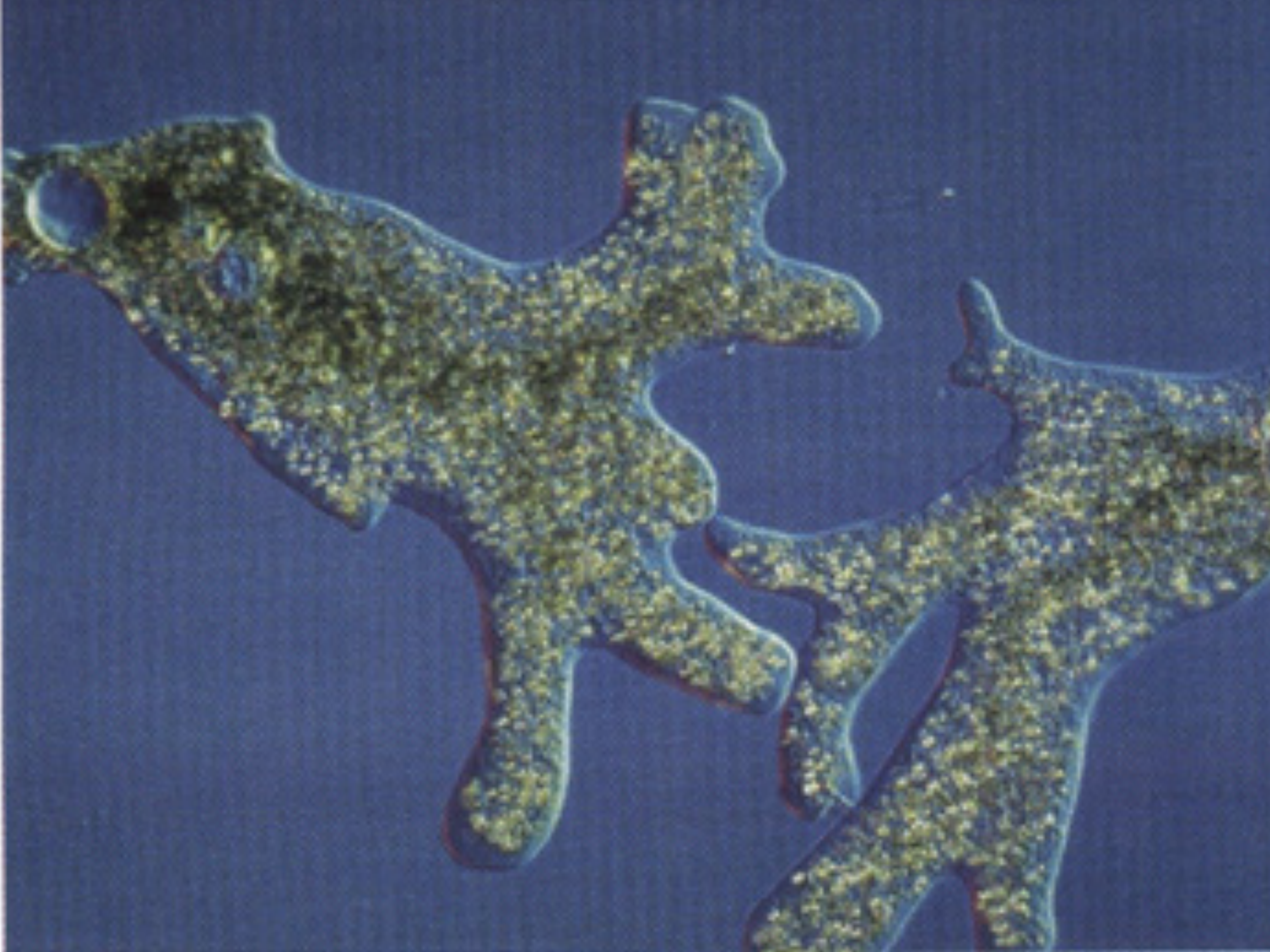
- Heads are worth 0
- Every node (including the ground), with children worth α_i, is worth $\Sigma\omega^{\alpha_i}$
- The kth step replaces a term $\omega^{\alpha+1}$ with $\omega^{\alpha}k$
- But if a head sprouting from the ground is cut, the total decreases by 1

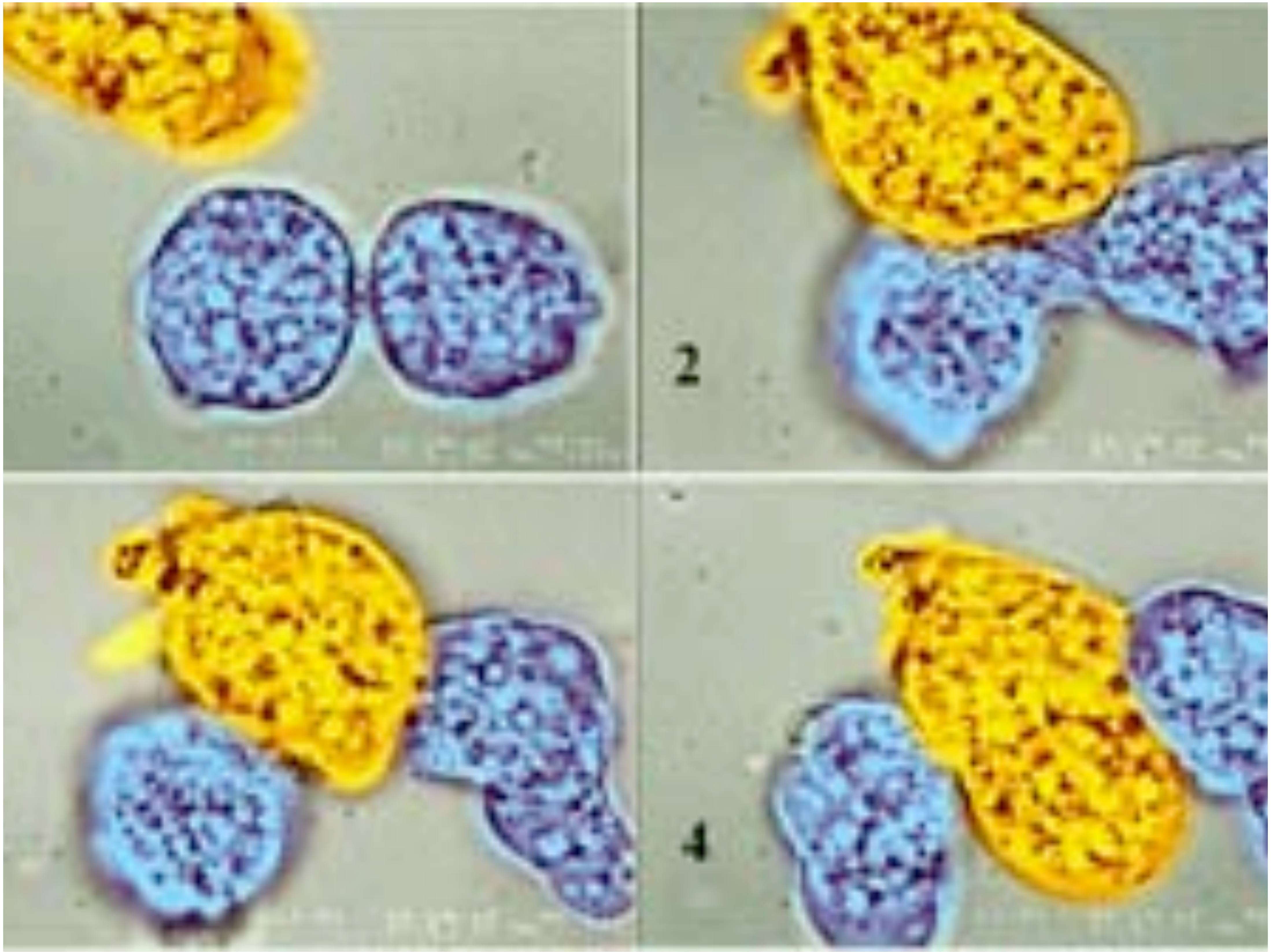
Hercules Defeats Hydra

- Cannot be proved in Peano Arithmetic [Paris & Kirby]
- Requires induction up to ε_0
  - Natural numbers do not suffice
  - Sophisticated variants require more

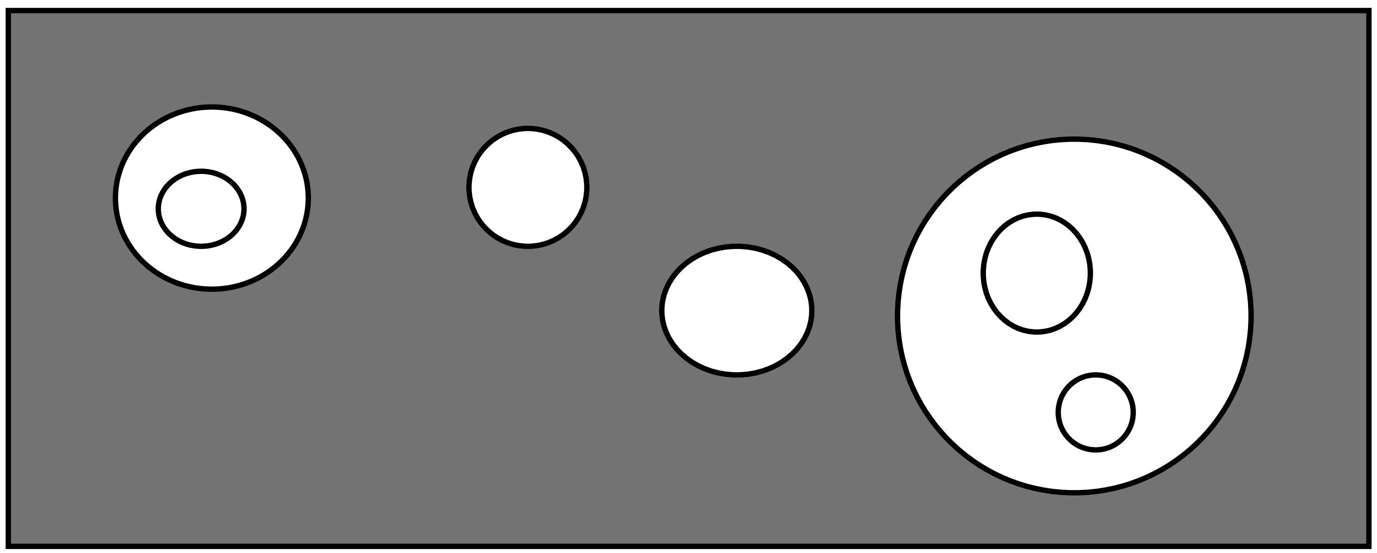
Termination

4. Well-Founded Orderings

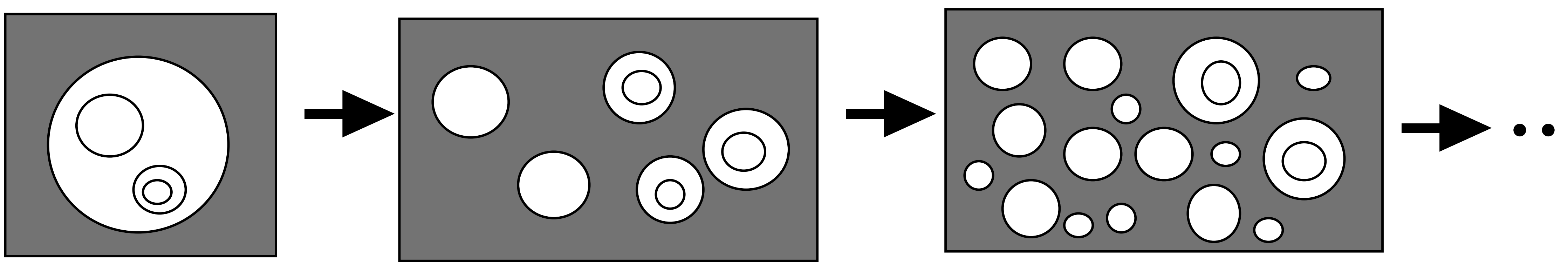


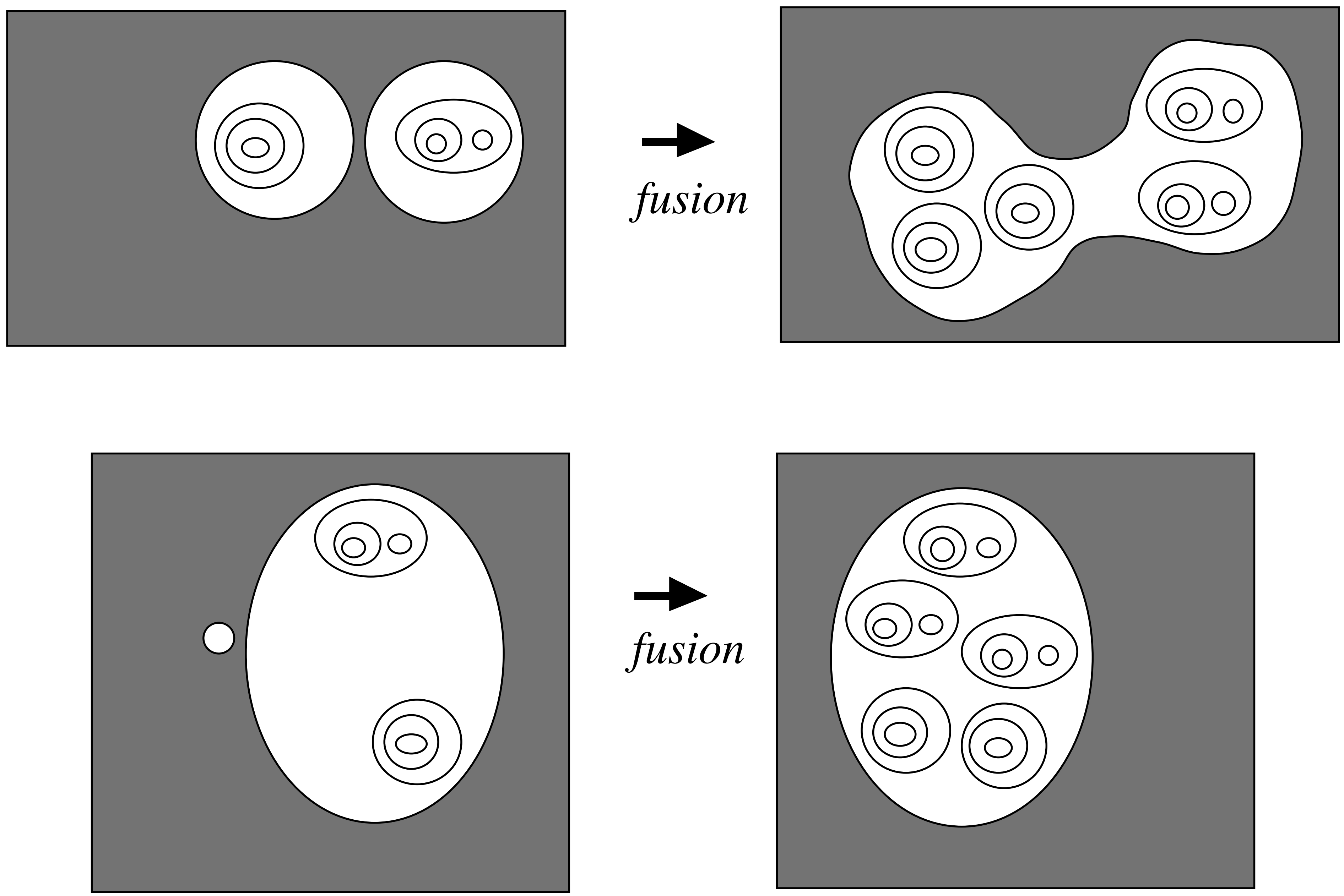

2
4

Amoebae



Fission




fusion
fusion

Colony Dies Out

- depth(o) = 0
- depth((a1 ... an)) = 1+max{depth{ai}}
- { (depth(a),|a|) : subcolony a }
- outer fission: depth decreases
- fusion: size decreases

Colony Dies Out

- $d(a)$ = depth(a)
- $\#_d(a)$ = number in a of depth d
- $\{ (d(a), \#_{d(a)}(a), \#_{d(a)-1}(a), \ldots) : \text{colony } a \}$
- fission: depth decreases
- fusion: size decreases

Big Picture

- Programs are state-transition systems
- Choose a well-founded order on states
- Show that transitions are decreases

Real Picture

- Programs are state-transition systems
- Choose a function for “ranking” states
- Choose a well-founded order on ranks
- Show that transitions always decrease rank

Imaginary Picture

- Programs are state-transition systems
- Choose a function for “ranking” states
- Choose a well-founded order on ranks
- Show that transitions eventually decrease rank

Nested Loops

```
r := 1
u := 1
loop    v := u
        until r≥n
        s := 1
        loop  u := u+v
              s := s+1
              while s≤r
              repeat
```

$\omega^2 (n - r) + \omega(r - s) + k$

Per Iteration

```
r := 1
u := 1
loop    v := u
        until r≥n
        s := 1
        loop    u := u+v
                s := s+1
                while s≤r
                repeat
```

$\omega(n-r)+r+1-s$

Lexicographic

```
r := 1
u := 1
loop    v := u
        until r≥n
        s := 1
        loop   u := u+v
               s := s+1
               while s≤r
               repeat
```

(n-r,r+1-s)

Invariants

```
r := 1
u := 1
loop    v := u                      1≤r≤n
        until r≥n
        s := 1
        loop   u := u+v             1≤s≤r+1
               s := s+1
               while s≤r
               repeat
```

Well-Founded

- No infinite descending sequences

  $x1 > x2 > x3 > \ldots$

Well-Founded

$>$ is a wfo of X

$$\frac{\forall x \in X.\ [\forall y < x.\ P(y)] \Rightarrow P(x)}{\forall x \in X.\ P(x)}$$

Why?

David Gries

- Under the reasonable assumption that nondeterminism is bounded, the two methods are equivalent.... In this situation, we prefer using strong termination.

$n := 0$
while $x > 0$ **do**
    $n := n + 1$
    $y := 0$; **while** $y^2 + 2y \leq x$ **do** $y := y + 1$
    if $x = y^2$
    then $x := y - 1$
    else $s := 0$
        $r := 0$; **while** $r^2 + 2r \leq x - y^2$ **do** $r := r + 1$
        while $x > y^2 + r^2$ **do**
            $y := 0$; **while** $y^2 + 2y \leq x$ **do** $y := y + 1$
            $s := s + (s + y^2 + y - x)^2$
            $x := x - y^2$
            $r := 0$; **while** $r^2 + 2r \leq x - y^2$ **do** $r := r + 1$
        for $i := 1$ **to** n **do** $x := r^2 + r - 1$
        while $s > 0$ **do**
            $r := 0$; **while** $r^2 + 2r \leq s$ **do** $r := r + 1$
            $x := x + (x + r^2 + r - s)^2$
            $s := s - r^2$

Contra-Gries

- To prove terminating with a natural (strong) ranking function requires ε_0-induction

All-Purpose Ranks

$0 < 1 < 2 < \cdots$

$< \omega < \omega+1 < \omega+2 < \cdots$

$< \omega 2 < \omega 2+1 < \cdots < \omega 3 < \cdots < \omega 4 < \cdots$

$< \omega^2 < \omega^2+1 < \cdots < \omega^2+\omega < \omega^2+\omega+1 < \cdots$

$< \omega^3 < \omega^3+1 < \cdots < \omega^4 < \cdots < \omega^5 < \cdots$

Ordinals

$0, 1, 2, \ldots,$

$\omega, \omega+1, \omega+2, \ldots,$

$\omega 2, \omega 2+1, \ldots, \omega 3, \ldots,$

$\omega^2, \ldots, \omega^2+\omega 2+3, \ldots, \omega^3, \ldots,$

$\omega^\omega, \ldots, \omega^{\omega^\omega}, \ldots,$

$\varepsilon_0, \varepsilon_0+1, \ldots, \varepsilon_0 2+\omega^\omega+\omega 2+3, \ldots,$

$\varepsilon_1, \ldots, \varepsilon_{\varepsilon_0}, \ldots,$

THE GROWTH

$0, 1, 2, 3, \ldots, \omega, \omega+1, \omega+2, \omega+3, \ldots, \omega+\omega = \omega 2, \omega 2+1, \omega 2+2, \omega 2+3, \ldots, \omega 3, \omega 3+1, \omega 3+2, \omega 3+3,$

$\omega\omega = \omega^2, \omega^2+1, \omega^2+2, \ldots, \omega^2+\omega, \ldots, \omega^2+\omega 2, \ldots, \omega^2+\omega 3, \ldots, \omega^2 2, \ldots, \omega^2 3, \ldots, \omega^3, \ldots, \ldots,$

$\ldots, \omega^\omega\omega^2, \ldots, \omega^\omega\omega^3, \ldots, \omega^\omega\omega^\omega = \omega^{\omega 2}, \ldots, \omega^{\omega 3}, \ldots, \ldots, \omega^{\omega^2}, \ldots, \omega^{\omega^\omega}, \ldots, \omega^{\omega^{\cdot^{\cdot^{\cdot}}}} = \varepsilon_0, \varepsilon_0+1, \ldots,$

$\varepsilon_0+\omega 2, \ldots, \varepsilon_0+\omega^2, \ldots, \varepsilon_0+\omega^\omega, \ldots, \varepsilon_0+\varepsilon_0 = \varepsilon_0 2, \ldots, \varepsilon_0 3, \ldots, \varepsilon_0\omega, \ldots, \varepsilon_0\omega 2, \ldots, \varepsilon_0\omega^2, \ldots,$

$\varepsilon_0\varepsilon_0 = \varepsilon_0^2, \ldots, \varepsilon_0^3, \ldots, \varepsilon_0^\omega, \ldots, \varepsilon_0^{\omega^\omega}, \ldots, \varepsilon_0^{\varepsilon_0}, \ldots, \varepsilon_0^{\varepsilon_0^{\cdot^{\cdot^{\cdot}}}} = \varepsilon_1, \ldots, \varepsilon_2, \ldots, \varepsilon_3, \ldots, \varepsilon_\omega$

$\varepsilon_{\omega^\omega}, \ldots, \varepsilon_{\varepsilon_0}, \ldots, \varepsilon_{\varepsilon_1}, \ldots, \varepsilon_{\varepsilon_\omega}, \ldots, \varepsilon_{\varepsilon_{\omega^\omega}}, \ldots, \varepsilon_{\varepsilon_{\omega^{\omega}}}, \ldots, \varepsilon_{\varepsilon_{\varepsilon_0}}, \ldots, \varepsilon_{\varepsilon_{\varepsilon_{\cdot_{\cdot_{\cdot}}}}} = \eta_0, \ldots, \eta_1, \ldots$

$\eta_{\varepsilon_0}, \ldots, \eta_{\varepsilon_\omega}, \ldots, \eta_{\varepsilon_{\varepsilon_0}}, \ldots, \eta_{\varepsilon_{\varepsilon_{\cdot_{\cdot_{\cdot}}}}} = \eta_{\eta_0}, \ldots, \eta_{\eta_{\eta_{\cdot_{\cdot_{\cdot}}}}} = \zeta_0, \ldots, \ldots\ldots\ldots$

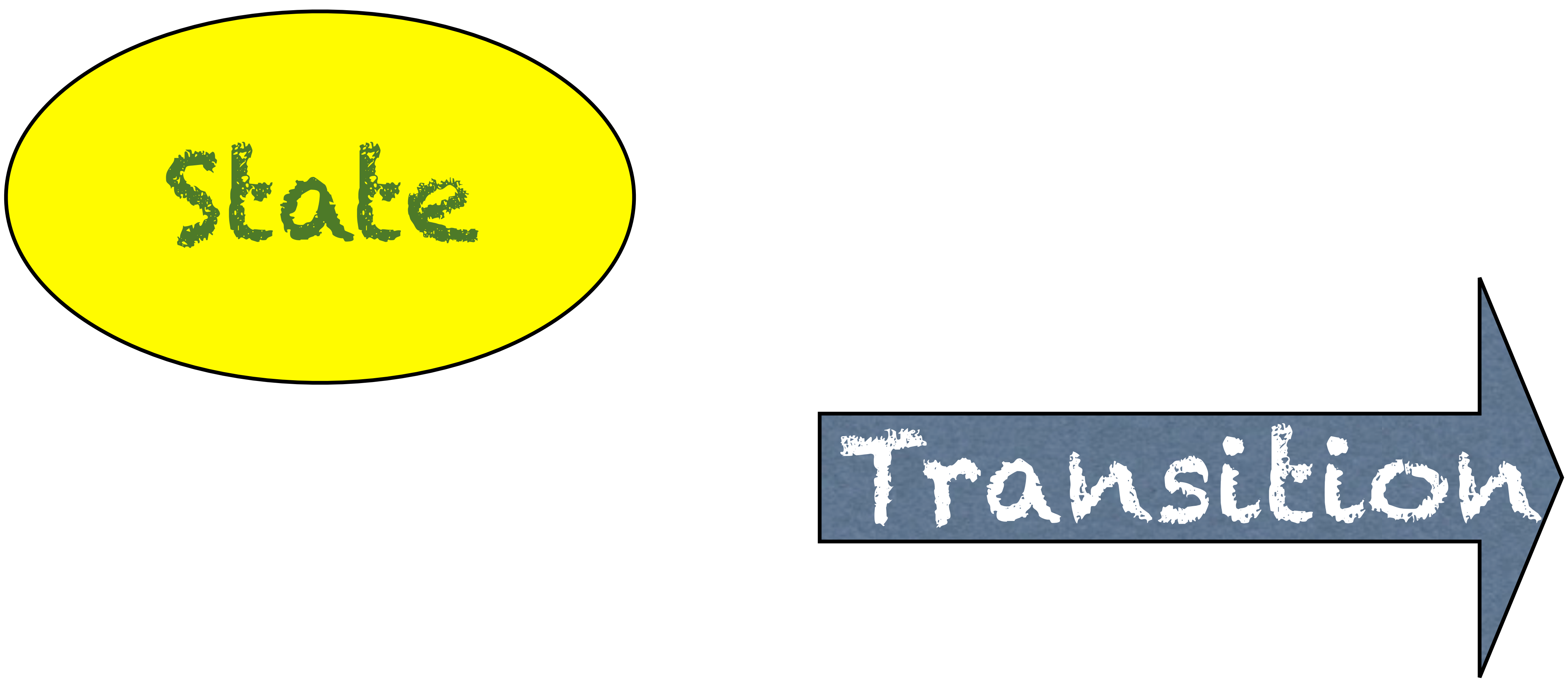
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

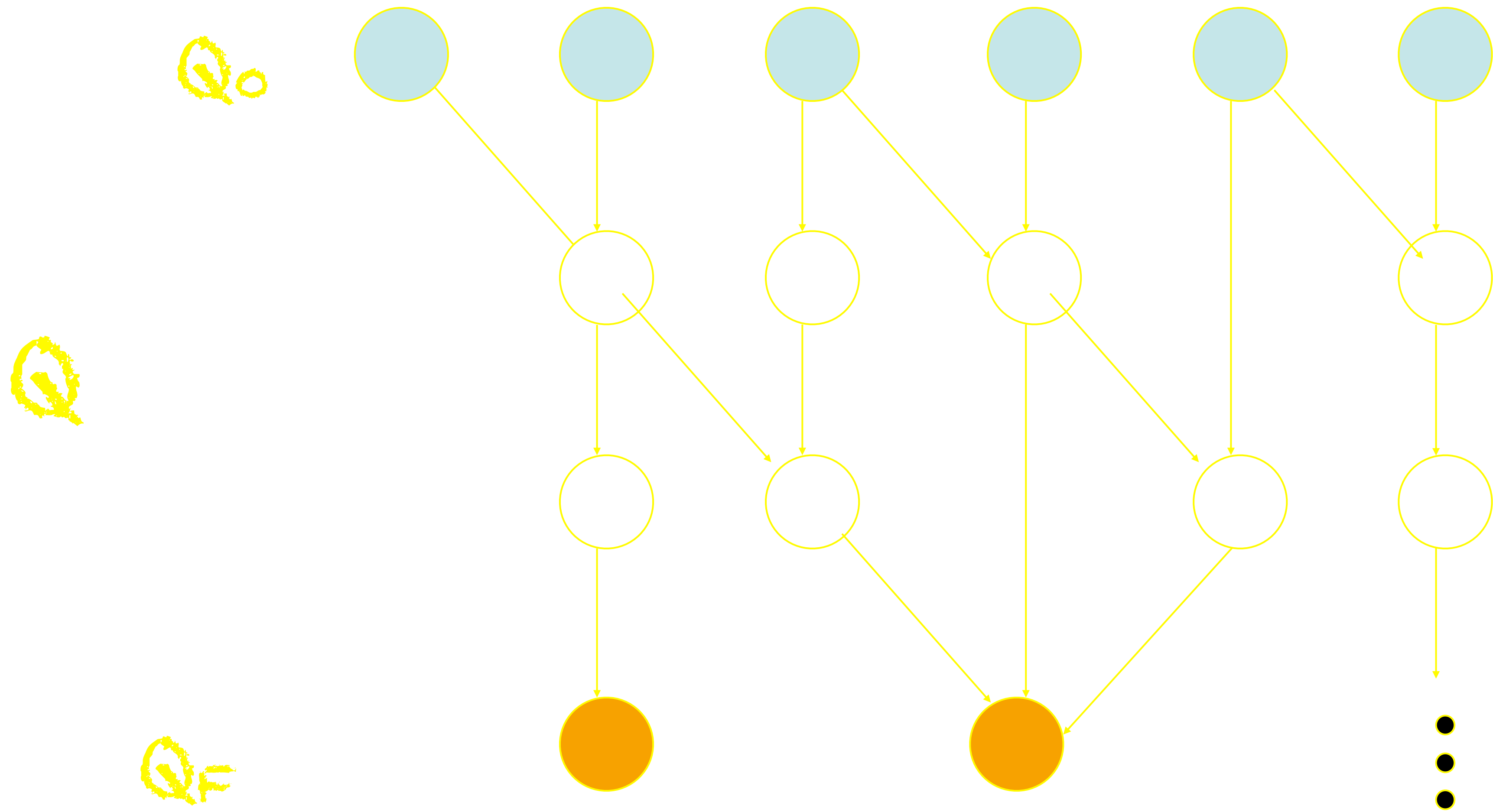
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

Transition System





Q0
Q
QF

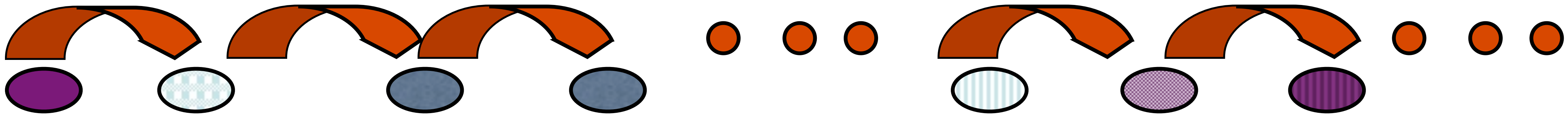
Well-Founded

- States Q
- Algorithm $R \subseteq Q \times Q$
- Well-founded order $>$ on Q
- $R \subseteq >$

