An introduction to Hoare-style program verification

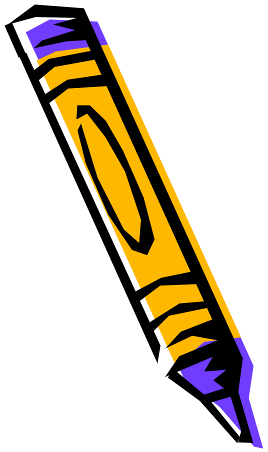
Swarat Chaudhuri

Adaptation of slides by K. Rustan M. Leino

Is this program correct?

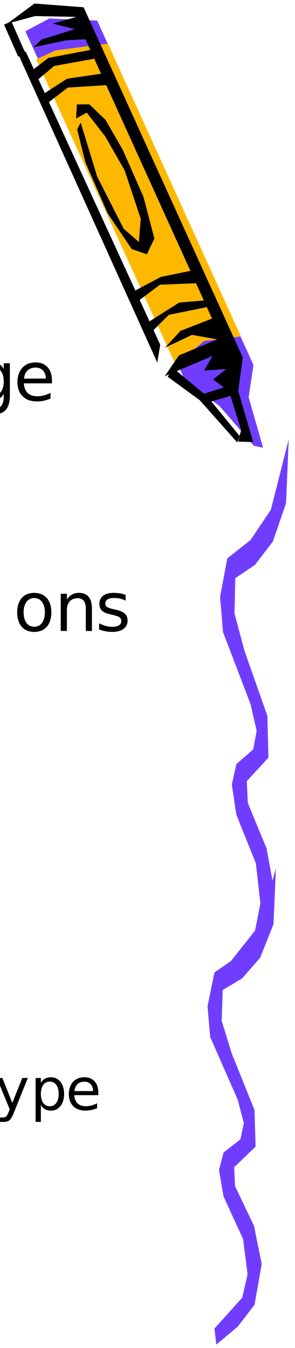
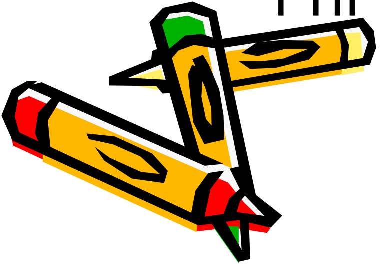
- How do we know?

```
int Find(float[] a, int m, int n,
float x) {
while (m < n) {
int j = (m+n) / 2;
if (a[j] < x) {
m = j+1;
} else if (x < a[j]) {
n = j-1;
} else {
return j;
}
}
return -1;
}
```



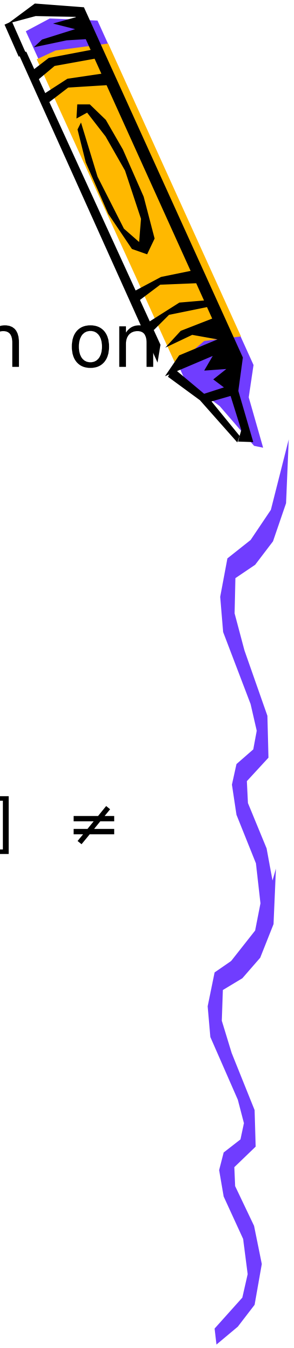
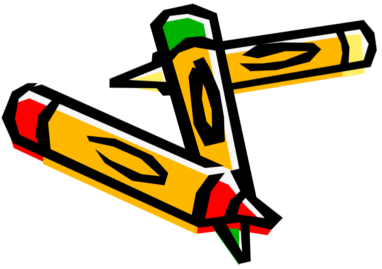
Making sense of programs

- Program semantics defines a language
  - e.g., Hoare logic, Dijkstra's weakest preconditions
- Specifications record design decisions
  - Generalization of type annotations
- Tools amplify human effort
  - manage details
  - find inconsistencies
  - ensure quality
  - you have already seen type checking, type inference

State predicates

- A predicate is a boolean function on the program state
- Examples:
  - $x = 8$
  - $x < y$
  - $m \leq n \Rightarrow (\forall j \mid 0 \leq j < a.\mathrm{length} \cdot a[j] \neq \mathrm{NaN})$
  - true
  - false