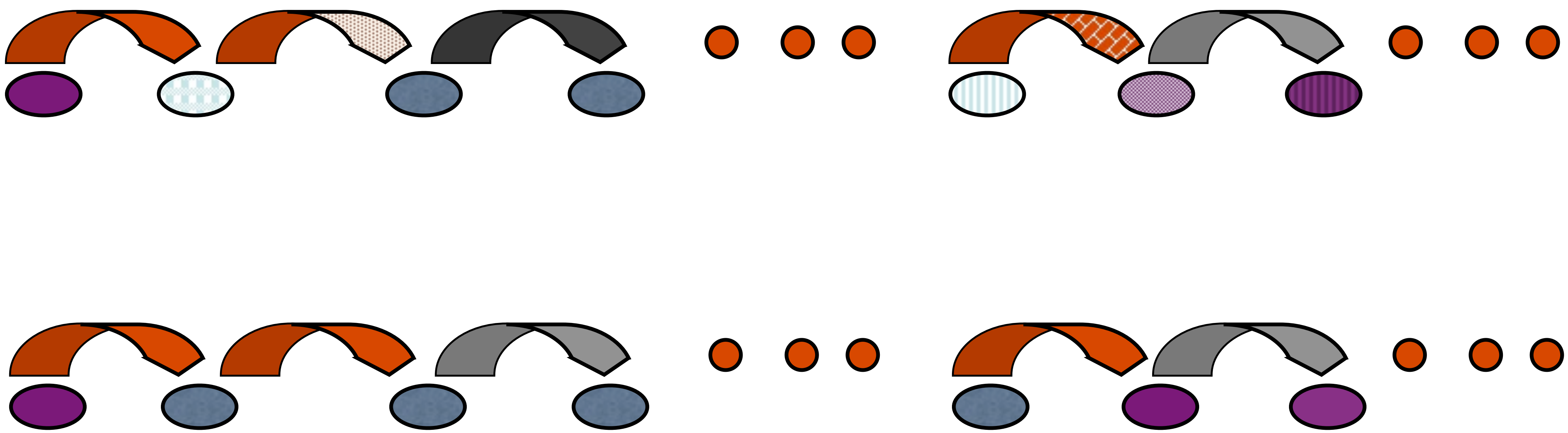
All-Purpose Ranking

- $r : Q \to \mathrm{Ord}$
- $r(x) = \sup \{ r(y)+1 : x \to y \}$

Computation



Abstraction



Frank Ramsey

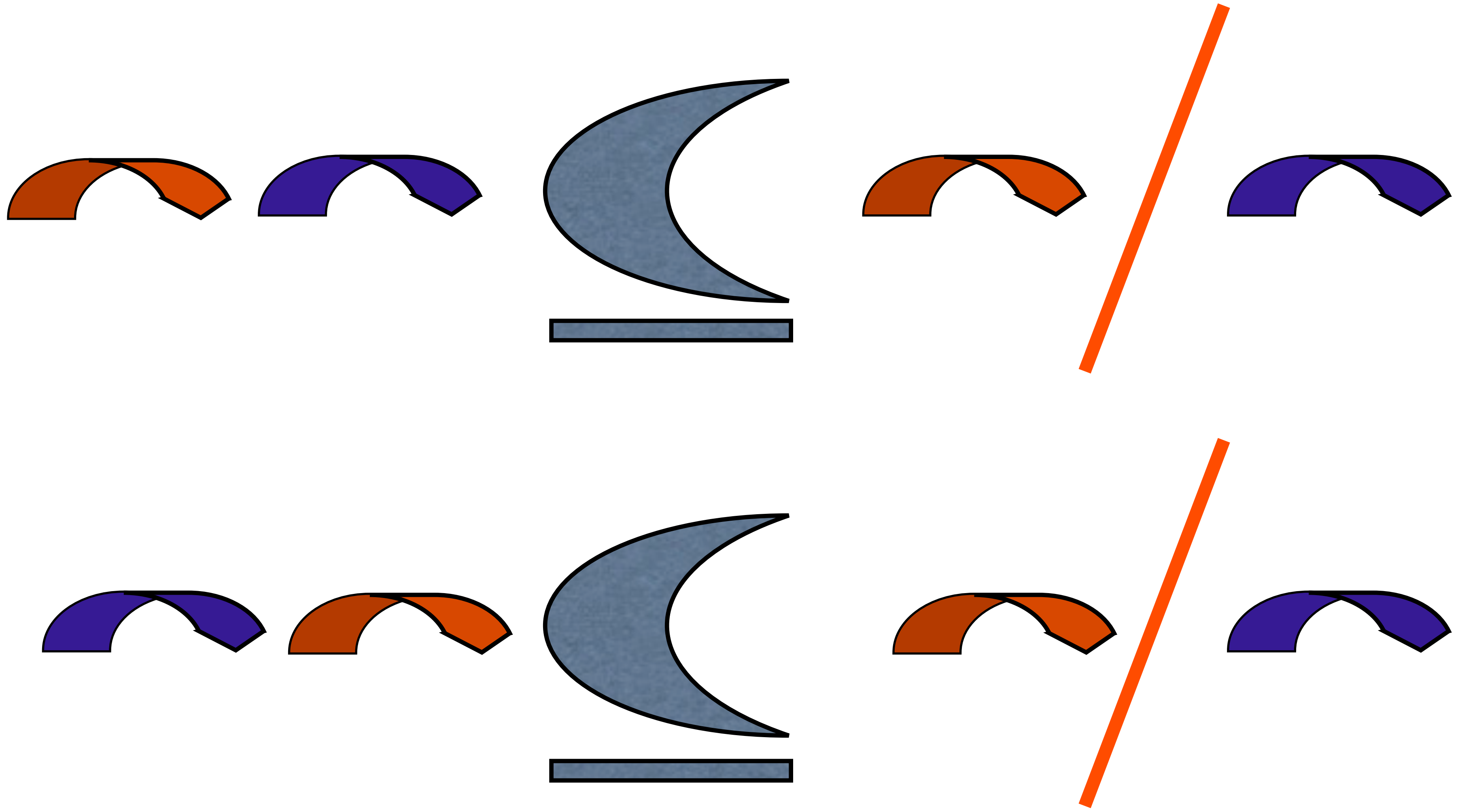
Ramsey's Theorem

Infinite complete graph

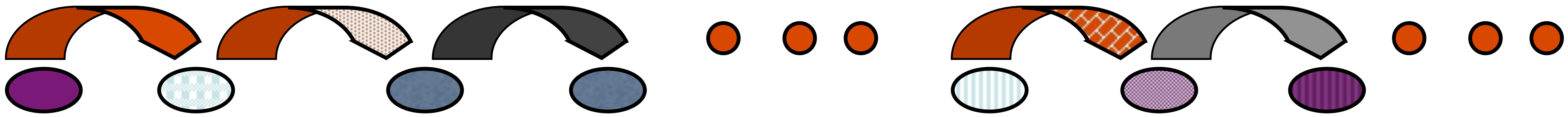
Finitely colored edges

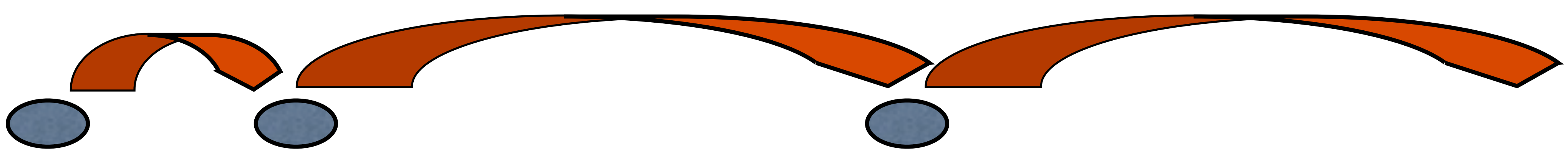
Monochrome infinite clique

Closure

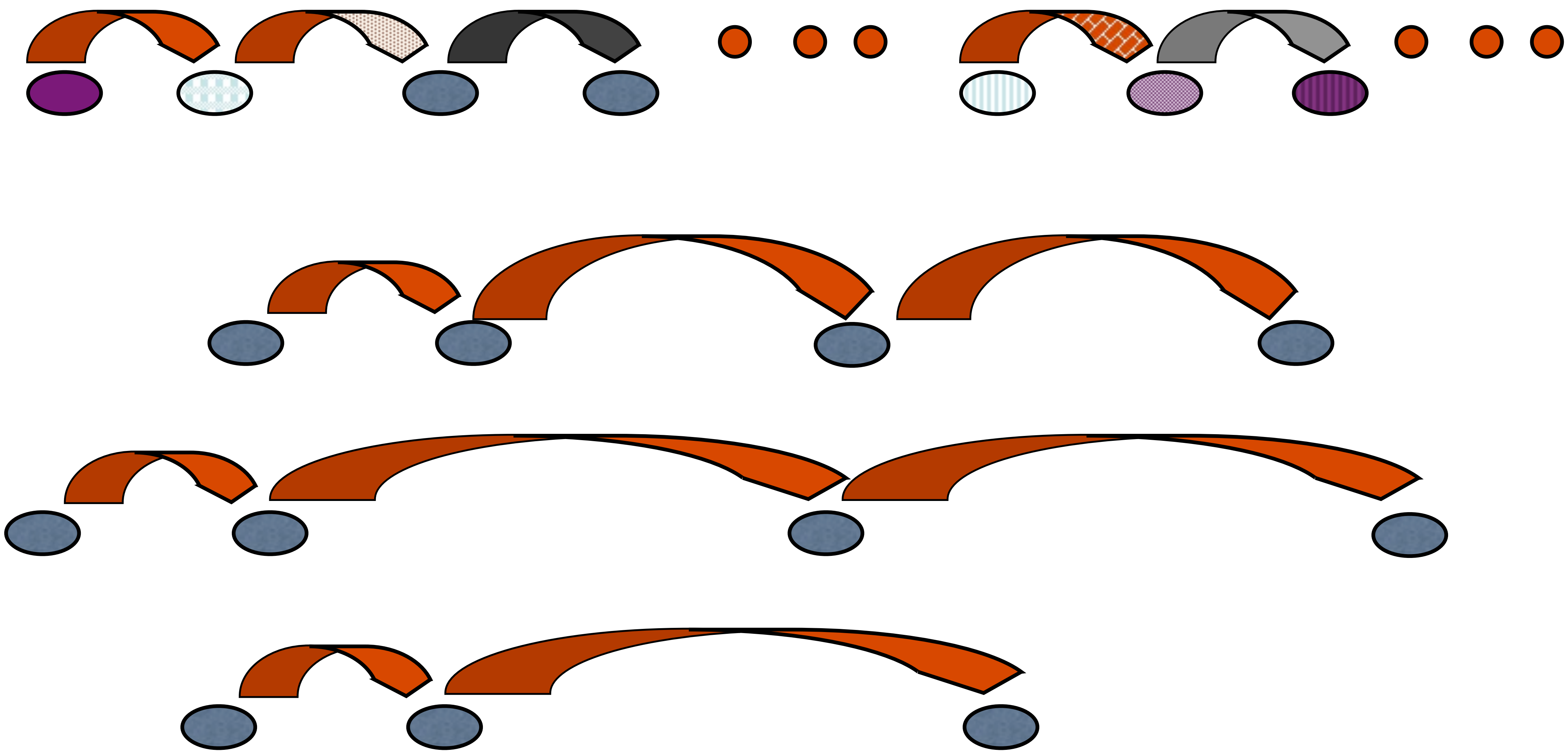


Proof

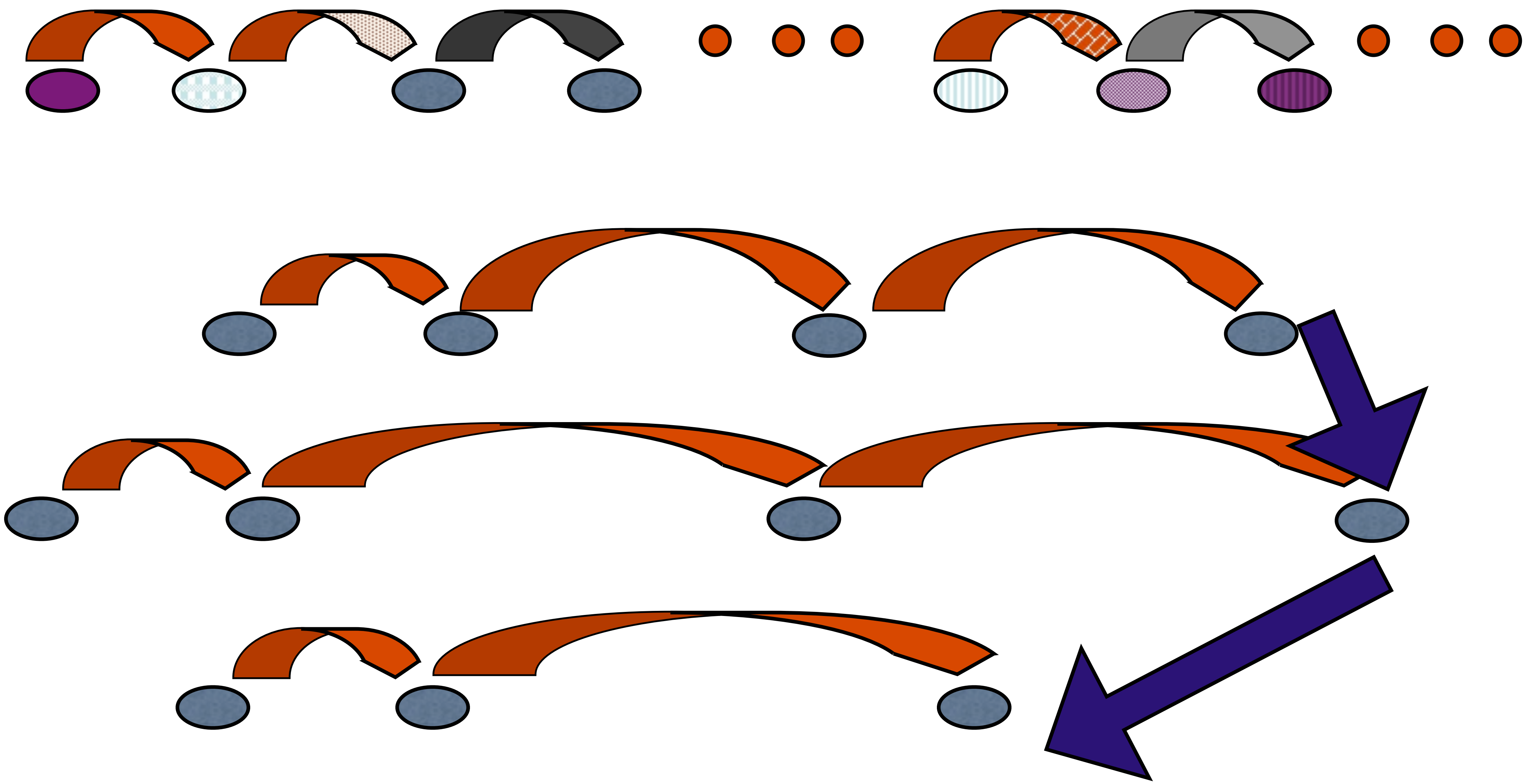




Proof



Proof



Disjunctive Orders

- States Q
- Algorithm $R \subseteq Q \times Q$
- Transitive closure R^+
- Well-founded orders $>$ and $\sqsupset$ on Q
- $R^+ \subseteq \; > \cup \sqsupset$

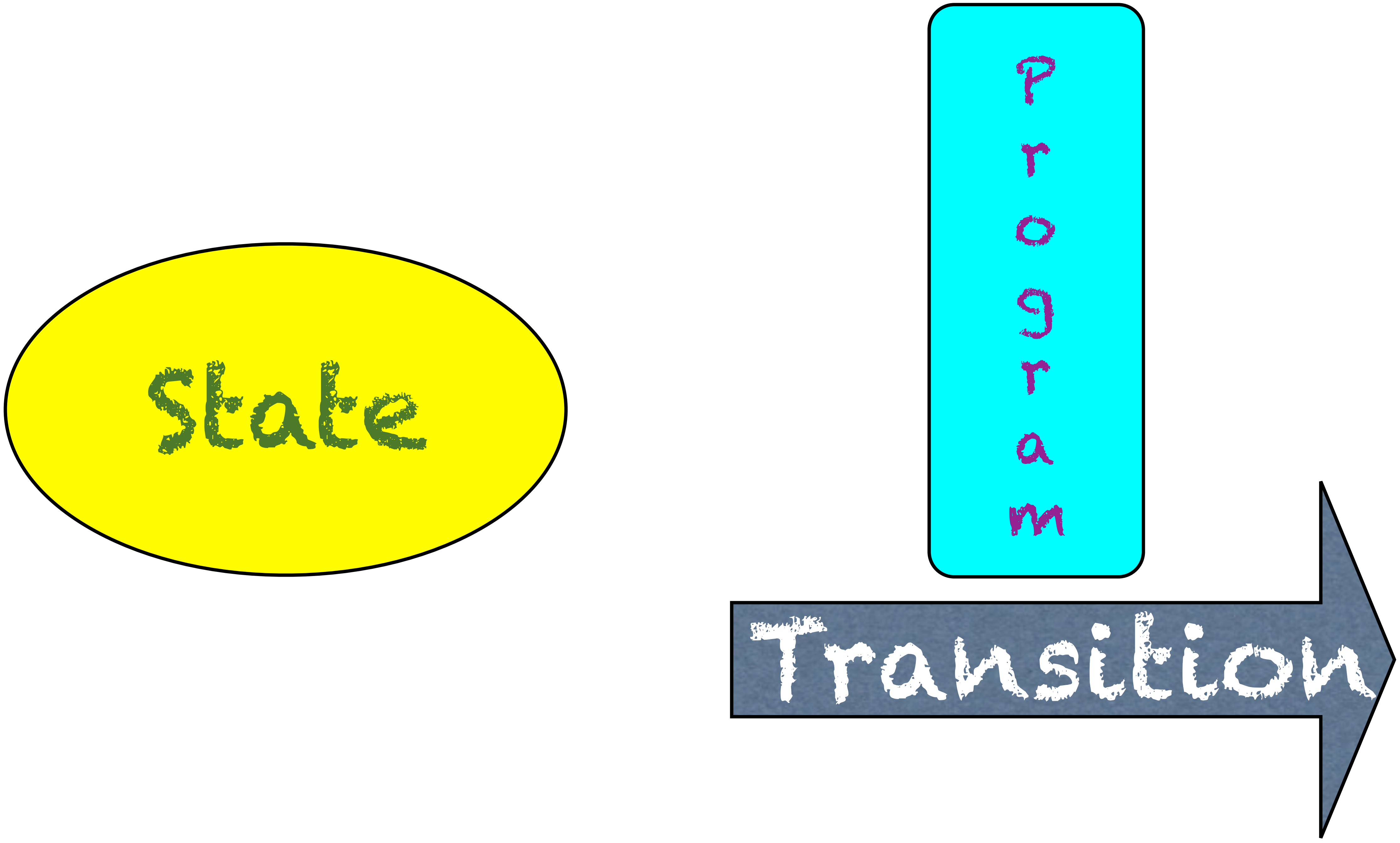
Ranking Method

- States Q
- Algorithm $R \subseteq Q \times Q$
- Well-founded order $\succ$ on W
- Ranking function $r : Q \rightarrow W$
- Define $X > Y$ if $r(X) \succ r(Y)$

Invariants

- States Q
- Algorithm $R \subseteq Q \times Q$
- Well-founded order $\succ$ on W
- Ranking function $r : Q \rightarrow W$
- Define $X > Y$ if $r(X) \succ r(Y)$

Algorithmic System



Classical Algorithms

- Every algorithm can be expressed precisely as a set of conditional assignments, executed in parallel repeatedly.

if c then f(s1,...,sn) := t

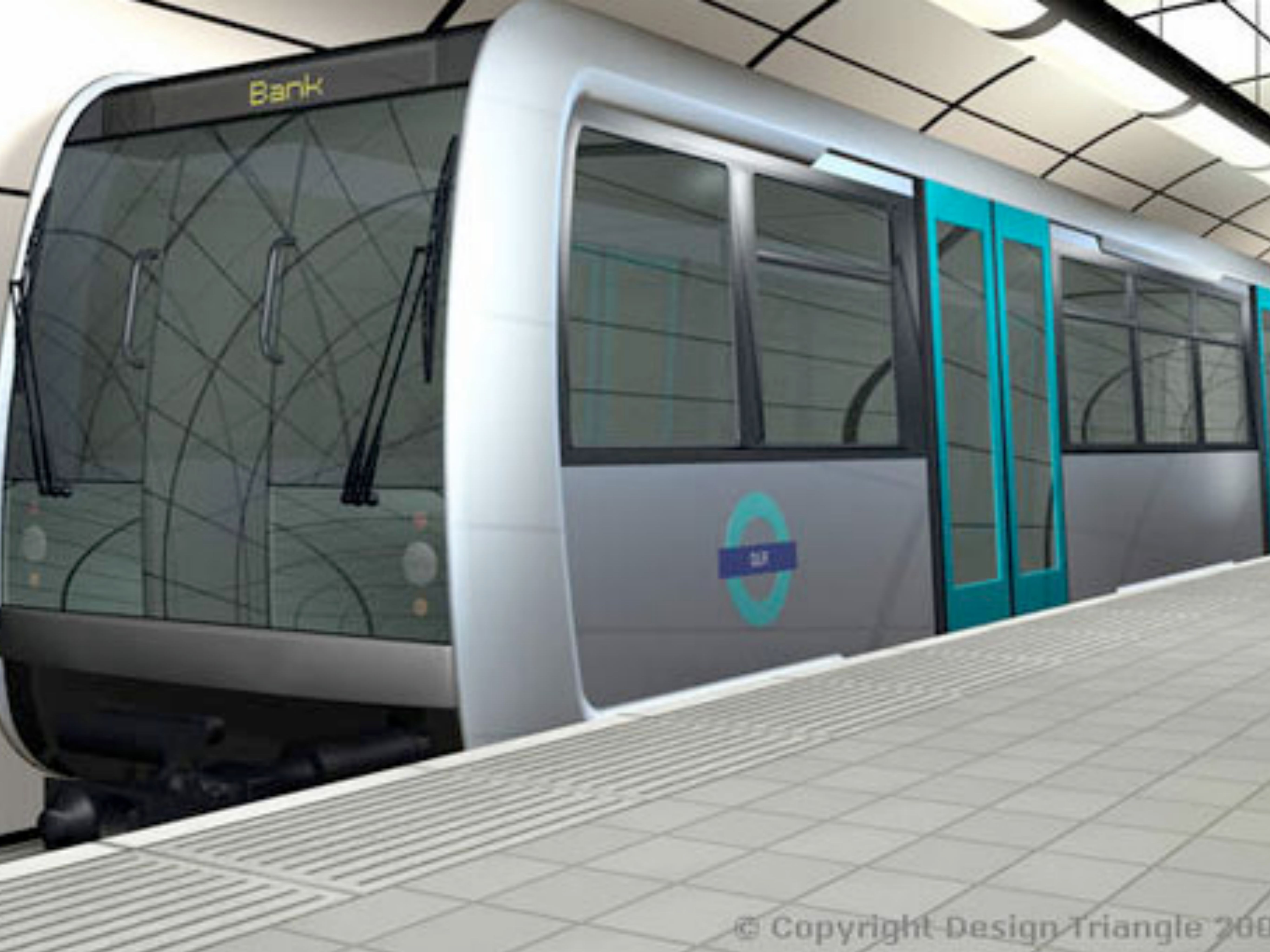
if c then f(s1,...,sn) := t

Practical Method

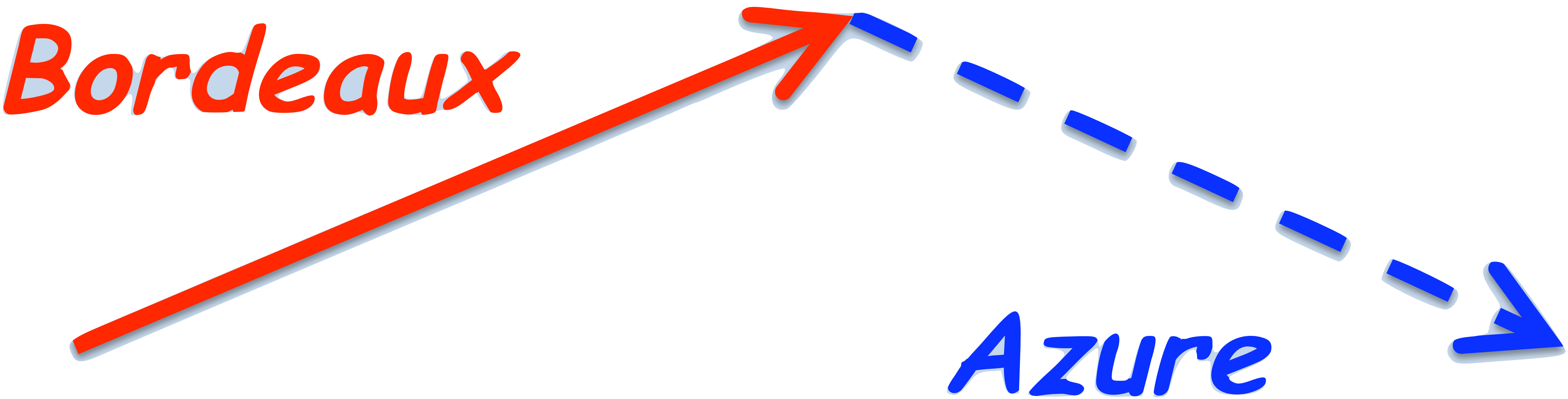
- States Q
- Algorithm $R \subseteq Q \times Q$
- Well-founded order $\succ$ on W
- Ranking function $r : Q \rightarrow W$
- Define $X > Y$ if $r(X) \succ r(Y)$

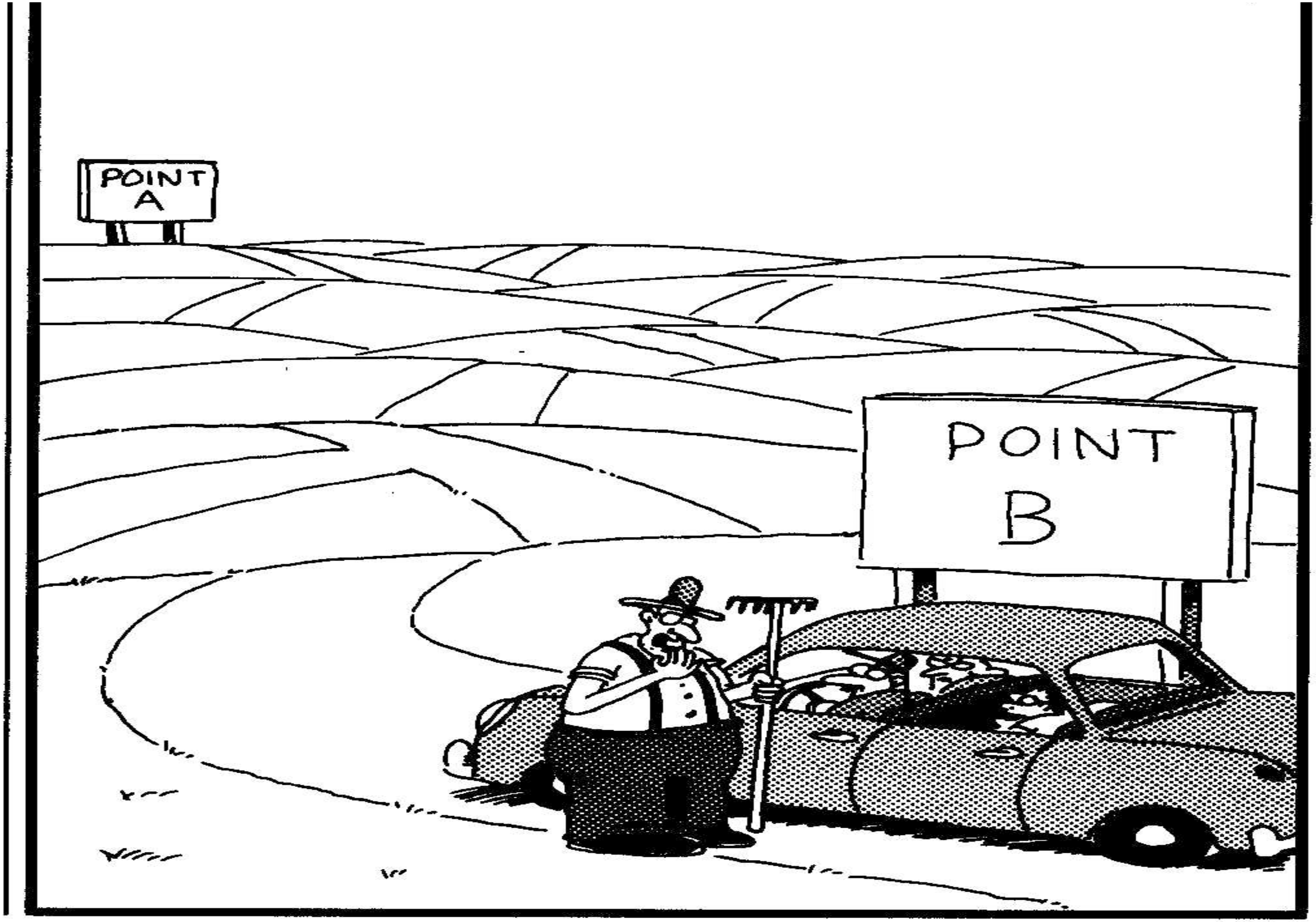
DLR


Town centre
Woolwich New Rd
National Rail
Bank
www.dlr.co.uk
95
DLR


Bank
© Copyright Design Triangle

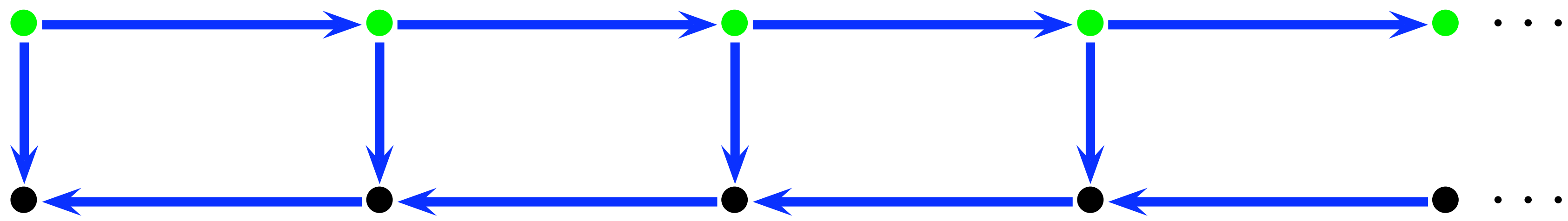
Color Code



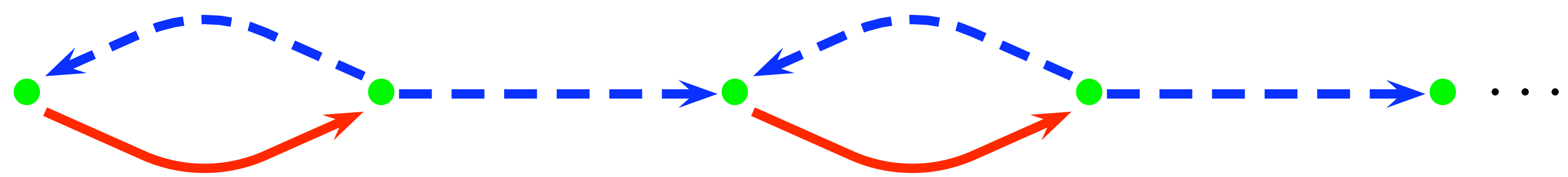


"Well, lemme think. . . . You've stumped me, son. Most folks only wanna know how to go the other way."

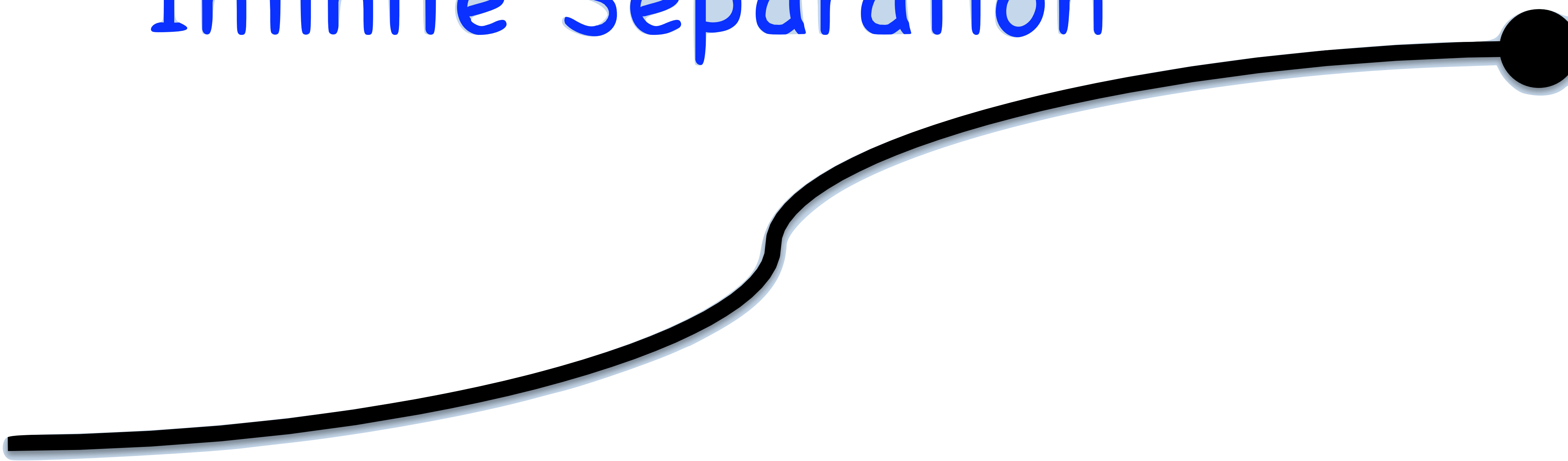
Mortal (black) nodes on bottom and immortal (green) nodes on top



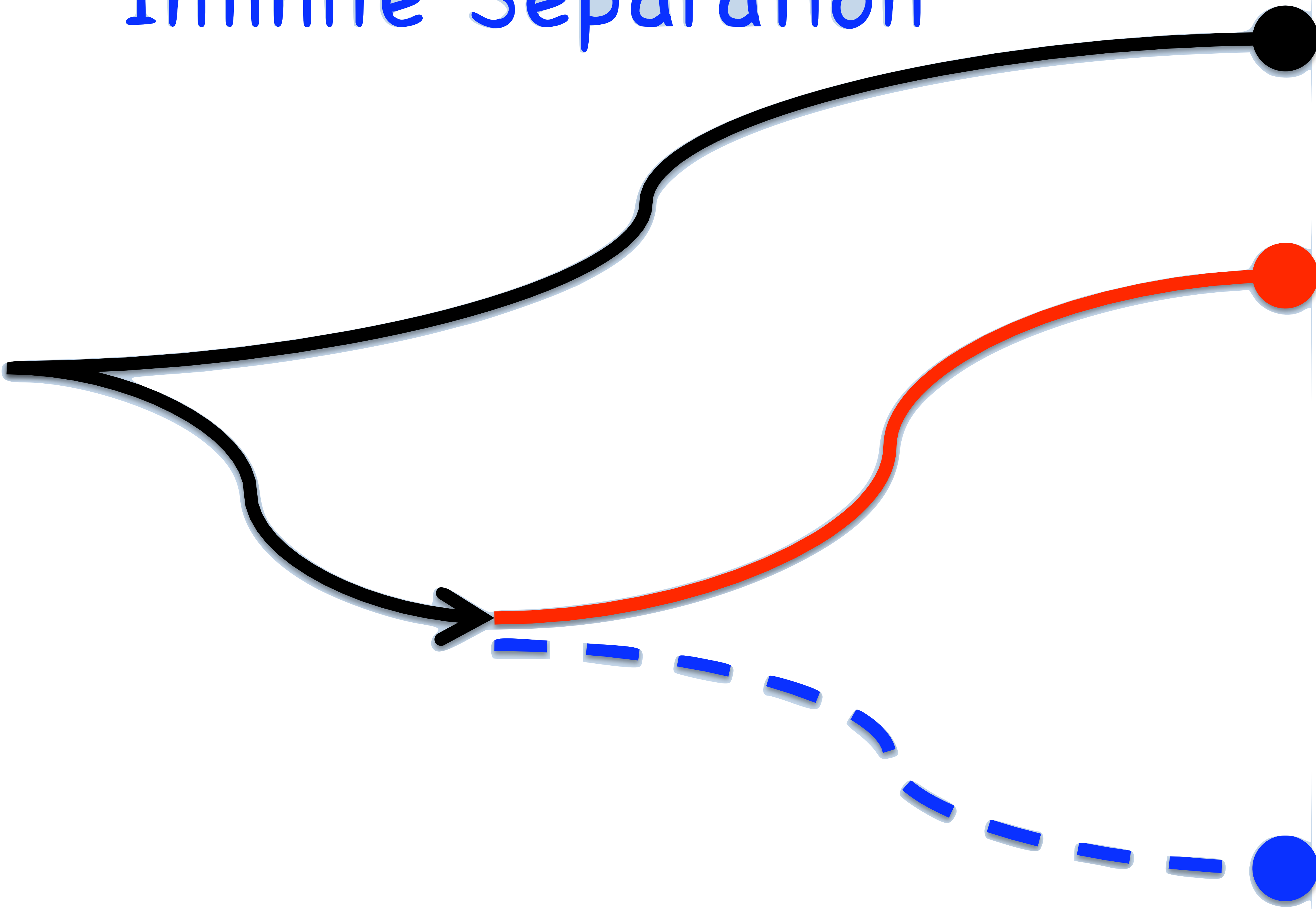
Mortal in each alone (dashed **Azure** or solid **Bordeaux**), but immortal in their union



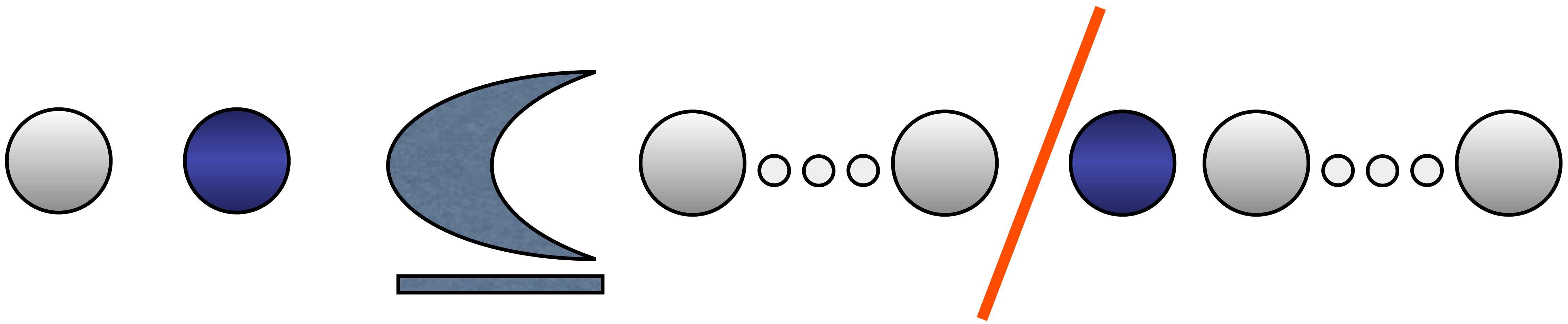
Infinite Separation



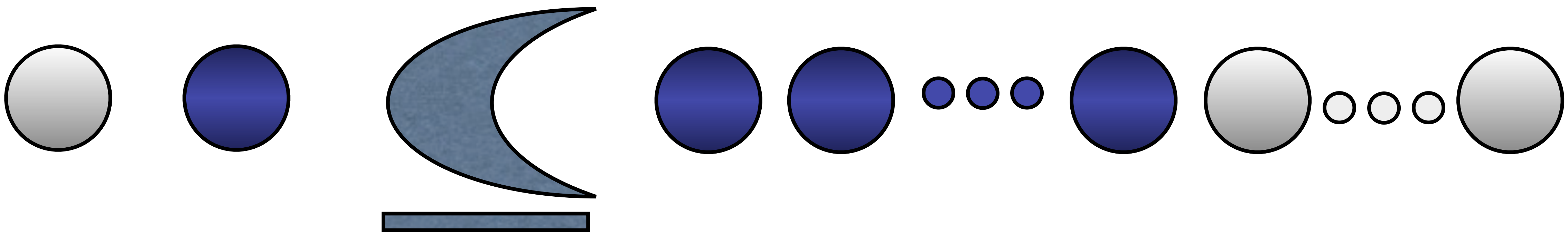
Infinite Separation



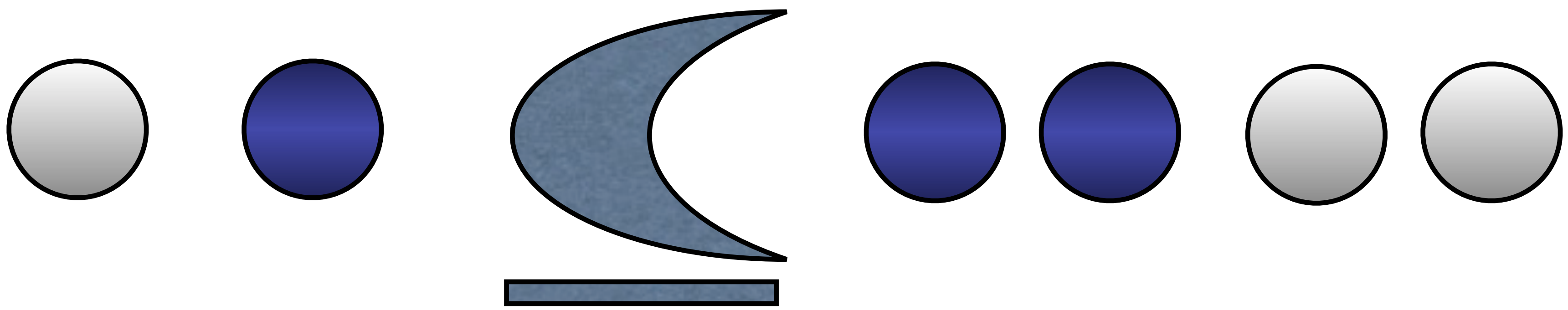
Enough?