Hoare triples

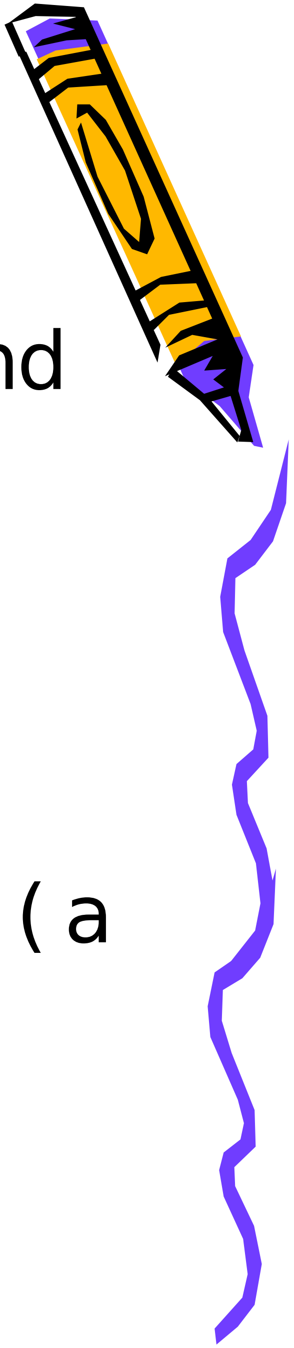
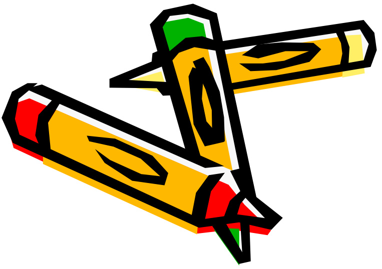
- For any predicates P and Q and any program S,

$$\{P\}\ S\ \{Q\}$$

postcondition

precondition

says that if S is started in (a state satisfying) P, then it terminates in Q

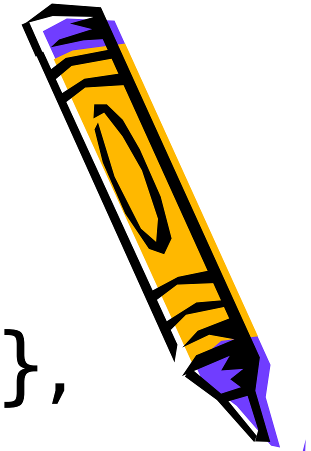
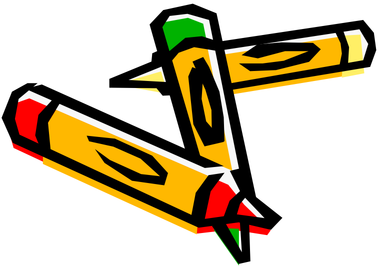



Examples

- {true} x := 12 {x = 12}
- {x < 40} x := 12 {10 ≤ x}
- {x < 40} x := x+1 {??}
- {m ≤ n} j := (m+n)/2 {??}
- {0 ≤ m < n ≤ a.length ∧ a[m] = x}
  r := Find(a, m, n, x)
  {??} m ≤ r
- {false} S {$x^n + y^n = z^n$}

Precise triples

- If {P} S {Q} and {P} S {R}, then does
  {P} S {Q ∧ R}
  hold?

Precise triples

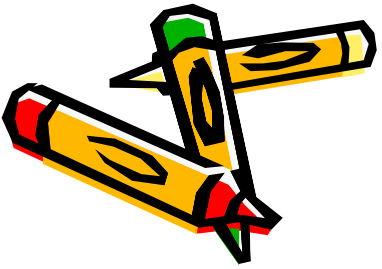
- If {P} S {Q} and {P} S {R}, then does

  {P} S {Q ∧ R}

  hold? yes
- The most precise Q such that

  {P} S {Q}

  is called the strongest postcondition of S with respect to P.



Weakest preconditions

- If {P} S {R} and {Q} S {R}, then

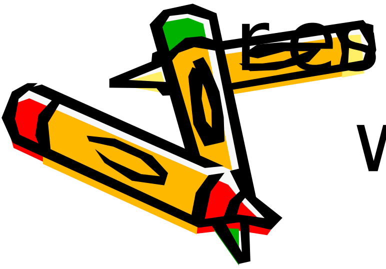
  {P v Q} S {R}

  holds.
- The most general P such that

  {P} S {R}

  is called the weakest precondition of S with respect to R,

  written wp(S, R)

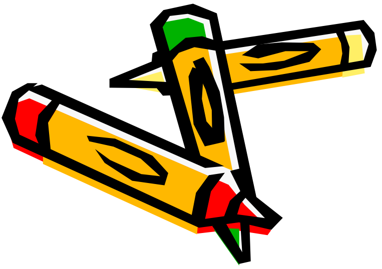


Triples and wp

$\{P\}\ S\ \{Q\}$

if and only if

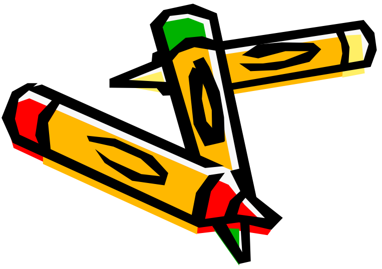
$P \Rightarrow wp(S, Q)$



Program semantics

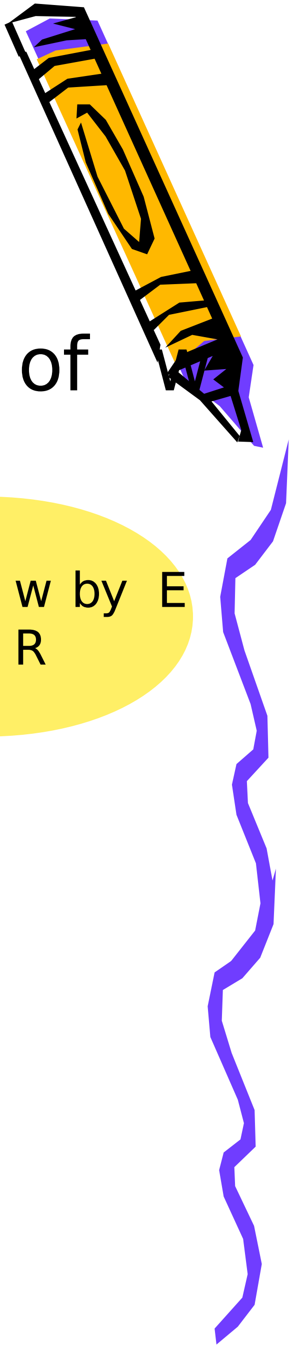
– skip

- no-op
- wp(skip, R) ≡ R
- wp(skip, $x^n + y^n = z^n$)
  ≡ $x^n + y^n = z^n$



Program semantics

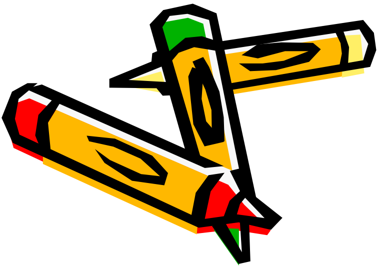
– assignment



- evaluate E and change value of w to E
- wp(w := E, R) ≡ R[w

replace w by E in R

- wp(x := x + 1, $x \leqq 10$)
  ≡ $x+1 \leqq 10$
  ≡ $x < 10$
- wp(x := 15, $x \leqq 10$)
- wp(y := x + 3*y, $x \leqq 10$)
- wp(x, y := y, x, $x < y$)



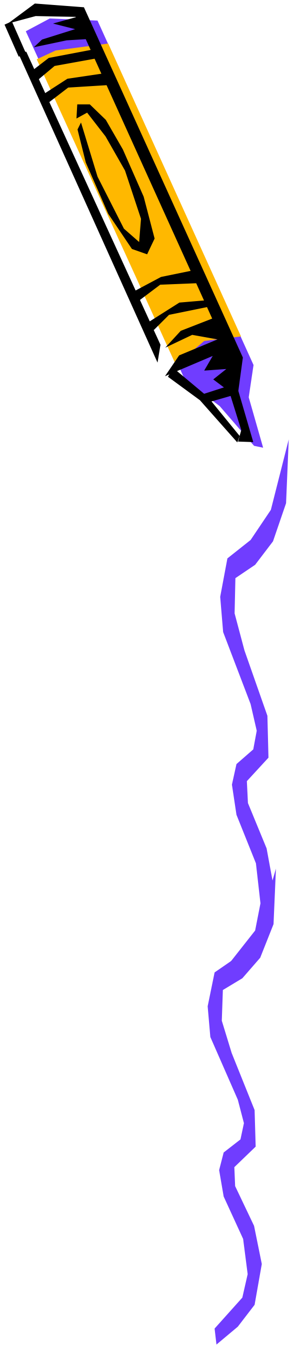
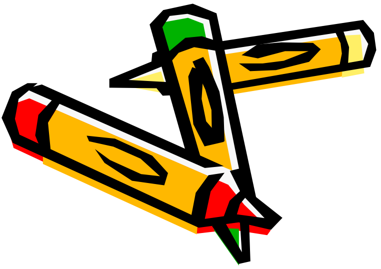
Program semantics

– assert

- if P holds, do nothing, else don't terminate
- wp(assert P, R) ≡ P ∧ R

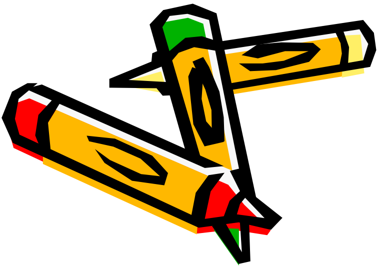
- wp(assert x < 10, 0 ≦ x)

  ≡ 0 ≦ x < 10
- wp(assert x = y*y, 0 ≦ x)
- wp(assert false, x ≦ 10)