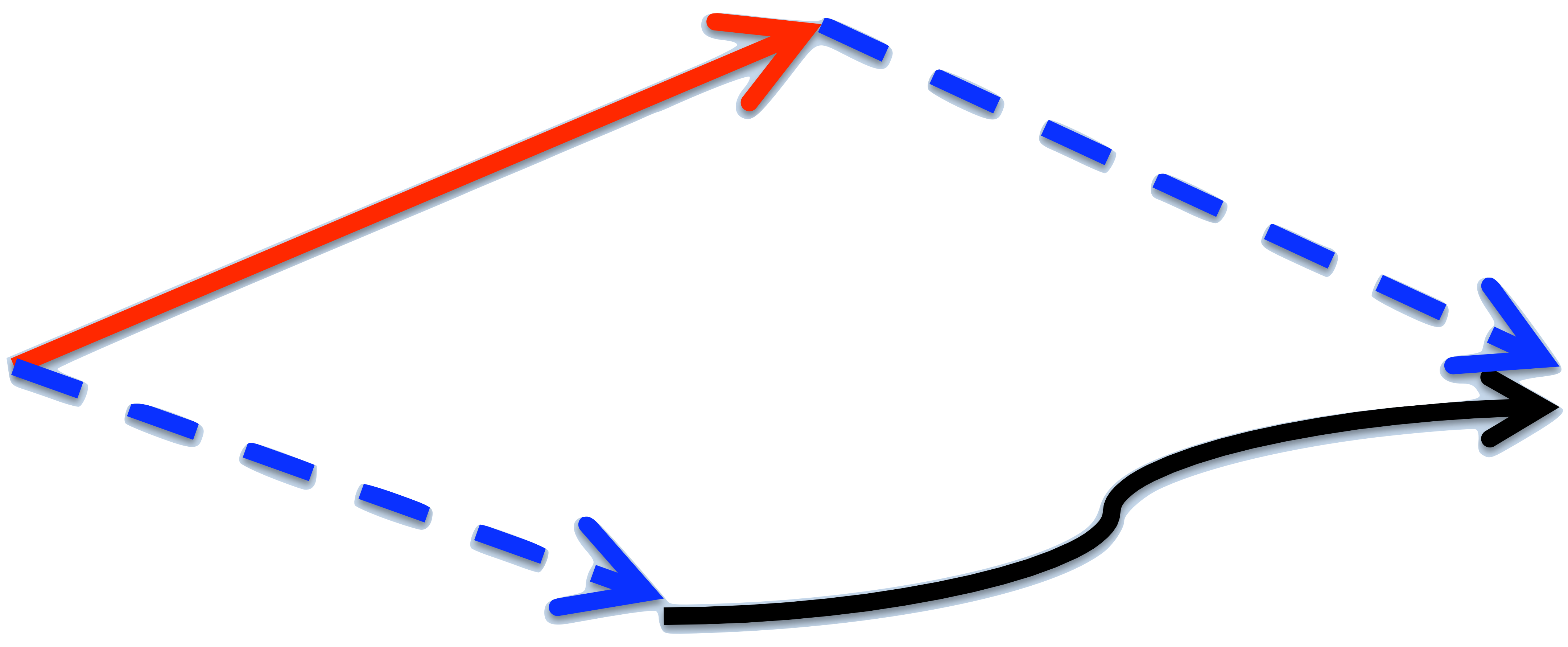
Enough?



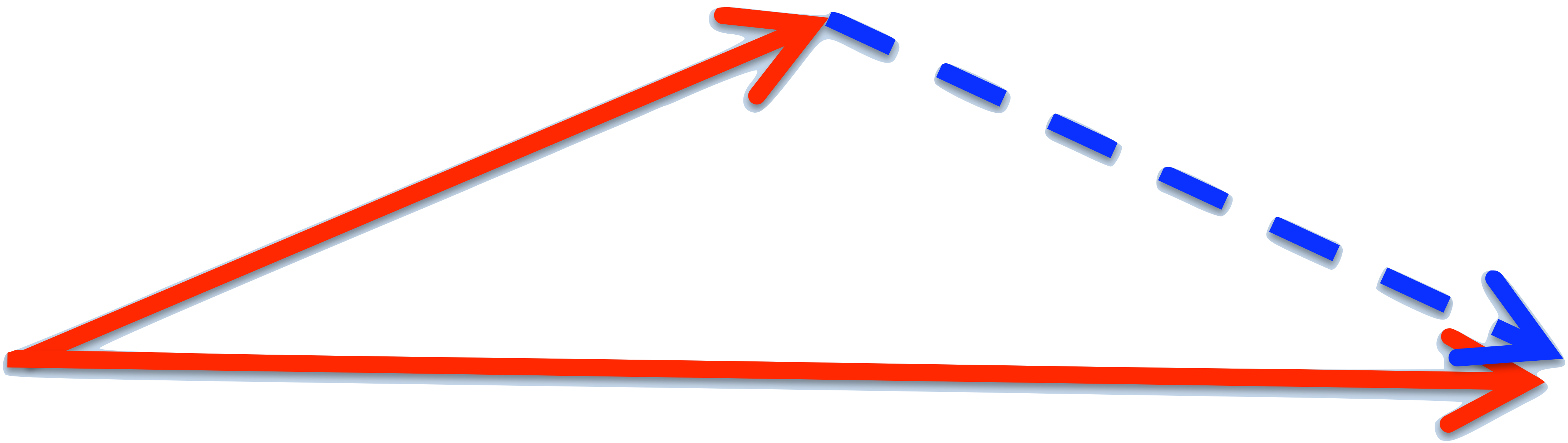
Enough?



Jumping

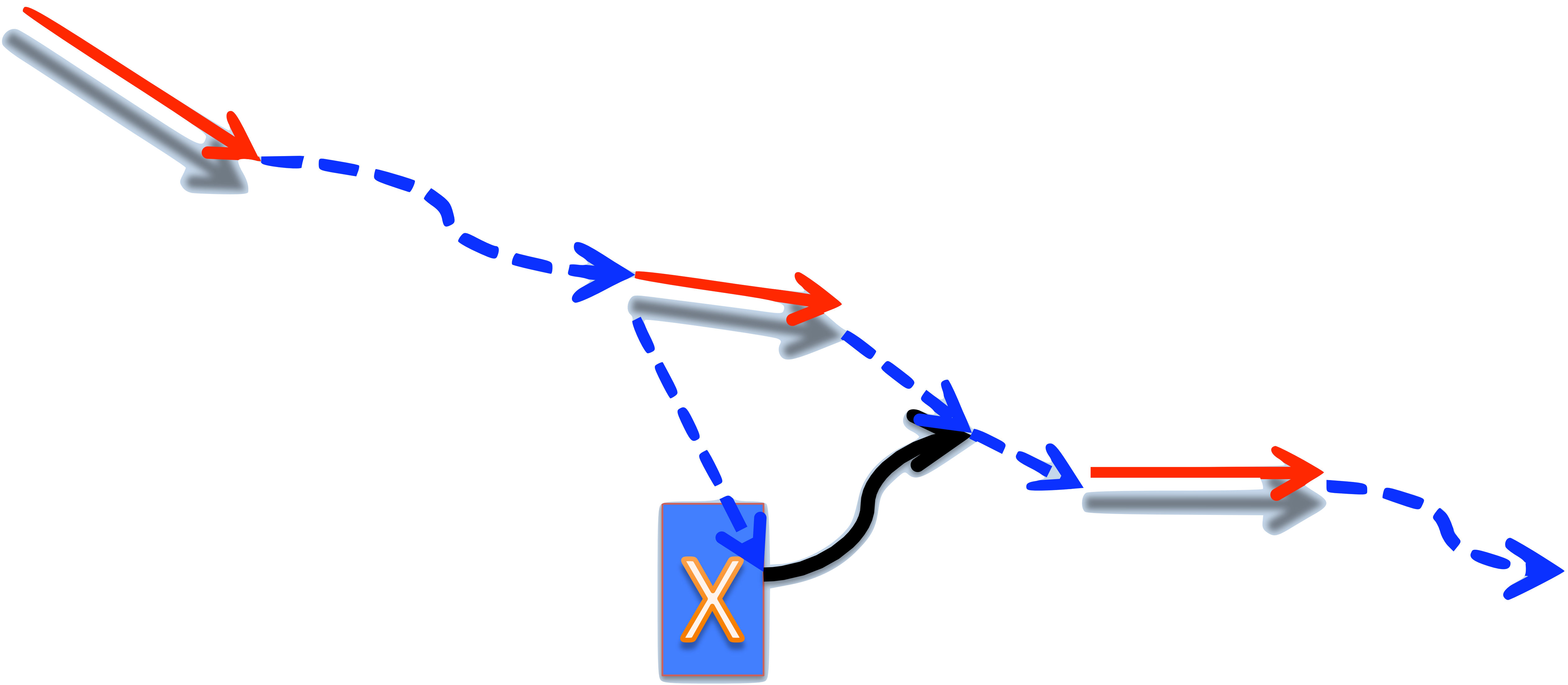


Jumping

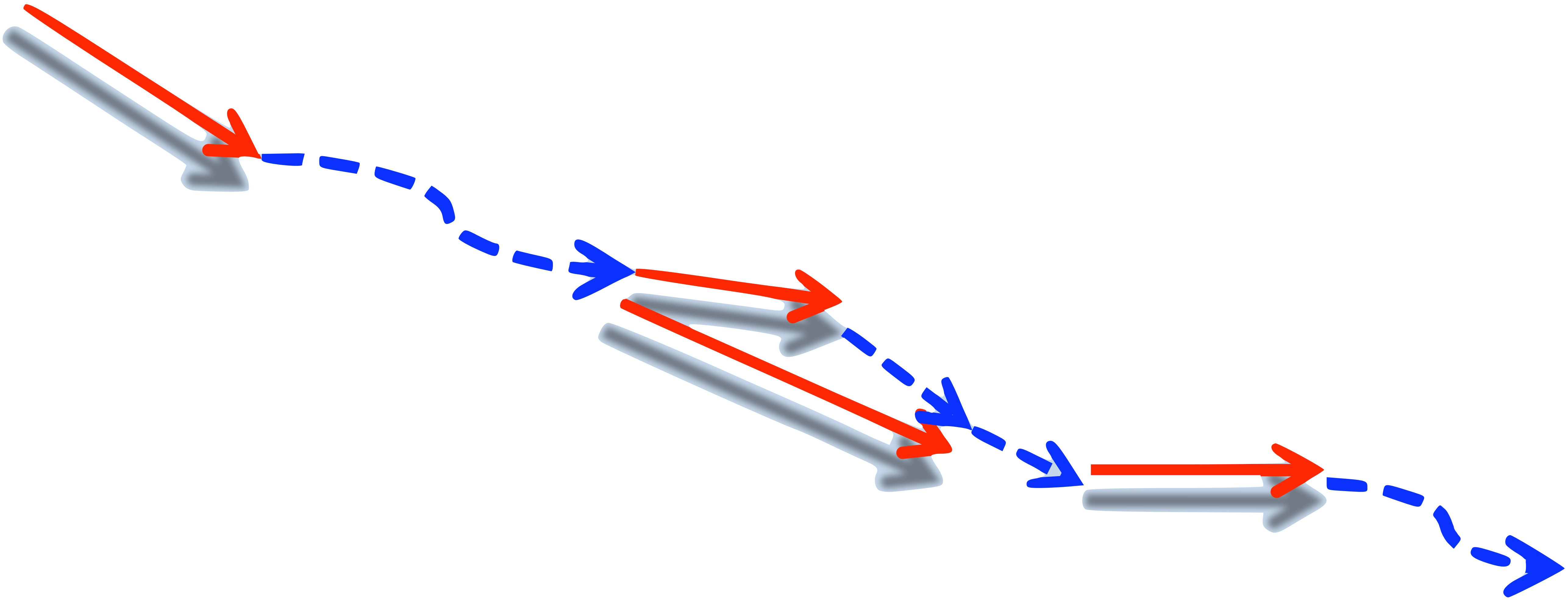



ONLY

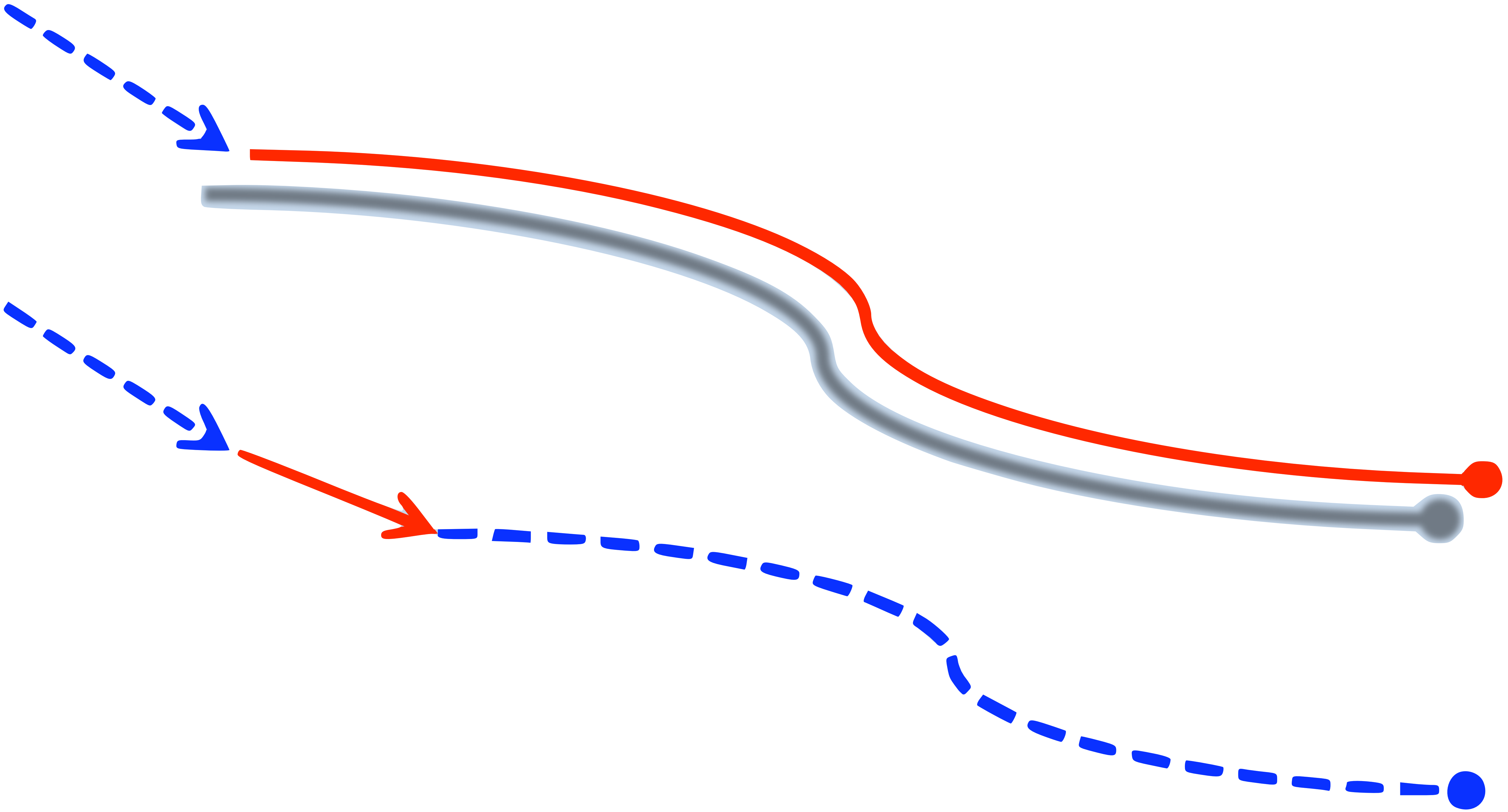
Constriction + Jumping



Constriction + Jumping



Constriction + Jumping



Termination

5. Well-Quasi Orderings

- $Dt = 1$
- $Dc = 0$
- $D(x+y) = Dx+Dy$
- $D(xy) = yDx+xDy$

CONTRIBUTIONS TO MECHANICAL MATHEMATICS

by

Renato Iturriaga

May 27, 1967

Carnegie-Mellon University
Pittsburgh, Pennsylvania

Accession Number : AD0670558

Title : TERMINATION OF ALGORITHMS.

Descriptive Note : Doctoral thesis,

Corporate Author : CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE

Personal Author(s) : Manna,Zohar

Report Date : APR 1968

Pagination or Media Count : 105

Abstract : The thesis contains two parts which are self-contained units. In Part 1 we present several results on the relation between the problem of termination and equivalence of programs and abstract programs, and the first order predicate calculus. Part 2 is concerned with the relation between the termination of interpreted graphs, and properties of well-ordered sets and graph theory. (Author)

Descriptors : (*COMPUTER PROGRAMMING, ALGORITHMS), COMPUTER PROGRAMS, NUMERICAL ANALYSIS, SET THEORY, GRAPHICS, THEORY, FLOW CHARTING, SEQUENCES(MATHEMATICS), COMPATIBILITY, MATRICES(MATHEMATICS), THESES

Subject Categories : Theoretical Mathematics

Disjunctiveness

while c do

$A \mid B$

a,b wfo

$(A \cup B)^+ \subseteq a \cup b$

Disjunctiveness

while $x > 0$ and $y > 0$ do

$$\begin{array}{l|l} x := x-1 & y := y-1 \\ y := ? & \end{array}$$

$x_i > x_j \vee y_i > y_j$ for $i>j$

need $x_i \geq x_j$

Jumping

while c do

A | B

while c do A   while c do B

$BA \subseteq A(A \cup B)^* \cup B$

Jumping

while $x > 0$ and $y > 0$ do

$$\begin{array}{l|l} x := x-1 & y := y-1 \\ y := ? & \end{array}$$

$BA \subseteq A$

Jumping

while $x > 0$ and $y > 0$ do

$$\begin{array}{l} x := x-1 \\ y := x+y \end{array} \;\Big|\; y := y-1$$

$BA \subseteq AB$

Disjunctiveness

while $x > 0$ and $y > 0$ do

$$\begin{array}{l|l} x := x-1 & y := y-1 \\ y := xy & \end{array}$$

$BA \subseteq AB^*$

Fairness

```
s := true

n := 0

while s do

        n := n+1   |   s := false
```

Fairness

s := ?

n := 0

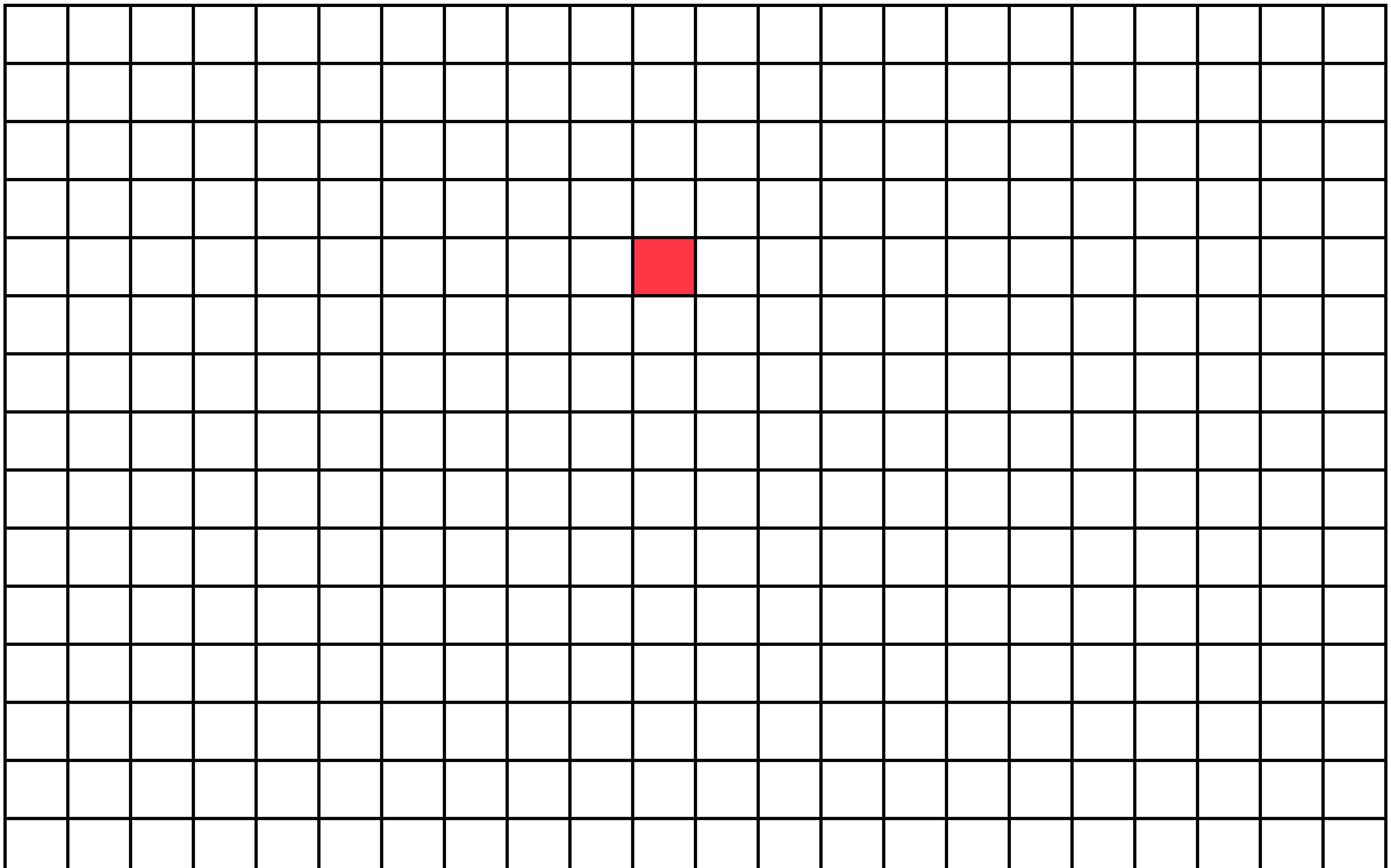
while s>0 do

n := n+1 | s := s-1

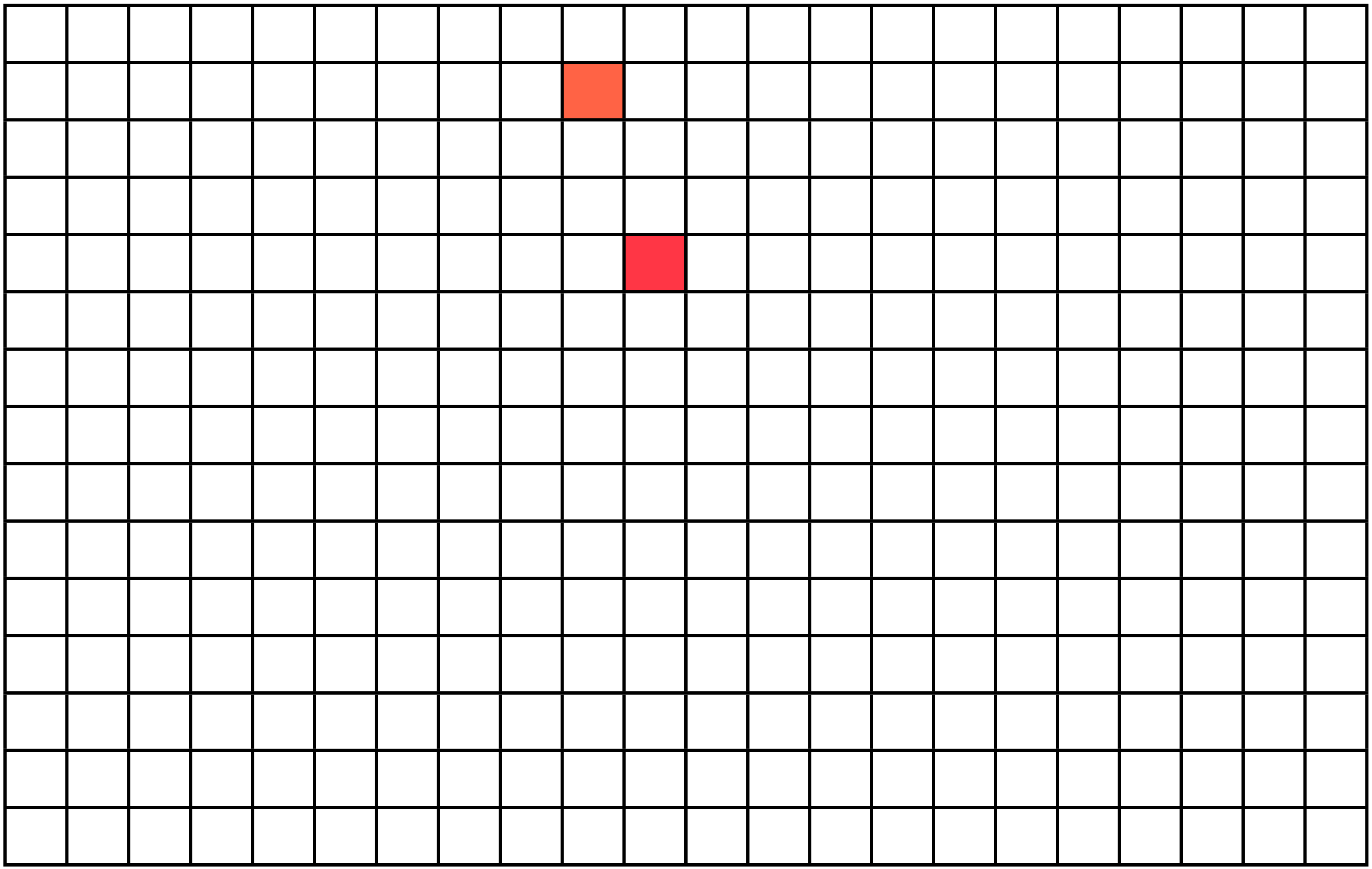
Grid Game

- Given (upper-right) grid coordinates $(x0,y0)$
- Choose (xj,yj) to prolong game s.t.
  - $xj < xi$ OR $yj < yi$ for all $i<j$

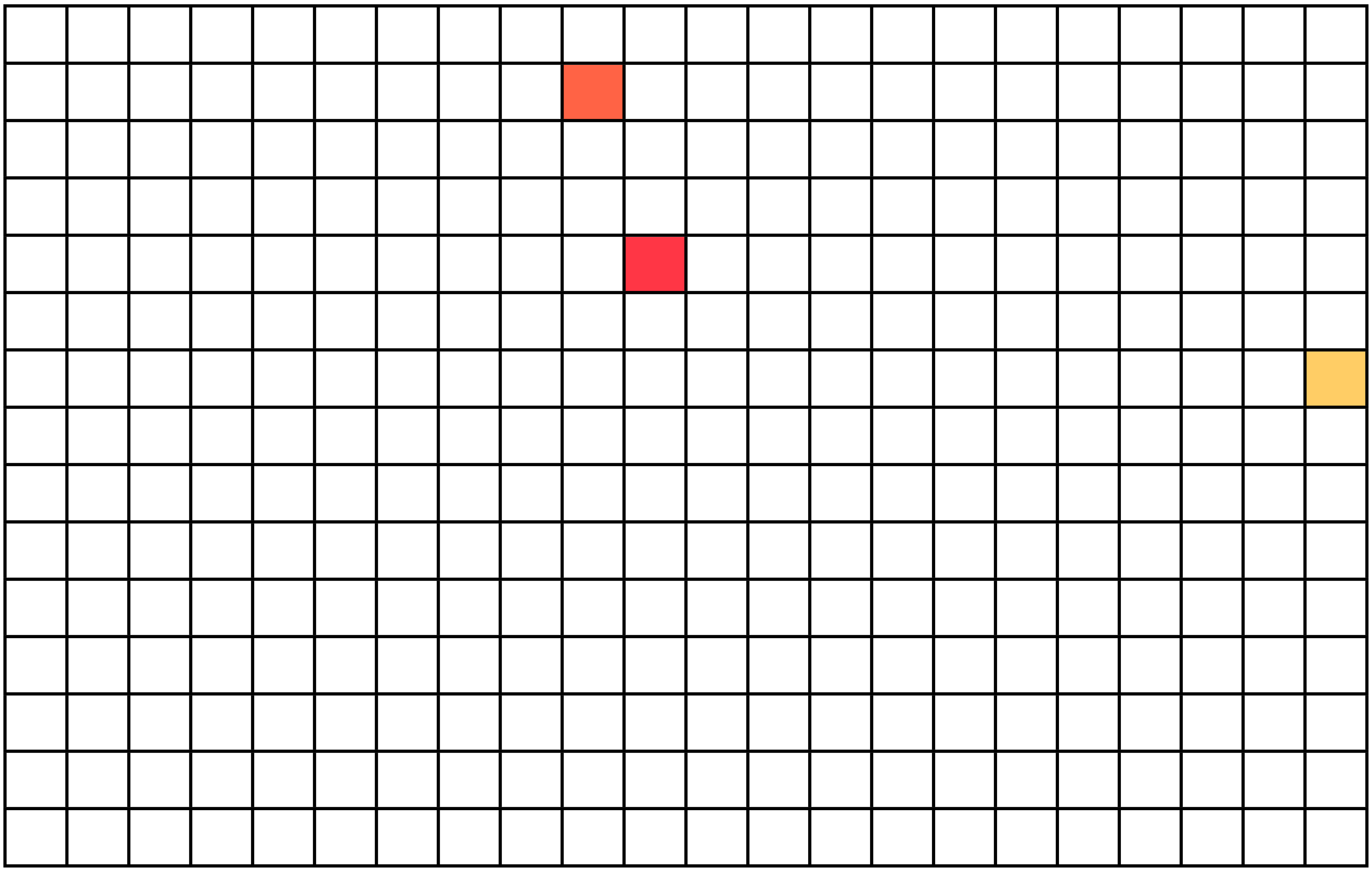
Grid Game



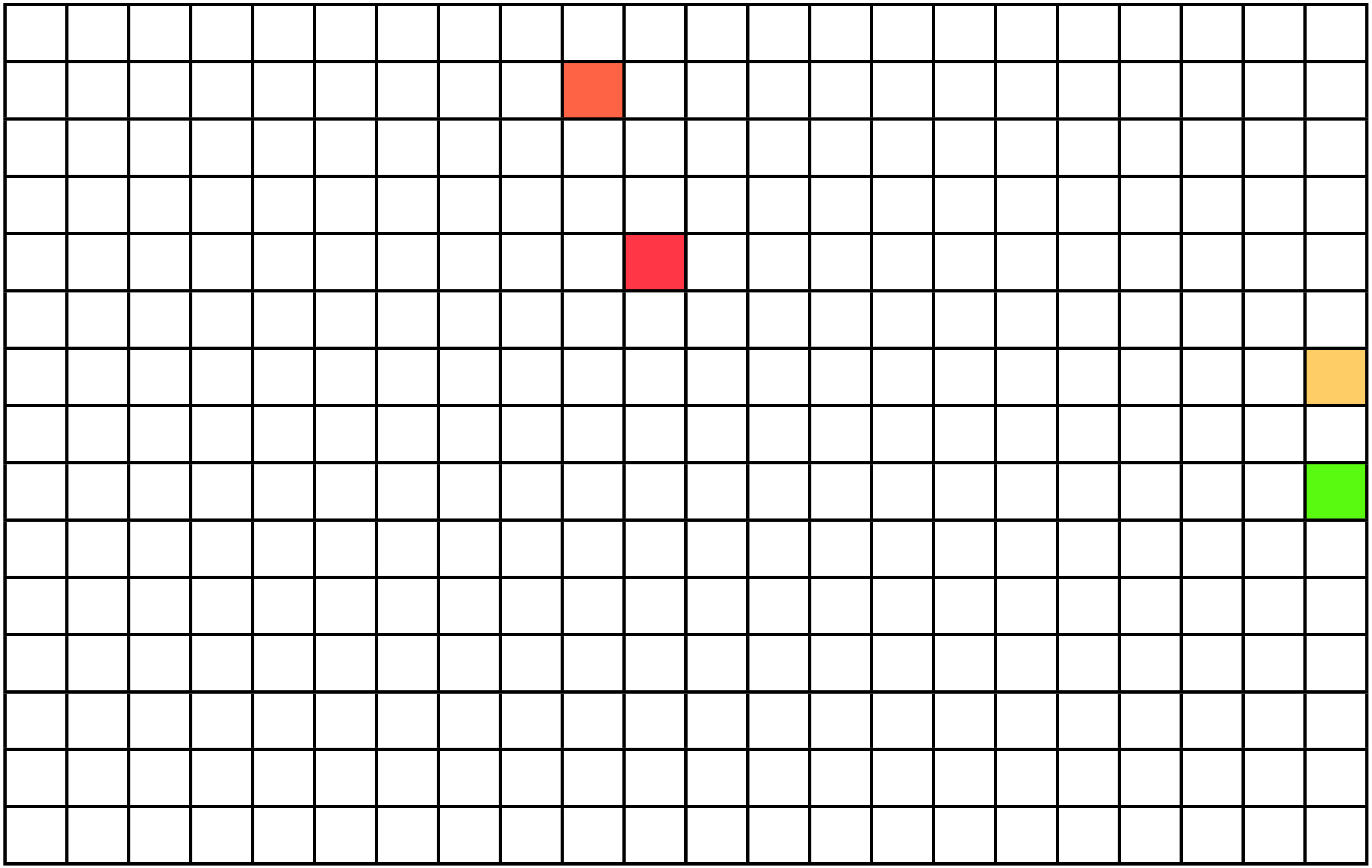
Grid Game



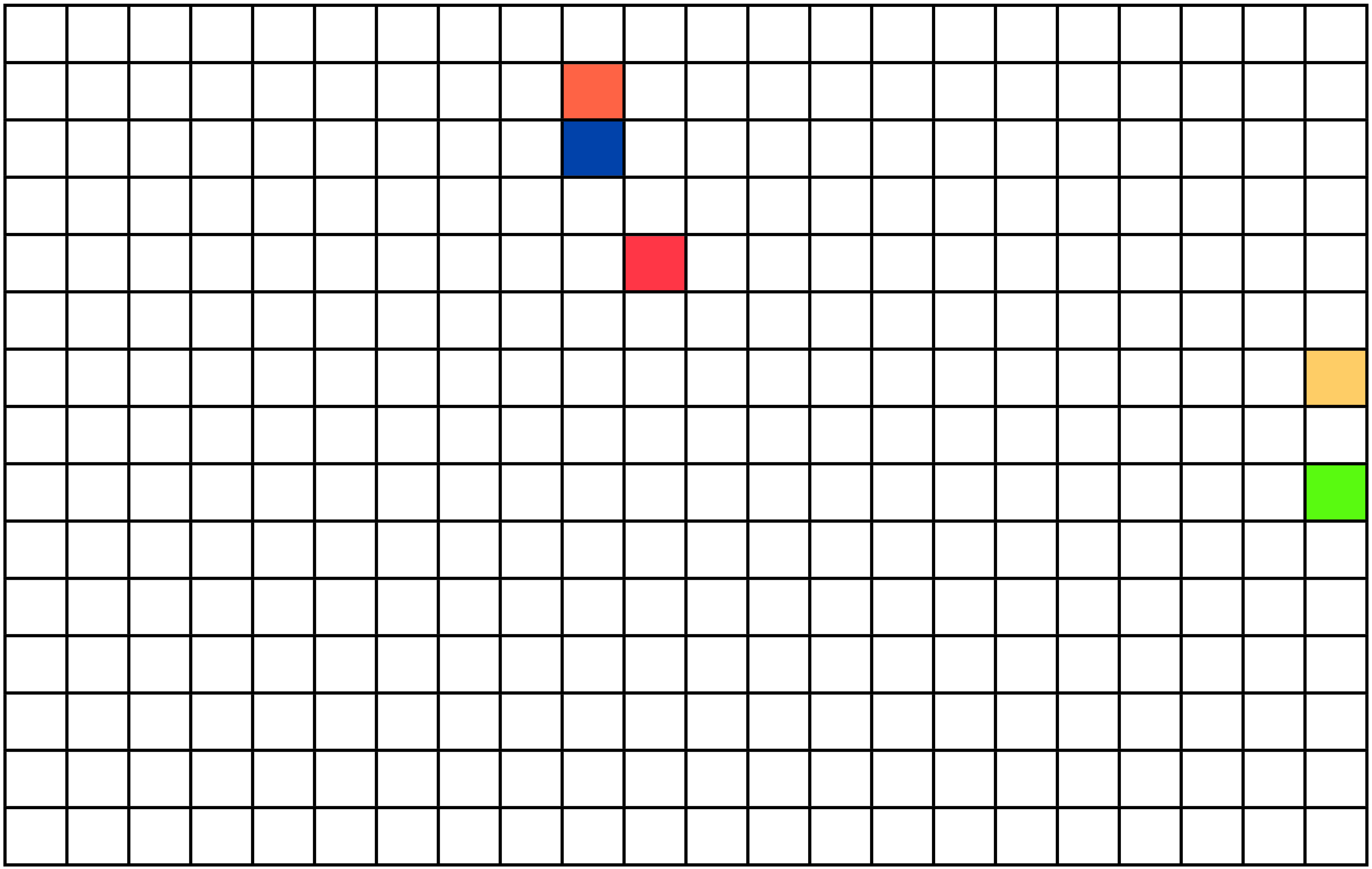
Grid Game



Grid Game



Grid Game



Tricolor

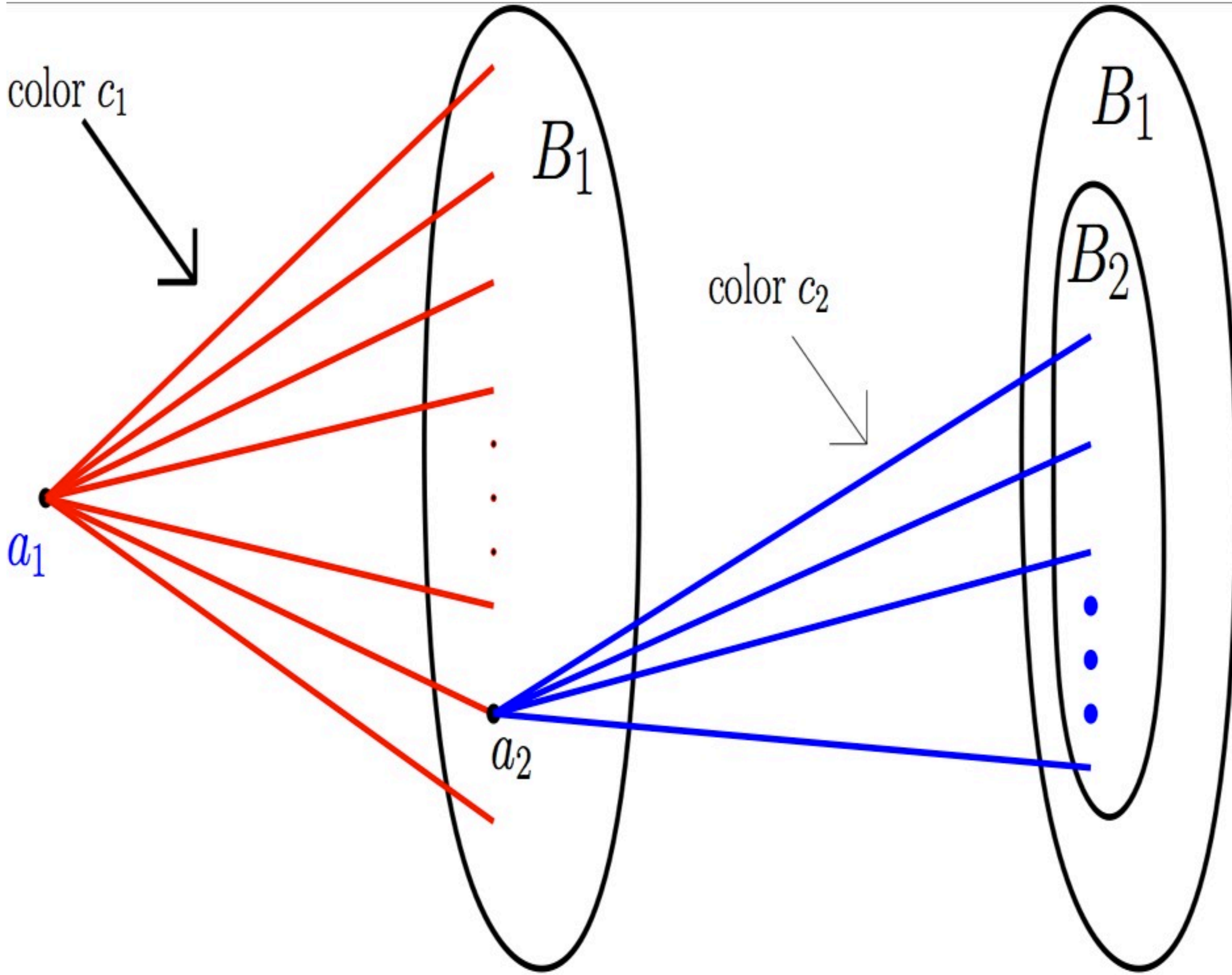
- Color pairs i<j of points
  - Purple if xi > xj and yi > yj
  - Blue if only xi > xj
  - Red if only yi > yj
- Consider sequence of points
  - Ramsey contradicts well-foundedness

Ramsey's Theorem

- Two colors: yes and no
- Extend yes as long as possible
- If can forever, then done (all yes)
- If not, then repeat

Ramsey's Theorem

- Reduce more than 2 colors to 2 (color-blindness). Repeat.
- For 2: Form sequence of nodes
  a1 a2 a3 ...   by repeatedly taking monochromatically-connected subsets



color c_1
B_1
a_1
a_2
color c_2
B_1
B_2

$S := V$

$R := \emptyset$

do forever

$\quad x :\in S$

$\quad R := R \cup \{x\}$

$\quad S := S \setminus \{x\}$

$\quad W := \{s \in S \mid c(x, s) = \text{white}\}$

$$S := \begin{cases} W & \textbf{if } |W| = \infty \\ S \setminus W & \textbf{otherwise} \end{cases}$$

$W := \{x \in R \mid \forall y \in R.\, y \neq x \to c(x, y) = \text{white}\}$

$$\textbf{return} \begin{cases} W & \textbf{if } |W| = \infty \\ R \setminus W & \textbf{otherwise} \end{cases}$$

Ramsey's Theorem

Infinite complete multi-graph

Finitely colored multi-edges

can have multiple multi-edges

Monochrome infinite clique

Quasi-ordering

- Greater or equivalent
  - Transitive
  - Reflexive

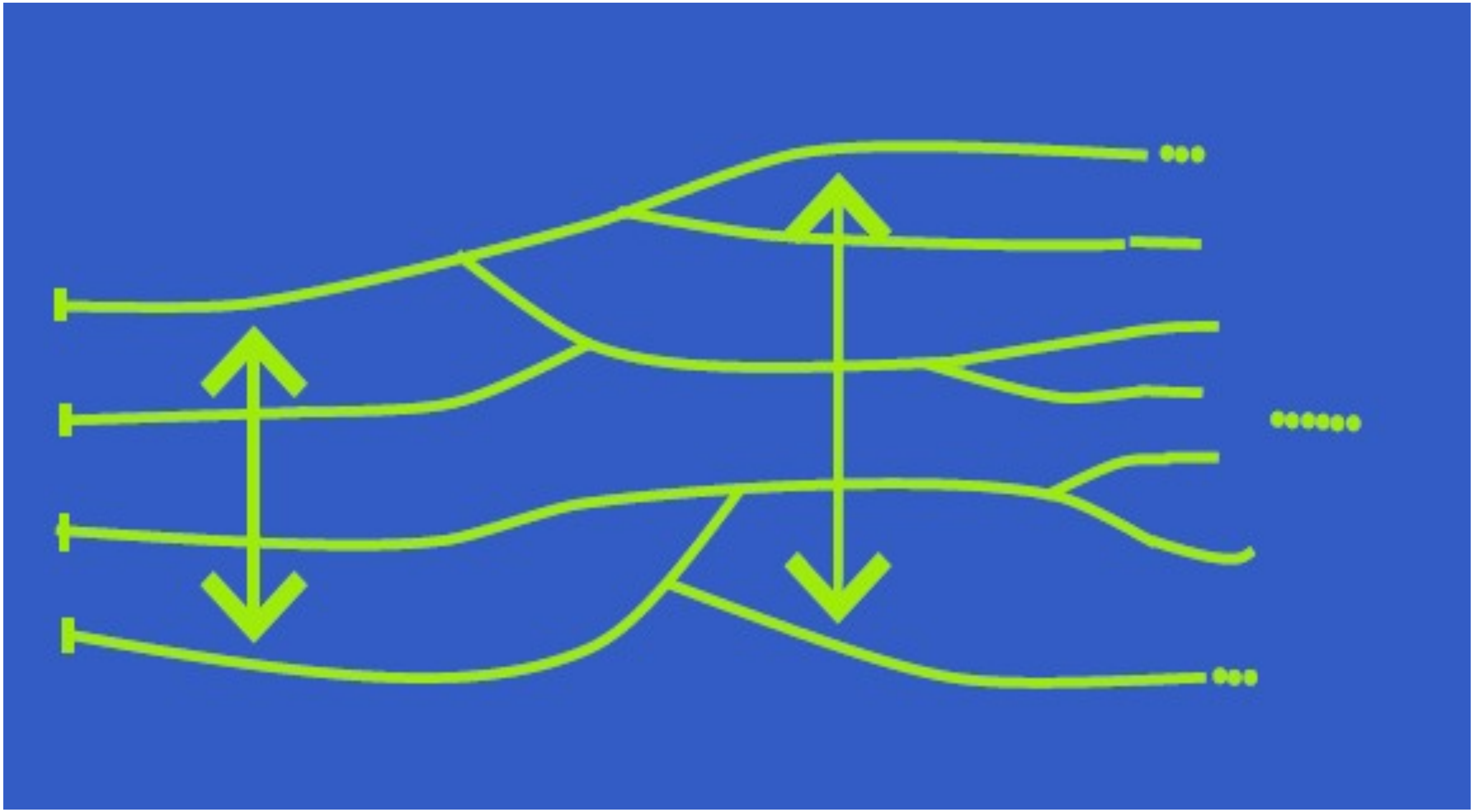
Quasi-ordering

- Equivalence (both directions)
- Strict part (only one)

Well-quasi-ordering

- Well-founded
  - no infinite strictly-descending sequences
- No infinite anti-chains

Wqo



A THEOREM ON PARTIALLY ORDERED SETS (Summary)

Michael Rabin

In the following note we give a condition for the finiteness of a partially ordered set. This theorem was established in order to prove the finiteness of certain classes of ideals.

Theorem.

Assumption: Let the partially ordered set M satisfy the following conditions:

a) The maximum condition (that is, the ascending chain condition).

b) The minimum condition (that is, the descending chain condition).

c) Every subset of M in which all pairs of elements are uncomperable, is finite.

Conclusion: M is finite.

The crucial point of the proof lies in the following general principle.

Equivalent Properties

- Wqo
- Every infinite sequence has an ordered pair

Well-Quasi-Order

Definition. *A set A is Well Quasi Ordered under $\precsim$ if for all infinite sequences from A:*

$$a_1, a_2, a_3, \ldots$$

there exists some $i < j$ such that $a_i \precsim a_j$.

Equivalent Properties

- Standard: wf and no inf antichain
- Simple: Every infinite sequence has an ordered pair
- Useful: Every infinite sequence contains an infinite non-decreasing chain
  - Why? -- Ramsey

Properties

- Every refinement (more order) is also wqo
- Every linearization (refinement s.t. all equivalence classes are comparable) is well-ordered

Dickson's Lemma

- Order (n-) tuples in product ordering
  - All components are in order

- Tuples of wqos are wqo

Good

- A pair is good if it is ordered
- A sequence is good if it has a good pair
- A set is good (wqo) if all sequences are good

Bad

- A sequence is bad if there is no good pair
- It is good if it has at least one pair

Good & Bad

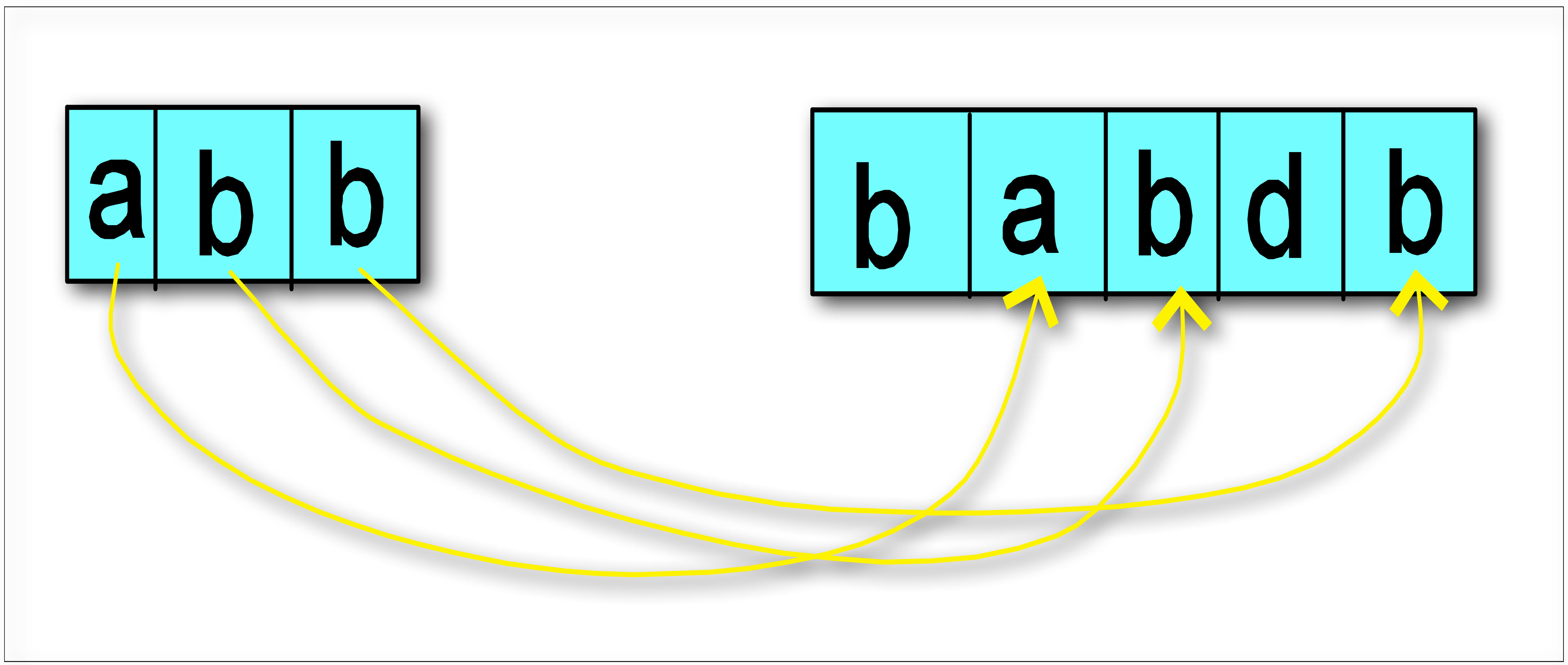
- A qo is a wqo if all sequences are good

- A sequence is bad if it is not good
- If a set is not good, then there is a minimal counterexample (bad sequence)

Higman's Lemma

- Every infinite sequence of words (over a finite alphabet) includes an embedding.

Homeomorphic



Higman's Lemma

- Suppose a finite or infinite alphabet is wqo
- Extend order to string embedding
  - letters map in order to bigger or equivalent ones
- Strings are wqo

Precedence

- Example, Σ

  $a_0<a_1<a_2<\ldots$

  $b_0<b_1<b_2<\ldots$

  …

  $z_0<z_1<\ldots$

Minimal Bad Sequence

- acd eef afda ...
- afda ab acd ...
- ab eef afda ...
- ab acd eef afda ...
- ab afda acd ...

Minimal Bad Sequence

- acd eef afda ...
- afda ab acd ...
- ab eef afda ...
- ab acd eef afda ...
- ab afda acd ...

Minimal Bad Sequence

- ab eef afda ...
- ab acd eef afda ...
- ab afda acd ...

Minimal Bad Sequence

- ab eef afda ...
- ab acd eef afda ...
- ab afda acd ...

Minimal Bad Sequence

- ab acd eef afda ...

Minimal Bad Sequence

- ab acd eef afda ...

Minimal Bad Sequence

- ab acd afda ...

Proof

- Consider minimal bad sequence
  - $\alpha_1 x_1\ \alpha_2 x_2\ \alpha_3 x_3\ \ldots\ \alpha_i x_i\ \ldots\ \alpha_j x_j\ \ldots$
- Extract subsequence with first letters $\alpha_{i1}\ \alpha_{i2}\ \alpha_{i3}\ \ldots$ ordered
- Consider rests $x_{i1}\ x_{i2}\ x_{i3}\ \ldots$

- Tails (or substrings) of minimal bad sequence are good
  - Why?
  - Suppose bad tails $x_9 \ldots x_3\, x_{18} \ldots$
  - Consider $x_3\, x_{18} \ldots$ (where 3 min index)
  - $\alpha_1 x_1\ \alpha_2 x_2\ x_3\, x_{18} \ldots$ would be smaller than

Contradiction

- ab acd   afda ...   aacafad ...

Corollary: Bag

- Given wfo > on elements X, consider bag order
- Extend (by Zorn's Lemma) to total well-order >; X is wqo by ≥
- By Higman, sequences X* are wqo
- Were there an infinite descending sequence {bi} of multisets wrt >, it would be decreasing wrt >
- By Higman, there's a pair bj ≤ bk; by bag order

Termination

6. Tree Orderings

Symbolic

- $Dt = 1$
- $Dc = 0$
- $D(x+y) = Dx + Dy$
- $D(xy) = xDy + yDx$
- ...

Exponential

- $[Dx] = 3^{[x]}$
- $[t] = [c] = 3$
- $[x+y] = \ldots = [xy] = [x] + [y]$

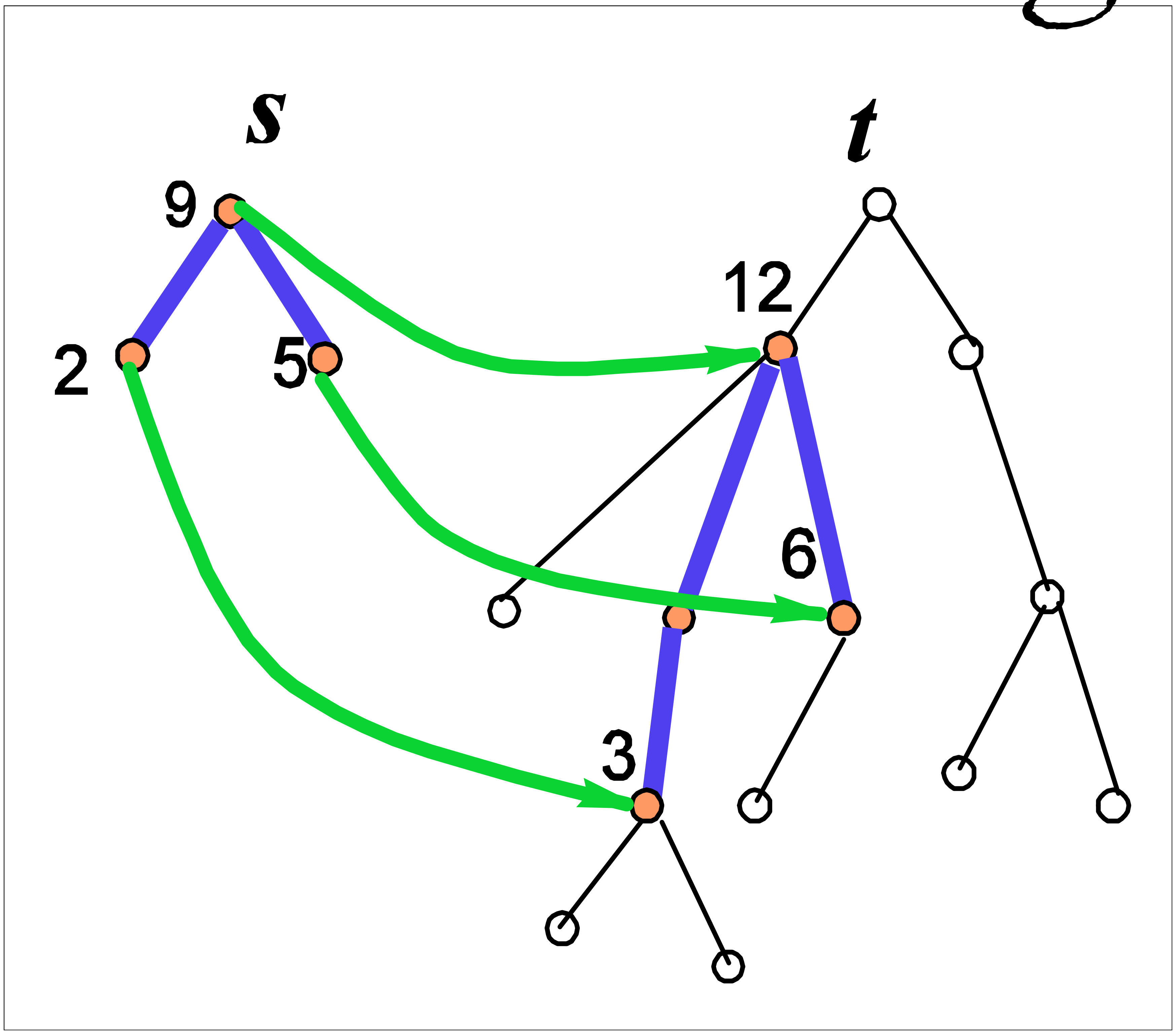
WQO

- Standard: wf and no inf antichain
- Simple: Every infinite sequence has an ordered pair
- Useful: Every infinite sequence contains an infinite non-decreasing chain
  - Why? -- Ramsey