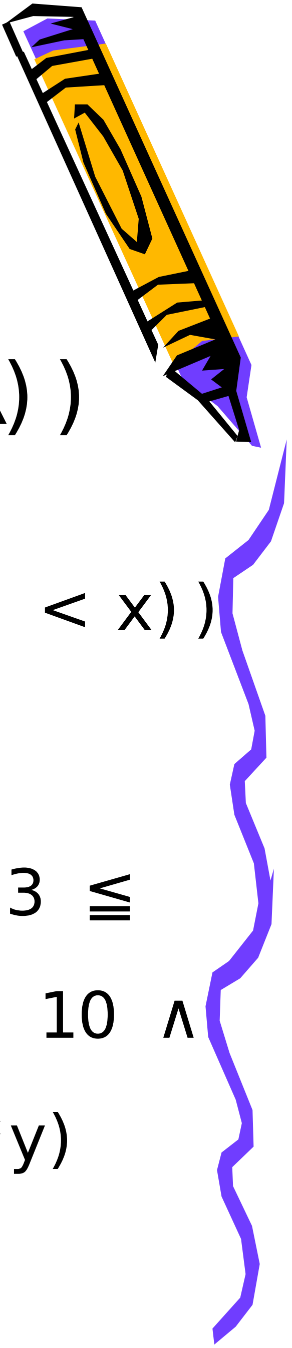



Program compositions

- If $\{P\}\ S\ \{Q\}$ and $\{Q\}\ T\ \{R\}$, then $\{P\}\ S;\ T\ \{R\}$
- If $\{P \land B\}\ S\ \{R\}$ and $\{P \land \neg B\}\ T\ \{R\}$, then $\{P\}$ if B then S else T end $\{R\}$



Program semantics – sequential composition

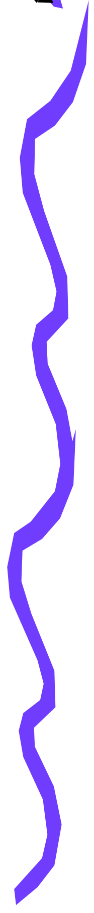


- wp(S; T, R) ≡ wp(S, wp(T, R))
- wp(x := x+1 ; assert x ≦ y, 0 < x)
  ≡ wp(x := x+1, wp(assert x ≦ y, 0 < x))
  ≡ wp(x := x+1, 0 < x ≦ y)
  ≡ 0 < x+1 ≦ y
  ≡ 0 ≦ x < y
- wp(y := y+1 ; x := x + 3*y, y ≦ 10 ∧ 3 ≦ x)
  ≡ wp(y := y+1, wp(x := x+3*y, y ≦ 10 ∧ 3 ≦ x))
  ≡ wp(y := y+1, y ≦ 10 ∧ 3 ≦ x+3*y)
  ≡ y+1 ≦ 10 ∧ 3 ≦ x+3*(y+1)
  ≡ y < 10 ∧ 3 ≦ x + 3*y + 3
  ≡ y < 10 ∧ 0 ≦ x + 3*y

Program semantics —conditional composition

- $wp(\text{if } B \text{ then } S \text{ else } T \text{ end}, R) \equiv$
  $(B \Rightarrow wp(S, R)) \wedge (\neg B \Rightarrow wp(T, R)) \equiv$
  $(B \wedge wp(S, R)) \vee (\neg B \wedge wp(T, R))$

- $wp(\text{if } x < y \text{ then } z := y \text{ else } z := x \text{ end}, 0 \leqq z)$
  $\equiv \quad (x < y \wedge wp(z := y, 0 \leqq z)) \vee$
  $(\neg(x < y) \wedge wp(z := x, 0 \leqq z))$
  $\equiv \quad (x < y \wedge 0 \leqq y) \vee (y \leqq x \wedge 0 \leqq x)$
  $\equiv \quad 0 \leqq y \vee 0 \leqq x$
- $wp(\text{if } x{\neq}10 \text{ then } x := x{+}1 \text{ else } x := x + 2 \text{ end}, x \leqq 10)$
  $\equiv \quad (x{\neq}10 \wedge wp(x := x{+}1, x \leqq 10)) \vee$
  $(\neg(x{\neq}10) \wedge wp(x := x{+}2, x \leqq 10))$
  $\equiv \quad (x{\neq}10 \wedge x{+}1 \leqq 10) \vee (x{=}10 \wedge x{+}2 \leqq 10)$
  $\equiv \quad (x{\neq}10 \wedge x < 10) \vee \text{false}$
  $\equiv \quad x < 10$