Corollary

- Multiset ordering
- Bounded-arity tree ordering

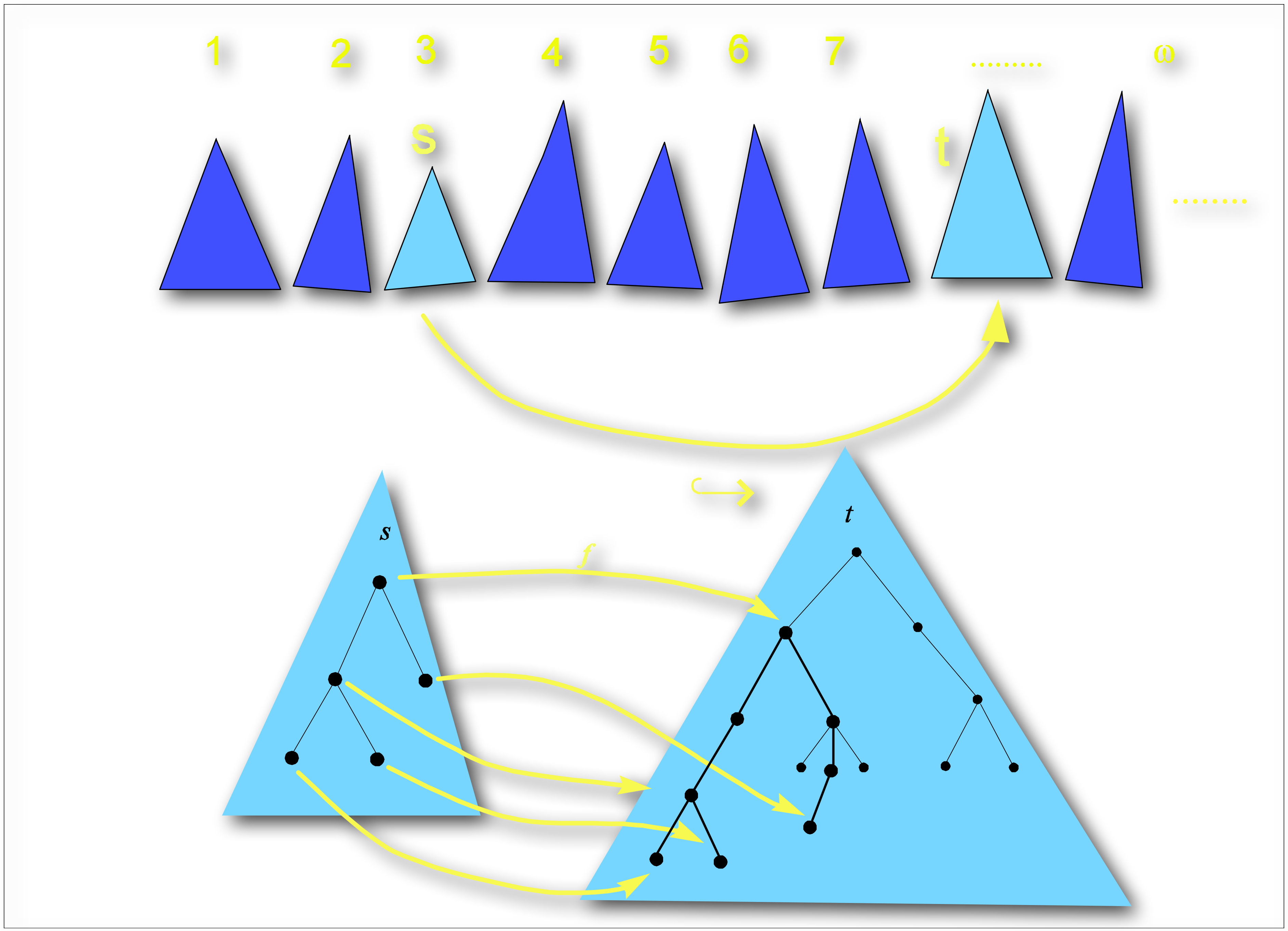
Tree Embedding

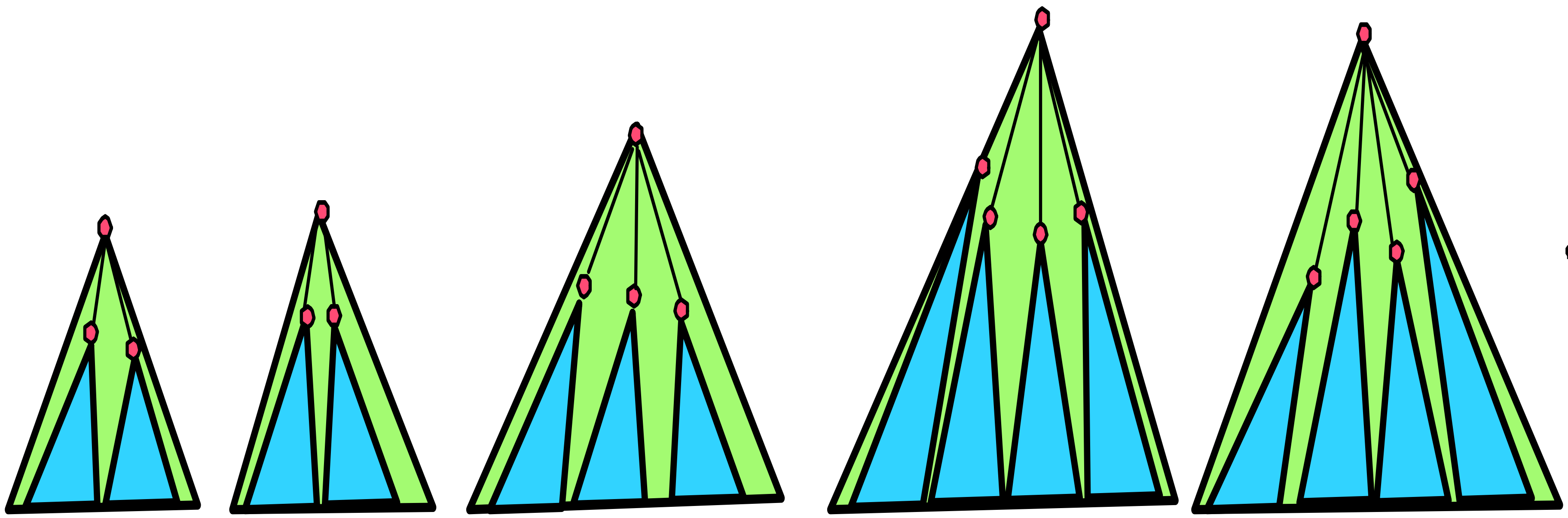


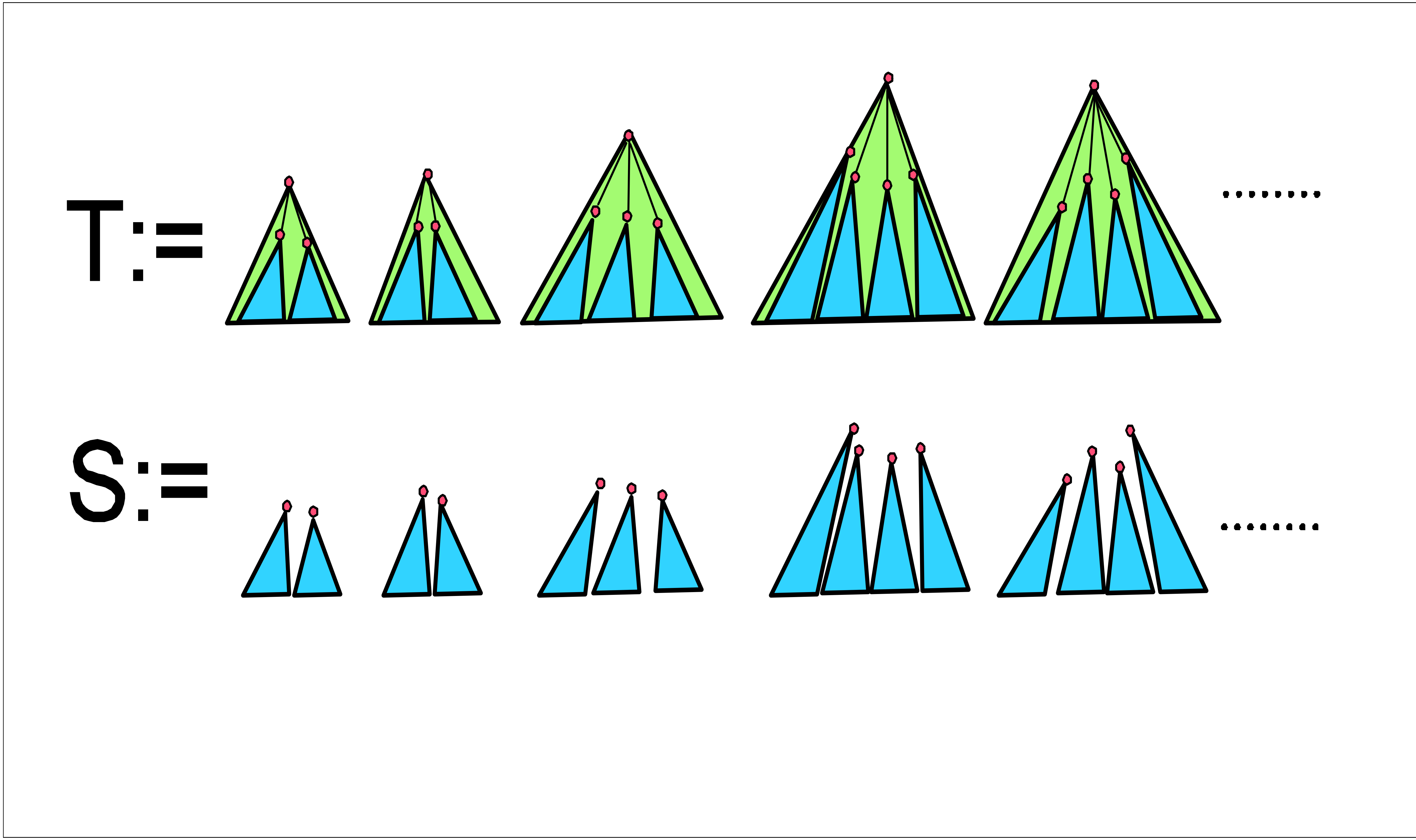
Kruskal's Tree Theorem

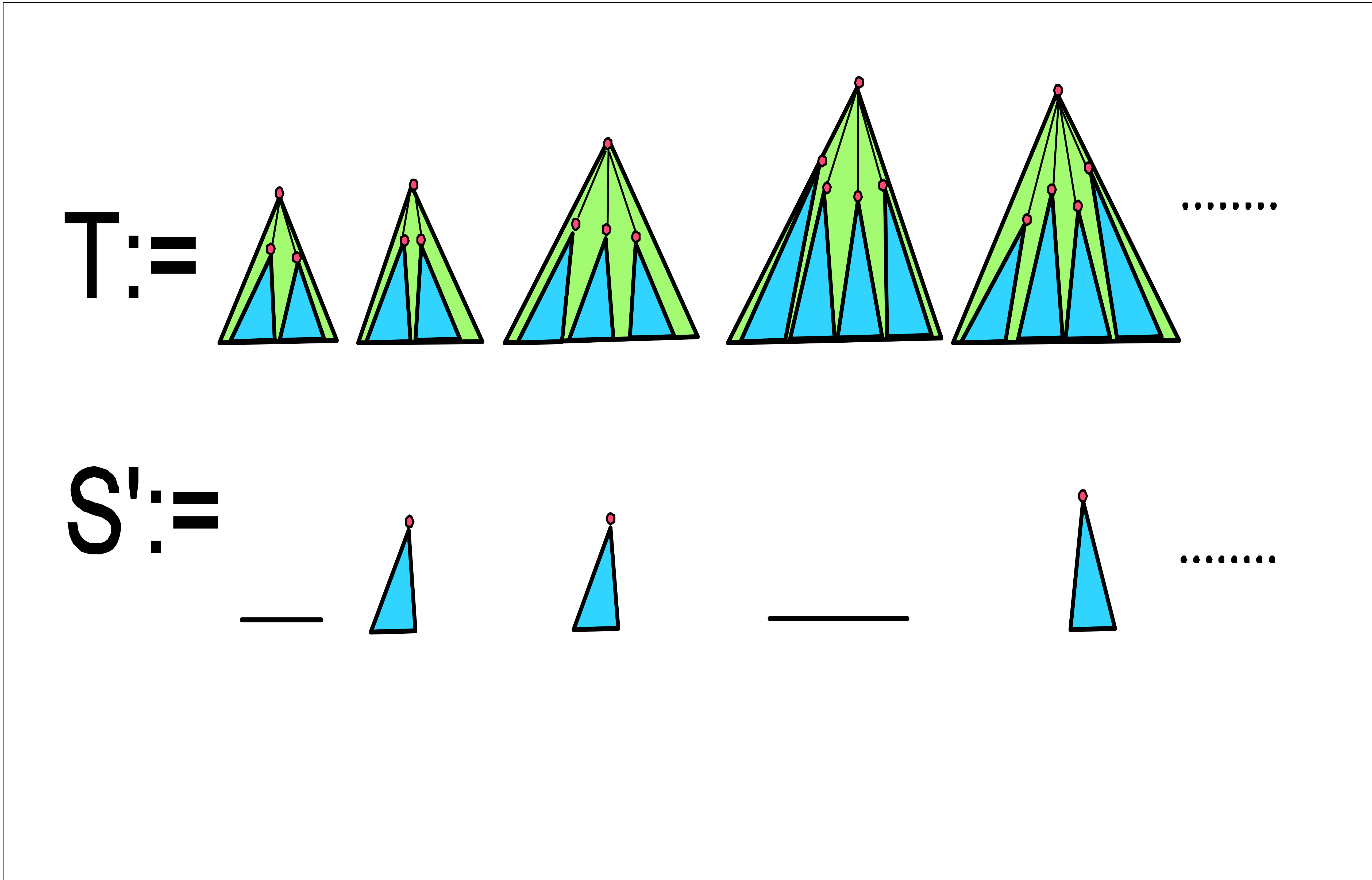
- Every infinite sequence of trees (over a wqo alphabet) includes an embedding.

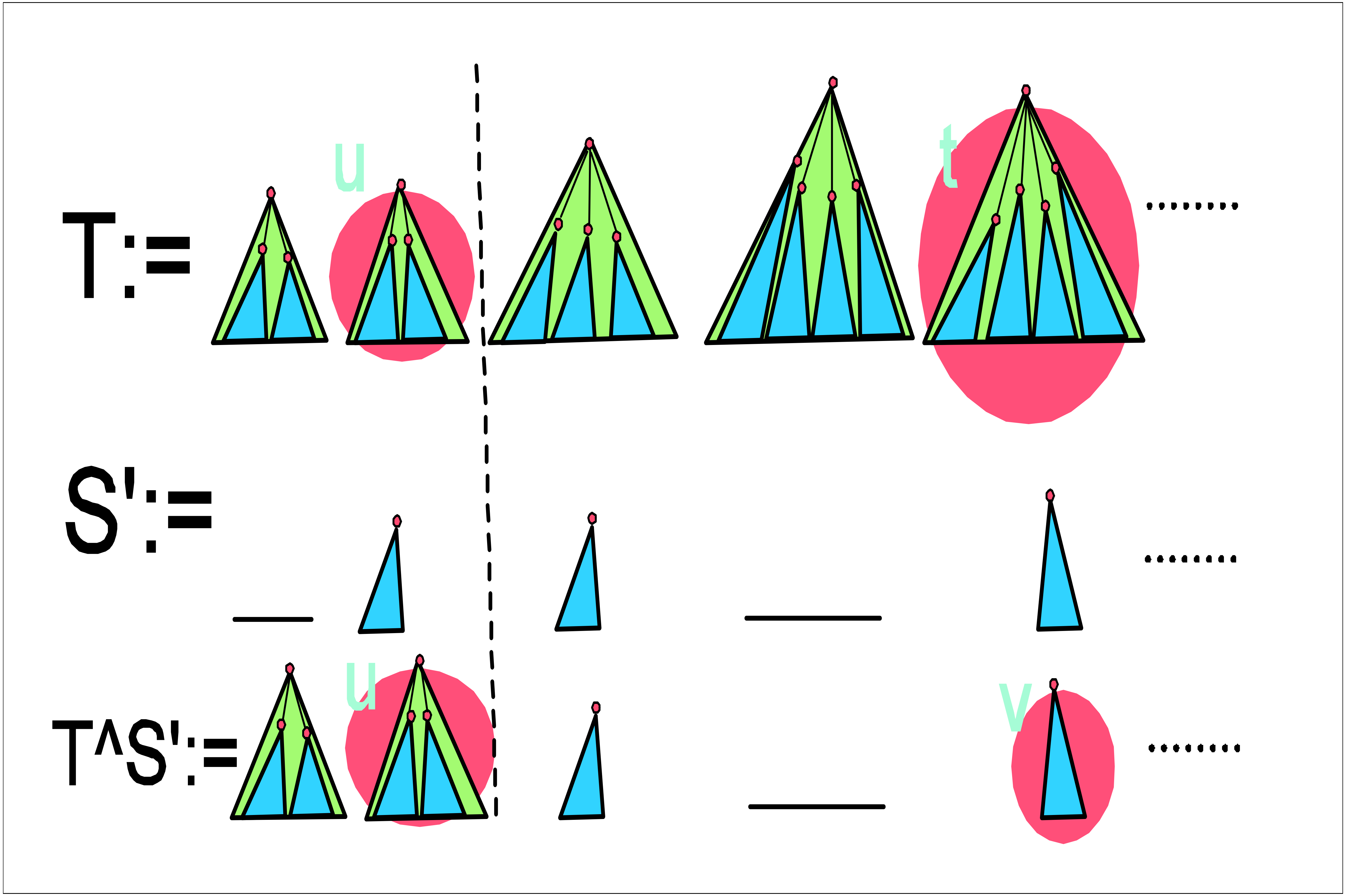
Good Sequence



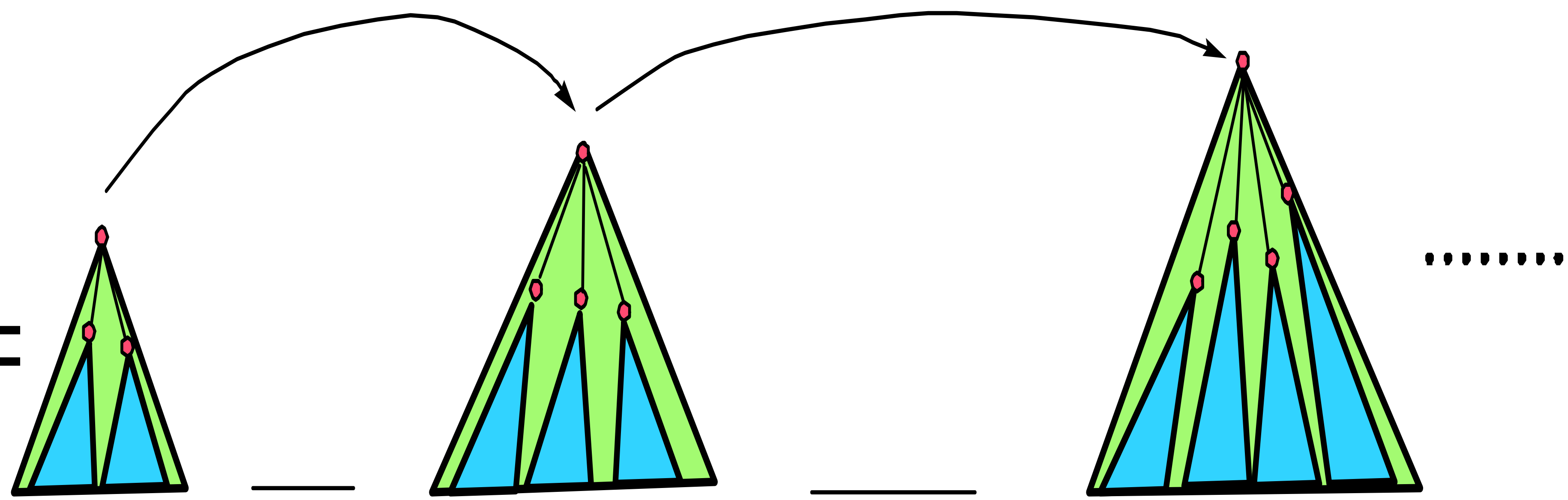
T:=  ....

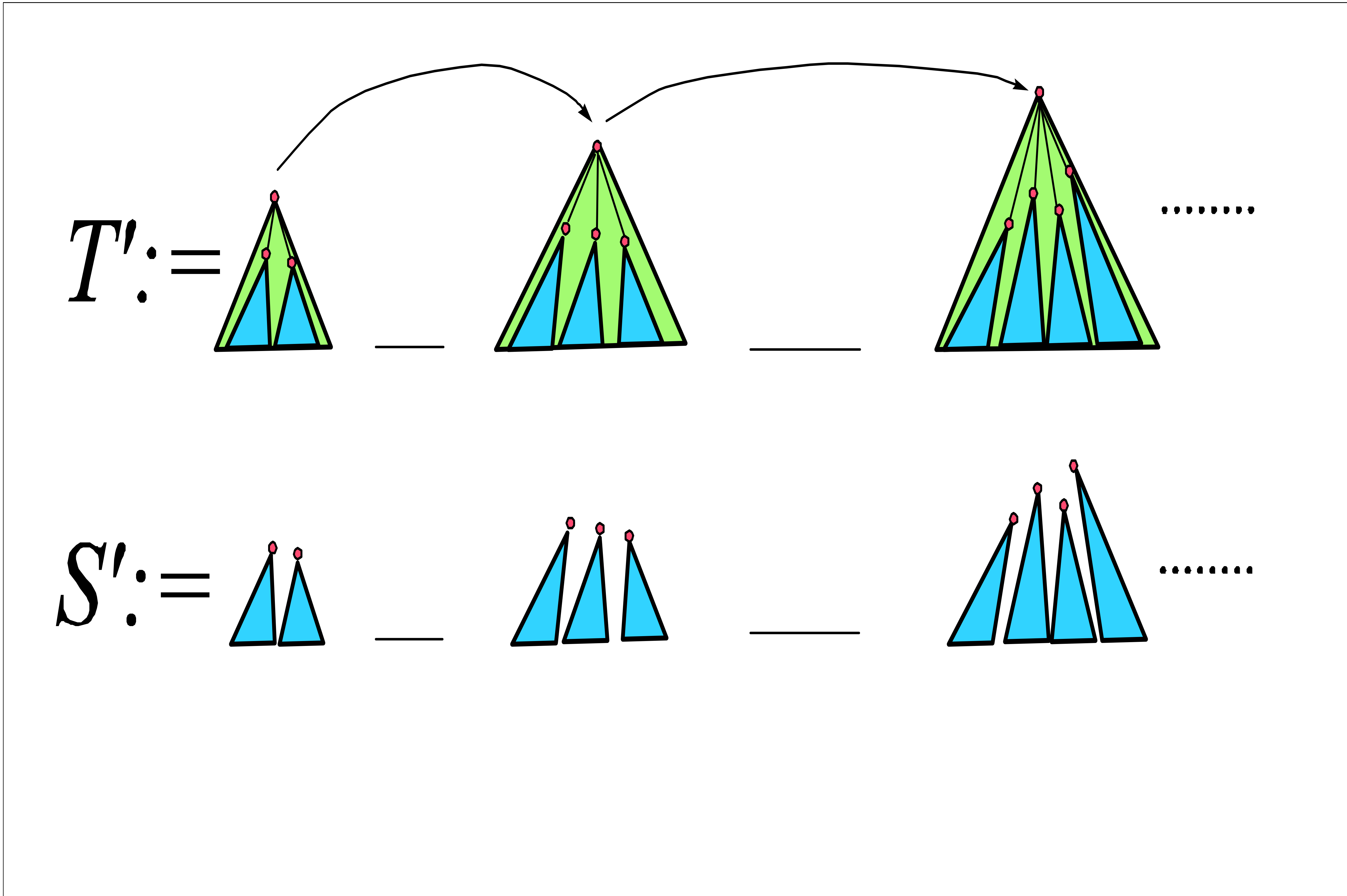

T:=
S:=


T:=
S':=

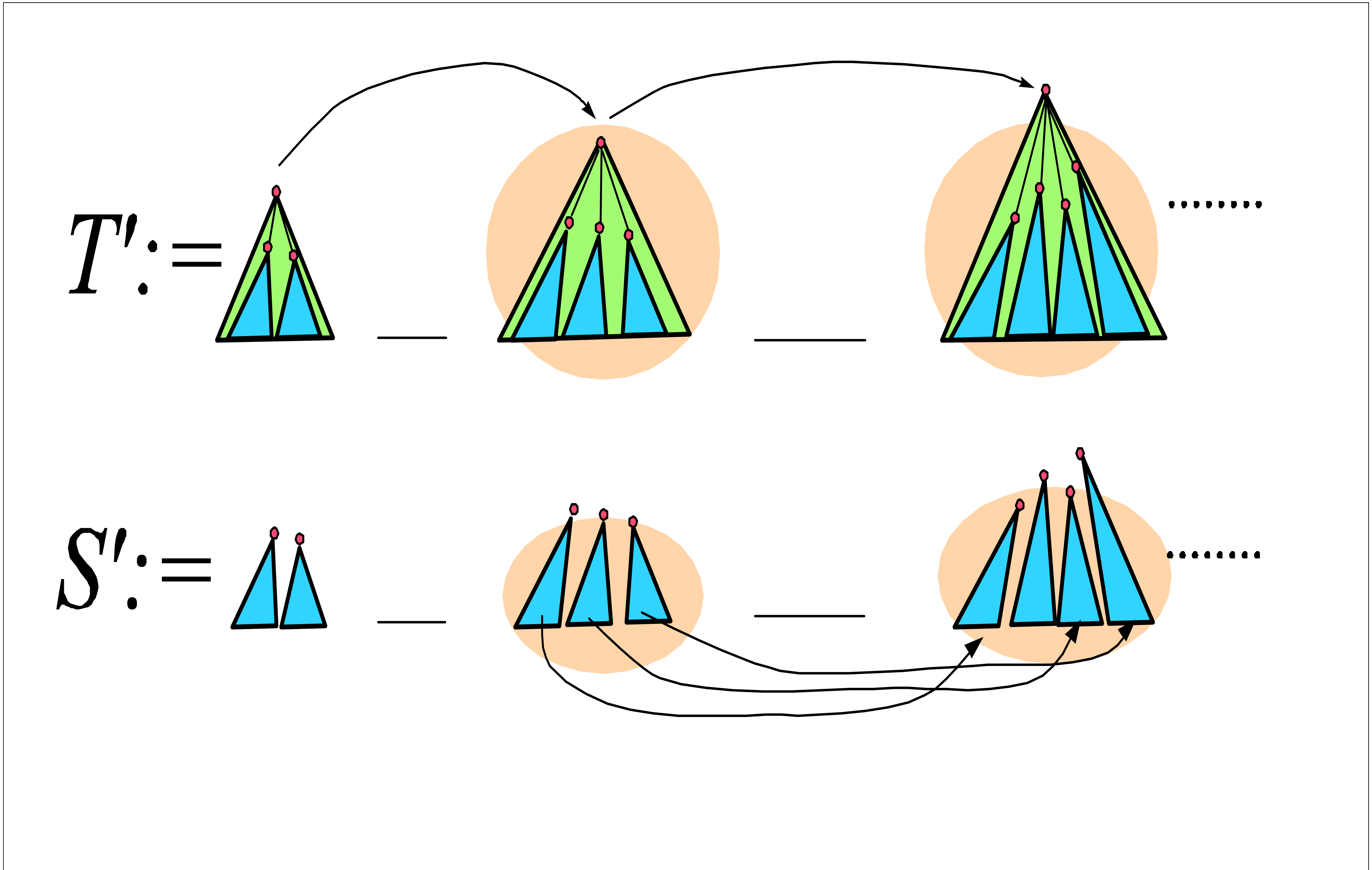


T:=
u
t
S':=
T^S':=
u
v

$T' :=$ 

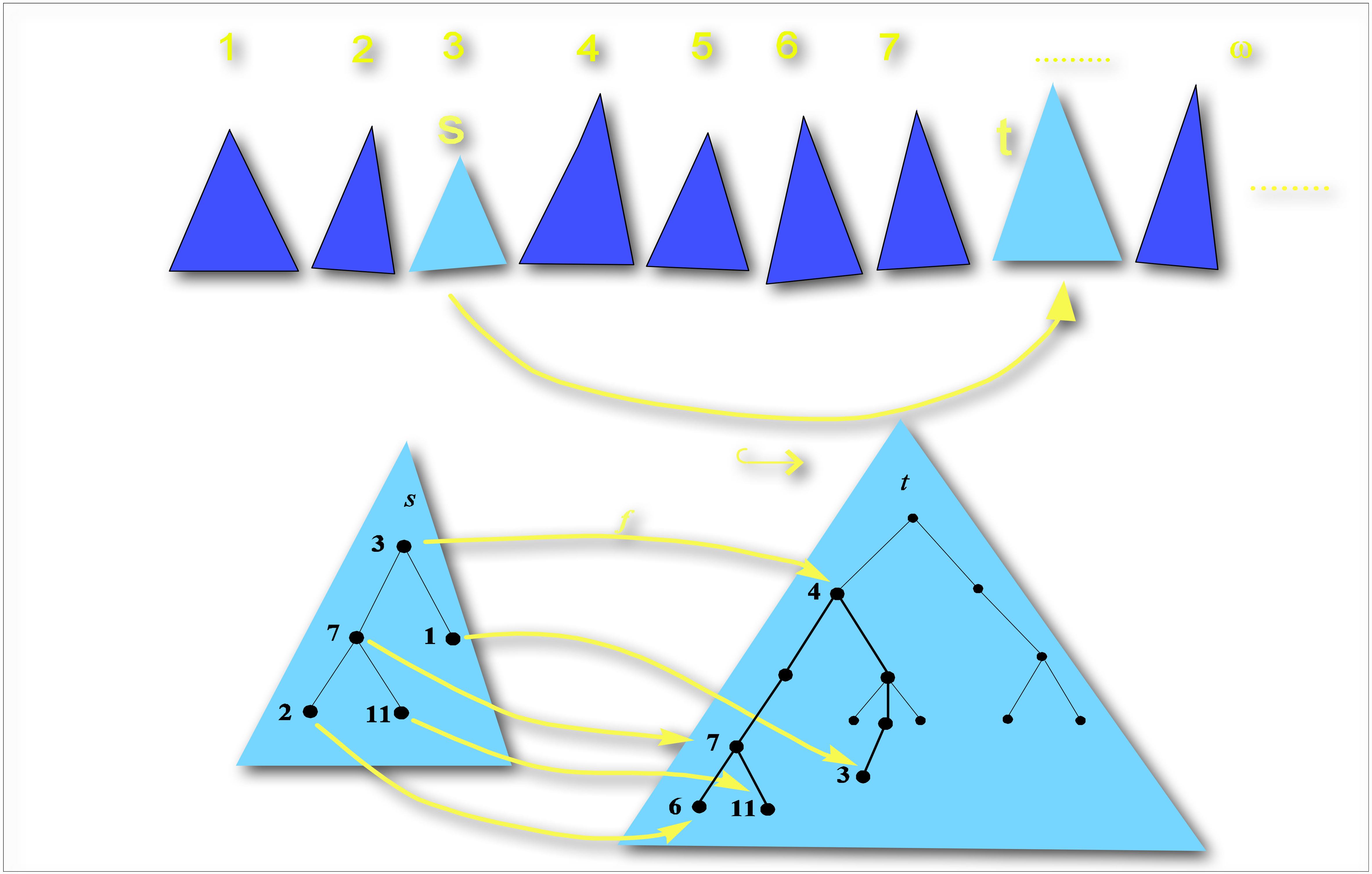


$T':=$
$S':=$

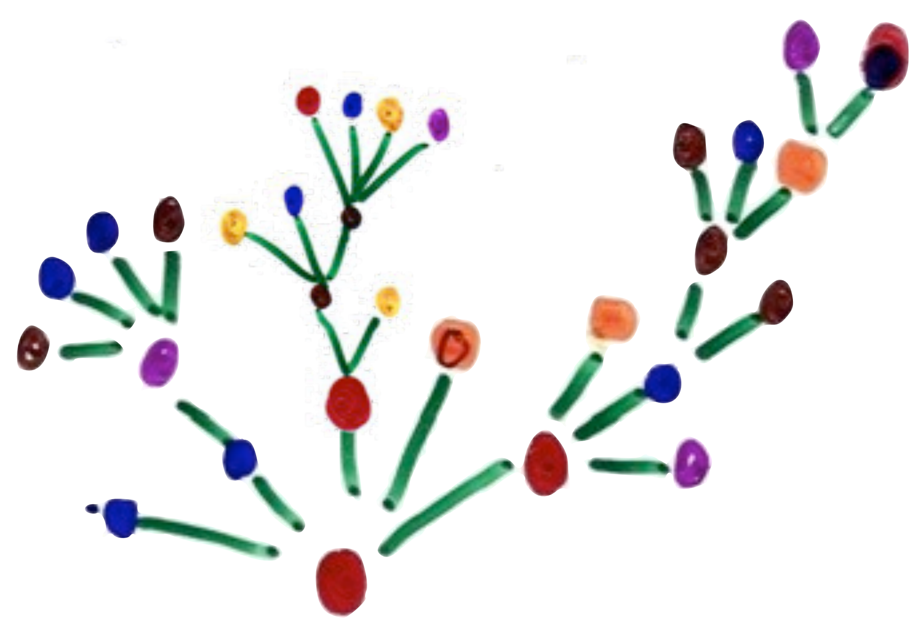


T':=
S':=

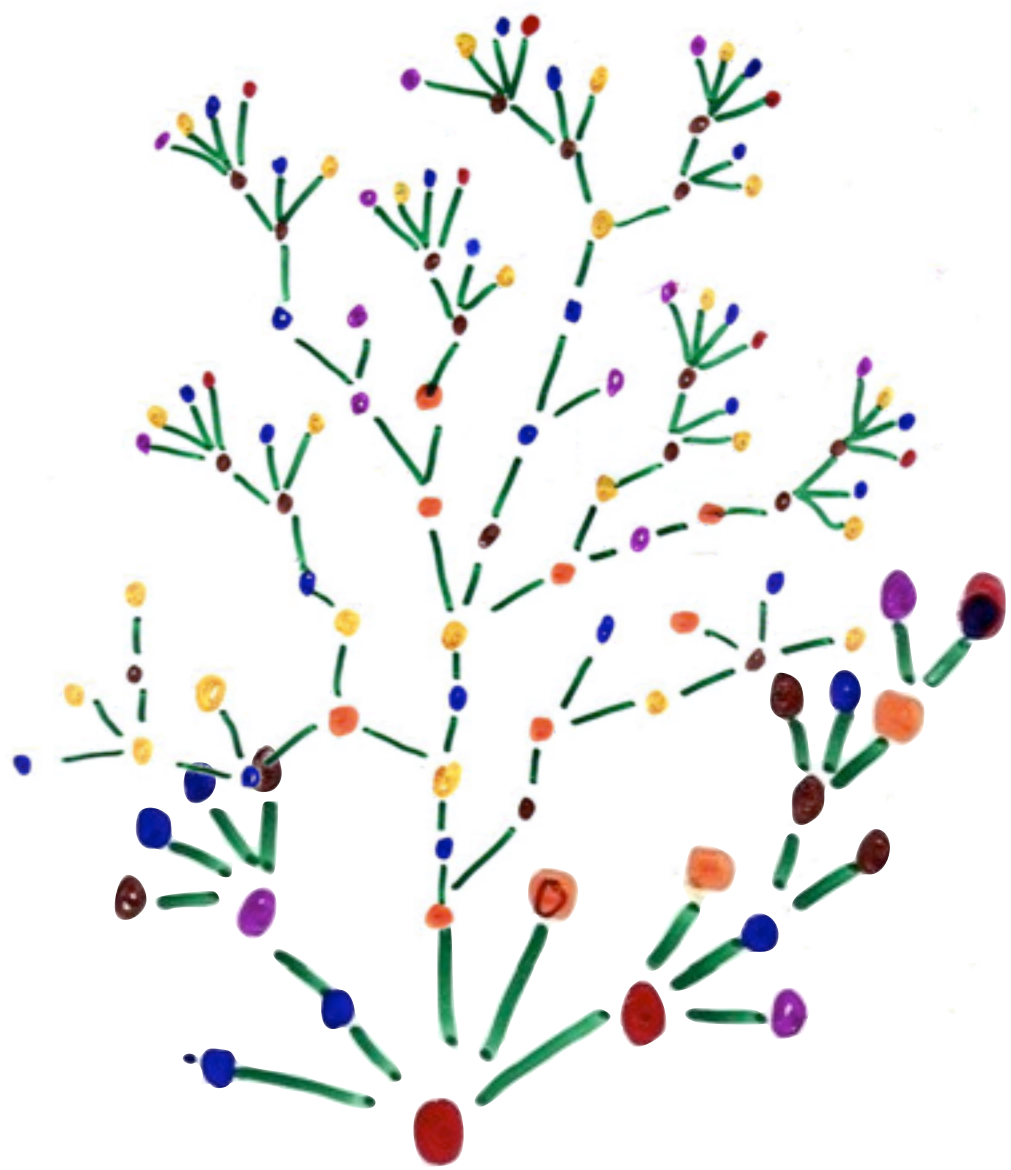
Labels



Gremlins



Gremlins



Multiset Path Order

- $s = f(s_1,\ldots,s_m) \quad t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(f,\{s_1,\ldots,s_m\}) >_{lex} (g,\{t_1,\ldots,t_n\})$
  - and $s > t_j$ for all j

Symbolic

- $Dt = 1$
- $Dc = 0$
- $D(x+y) = Dx + Dy$
- $D(xy) = xDy + yDx$
- ...