Example

(x != null ==> x != null && x.f >= 0) &&
(x == null ==> z-1 >= 0)

```
if (x != null) {
    n = x.f;
} else {
    n = z-1;
    z++;
}
a = new char[n];
```

x != null && x.f >= 0

z-1 >= 0

n >= 0

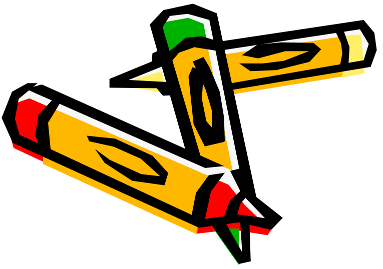
true

A good exercise

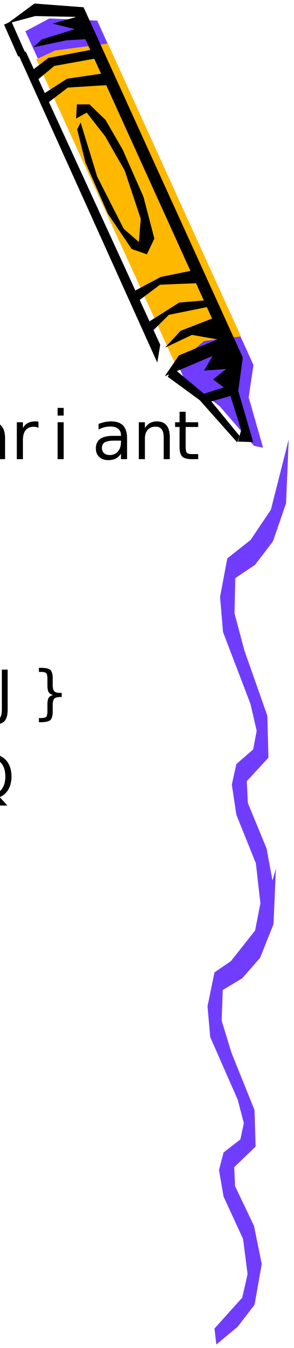
Define

change w such that P

by giving its weakest precondition



Loops



To prove

$\{P\}$ while B do S end $\{Q\}$

find invariant J and well-founded variant function vf such that:

- invariant holds initially: $P \Rightarrow J$
- invariant is maintained: $\{J \wedge B\}\ S\ \{J\}$
- invariant is sufficient: $J \wedge \neg B \Rightarrow Q$
- variant function is bounded:
  $J \wedge B \Rightarrow 0 \leqq vf$
- variant function decreases:
  $\{J \wedge B \wedge vf=VF\}\ S\ \{vf<VF\}$

Review

- {P} skip {P}
- {P[w := E]} w := E {P}
- {P ∧ B} assert B {P}
- if {P} S {Q} and {Q} T {R}, then {P} S ; T {R}
- if {P ∧ B} S {R} and {P ∧ ¬B} T {R}, then {P} if B then S else T end {R}

Loops

To prove

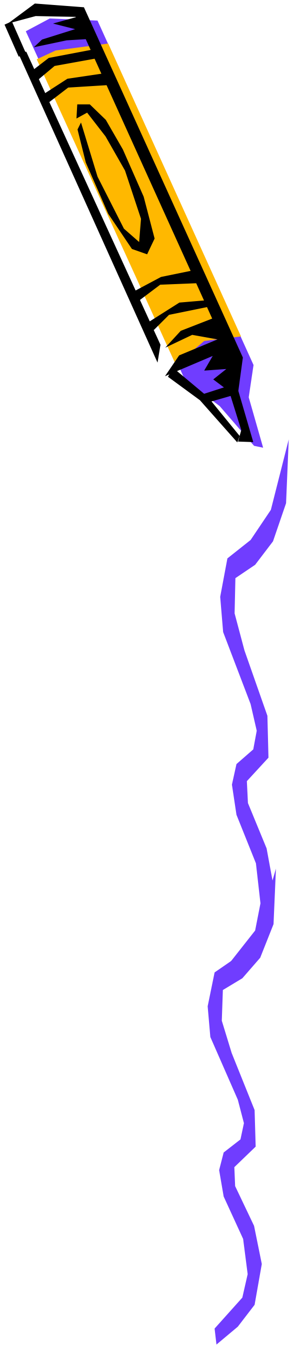
$\{P\}$ **while** B **do** S **end** $\{Q\}$

prove

```
$\{P\}$ $\{J\}$
while $B$ do
    $\{J \land B\}$ $\{0 \leqq vf\}$
    $\{J \land B \land vf = VF\}$ $S$ $\{J \land vf < VF\}$
end
$\{J \land \neg B\}$ $\{Q\}$
```

Example: Array sum

```
{$0 \leqq N$}
k := 0; s := 0;
while $k \neq N$
do s:=s+a[k];
end k:=k+1
{$s = (\Sigma i \mid 0 \leqq i < N \cdot a[i])$}
```



Example: Array sum

```
{0≤N} k := 0; s := 0; {J}
while k ≠ N do
  k := 0; s := 0;
}
while k ≠ N  {J ∧ k≠N}  {0 ≤ vf}
do {J ∧ k≠N ∧ vf = VF}  s := s+a[k];
end  s := s+a[k] ; k := k+1
     k := k+1
{J ∧ vf < VF}
{s = (Σi | 0≤i<N • a[i])}
end
{J ∧ ¬(k≠N)} {s = (Σi | 0≤i<N
• a[i])}
```

Example: Array sum

```
$\{0 \leqq N\}$ k := 0; s := 0; $\{J\}$
while k ≠ N do
    $\{J \wedge k \neq N\}$ $\{0 \leqq vf\}$
    $\{J \wedge k \neq N \wedge vf = VF\}$
      s:=s+a[k] ; k:=k+1
    $\{J \wedge vf < VF\}$
end
$\{J \wedge \neg(k \neq N)\}$ $\{s = (\Sigma i \mid 0 \leqq i < N$
$\bullet\ a[i])\}$
```