Distributivity

- $x(y+z) = xy + xz$

DNF

- $\neg\neg x = x$
- $\neg(x \vee y) = (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) = (\neg x) \vee (\neg y)$
- $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x = (y \wedge x) \vee (z \wedge x)$

Simplification Order

- $f(\ldots, s_i, \ldots) > s_i$
- $s_i > t_i \Rightarrow f(\ldots, s_i, \ldots) > f(\ldots, t_i, \ldots)$
- Finite alphabet

Simplification Order

- $f(\ldots,s_i,\ldots) > s_i$
- $s_i > t_i \Rightarrow f(\ldots,s_i,\ldots) > f(\ldots,t_i,\ldots)$
- $f > g \Rightarrow f(\ldots,s_i,\ldots) > g(\ldots,s_i,\ldots)$

Lexicographic Path

- $s = f(s_1,\ldots,s_m) \quad t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(f, s_1,\ldots,s_m) >_{lex} (g, t_1,\ldots,t_n)$
  - and $s > t_j$ for all j

Recursive Path Order

- $s = f(s_1,\ldots,s_m) \quad t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(f, s_1, \ldots, \{s_i, \ldots, s_m\}) >_{lex} (g, t_1, \ldots, \{t_i, \ldots, t_n\})$
  - and $s > t_j$ for all j

Weak Simplification Order

- $f(\ldots,s_i,\ldots) \gtrsim s_i$
- $s_i \gtrsim t_i \Rightarrow f(\ldots,s_i,\ldots) \gtrsim f(\ldots,t_i,\ldots)$

Simplification Ordering

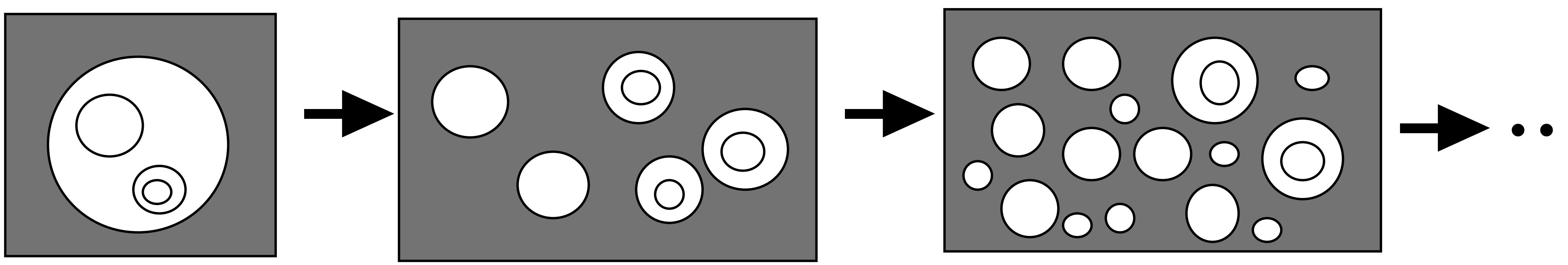
- (Weakly) Monotonic
- (Weakly) Subterm

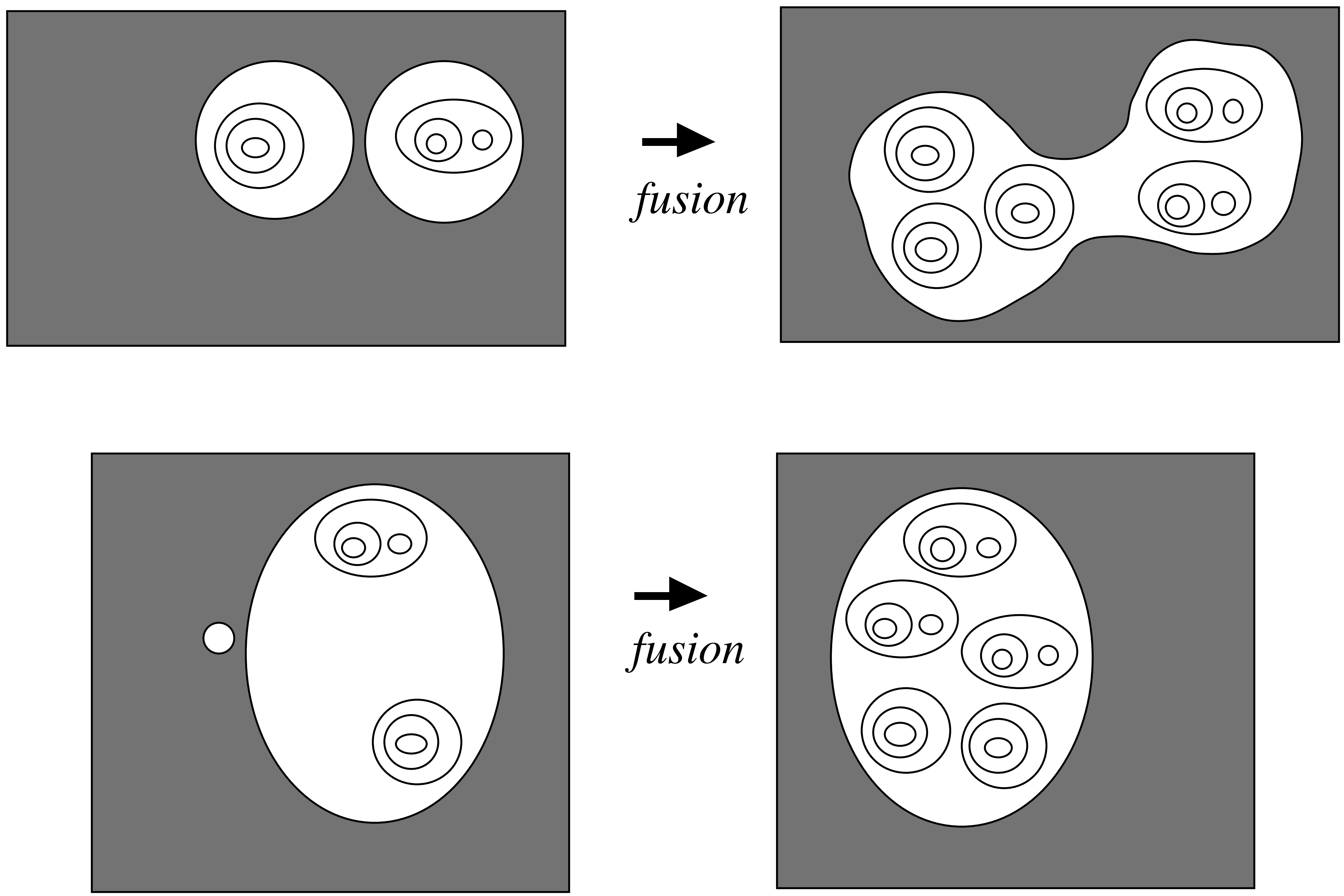
- They are well-quasi-orders

Termination

7. Rewriting

Fission




fusion
fusion

Better

- d(a) = depth(a)
- { {d(a) : a in A} : colony A }
- fission: depth decreases
- fusion: one deep item removed

DNFO

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

DNF1

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y)$
- $x \wedge (y \wedge z) \Rightarrow (x \wedge y) \wedge z$
- $x \vee (y \vee z) \Rightarrow (x \vee y) \vee z$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

DNF2

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y)$
- $(x \wedge y) \wedge z \Rightarrow x \wedge (y \wedge z)$
- $x \vee (y \vee z) \Rightarrow (x \vee y) \vee z$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

DNF3

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg\neg\neg x) \wedge (\neg\neg\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg\neg\neg x) \vee (\neg\neg\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

DNF3

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg\neg\neg x) \wedge (\neg\neg\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg\neg\neg x) \vee (\neg\neg\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

$\neg\neg(a \wedge (b \vee c))$

DNF4

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z) \vee (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (x \wedge y) \vee (x \wedge z) \vee (x \wedge y) \vee (x \wedge z)$
- $x \vee x \Rightarrow x$

DNF5

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y) \wedge (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y) \vee (\neg x) \vee (\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $x \vee x \Rightarrow x$
- $x \wedge x \Rightarrow x$

DNF6

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg\neg\neg x) \wedge (\neg\neg\neg y) \wedge (\neg\neg\neg x) \wedge (\neg\neg\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg\neg\neg x) \vee (\neg\neg\neg y) \vee (\neg\neg\neg x) \vee (\neg\neg\neg y)$
- $x \vee x \Rightarrow x$
- $x \wedge x \Rightarrow x$

DNF7

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y) \wedge (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y) \vee (\neg x) \vee (\neg y)$
- $x \vee x \Rightarrow x$
- $x \wedge x \Rightarrow x$

Symbolic Computation

- $Dt = 1$
- $Dc = 0$
- $D(x+y) = Dx + Dy$
- $D(xy) = xDy + yDx$
- …

Rewriting

- $Dt \Rightarrow 1$
- $Dc \Rightarrow 0$
- $D(x+y) \Rightarrow Dx + Dy$
- $D(xy) \Rightarrow xDy + yDx$
- …

Factorial

- $x+0 \Rightarrow x$
- $x+s(y) \Rightarrow s(x+y)$
- $x*0 \Rightarrow 0$
- $x*s(y) \Rightarrow y + x*y$
- $f(0) \Rightarrow s(0)$
- $f(s(x)) \Rightarrow s(x)*f(x)$
-

Factorial

- $x+0 \Rightarrow x$
- $x+s(y) \Rightarrow s(x+y)$
- $x*0 \Rightarrow 0$
- $x*s(y) \Rightarrow y + x*y$
- $f(0) \Rightarrow s(0)$
- $f(s(x)) \Rightarrow s(x)*f(p(s(x)))$
- $p(s(x)) \Rightarrow x$

Termination

- If s[x] ⇨ t[x] is a rule
- then c[s[v]] ⇨ c[t[v]] is a rewrite
- Want c[s[v]] > c[t[v]] in some wfo
- Want monotonicity
  - $s > t \Rightarrow f(\ldots,s,\ldots) > f(\ldots,t,\ldots)$

Exponential Interpretation

- $[Dx] = 3^{[x]}$
- $[t] = [c] = 3$
- $[x+y] = \ldots = [xy] = [x] + [y]$

Polynomial Interpretation

- $[Dx] = [x]^2$
- $[x+y] = \ldots = [xy] = [x] + [y]$
- Eventually positive
  - $x^2 + y^2 + 2xy - x^2 - y^2 - x - y = 2xy - x - y$
  - Derivatives: $2x-1$; $2y-1$

Multiset Path Order

- $s = f(s_1,\ldots,s_m) \quad t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(f,\{s_1,\ldots,s_m\}) >_{lex} (g,\{t_1,\ldots,t_n\})$
  - and $s > t_j$ for all j

Lexicographic Path Order

- $s = f(s_1,\ldots,s_m)$ $t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(f,s_1,\ldots,s_m) >_{lex} (g,t_1,\ldots,t_n)$
  - and $s > t_j$ for all j

Boyer & Moore

- $\mathrm{if}(\mathrm{if}(x,y,z),u,v) \Rightarrow \mathrm{if}(x,\mathrm{if}(y,u,v),\mathrm{if}(z,u,v))$

Recursive Path Order

- $s = f(s_1,\ldots,s_m) \quad t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(f, s_1, \ldots, \{s_i, \ldots, s_m\}) >_{lex} (g, t_1, \ldots, \{t_i, \ldots, t_n\})$
  - and $s > t_j$ for all j

Simplification Order

- Suppose finite vocabulary
- Subterm: $f(\ldots,s,\ldots) > s$
- Monotonic: $s > t \Rightarrow f(\ldots,s,\ldots) > f(\ldots,t,\ldots)$
- Must be well-founded

Weak Simplification Order

- Weak subterm: $f(\ldots,s_i,\ldots) \gtrsim s_i$
- Weak monotonicity:
  $s_i \gtrsim t_i \Rightarrow f(\ldots,s_i,\ldots) \gtrsim f(\ldots,t_i,\ldots)$
- Well-quasi-order by Kruskal
- Enough for termination of rewriting
  - Why?

Total Order

- Suppose finite vocabulary
- Monotonic: $s > t \Rightarrow f(\ldots,s,\ldots) > f(\ldots,t,\ldots)$
- Well-founded iff subterm

Semantic Path Order

- $s = f(s_1,\ldots,s_m)$ $\quad t = g(t_1,\ldots,t_n)$ $\quad >$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if
  - $(s, s_1,\ldots,s_m) >_{lex} (t, t_1,\ldots,t_n)$
  - and $s > t_j$ for all j
- require $s \Rightarrow t \Rightarrow f(\ldots s \ldots) \geq f(\ldots t \ldots)$

Proof

- Extend base order to a total w.f. order
- Consider minimal bad sequence
- Subterms are well-founded
- No use of $s_i \gtrsim t$ case
- So base order decreases and stabilizes

Termination

8. Semantic Path Order

DNF3

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg\neg\neg x) \wedge (\neg\neg\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg\neg\neg x) \vee (\neg\neg\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

$\neg\neg(a \wedge (b \vee c))$

DNF4

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg x) \wedge (\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg x) \vee (\neg y)$
- $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z) \vee (x \wedge y) \vee (x \wedge z)$
- $(y \vee z) \wedge x \Rightarrow (x \wedge y) \vee (x \wedge z) \vee (x \wedge y) \vee (x \wedge z)$
- $x \vee x \Rightarrow x$

DNF5

- $\neg\neg x \Rightarrow x$
- $\neg(x \lor y) \Rightarrow (\neg x) \land (\neg y) \land (\neg x) \land (\neg y)$
- $\neg(x \land y) \Rightarrow (\neg x) \lor (\neg y) \lor (\neg x) \lor (\neg y)$
- $x \land (y \lor z) \Rightarrow (x \land y) \lor (x \land z)$
- $(y \lor z) \land x \Rightarrow (x \land y) \lor (x \land z)$
- $x \lor x \Rightarrow x$
- $x \land x \Rightarrow x$

DNF6

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg\neg\neg x) \wedge (\neg\neg\neg y) \wedge (\neg\neg\neg x) \wedge (\neg\neg\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg\neg\neg x) \vee (\neg\neg\neg y) \vee (\neg\neg\neg x) \vee (\neg\neg\neg y)$
- $x \vee x \Rightarrow x$
- $x \wedge x \Rightarrow x$

DNF6

- $\neg\neg x \Rightarrow x$
- $\neg(x \vee y) \Rightarrow (\neg\neg\neg x) \wedge (\neg\neg\neg y) \wedge (\neg\neg\neg x) \wedge (\neg\neg\neg y)$
- $\neg(x \wedge y) \Rightarrow (\neg\neg\neg x) \vee (\neg\neg\neg y) \vee (\neg\neg\neg x) \vee (\neg\neg\neg y)$
- $x \vee x \Rightarrow x$
- $x \wedge x \Rightarrow x$

$\neg\neg(x \vee y)$

Labeling

- ffx ⇨ fgfx
- ffx ⇨ fgfx
- fffx ⇨ ffgfx

Semantic Path Order

- Given a well-founded term order $\gtrsim$
- $s = f(s_1,\ldots,s_m) \quad t = g(t_1,\ldots,t_n)$
- $s > t$ if $s_i \gtrsim t$ for some i
- $s > t$ if $(s,s_1,\ldots,s_m) >_{lex} (t,t_1,\ldots,t_n)$
  - and $s > t_j$ for all j
- $s \approx t$ iff $(s,s_1,\ldots,s_m) \approx (t,t_1,\ldots,t_n)$

Semantic Path Order

- require $s \Rightarrow t \Rightarrow f(\ldots s \ldots) \geq f(\ldots t \ldots)$

Proof

- ~~Extend base order to a total w.f. order~~
- Consider minimal bad sequence
- Subterms are well-founded
- No use of $s_i \gtrsim t$ case
- So base order decreases and stabilizes

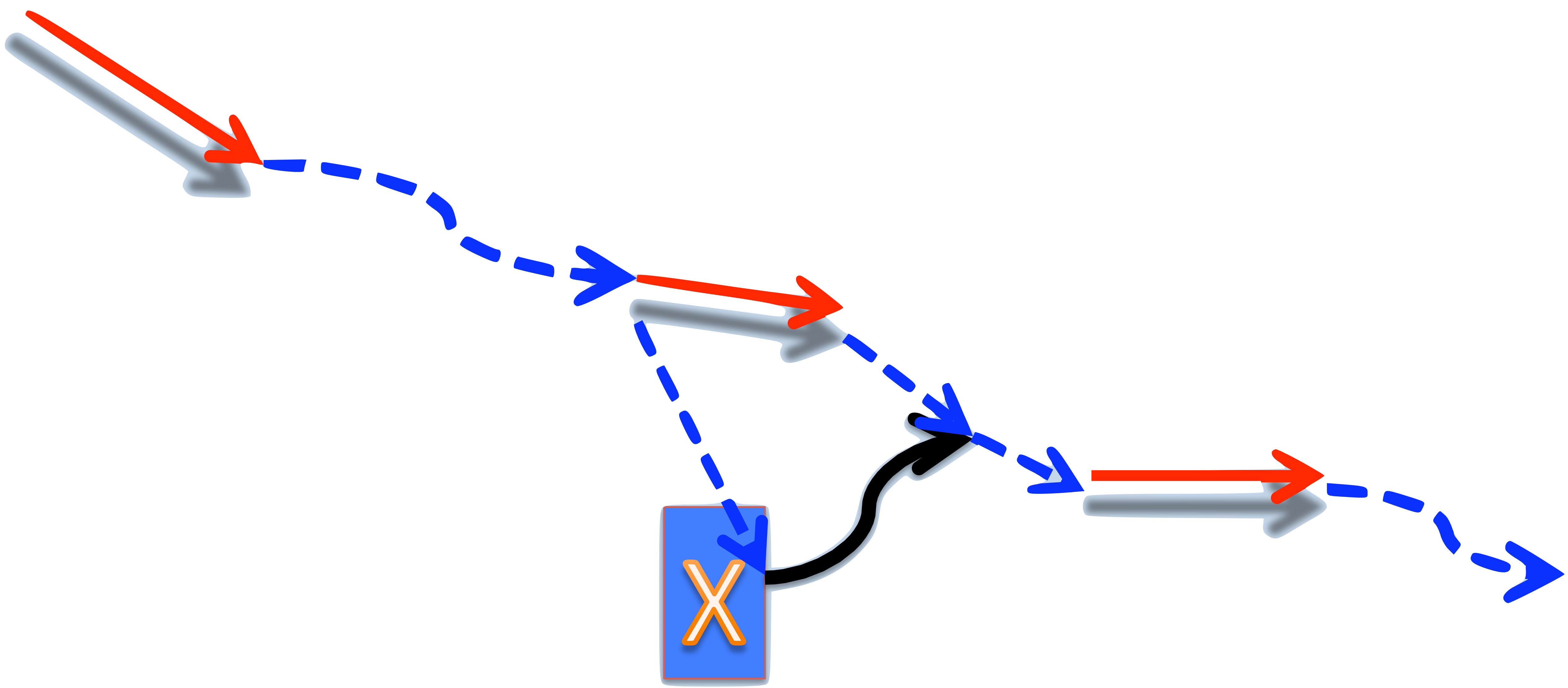
Jumping

- Let $P = R \cup B$
- If $s\ R \cup B\ t$
  - then $s\ R\ t$
  - or $s\ B\ v_1\ P\ v_2\ P \ldots P\ v_n\ P\ t$
- In short $RB \subseteq R \cup BP^*$
- Hence (induction) $RB^* \subseteq R \cup BP^*$

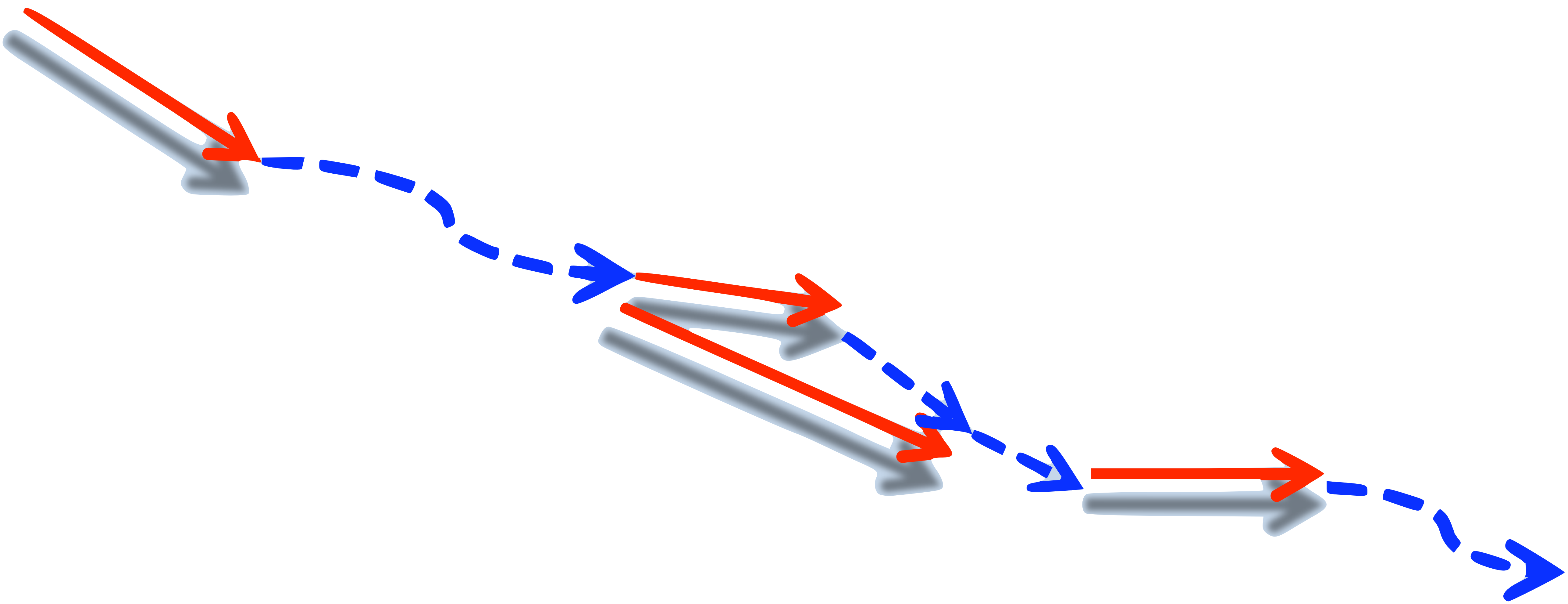
Constricting

- Let $P = R \cup B$
- If there is an immortal purple chain
  $s_1 \; P \; s_2 \; P \; s_3 \; P \ldots$
- then there is an immortal constricting chain
  $s_1 \; BB\ldots B \; t_1 \; R \; u_1 \; BB\ldots B \; t_2 \; R \ldots$
  - R only when "necessary"
  - if $t_i \; B \; v$, then v is mortal

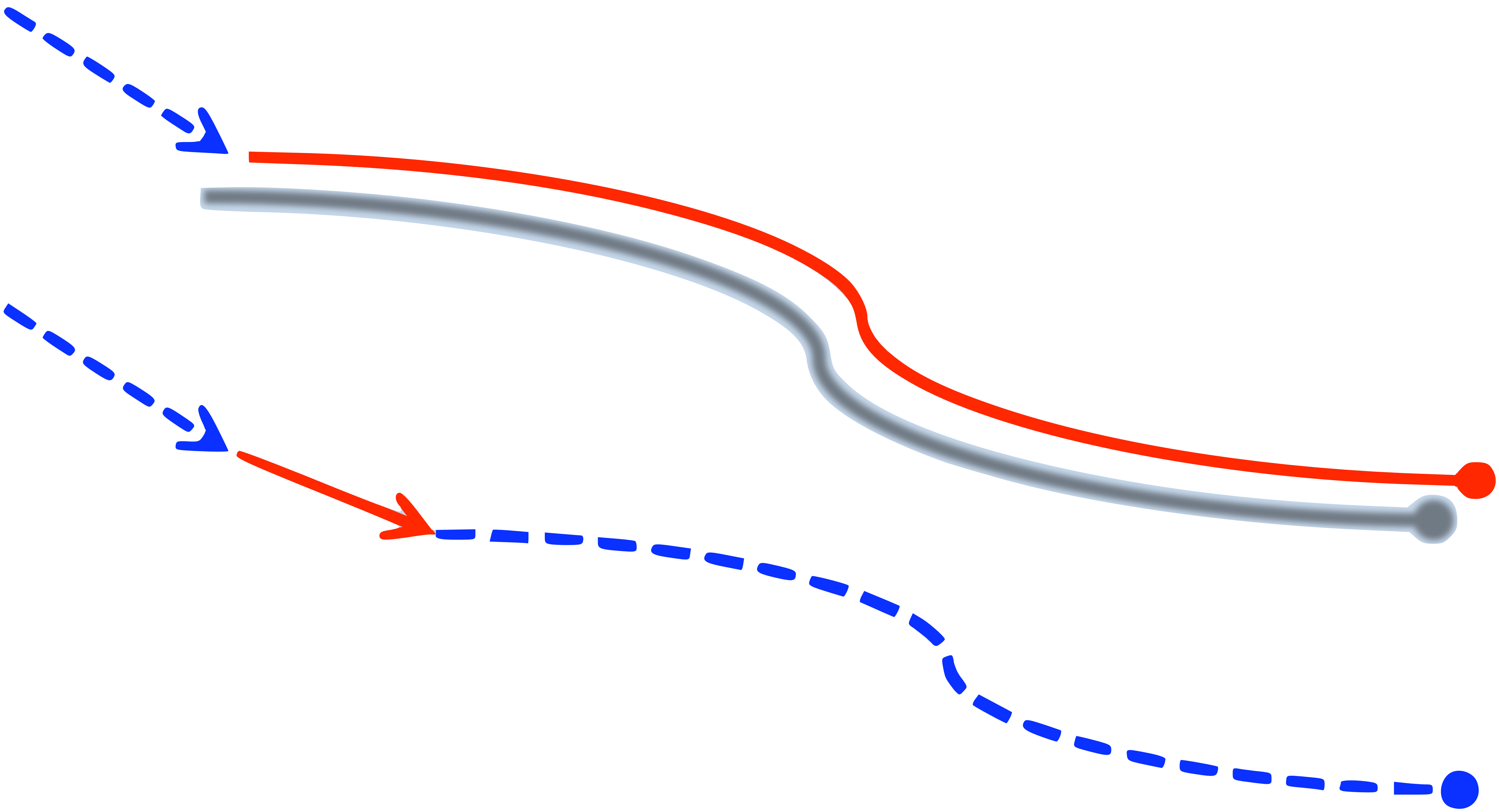
Constriction + Jumping



Constriction + Jumping



Constriction + Jumping



Constricted Jumping

- Constricted s_1 BB...B t_2 $\underline{R}$ t_3 BB...B t_4 $\underline{R}$...
- Jumping RB* ⊆ R ∪ BP*
- <u>Jumping</u> $\underline{R}$B* ⊆ $\underline{R}$
- s_1 BB...B t_2 $\underline{R}$ t_4 $\underline{R}$...

Jumping Union

- If B jumps over R
- then union well-founded iff both are
- $s_1\ BB\ldots B\ t_1\ \underline{R}B^*\ t_2\ \underline{R}B^*\ t_2\ \underline{R}B^*\ \ldots$
    - $s_1\ BB\ldots B\ t_1\ \underline{R}\ t_2\ \underline{R}\ t_3\ \underline{R}\ \ldots$
- $s_1\ BB\ldots B\ t_1\ \underline{R}B^*\ t_2\ \underline{R}B^*\ t_2\ \underline{R}BBBB\ldots$
    - $s_1\ BB\ldots B\ t_1\ \underline{R}\ \underline{R}\ \underline{R}\ u_k\ BBBB\ldots$

Lifting

- For any immortal red chain
  s_1 R s_2 R s_3 R...
- there is also an immortal purple chain after taking an immediate blue turn
  s_1 B t_1 P t_2 P...
- Example: R is multiset; B is subset

Lifting Union

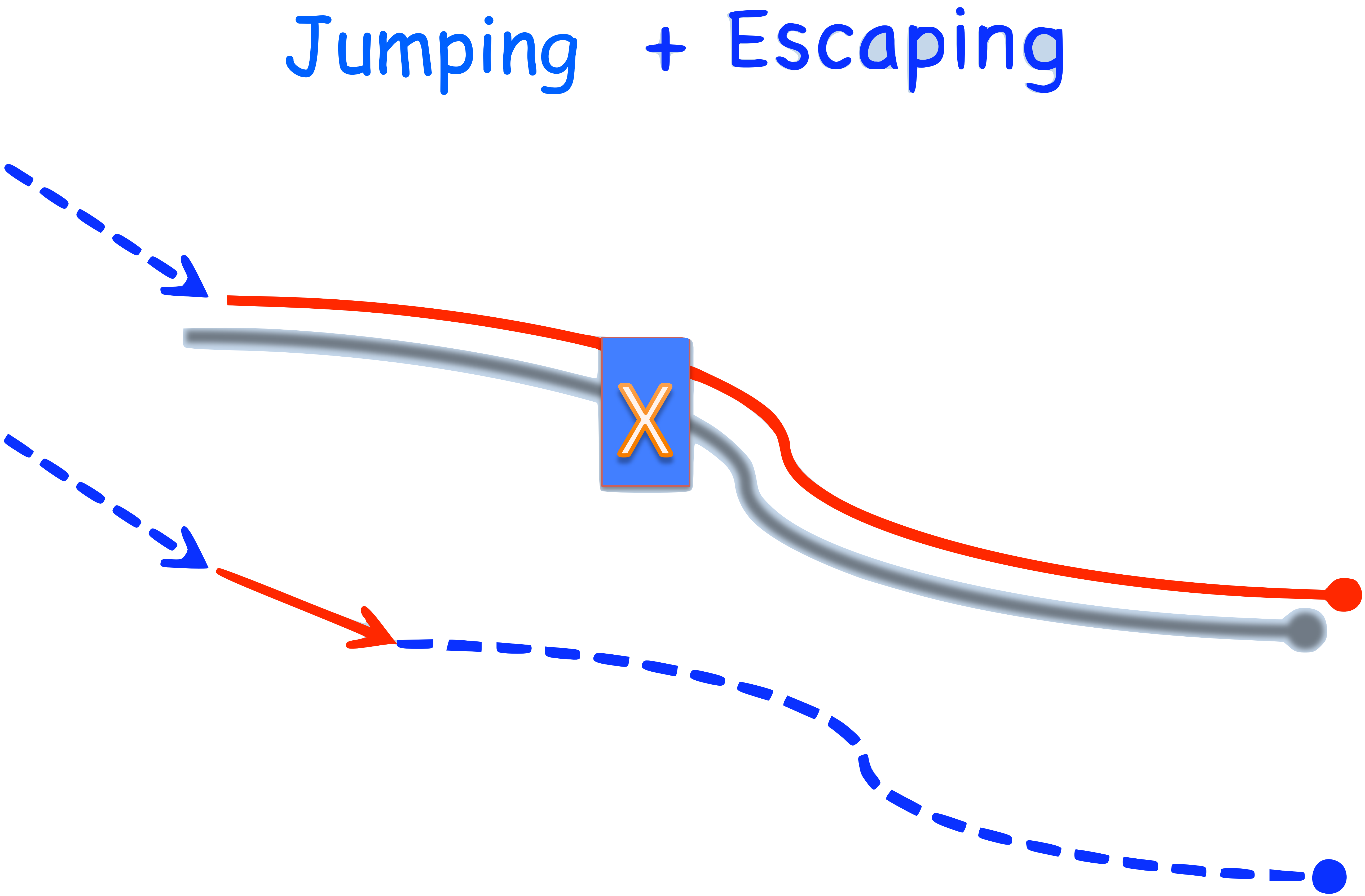
- If B jumps over R
- and B lifts to R
- then union well-founded iff B is
  - s_1 BB…B t_1 <u>R</u> t_2 <u>R</u> t_3 <u>R</u> … XXX
  - s_1 BB…B t_1 <u>R</u> <u>R</u> <u>R</u> u_k BBBB…

Nested Multisets

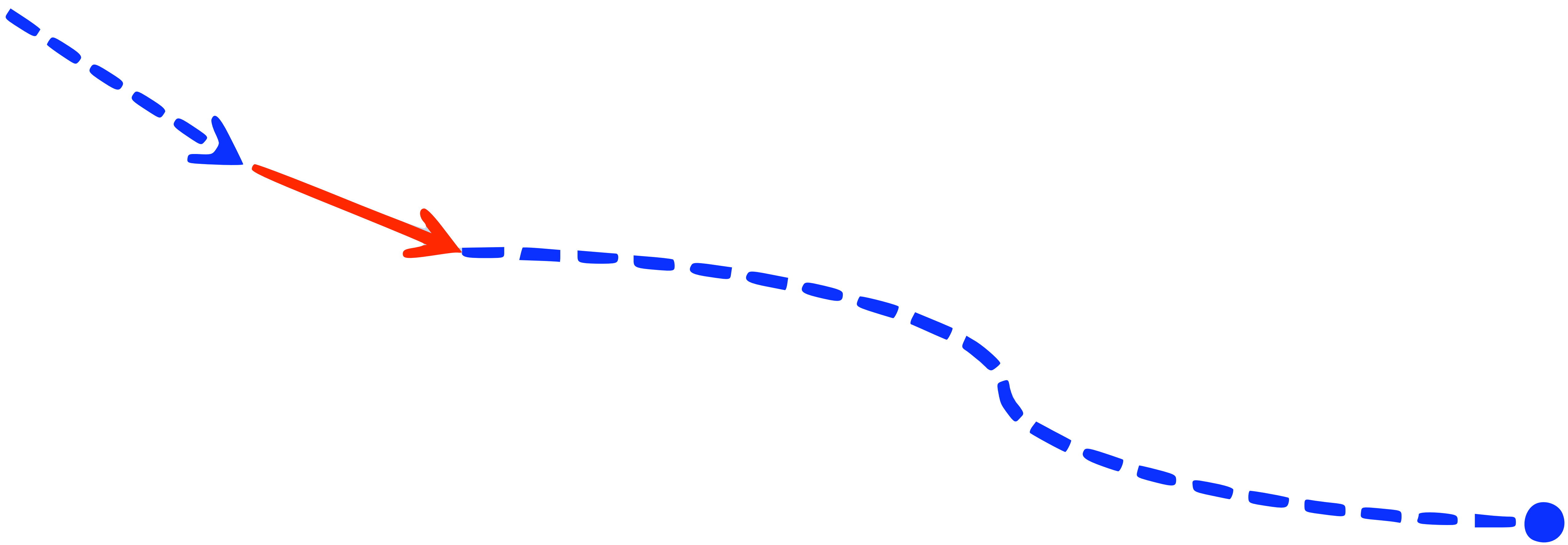
- subset jumps over multiset
- subset lifts to multiset
- well-founded since subset is

Escaping

- For any immortal red chain
  s_1 R s_2 R s_3 R...
- there is also an immortal purple chain after some blue turn
  s_1 R s_2 R ... R s_k B t_1 P t_2 P...


Jumping + Escaping
X

Jumping + Escaping



Escaping Union

- If B jumps over R
- and B escapes from R
- then union well-founded iff B is
  - s_1 BB...B t_1 <u>R</u> t_2 <u>R</u> t_3 <u>R</u> ... XXX
  - s_1 BB...B t_1 <u>R</u> <u>R</u> <u>R</u> u_k BBBB...

Termination

9. Dependencies

Assumption

- Simplification orders
- Assume fixed or bounded arity
- Otherwise need another condition
  - $f(\ldots s \ldots) \gtrsim f(\ldots\ \ldots)$

Substitutions

- substitution $\{x_i \mapsto u_i\}$
- apply $t\{x_i \mapsto u_i\}$, replace each occurrence of variable x_i in t with term u_i
- compose $\{x_i \mapsto u_i\}\sigma = \{x_i \mapsto u_i\sigma\}$

Unifiers

- substitution σ unifies terms s and t if $s\sigma = t\sigma$
- substitution μ more general than σ if there's a τ (not a renaming) such that $\sigma = \mu\tau$
- if there is a unifier, then there is a unique most general one μ (unique up to renaming)

Unifiers

- x,y distinct variables
- f,g distinct symbols

- $mgu(x,x) = \emptyset$; $mgu(x,y) = \{x \mapsto y\}$
- $mgu(x,t) = \{x \mapsto t\}$, t does not contain x
- $mgu(x,t) = fail$, t contains x (but isn't t x)
- $mgu(f(\underline{s}),g(\underline{t})) = fail$; $mgu(f(),f()) = \emptyset$
- $mgu(f(u,\underline{s}),f(v,\underline{t})) = \mu \cup mgu(f(\underline{s}\mu),f(\underline{t}\mu))$ where $\mu = mgu(u,v)$

Non-termination

- Can use most general unifier to look for examples of nontermination
  - Given two derivations $s \dashrightarrow t$ and $u \dashrightarrow v$
    - renamed so that the two have distinct variables
    - rules are one-step derivations
  - extend (if possible) by mgu μ of u and <u>nonvariable</u> subterm of t
    - $s\mu \dashrightarrow t\mu = r\mu[u\mu] \dashrightarrow r\mu[v\mu]$

Jumping

- Let $P = R \cup B$
- If $s \; R \cup B \; t$
  - then $s \; R \; t$
  - or $s \; B \; v_1 \; P \; v_2 \; P \; \ldots \; P \; v_n \; P \; t$
- In short $RB \subseteq R \cup BP^*$
- Hence (induction) $RB^* \subseteq R \cup BP^*$

Jumping Union

- If B jumps over R
- then union well-founded iff both are
- s_1 BB...B t_1 $\underline{R}$B* t_2 $\underline{R}$B* t_2 $\underline{R}$B* ...
  - s_1 BB...B t_1 $\underline{R}$ t_2 $\underline{R}$ t_3 $\underline{R}$ ...
- s_1 BB...B t_1 $\underline{R}$B* t_2 $\underline{R}$B* t_2 $\underline{R}$BBBB...
  - s_1 BB...B t_1 $\underline{R}$ $\underline{R}$ $\underline{R}$ u_k BBBB...

Escaping

- For any immortal red chain
  s_1 R s_2 R s_3 R...
- there is also an immortal purple chain after some blue turn
  s_1 R s_2 R ... R s_k B t_1 P t_2 P...

Escaping Union

- If B jumps over R
- and B escapes from R
- then union well-founded iff B is
  - s_1 BB...B t_1 $\underline{R}$ t_2 $\underline{R}$ t_3 $\underline{R}$ ... XXX
  - s_1 BB...B t_1 $\underline{R}$ $\underline{R}$ $\underline{R}$ u_k BBBB...