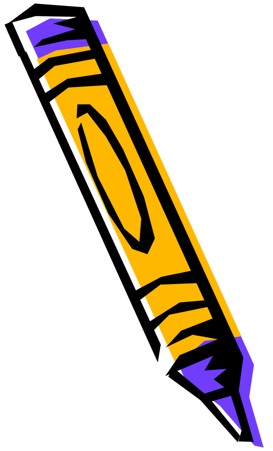
Example: Array sum

```
{$0 \leq N$} k := 0; s := 0; {J}
while $k \neq N$ do
    {$J \wedge k \neq N$} {$0 \leq vf$}
    {$J \wedge k \neq N \wedge vf = VF$}
      s:=s+a[k] ; k:=k+1
    {$J \wedge vf < VF$}
end
{$J \wedge k = N$} {$s = (\Sigma i \mid 0 \leq i < N \cdot a[i])$}
```

- J : $s = (\Sigma i \mid 0 \leq i < k \cdot a[i]) \wedge 0 \leq k \leq N$
- vf : $N - k$

Example: Array sum

Initialization



$\{0 \leqq N\}$

$\{0 = (\Sigma i \mid 0 \leqq i < 0 \cdot a[i]) \wedge 0 \leqq 0 \leqq N\}$

$k := 0;$

$\{0 = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \wedge 0 \leqq k \leqq N\}$

$s := 0;$

$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \wedge 0 \leqq k \leqq N\}$

Example: Array sum -Invariance

$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \land 0 \leqq k \leqq N \land k \neq N$
$\land\ N-k=VF\}$

$\{s+a[k] = (\Sigma i \mid 0 \leqq i < k \cdot a[i])+a[k] \land 0 \leqq k < N$
$\land\ N-k-1<VF\}$

$s := s + a[k];$

$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i])+a[k] \land 0 \leqq k < N$
$\land\ N-k-1<VF\}$

$\{s = (\Sigma i \mid 0 \leqq i < k+1 \cdot a[i]) \land 0 \leqq k+1 \leqq N$
$\land\ N-(k+1)<VF\}$

$k := k+1;$

$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \land 0 \leqq k \leqq N \land N-k<VF\}$

In-class exercise: computing cubes

```
{$0 \leqq N$}
k := 0;   r := 0;   s := 1;   t := 6;
while $k \neq N$ do
    a[k] := r;
    r := r + s;
    s := s + t;
    t := t + 6;
    k := k + 1
end
{$(\forall i \mid 0 \leqq i < N \cdot a[i] = i^3)$}
```

Computing cubes
– Guessing the invariant

- From the postcondition
  $(\forall i \mid 0 \leq i < N \cdot a[i] = i^3)$
  and the negation of the guard
  $k = N$
  guess the invariant
  $(\forall i \mid 0 \leq i < k \cdot a[i] = i^3) \wedge 0 \leq k \leq N$
- From this invariant and variant function $N-k$, it follows that the loop terminates

Computing cubes — Maintaining the invariant

```
while $k \neq N$ do
    $\{(\forall i \mid 0 \leq i < k \cdot a[i] = i^3) \wedge 0 \leq k \leq N \wedge k \neq N\}$
    $\{(\forall i \mid 0 \leq i < k \cdot a[i] = i^3) \wedge r = k^3 \wedge 0 \leq k < N\}$
    a[k] := r;
    r := r + s;
    s := s + t;
    t := t + 6;
    $\{(\forall i \mid 0 \leq i < k \cdot a[i] = i^3) \wedge$
    $\{(\forall i \mid 0 \leq i < k+1 \cdot a[i] = i^3) \wedge 0 \leq k+1 \leq N\}$
    k := k + 1
    $\{(\forall i \mid 0 \leq i < k \cdot a[i] = i^3) \wedge 0 \leq k \leq N\}$
end
```

Add this to the invariant, and then try to prove that it is maintained

Computing cubes

– Maintaining the invariant

```
while k≠N do
   {$r = k^3 \wedge$ ..}
   {$r + s = k^3 + 3*k^2 + 3*k + 1$}
   a[k] := r;
   r := r + s;
   s := s + t;
   t := t + 6;
   {$r = k^3 + 3*k^2 + 3*k + 1$}
   {$r = (k+1)^3$}
   k := k + 1
   {$r = k^3$}
end
```

Add
$s = 3*k^2 + 3*k + 1$
to the invariant, and then try to prove that it is maintained

Computing cubes

~~—~~Maintaining the invariant

```
while k≠N do
    {s = 3*k$^2$ + 3*k + 1 ∧ ..}
    {s + t = 3*k$^2$ + 6*k + 3 + 3*k + 3 + 1}
    a[k] := r;
    r := r + s;
    s := s + t;
    t := t + 6;
    {s = 3*k$^2$ + 6*k + 3 + 3*k + 3 + 1}
    {s = 3*(k+1)$^2$ + 3*(k+1) + 1}
    k := k + 1
    {s = 3*k$^2$ + 3*k + 1}
end
```