Top & Not

- Two parts to rewriting $\Rightarrow$
  - instance of rule $\Rightarrow_{top}$
  - within a context $\Rightarrow_{in}$

Top | Not

- Immediate subterm: $f(\ldots t \ldots) \rhd t$
- If $s1 \Rightarrow s2 \Rightarrow s3 \Rightarrow \ldots$
  - Either $si \Rightarrow_{top} \ldots sj \Rightarrow_{top} \ldots sk \Rightarrow_{top}$
  - Or $s1 \Rightarrow \ldots \Rightarrow sk \rhd t1 \Rightarrow t2 \Rightarrow \ldots$

Facts

- $f(\ldots s \ldots u \ldots) \Rightarrow_{in} f(\ldots t \ldots u \ldots) \rhd t$
  - $f(\ldots s \ldots u \ldots) \rhd s \Rightarrow t$
- $f(\ldots s \ldots u \ldots) \Rightarrow_{in} f(\ldots t \ldots u \ldots) \rhd u$
  - $f(\ldots s \ldots u \ldots) \rhd u$

Dependencies

- Let ➤ be $\Rightarrow_{top} \rhd^*$
- Rule s $\Rightarrow$ t[u]
    - s ➤ u
    - exclude variable u

Dependency Pairs

- R rewrite step
- T top step
- I inner step (not at top)
- D dependency pair (includes top step)
- A subterm

Dependencies

- $B = D \cup I$
- $R \subseteq B$
- $DA \subseteq D \cup A^+ \subseteq B \cup A^+$
- $IA \subseteq A \cup AR \subseteq A \cup AB$
- $BA \subseteq B \cup A^+ \cup AB$
- A jumps over B ($D \cup I$)

Dependencies

- Show $B = D \cup I$ is terminating
- $D \subseteq >$
- $I \subseteq \gtrsim$
- $>$ well-founded
- $\gtrsim > \subseteq >$ "compatible"

Proof

- Infinite D & I, with infinitely many Ds
- A escapes from I and jumps over I
- Can't have infinite tail of only I
- So show I*D terminates
- $I^*D \subseteq \gtrsim > \subseteq >$

Advantage

- Must have infinitely many D steps at top
- So enough to show other steps $\gtrsim$

Quotient

- $x - 0 \Rightarrow x$
- $sx - sy \Rightarrow x - y$
- $0 \div sy \Rightarrow 0$
- $sx \div sy \Rightarrow s(\ [x-y] \div sy\ )$

Rules

- $x - 0 \gtrsim x$
- $sx - sy \gtrsim x - y$
- $0 \div sy \gtrsim 0$
- $sx \div sy \gtrsim s(\, [x-y] \div sy\,)$

Drop Subtrahend

LPO with only first argument of -

- $-x \gtrsim x$
- $-sx \gtrsim -x$
- $0 \dot{-} sy \gtrsim 0$
- $sx \dot{-} sy \gtrsim s(-x \dot{-} sy)$

Pairs

- $sx - sy > x - y$
- $sx \div sy > (x-y) \div sy$
- $sx \div sy > x-y$

Pairs

- $-sx > -x$
- $sx \div sy > -x \div sy$
- $sx \div sy > -x$

Dependency Graph

$$
\begin{tikzcd}
\boxed{\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x,y),\mathsf{s}(y))} \arrow[loop above] \arrow[d] \\
\boxed{\mathsf{D}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)} \arrow[d] \\
\boxed{\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{M}(x,y)} \arrow[loop below]
\end{tikzcd}
$$

Termination

10. Recursion

$$
\begin{aligned}
x - \mathsf{0} &\to x \\
\mathsf{s}(x) - \mathsf{s}(y) &\to x - y \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) &\to \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) \\
\mathsf{0} + y &\to y \\
\mathsf{s}(x) + y &\to \mathsf{s}(x + y) \\
(x - y) - z &\to x - (y + z)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{le}(0, y) &\to \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), 0) &\to \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{le}(x, y) \\
\mathsf{minus}(0, y) &\to 0 \\
\mathsf{minus}(\mathsf{s}(x), y) &\to \mathsf{if}_{\mathsf{minus}}(\mathsf{le}(\mathsf{s}(x), y), \mathsf{s}(x), y) \\
\mathsf{if}_{\mathsf{minus}}(\mathsf{true}, \mathsf{s}(x), y) &\to 0 \\
\mathsf{if}_{\mathsf{minus}}(\mathsf{false}, \mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{minus}(x, y)) \\
\mathsf{quot}(0, \mathsf{s}(y)) &\to 0 \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y)))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{le}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{le}(x, y) \\
\mathsf{pred}(\mathsf{s}(x)) &\to x \\
\mathsf{minus}(x, \mathsf{0}) &\to x \\
\mathsf{minus}(x, \mathsf{s}(y)) &\to \mathsf{pred}(\mathsf{minus}(x, y)) \\
\mathsf{gcd}(\mathsf{0}, y) &\to y \\
\mathsf{gcd}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{s}(x) \\
\mathsf{gcd}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{if}_{\mathsf{gcd}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{gcd}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{gcd}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{gcd}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{gcd}(\mathsf{minus}(y, x), \mathsf{s}(x))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{le}(x, y) \\
\mathsf{app}(\mathsf{nil}, y) &\to y \\
\mathsf{app}(\mathsf{add}(n, x), y) &\to \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{low}(n, \mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{low}(n, \mathsf{add}(m, x)) &\to \mathsf{if}_{\mathsf{low}}(\mathsf{le}(m, n), n, \mathsf{add}(m, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{add}(m, x)) &\to \mathsf{add}(m, \mathsf{low}(n, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{add}(m, x)) &\to \mathsf{low}(n, x) \\
\mathsf{high}(n, \mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{high}(n, \mathsf{add}(m, x)) &\to \mathsf{if}_{\mathsf{high}}(\mathsf{le}(m, n), n, \mathsf{add}(m, x)) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{add}(m, x)) &\to \mathsf{high}(n, x) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) &\to \mathsf{add}(m, \mathsf{high}(n, x)) \\
\mathsf{quicksort}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{quicksort}(\mathsf{add}(n, x)) &\to \mathsf{app}(\mathsf{quicksort}(\mathsf{low}(n, x)), \\
&\qquad\qquad \mathsf{add}(n, \mathsf{quicksort}(\mathsf{high}(n, x))))
\end{aligned}
$$

Apply

```
apply(t,σ)  :=
    if var?(t)
    then if σ ={}
        then t
        else let {x↦u}∪σ'=σ in
            if t=x
            then u
            else apply(t,σ')
    else let f(t1,…,tn)=t in
        f(apply(t1,σ),…,apply(tn,σ))
```

Occur?

```
occur?(x,t) :=
    if var?(t)
    then (x=t)
    else let f(t1,...,tn)=t in
        occur?(x,t1) ∨...∨ occur?(x,tn)
```

Unify

```
unify(s,t) :=
    if var?(s)
    then if var?(t)
            then if s=t then {} else {s↦t}
            else if occur?(s,t)
                    then fail
                    else {s↦t}
    else let f(s1,...,sm) = s & g(t1,...,tn) = t in
        if f≠g
        then fail
        else if m=0 [assuming m=n]
            then {}
            else let σ = unify(s1,t1) in
                let τ = unify(apply(f(s2,...,sm), σ),apply(f(t2,...,tn), σ)) in
                τ ∪ στ [composition of substitutions....]
```

Primitive Recursion

- f(n,x,...,z) :=

  if n=0

  then g(x,...,z)

  else h(f(n-1,x,...,z),n-1,x,...,z)

Inductive Definitions

- Constructors
  - 0, s(0), s(s(0)), ...
  - e, a(e), b(e), a(a(e)), ...
  - e, b(e,e), b(b(e,e),e), ...

Structural Induction

- a(x,*y*) := if x=() then *y* else c(hd(x),a(tl(x),*y*))
- r(x) := if x=() then () else a(r(tl(x)),c(hd(x),()))

Functions

- Basic (e.g. arithmetic, boolean)
- Constructors (e.g. lists, trees)
- Conditional (if c then a else b)
- Defined (recursively, perhaps)

Definitions

- $f(x,y,\ldots,z) := t[x,y,\ldots,z]$
- $e(m,n) := \text{if } n=0 \text{ then } 1 \text{ else } m \times e(m,n-1)$

Evaluations

- $\mathrm{if}(T,x,y) \Rightarrow x$
- $\mathrm{if}(F,x,y) \Rightarrow y$
- $\mathrm{if}(c,x,y) \Rightarrow \mathrm{if}(c',x,y)$
- $f(x,y) \Rightarrow t[x,y]$
- $f(x,y) \Rightarrow f(x',y)$
- $f(x,y) \Rightarrow f(x,y')$

Inner/Outer

- $\mathrm{if}(T,x,y) \Rightarrow x$
- $\mathrm{if}(F,x,y) \Rightarrow y$
- $\mathrm{if}(c,x,y) \Rightarrow \mathrm{if}(c',x,y)$
- $f(x,y) \Rightarrow t[x,y]$
- $f(x,y) \Rightarrow f(x',y)$
- $f(x,y) \Rightarrow f(x,y')$

Inner & Outer

- N: normative; nothing above
- A: applicative; nothing below
- I: inner; something above (not normal)
- O: outer; something below

91 Example

- f(x) := if x>100

  then x-10

  else f(f(x+11))

Example

- $f(x,y) := \text{if } x=0$

  $\text{then } 2$

  $\text{else } f(x-1,f(x+y,y))$

Example

- f(x,y) := if x=0

  then 0

  else if x=1

  then f(0,f(1,y))

  else f(x-2,y+1)

Example

- $f(1,1) = f(0, f(1,1)) = ???$

In vs. Out

- If any computation is terminating, then outermost (normal order) is terminating.
- If any computation is non-terminating, then innermost (applicative order) is non-terminating.

Normal is Very Good

- Suppose not
- Consider minimal counterexample
- u NNNNINNIINNNNIII v ; v value
- I N ≈ I O ⊆ N A*
- So: u N…N I…I v
- But can't have Iv, so u N* v

Applicative is Very Bad

- If u O v, then
  - there are u' v' v'' such that
  - u $A^!$ u' A v' $A^!$ v''
  - v A^* v' $A^!$ v''
  - $A^!$ means as much as possible

Termination

11. Eventuality

Transformation

Transitions

- Program: s1 ⇝ s2 ⇝ s3 ⇝ ...
- Transformation si ↦ si
- Schema: s1 ⇝ s2 ⇝ s3 ⇝ ...
- s ⇝ s' if s ⇝ s'

Homework

Example

$x - 0 \Rightarrow x$

$sx - sy \Rightarrow x - y$

$0 \div sy \Rightarrow 0$

$sx \div sy \Rightarrow s((x-y) \div sy)$

$0 + y \Rightarrow y$

$sx + y \Rightarrow s(x+y)$

$(x-y) - z \Rightarrow x - (y+z)$

Easy Rules

$x - 0 \Rightarrow x$

$0 \div sy \Rightarrow 0$

$0 + y \Rightarrow y$

Precedence

÷ , + > s > ~ (lr?)

Hard Rule

$$sx \div sy \Rightarrow s((x-y) \div sy)$$

Solution

$$sx \div sy \Rightarrow s((x-\cancel{y}) \div sy)$$

Problem

$$sx \div sy \Rightarrow s((x-y) \div sy) \Rightarrow s((u+v) \div sy)$$

Pairs

$sx - sy \Rightarrow x - y$

$sx \div sy \Rightarrow (x-y) \div sy$ $\quad$ $sx \div sy \Rightarrow x-y$

$sx + y \Rightarrow x + y$

$(x-y) - z \Rightarrow x - (y+z)$ $\quad$ $(x-y) - z \Rightarrow y + z$

Pairs - Colored

$sx - sy \Rightarrow x - y$

$sx \div sy \Rightarrow (x-y) \div sy \qquad sx \div sy \Rightarrow x-y$

$sx + y \Rightarrow x + y$

$(x-y) - z \Rightarrow x - (y+z) \qquad (x-y) - z \Rightarrow y + z$

Pairs - Separated

$$\boxed{sx - sy \Rightarrow x - y}$$

$$\boxed{sx \div sy \Rightarrow (x-y) \div sy \qquad sx \div sy \Rightarrow x-y}$$

$$\boxed{sx + y \Rightarrow x + y}$$

$$\boxed{(x-y) - z \Rightarrow x - (y+z) \qquad (x-y) - z \Rightarrow y + z}$$

Pairs - Separated

$$\boxed{sx + y \Rightarrow x + y}$$

Pairs - Separated

$sx \div sy \Rightarrow (x-y) \div sy$ $\qquad$ $sx \div sy \Rightarrow x-y$

Pairs - Separated

$$sx \div sy \Rightarrow x- \qquad \div sy \qquad\qquad sx \div sy \Rightarrow x -$$

Pairs - Separated

$$sx - sy \Rightarrow x - y$$

$$(x-y) - z \Rightarrow x - (y+z) \qquad (x-y) - z \Rightarrow y + z$$

Rules

$x - 0 \Rightarrow x$

$sx - sy \Rightarrow x - y$

$0 \div sy \Rightarrow 0$

$sx \div sy \Rightarrow s((x-y) \div sy)$

$0 + y \Rightarrow y$

$sx + y \Rightarrow s(x+y)$

$(x-y) - z \Rightarrow x - (y+z)$

Rules -

$x - \;\Rightarrow\; x$

$sx - \;\Rightarrow\; x -$

$0 \div sy \Rightarrow 0$

$sx \div sy \Rightarrow s((x- \;) \div sy)$

$0 + y \Rightarrow y$

$sx + y \Rightarrow s(x+y)$

$(x-) \; - \;\Rightarrow\; x -$

Rules $\gtrsim$

$$x - \;\gtrsim\; x$$

$$sx - \;\gtrsim\; x -$$

$$0 \div sy \;\gtrsim\; 0$$

$$sx \div sy \;\gtrsim\; s((x- \;) \div sy)$$

$$0 + y \;\gtrsim\; y$$

$$sx + y \;\gtrsim\; s(x+y)$$

$$(x-) \; - \;\gtrsim\; x -$$

$$0 \le y \Rightarrow T$$

$$sx \le 0 \Rightarrow F$$

$$sx \le sy \Rightarrow x \le y$$

$$0 - y \Rightarrow 0$$

$$sx - y \Rightarrow if(sx \le y, sx, y)$$

$$if(T, sx, y) \Rightarrow 0$$

$$if(F, sx, y) \Rightarrow s(x - y)$$

$$0 \div sy \Rightarrow 0$$

$$sx \div sy \Rightarrow s((x - y) \div sy)$$

$$0 \leq y \Rightarrow T$$

$$sx \leq y \Rightarrow F$$

$$sx \leq sy \Rightarrow x \leq y$$

$$psx \Rightarrow x$$

$$x - 0 \Rightarrow x$$

$$x - sy \Rightarrow p(x-y)$$

$$gcd(sx,0) \Rightarrow s(x)$$

$$gcd(sx,sy) \Rightarrow if(y\leq x,sx,sy)$$

$$if(T,sx,sy) \Rightarrow gcd(x-y,sy)$$

$$if(F,sx,sy) \Rightarrow gcd(y-x,sx)$$

$$
\begin{aligned}
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{le}(x, y) \\
\mathsf{app}(\mathsf{nil}, y) &\to y \\
\mathsf{app}(\mathsf{add}(n, x), y) &\to \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{low}(n, \mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{low}(n, \mathsf{add}(m, x)) &\to \mathsf{if}_{\mathsf{low}}(\mathsf{le}(m, n), n, \mathsf{add}(m, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{add}(m, x)) &\to \mathsf{add}(m, \mathsf{low}(n, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{add}(m, x)) &\to \mathsf{low}(n, x) \\
\mathsf{high}(n, \mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{high}(n, \mathsf{add}(m, x)) &\to \mathsf{if}_{\mathsf{high}}(\mathsf{le}(m, n), n, \mathsf{add}(m, x)) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{add}(m, x)) &\to \mathsf{high}(n, x) \\
\mathsf{if}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) &\to \mathsf{add}(m, \mathsf{high}(n, x)) \\
\mathsf{quicksort}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{quicksort}(\mathsf{add}(n, x)) &\to \mathsf{app}(\mathsf{quicksort}(\mathsf{low}(n, x)), \\
&\qquad\quad \mathsf{add}(n, \mathsf{quicksort}(\mathsf{high}(n, x))))
\end{aligned}
$$

Dataflow

Top Graph

- Pierre Réty & al. (1987) : Narrowing

- Jürgen Giesl & al. (2000) : Rewriting

Argument Graph

- Shuki Sagiv & al. (1991) : Logic languages
- Neil Jones & al. (2000) : Functional languages

Induction

Leaves

```
leaves(t) :=
    if leaf(t)
    then 1
    else leaves(left(t)) + leaves(right(t))
```

Counting Leaves

```
s := push(t,empty)
n := 0
loop while s ≠ empty
    h := top(s)
    s := pop(s)
    if leaf(h)
    then n := n + 1
    else s := push(left(h),push(right(h),s))
```

Correctness

- if s=t.e and n=0
- then eventually s=e and n=#(t)

Lemma

- if s=t.r and n=k
- then eventually s=r and n=k+#(t)

Induction (1)

- if s=leaf.r and n=k
- then eventually s=r and n=k+#(leaf)
- then eventually s=r and n=k+1

Induction (2)

- if s=b(lt,rt).r and n=k
- then s=lt.rt.r and n=k
- then eventually s=rt.r and n=k+#(lt)
- then eventually s=r and n=k+#(lt)+#(rt)
- then eventually s=r and n=k+#b(lt,rt)

Termination

- if s=t.e
- then eventually s=e

Lemma

- if s=t.r
- then eventually s=r

Ackermann

```
t := 1
s[t] := m
loop m := s[t]
     t := t-1
     if m=0
     then n := n+1
     else if n=0
     then t := t+1
          s[t] := m-1
          n := 1
     else t := t+2
          s[t-1] := m-1
          s[t] := m
          n := n-1
     until t=0
```

Termination

- If $t=k$ then eventually $t=k-1$ and $s[0:k-1]$ same
- Induction on (m,n) just after $m := s[t]$
- Case 1, $m=0$: $t' = t-1$
- Case 2, $m>0$, $n=0$: $t' = t$; $m' = m-1$
- Case 3, $m,n>0$: $t' = t+1$; $m' = m$; $n' = n-1$; $s[t'] = m-1$
- By induction, eventually $t''=t$; $m'' = m-1$

Termination

12. Typing

Grades

- 10% - participation & exercises
- 90% - term paper

- Alonzo Church (1903-1995)
  - invented lambda calculus (1932)
  - first programming-language researcher (sans computers)
  - Turing's advisor

A SET OF POSTULATES FOR THE FOUNDATION OF LOGIC.[1]

By Alonzo Church.[2]

1. **Introduction.** In this paper we present a set of postulates for the foundation of formal logic, in which we avoid use of the free, or real, variable, and in which we introduce a certain restriction on the law of excluded middle as a means of avoiding the paradoxes connected with the mathematics of the transfinite.

free and bound variables

In consequence of this abstract character of the system which we are about to formulate, it is not admissible, in proving theorems of the system, to make use of the meaning of any of the symbols, although in the application which is intended the symbols do acquire meanings. The initial set of postulates must of themselves define the system as a formal structure, and in developing this formal structure reference to the proposed application must be held irrelevant. There may, indeed, be other applications of the system than its use as a logic.

symbols do not have pre-conceived meanings

In consequence of this abstract character of the system which we are about to formulate, it is not admissible, in proving theorems of the system, to make use of the meaning of any of the symbols, although in the application which is intended the symbols do acquire meanings. The initial set of postulates must of themselves define the system as a formal structure, and in developing this formal structure reference to the proposed application must be held irrelevant. There may, indeed, be other applications of the system than its use as a logic.

symbols do not have pre-conceived meanings

Proof terms, well-formed objects

An occurrence of a variable **x** in a given formula is called an occurrence of **x** as a *bound variable* in the given formula if it is an occurrence of **x** in a part of the formula of the form $\lambda \mathbf{x}[\mathbf{M}]$; that is, if there is a formula **M** such that $\lambda \mathbf{x}[\mathbf{M}]$ occurs in the given formula and the occurrence of **x** in question is an occurrence in $\lambda \mathbf{x}[\mathbf{M}]$. All other occurrences of a variable in a formula are called occurrences as a *free variable*.

A formula is said to be *well-formed* if it is a variable, or if it is one

Lambda Calculus

- Everything is a function
- For example, $\lambda x.x$ is the identity function
- $\lambda y.\lambda x.x$ is a constant function, always returning identity

Lambda Terms

- Constants C; Variables X
- L = constant | variable | application | abstraction
- L ::= C | X | (LL) | λX.L

Positions

- Dewey decimal system
- Number children, left to right
- Path to position gives "address"

Free Occurrences

- Constants C; Variables X
- $L := C \mid X \mid (LL) \mid \lambda X.L$
- $F_x(c) = \{\}$ $\quad F_x(x) = \{e\}$
- $F_x(st) = 0.F_x(s) \cup 1.F_x(t)$
- $F_x(\lambda x.s) = \{\}$
- $F_x(\lambda y.s) = 1.F_x(s)$

Lambda Calculus

- β-rule: $(\lambda x.s)t \to s[x \mapsto t]$
- Replace (all free) x in s with t

Substitution

- $x[x \mapsto t] = t$
- $y[x \mapsto t] = y$
- $c[x \mapsto t] = c$
- $(su)[x \mapsto t] = s[x \mapsto t]\ u[x \mapsto t]$
- $(\lambda x.s)[x \mapsto t] = \lambda x.s$
- $(\lambda y.s)[x \mapsto t] = \lambda y.\ s[x \mapsto t]$

Beta Immortality

- $\lambda x.x(x)\ \lambda x.x(x) \rightarrow \lambda x.x(x)\ \lambda x.x(x)$

Completeness

- Every recursive function can be simulated by a pure lambda expression.
- Church numerals represent the naturals.
- Termination is undecidable.

Church Numerals

- n
- $\lambda f,x.f^{n}(x)$

Church Numerals

T	$\lambda x,y.x$
F	$\lambda x,y.y$
if(c,a,b)	$\lambda c,a,b.c(a,b)$
0	$\lambda f,x.x$
n++	$\lambda f,x.f(n(f,x))$
n--	hard
n=0	$n(\lambda x.F,T)$

Synagogue Numerals

• T	• $\lambda x,y.x$
• F	• $\lambda x,y.y$
• if(c,a,b)	• $\lambda c,a,b.c(a,b)$
• 0	• $\lambda x.x$
• n++	• $\lambda x.x(F,n)$
• n--	• n(F)
• n=0	• n(T)

Scheme

- (((lambda (x y) (y x)) (lambda (z) z) (lambda (z) (z z))) 5)
- (((lambda (z) (z z)) (lambda (z) z)) 5)
- (((lambda (z) z) (lambda (z) z)) 5)
- ((lambda (z) z) 5)
- 5

Inner vs. Outer

- Scheme uses innermost
- Haskell uses outermost

Recursor

- $Y := (\lambda x.(\lambda y.x(y(y)))(\lambda y.x(y(y))))$
- $Y(b)$: recursive function with body b
- fixpoint: $Y(b) = b(Y(b))$

- (Y(λf.λm,n.if(n=0,m,(f(m,n--))++)))) (3,4)

Currying

$\lambda x.\lambda y.A[x,y]$ instead of $\lambda x,y.A[x,y]$

$+$ is the binary addition function

$+(3)$ adds 3 to any number

$+(3)(4)$ evaluates to 7

Arithmetic (Rosser)

• 0	• $\lambda f.\lambda x.x$
• $n{+}{+}$	• $\lambda f.\lambda x.n(f)(f(x))$
• $m+n$	• $\lambda f.\lambda x.m(f)((n(f))(x))$
• mn	• $\lambda f.m(n(f))$
• m^n	• $\lambda f.n(m)(f)$

λ-calculus and first-order rewriting led to two important families of programming languages:

- functional programming languages: Lisp (1958), ML (1972), Haskell (1990), OCaml (1996), F# (2005), ...
- rewriting-based languages: OBJ (1976), Elan (1994), Maude (1996), ...

Simple Types

- Base types B (e.g. Nat)
- Arrow types [e.g. Nat → (Nat → Bool)]
- Each constant/variable has a type
- Type($\lambda x{:}\sigma.\ s{:}\tau$) = $\sigma \to \tau$
- Type($s{:}\sigma \to \tau\ t{:}\sigma$) = τ

Typing Rules

$$\frac{}{x : A \vdash x : A}\,\mathrm{Id}$$

$$\frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x.u : A \to B}\,\to\text{-I} \qquad \frac{\Gamma \vdash s : A \to B \qquad \Delta \vdash t : A}{\Gamma, \Delta \vdash st : B}\,\to\text{-E}$$

Typed Lambda Calculus

- β-rule: $(\lambda x{:}\sigma.\ s{:}\tau)\,t{:}\sigma \rightarrow s[x{:}\sigma \mapsto t{:}\sigma]{:}\tau$

Typed Beta Mortality

- $\lambda x{:}\sigma\to\tau.(x{:}\sigma\to\tau\ x{:}\sigma) : (\sigma\to\tau)\to\tau$

Termination

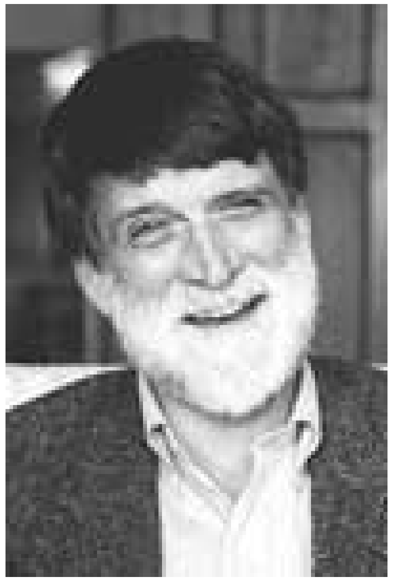
- Turing gave first proof
- Tait's proof
  - Induction on term structure
  - Induction on type structure

Termination of β-reduction alone?

in the simply-typed λ-calculus:

- $\rightarrow_\beta$ can be proved terminating by a direct induction on the type of the substituted variable (Sanchis 1967, van Daalen 1980)
  does not extend to rewriting where the type of substituted variables can increase, *e.g.* $\mathsf{f}(\mathsf{c}x) \rightarrow x$ with $x : \mathsf{A} \Rightarrow \mathsf{B}$

computability has been introduced for proving termination of β-reduction in typed λ-calculi (Tait, 1967) (Girard, 1970)



- every type T is mapped to a set $[\![T]\!]$ of computable terms
- every term $t : T$ is proved to be computable, *i.e.* $t \in [\![T]\!]$

Predicates

- S[t] : t is "terminating" (no infinite paths)
- C[t] : t is "computable" (typed terminating)
- N[t] : t is "normalizing" (has a normal form)

Facts

- $S[t]\ \&\ t \rightarrow u \Rightarrow S[u]$
- $S[t]\ \&\ t \rhd u \Rightarrow S[u]$
- $\{\ \forall u.\ t \rightarrow u \Rightarrow S[u]\ \} \Rightarrow S[t]$

Desiderata

1. $C[t] \Rightarrow S[t]$

2. $C[s]$ & $s \to t \Rightarrow C[t]$

3. $C[x] \qquad C[c]$

4. $\forall t \{ u(v) \to t \Rightarrow C[t] \} \Rightarrow C[u(v)]$

5. $C[u] \Leftrightarrow \forall v \{ C[v] \Rightarrow C[u(v)] \}$

Computability predicates

there are different definitions of computability (Tait Sat, Girard Red, Parigot SatInd, Girard $\mathrm{Bi}\bot$) but Girard's definition Red is better suited for handling *arbitrary* rewriting

let Red be the set of P such that:

- termination: $P \subseteq \mathrm{SN}(\rightarrow_\beta)$
- stability by reduction: $\rightarrow_\beta(P) \subseteq P$
- if t is neutral and $\rightarrow_\beta(t) \subseteq P$ then $t \in P$

neutral = not head-reducible after application (λxu is *not* neutral)

Termination

13. Higher-Order Orderings

Predicates

- S[t] : t is terminating
- C[t] : t is computable

Computability

Inductive definition of C[t]:

- Basic t: C[t] if S[t]
- Arrow t: C[t] if C[t(s)] for all computable s (of the right type)

Lemmas

0. Reducts of computable terms are computable
1. Computable terms are terminating
2. Applications are computable if all reducts are

Main. Computable substitutions yield computable terms

Lemma 0

- Reducts of computable terms are computable

$$C[t] \ \& \ t \rightarrow u \Rightarrow C[u]$$

Proof of Lemma 0

$C[t]$ & $t \rightarrow u \Rightarrow C[u]$

- Induction on type
- Basic t: $C[u]$ if $S[u]$ if $S[t]$ if $C[t]$
- Arrow $t:\sigma\rightarrow\tau$: By def, $C[t(s):\tau]$ for all computable s. By ind, $C[u(s):\tau]$, for all s. By def, $C[u]$.

Lemma 1

- Computable terms are terminating

$$C[t] \Rightarrow S[t]$$

Proof of Lemma 1

$C[t] \Rightarrow S[t]$

- Induction on type
- Basic t: By definition
- Arrow $t{:}\sigma\to\tau$

  By def, $C[t(s)]$ for all computable $s{:}\sigma$.

  By ind, $S[t(s){:}\tau]$. It must be that $S[t]$, too.

Neutrality

- applying creates no new redexes

  t neutral: redexes of t(s) are in t or s

- computable if reducts are

  C[t] if C[r] for all r s.t. t ➛ r

Lemma 2

Applications are neutral:

$C[s(t)]$ if $C[r]$ for all r s.t. $s(t) \to r$

Proof of Lemma 2

$C[s(t)]$ if $\forall r.\ s(t) \to r \Rightarrow C[r]$

- Induction on type of $s(t)$
- Basic: $S[s(t)]$ iff $S[r]\ \forall r$
- Arrow: Show $C[s(t)(u)]$ for each computable u. By ind, $C[r(u)]\ \forall r$ suffices, which is just $C[r]$.

Corollary

$C[(\lambda x.s)\ (t)]$ if $C[s\{x \mapsto t\}]$ & $C[t]$

By well-founded induction on s,t

Proof of Corollary

$C[s\{x\mapsto t\}]$ & $C[t] \Rightarrow C[(\lambda x.s)(t)]$

By L0, $S[s]$ & $S[t]$. Let $s \rightarrow s'$, $t \rightarrow t'$

So $C[s'\{x\mapsto t\}]$ & $C[t] \Rightarrow C[(\lambda x.s')(t)]$

$C[s\{x\mapsto t\}]$ & $C[t'] \Rightarrow C[(\lambda x.s)(t')]$

By L2, $C[(\lambda x.s)(t)]$ if $C[(\lambda x.s')(t)]$ & $C[(\lambda x.s)(t')]$ & $C[s\{x\mapsto t\}]$

But $C[t] \Rightarrow C[t']$ and $C[s\{x\mapsto t\}] \Rightarrow C[s'\{x\mapsto t\}]$

Lemma 3

S[t1] & … & S[tn] ⇒ C[x(t1)(t2)…(tn)]

- Induction on type of t = x(t1)(t2)…(tn)
- Basic t: Since only reducible inside terminating ti, S[t]. By def, C[t].
- Arrow t:σ→τ. For any computable s:σ, S[s] by L1. By ind, C[t(s):τ]. By def, C[t].

Main Lemma

- Computable substitutions yield computable terms

  Main: $C[u\sigma]$ for all u and computable σ

  - where $C[\sigma]$ if $C[t]$ for all $x \mapsto t$ in σ

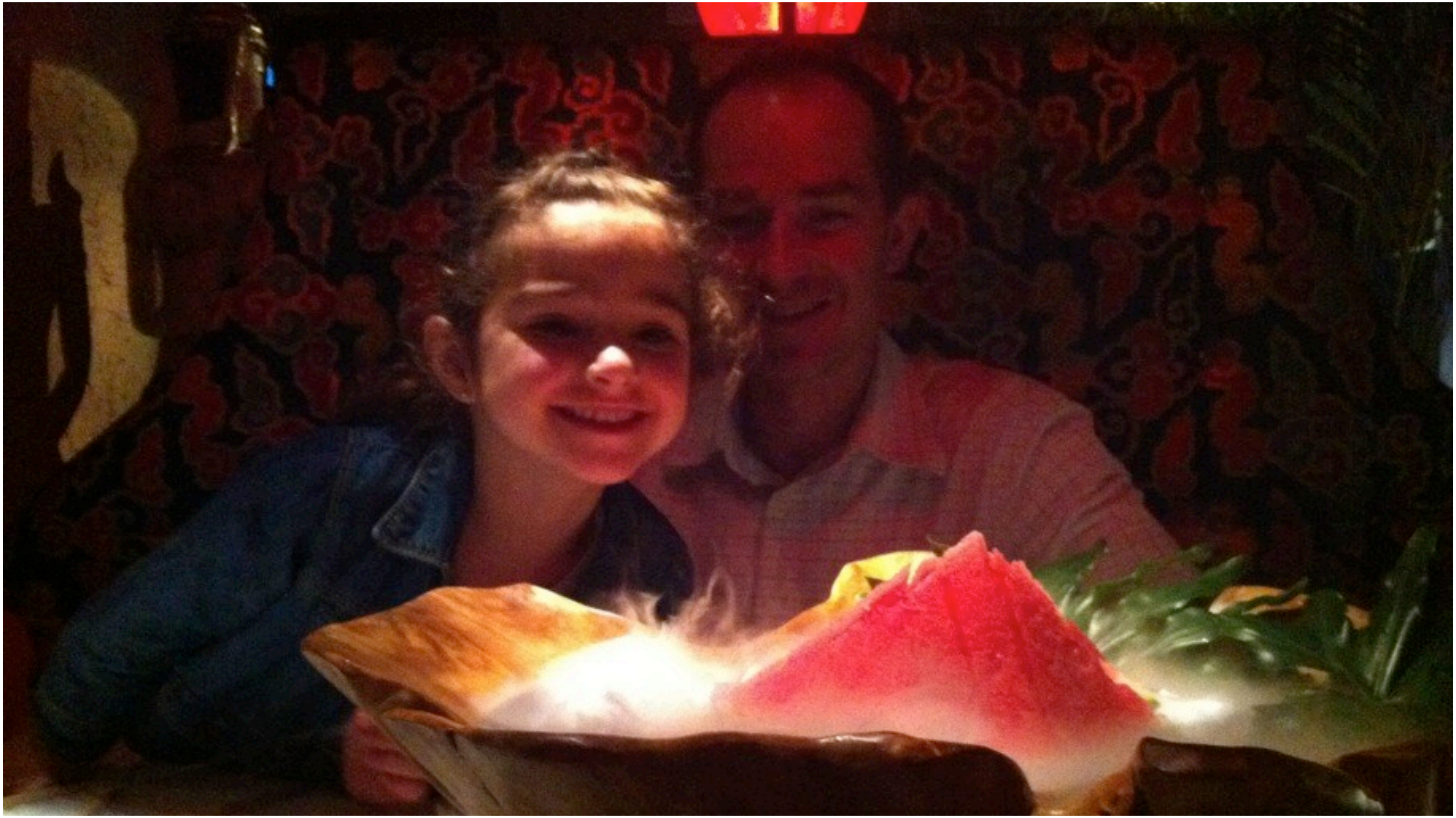
Proof of Main Lemma

$C[u\sigma]$ for computable σ

- Structural induction on u
- u constant: $u=u\sigma$ is basic and terminating; so $C[u]$ by def.
- u is variable x: If $x\sigma=x$, L3 applies; otherwise $x\sigma$ is computable.
- $u=t(s)$: $u\sigma=t\sigma(s\sigma)$. By ind, $C[t\sigma]$; by def, $C[t\sigma(s\sigma)]$, since $C[s\sigma]$ by ind.
- $u=\lambda x.s$: For computable t, let $\sigma'=\sigma-\{x\mapsto x\sigma\}\cup\{x\mapsto t\}$. By ind, $C[s\sigma']$. By L2c, $C[((\lambda x.s)\sigma)(t)]$, as $(\lambda x.s)\sigma = \lambda x.s(\sigma-\{x\mapsto x\sigma\})$ and $s(\sigma-\{x\mapsto x\sigma\})\{x\mapsto t\} = s\sigma'$. By def, $C[(\lambda x.s)\sigma]$.