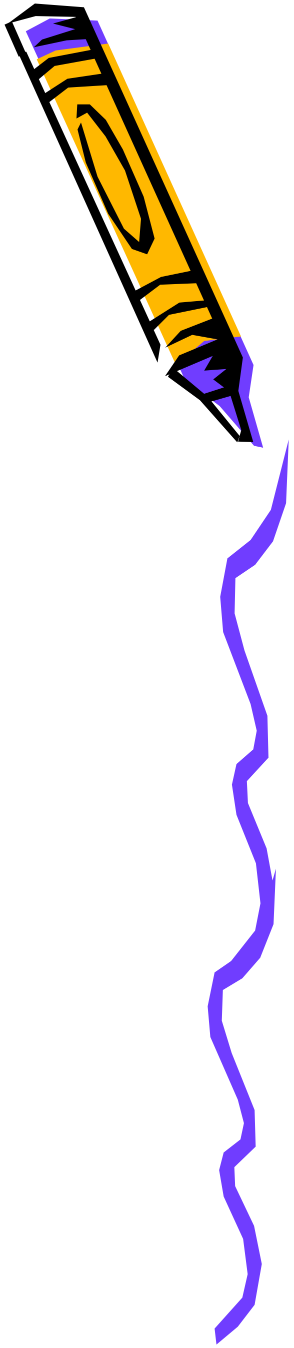
Add

$t = 6*k + 6$

to the invariant, and then try to prove that it is maintained

Computing cubes – Maintaining the invariant

```
while $k \neq N$ do
    {$t = 6*k + 6 \wedge ...$}
    {$t + 6 = 6*k + 6 + 6$}
    a[k] := r;
    r := r + s;
    s := s + t;
    t := t + 6;
    {$t = 6*k + 6 + 6$}
    {$t = 6*(k+1) + 6$}
    k := k + 1
    {$t = 6*k + 6$}
end
```



Computing cubes — Establishing the invariant

$\{0 \leqq N\}$

$\{(\forall i \mid 0 \leqq i < 0 \cdot a[i] = i^3) \wedge 0 \leqq 0 \leqq N \wedge$
$\quad 0 = 0^3 \wedge$
$\quad 1 = 3*0^2 + 3*0 + 1 \wedge$
$\quad 6 = 6*0 + 6\}$

k := 0; r := 0; s := 1; t := 6;

$\{(\forall i \mid 0 \leqq i < k \cdot a[i] = i^3) \wedge 0 \leqq k \leqq N \wedge$
$\quad r = k^3 \wedge$
$\quad s = 3*k^2 + 3*k + 1 \wedge$
$\quad t = 6*k + 6\}$

In-class exercise: computing cubes

– Answers

- Invariant:

  $(\forall i \mid 0 \leq i < k \cdot a[i] = i^3)$
  $\wedge$
  $0 \leq k \leq N \wedge$
  $r = k^3 \wedge$
  $s = 3*k^2 + 3*k + 1 \wedge$
  $t = 6*k + 6$

- Variant function:

  $N-k$

Chips: Does it terminate?

- You have a bag of R, Y, B chips.
- If one chip remains, you take it out.
- Otherwise, remove two chips at random:
  - If one of the two chips is R, you do not put chips back in bag.
  - If both are Y, you put one Y and five B chips in bag.
  - If one chip is B and the other is not R, put ten R chips in bag.

Variant function

- Lexicographic ordering:

  (#Y, #B, #R)