Theorem

- All typed terms are terminating
  - C[t] for all t
    - Main lemma (empty substitution)
  - S[t] for all t
    - By Lemma 1



Frédéric

Functional

- $D(\lambda x.y) \rightarrow \lambda x.0$
- $D(\lambda x.x) \rightarrow \lambda x.1$
- $D(\lambda x.\sin(F(x))) \rightarrow \lambda x.D(F(x)) \cdot \cos(F(x))$

Higher-Order Rewriting

- map(F,e) ➔ e
- map(F,x:*y*) ➔ F(x):map(F,*y*)

System T

- $\mathrm{rec}(0,u,F) \rightarrow u$
- $\mathrm{rec}(s(x),u,F) \rightarrow F(x,\mathrm{rec}(x,u,F))$
- $n! \rightarrow \mathrm{rec}(n,1, \lambda y,z.s(y)\cdot z)$

Mixing Problem

- $f(c(x)) \rightarrow x$
  - $f: A \rightarrow (A \rightarrow B) \quad c: (A \rightarrow B) \rightarrow A \quad x: A \rightarrow B$
- $w = \lambda z{:}A.f(z)(z)$
  - $w(c(w)) \rightarrow f(c(w))(c(w)) \rightarrow w(c(w)) \rightarrow$

Explicit Application

- @(s,t) for s(t)
- @(F,t) for F(t)

System T

- $rec(0,u,F) \rightarrow u$
- $rec(s(x),u,F) \rightarrow @(F,x,rec(x,u,F))$

Eta

- $\lambda x.f(x) =_\eta f$ (for $x \notin f$)
- eta long: $\lambda x.f(x)$

Higher-Order RPO

- precedence >
  - @ minimal
  - assume total (for simplicity)
- type order >
  - various conditions

Example Type Order

- $\sigma \rightarrow \tau > \tau$
- $\sigma \rightarrow \tau > a \Leftrightarrow \tau \geq a$ (base a)
- $\sigma \rightarrow \tau > \sigma' \rightarrow \tau' \Leftrightarrow \tau > \tau' \vee \sigma \geq \sigma' \rightarrow \tau'$
- well-founded even when enriched with $\sigma \rightarrow \tau > \sigma$

Higher-Order RPO

- $> = >^{\emptyset}$
- $>^{X}$ (keep track of variables X)
- $>^{X} = >^{X} \cap \geq$

Plain Cases

- $s = f(s1,\ldots,sm) >^X g(t1,\ldots,tn)$
  - if $f>g$ & $s >^X t1,\ldots,tn$
- $s = f(s1,\ldots,sm) >^X f(t1,\ldots,tn)$
  - if $\{s1,\ldots,sm\} > \{t1,\ldots,tn\}$ and $s >^X t1,\ldots,tn$
- $s = f(s1,\ldots,sm) >^X t$
  - if some $si \gtrsim^X t$

Variable Case

- $s >^{\{\ldots x \ldots\}} x$
  - if $s \neq x$

Lambda Cases

- $\lambda x{:}\alpha.w[x] >^X t$
  - if $w[z{:}\alpha] \gtrsim^X t$
- $s >^X \lambda y{:}\beta.w[y]$
  - if $s >^{X \cup \{z:\beta\}} w[z]$

Beta-Eta Cases

- $\lambda x.@(v,x) >^X t$
  - if $x \notin v$, $v \gtrsim^X t$
- $@(\lambda x.w[x],v) >^X t$
  - if $w[v] \gtrsim^X t$

Lambda-Lambda

- $\lambda x{:}\alpha.u[x] >^X \lambda y{:}\alpha.w[y]$
  - if $u[z{:}\alpha] >^X w[z]$
- $s = \lambda x{:}\alpha.u[x] >^X \lambda y{:}\beta.w[y]$
  - if $\alpha \neq \beta$ & $s >^X w[z{:}\beta]$

System T

- rec(0,u,F) ➔ u
- rec(s(x),u,F) ➔ @(F,x,rec(x,u,F)))

Brower Ordinals

- rec(0,U,V,W) ➙ U
- rec(s(X),U,V,W) ➙ @(V,X,rec(X,U,V,W))
- rec(lim(F), U, V, W) ➙
  @(W, F, λn.rec(@(F, n), U, V, W))
- a little more needed

Termination

14. Terminate

ε_0

- $0, 1, 2, \ldots, \omega, \omega+1, \omega+2, \ldots, \omega 2, \omega 2+1, \ldots, \omega 3, \ldots, \omega^2, \ldots, \omega^2+\omega 2+3, \ldots, \omega^3, \ldots, \omega^\omega, \ldots, \omega^{\omega^\omega}, \ldots$

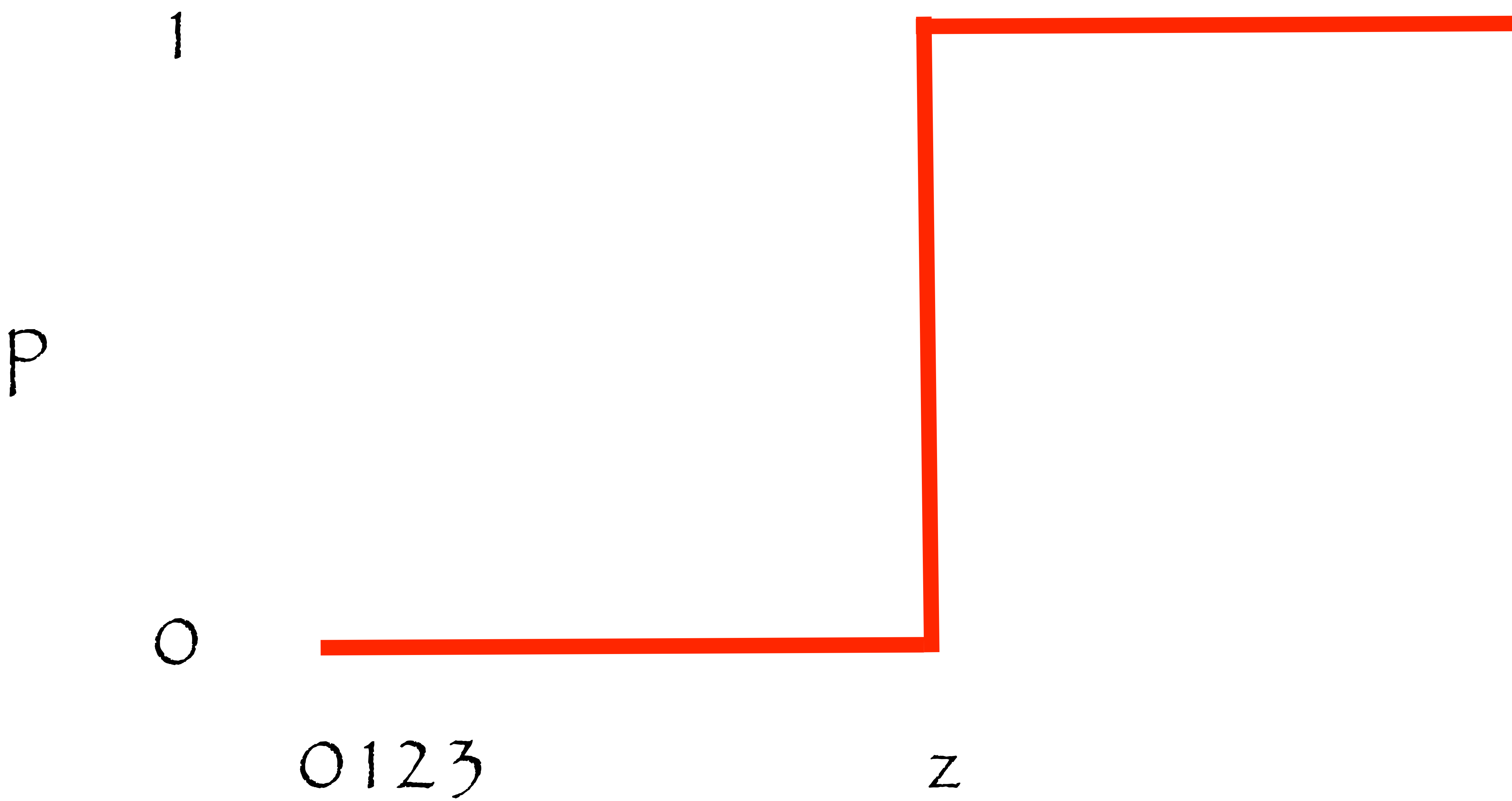
Ordinal Indexing

- $f_0, \ldots, f_{100}, \ldots, f_\omega, \ldots, f_{\omega 2}, \ldots, f_{\varepsilon 0}, \ldots.$

Defenestration

Graduate Students

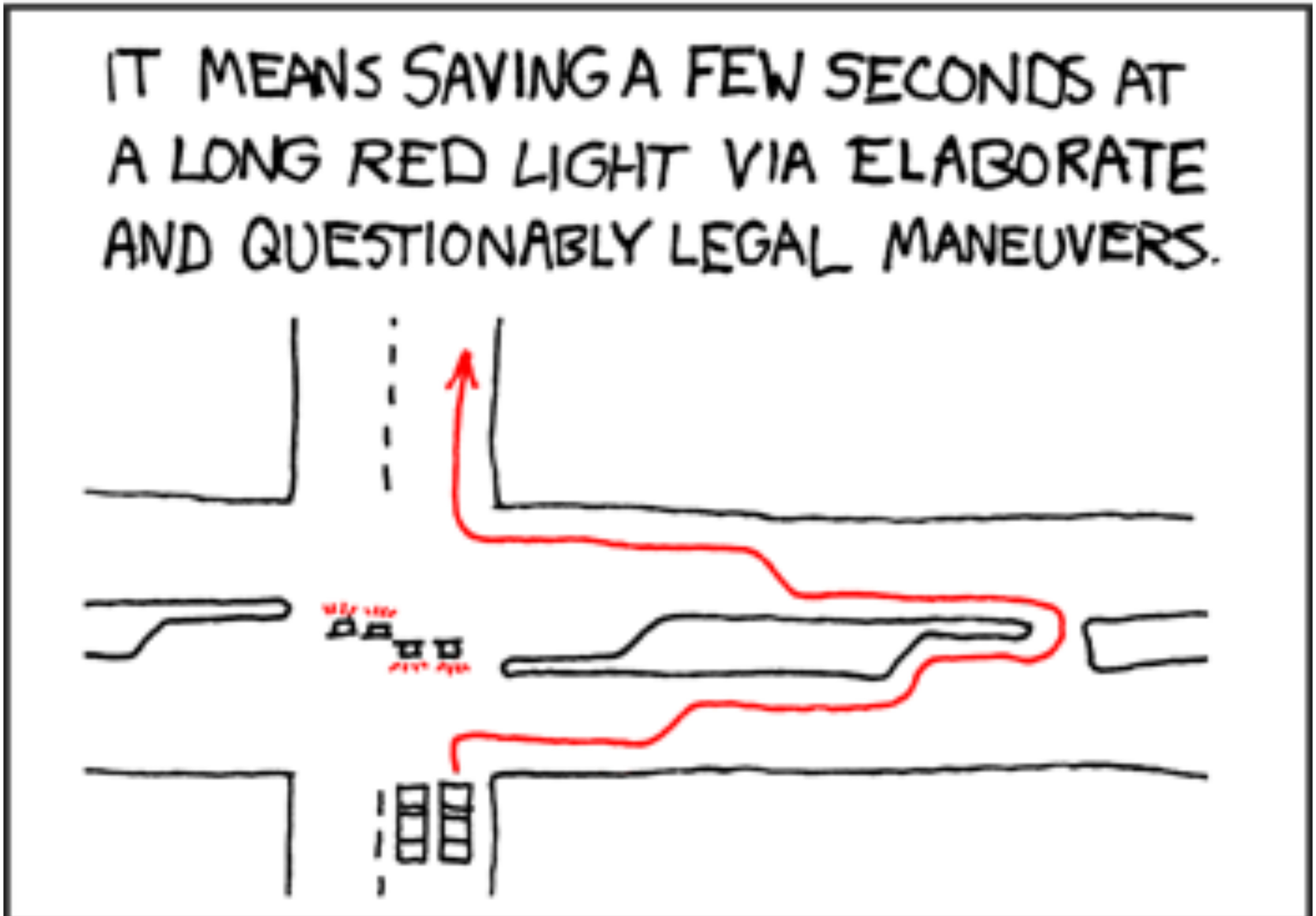
“Binary” Search

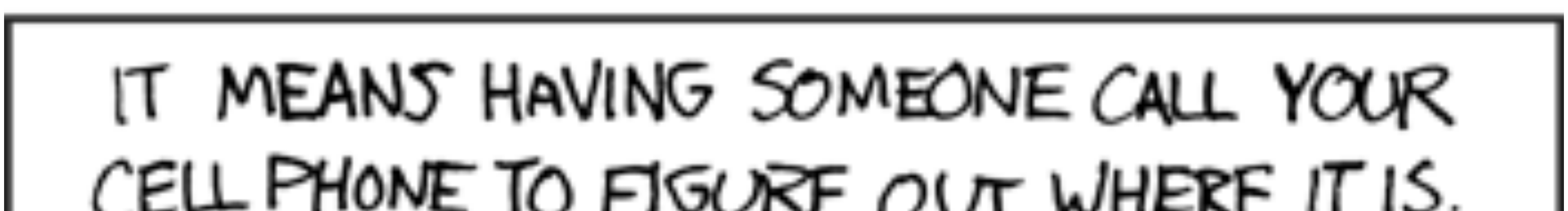


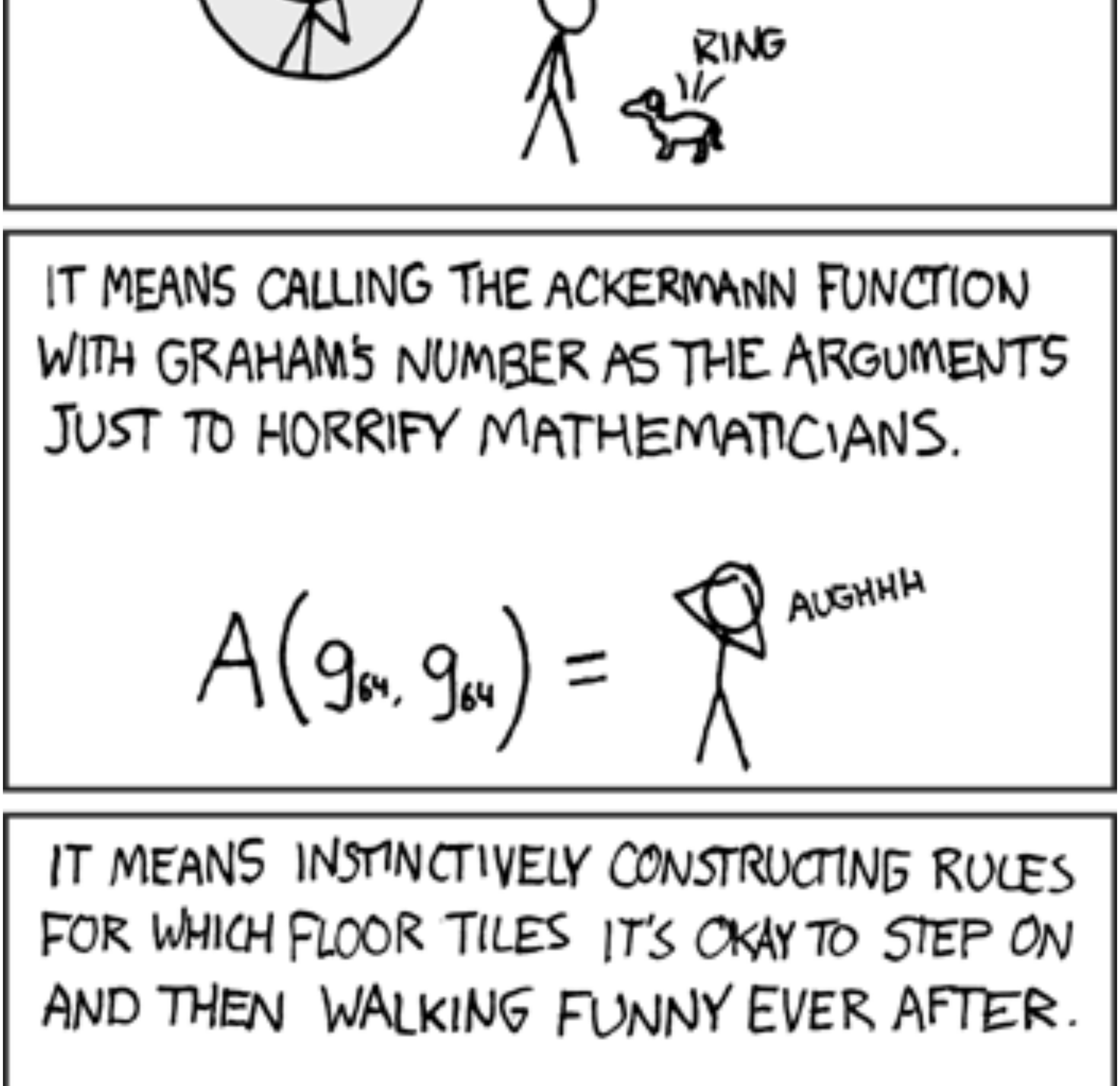
Unbounded Search

- Cost $c(z)$: number of queries $p(i)$ when answer is z
- There is a transfinite sequence of algorithms, each dramatically better than its predecessor.

WHAT DOES XKCD MEAN?






RING
IT MEANS CALLING THE ACKERMANN FUNCTION WITH GRAHAM'S NUMBER AS THE ARGUMENTS JUST TO HORRIFY MATHEMATICIANS.
$A(g_{64}, g_{64}) =$
AUGHHH
IT MEANS INSTINCTIVELY CONSTRUCTING RULES FOR WHICH FLOOR TILES IT'S OKAY TO STEP ON AND THEN WALKING FUNNY EVER AFTER.

Iterated Ackermann

- $A_1(n) := A(n,n)$
- $A_2(n) := A_1^n(n) = A_1(A_1(A_1(\ldots(n))))$
- …
- $A_k(n) := A_{k-1}^n(n)$

Knuth's Arrows

- $m\uparrow n = m^n$
- $m\uparrow\uparrow n = m\uparrow(m\uparrow(m\uparrow(m\uparrow\ldots\uparrow m)))$
- $m\uparrow^{k+1}n = m\uparrow^k(m\uparrow^k(m\uparrow^k(m\uparrow^k\ldots\uparrow^k m)))$

Cantor Normal Form

- 0, $\alpha+\beta$, ω^α
  - $n = \omega^0 + \omega^0 + \omega^0 + \ldots + \omega^0$
  - $\omega^\alpha n = \omega^\alpha + \omega^\alpha + \omega^\alpha + \ldots + \omega^\alpha$
- cnf: $\omega^\alpha n + \beta$
  - α, β in cnf; $\omega^\alpha n > \beta$
  - $\omega^{\alpha 1} + \omega^{\alpha 2} + \ldots + \omega^{\alpha n}$; $\alpha 1 \geq \alpha 2 \geq \ldots \geq \alpha n$

Fundamental Sequence

- $\lim_{n\to\omega} \lambda[n] = \lambda$
  - $(\alpha+\beta)[n] := \alpha+\beta[n]$
  - $\omega^{\alpha+1}[n] := \omega^{\alpha}n$
  - $\omega^{\lambda}[n] := \omega^{\lambda[n]}$

Fast Grzegorczyk

- $G_0(n) := n+1$
- $G_{\alpha+1}(n) := G_\alpha^{n+1}(n)$
- $G_\lambda(n) := G_{\lambda[n]}(n)$ (λ limit)

Hardy

- $H_0(n) := n$
- $H_{\alpha+1}(n) := H_\alpha(n+1)$
- $H_\lambda(n) := H_{\lambda[n]}(n)$ (λ limit)

Slow-Growing

- $g_0(n) := 0$
- $g_{\alpha+1}(n) := g_\alpha(n)+1$
- $g_\lambda(n) := g_{\lambda[n]}(n)$ (λ limit)

Gödel

- For any consistent axiomatization of arithmetic, there are true unprovable sentences.

Peano Arithmetic

- FO logic w/ =
- Numbers 0 and its successors
- $\forall_n \neg(s(n)=0)$
- $\forall_{m,n}\ s(m)=s(n) \Rightarrow m=n$
- $P(0) \wedge \forall_n(P(n) \Rightarrow P(s(n))) \Rightarrow \forall_n P(n)$

Definable

$F(x,z)$ defines $f(x)$ in L if

- $z=f(x)$ iff $F(x,z)$
- and these are provable:
  - $\forall_x \exists_z .\ F(x,z)$
  - $\forall_{x,z,z'} .\ F(x,z)\ \&\ F(x,z') \Rightarrow z=z'$

Gentzen

- The Peano axioms are consistent
  - Proof by ε_0 induction

Cut Elimination

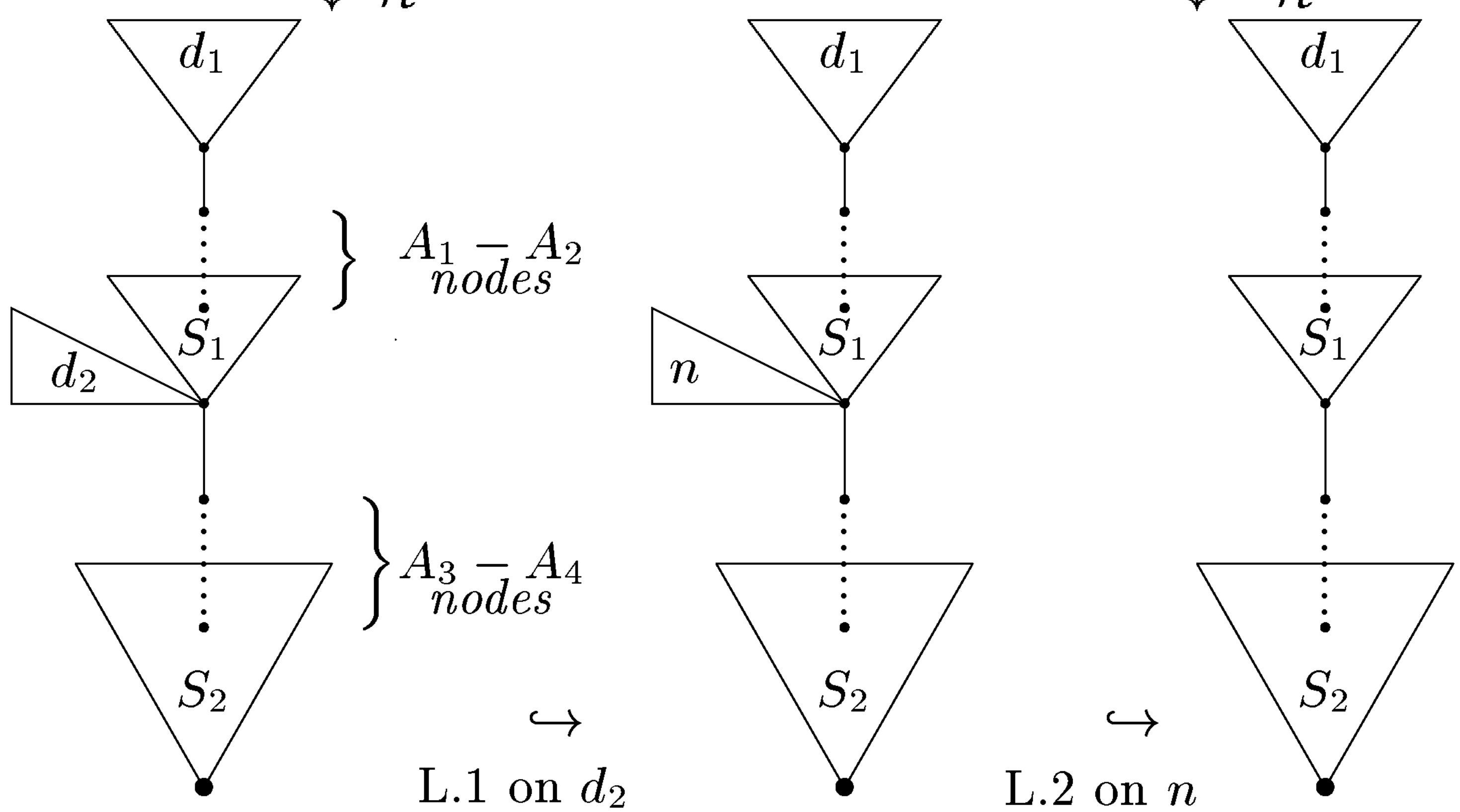
$D:$

$$
\begin{array}{ccc}
 & & D_1 \\
 & & \vdots \\
 & (A_1) & \dfrac{\Pi' \to \Xi'}{C, \Pi' \to \Xi'}\; I \\
D_2 & (A_2) & \\
\vdots & & \vdots\; s_1 \\
(A_3)\; \Gamma \to \Delta, C & & C, \Pi \to \Xi \\
\hline
(A_4) & \Gamma, \Pi \to \Delta, \Xi & I^* \\
 & \vdots\; s_2 & \\
 & \to &
\end{array}
$$

$D':$

$$
\begin{array}{cc}
 & D_1 \\
 & \vdots \\
(A_1) & \Pi' \to \Xi' \\
 & \overline{\quad exchanges \quad} \\
(A_2) & \Pi' \to \Xi' \\
 & \vdots\; s_1 \\
(A_3) & \Pi \to \Xi \\
 & \overline{\quad weakenings, exchanges \quad} \\
(A_4) & \Gamma, \Pi \to \Delta, \Xi \\
 & \vdots\; s_2 \\
 & \to
\end{array}
$$

$\downarrow \mathcal{H}$ $\qquad\qquad\qquad\qquad$ $\downarrow \mathcal{H}$



Conclusion

- There are true sentences about arithmetic that are not provable from the Peano axioms.
  - Hercules beats Hydra
  - Finitized Kruskal Theorem
  - Finitized Ramsey Theorem

Paris-Harrington

- $\forall$ n,k,m>0, $\exists$ N s.t. if we color each n-element subset of S = {1, 2, 3,..., N} with one of k colors, then $\exists$ Y $\subseteq$ S , |Y| $\geq$ m, such that all n element subsets of Y are monochrome, and |Y| $\geq$ min Y.

Finite Tree Theorem

- $\forall n \exists m$ s.t. for trees $T1,\ldots,Tm$, where each Tk has $k+n$ nodes, then $Ti \not\to Tj$ for some $i < j$.

Colored Finite Tree Theorem

- $\forall n \exists m$ s.t. for trees T1,…,Tm, where each Tk has up to k nodes, labeled in n colors, then Ti $\nrightarrow$ Tj for some i < j.

Kruskal Bound

- Tree(1) = 1 [length of sequence, 1 color]
- Tree(2) = 3
- Tree(3) > $2\uparrow$
$\uparrow$
$\uparrow$
$\uparrow$
$\uparrow\ldots$

Γ_0

- 0, $\alpha+\beta$, $\varphi_\alpha(\beta)$
  - $\varphi_0(\beta) = \omega^\beta$
  - $\varphi_{\alpha+1}(\beta) = \{\gamma : \varphi_\alpha(\gamma) = \gamma\}_\beta$
  - $\varphi_\lambda(\beta) = \lim_{\alpha<\lambda} \varphi_\alpha(\beta)$

Division

- A,B binary relations
- A/B is the relation s.t.
  - $(A/B) \circ B \subseteq A$
  - s (A/B) t if s A u for all u s.t. t B u

MPO

- $(f,\{b1,\ldots,bm\}) \triangleright b1,\ldots,bm$
- $(f,\{\})$
- $s > t$ if
    - $s \triangleright\geq t$ or
    - $s >_{lex} t$ and $s \not\triangleright t$

Abstract Path Order

- $s > t$ if
  - $s \rhd\geq t$ or
  - $s \gg t$ and $s >/\rhd\ t$
- $\rhd$ wfo
- $\rhd$ wfo escapes from $\gg$

Level i Subterm

- $\rhd_i$
- Subterm with i in node just above and >i from root to there

Ordinal Diagrams

- triples ⟨f,i,{b1,...,bm}⟩; think tree
- f: countably many, linearly ordered >
- level i: 1..N, linearly ordered >
- {...bi...} multiset of diagrams, ms order

Lexicographic Level

- $>_0$ is lexicographic
- $(f,i,x) >_0 (g,j,y)$ if
  - $f>g$
  - $f=g$, $i>j$
  - $f=g$, $i=j$, $x >_i y$

Higher Levels

- $s >_k t$ ($k>0$) if
  - $s \rhd_k \geq_k t$ or
  - $s >_{k-1} t$ and $s >_k / \rhd_k t$

Conditionals

$h(f(a)) \rightarrow c$

$h(x) \rightarrow k(x)$

$c \rightarrow k(f(a))$

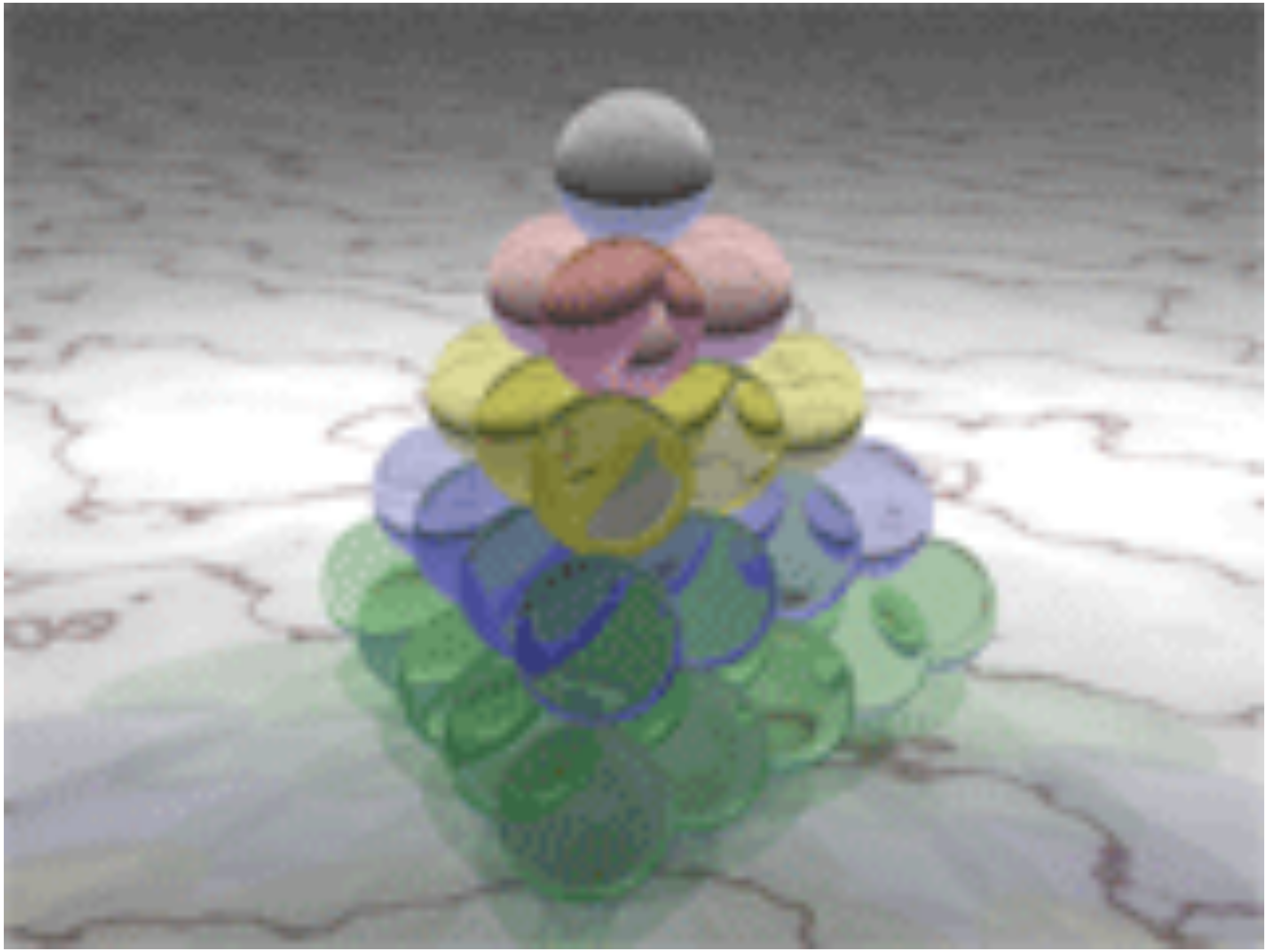
$a \rightarrow b$

$c \rightarrow k(g(b))$

$\underline{k(g(b)) \downarrow h(f(x))}$: $f(x) \rightarrow g(x)$

$$\sigma(h(t)) = (0, \sigma(t) * 2)$$
$$\sigma(f(t)) = (1, \sigma(t))$$
$$\sigma(c) = (0, (1, 1) * 1)$$
$$\sigma(k(t)) = (0, \sigma(t))$$
$$\sigma(a) = 1$$
$$\sigma(b) = 0$$
$$\sigma(g(t)) = (0, \sigma(t))$$

It's a Wrap



Kepler Conjecture

This is really the end