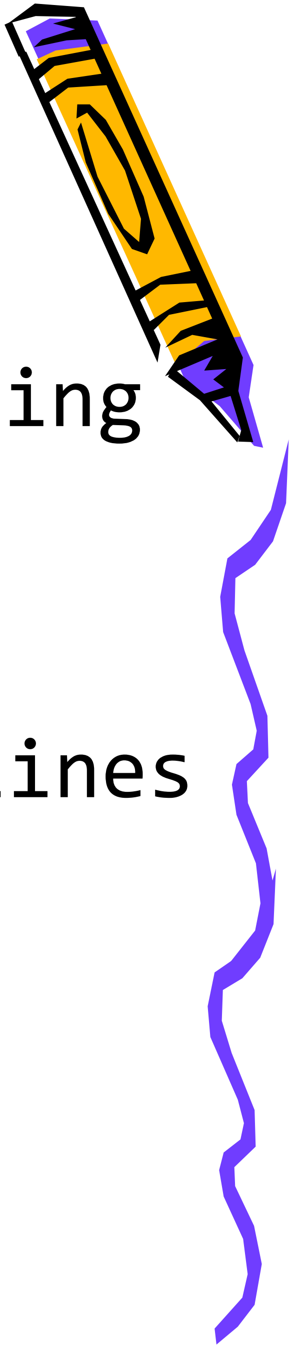
Dijkstra's map problem

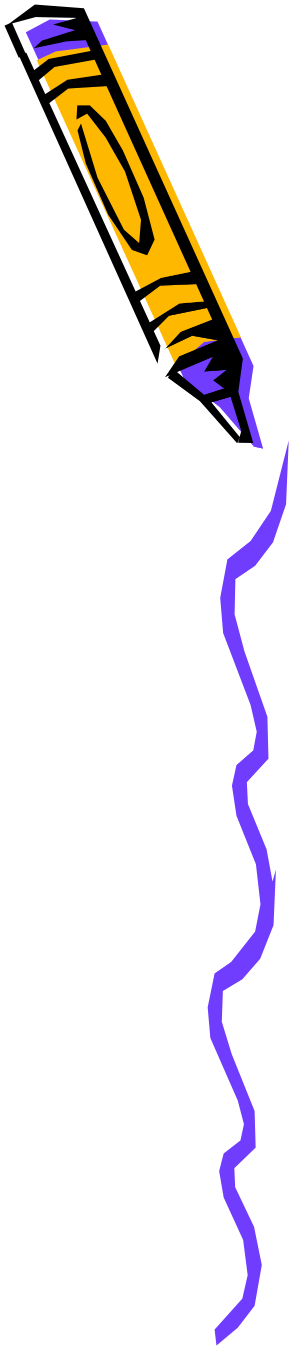
- Given
  - two sets of points in R^2 of equal cardinality
- Find
  - A one-to-one mapping such that mapping lines do not cross in R^2

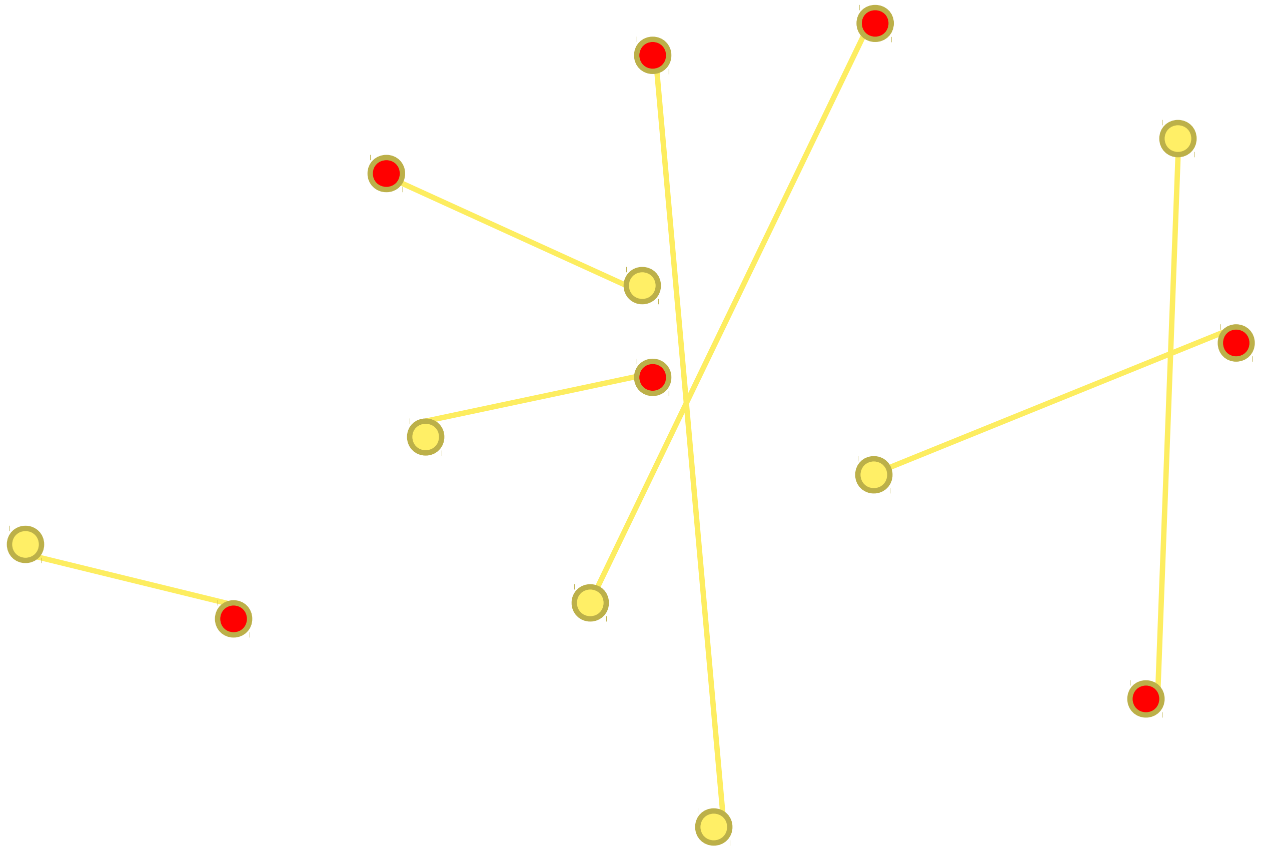
Example: Proposed algorithm

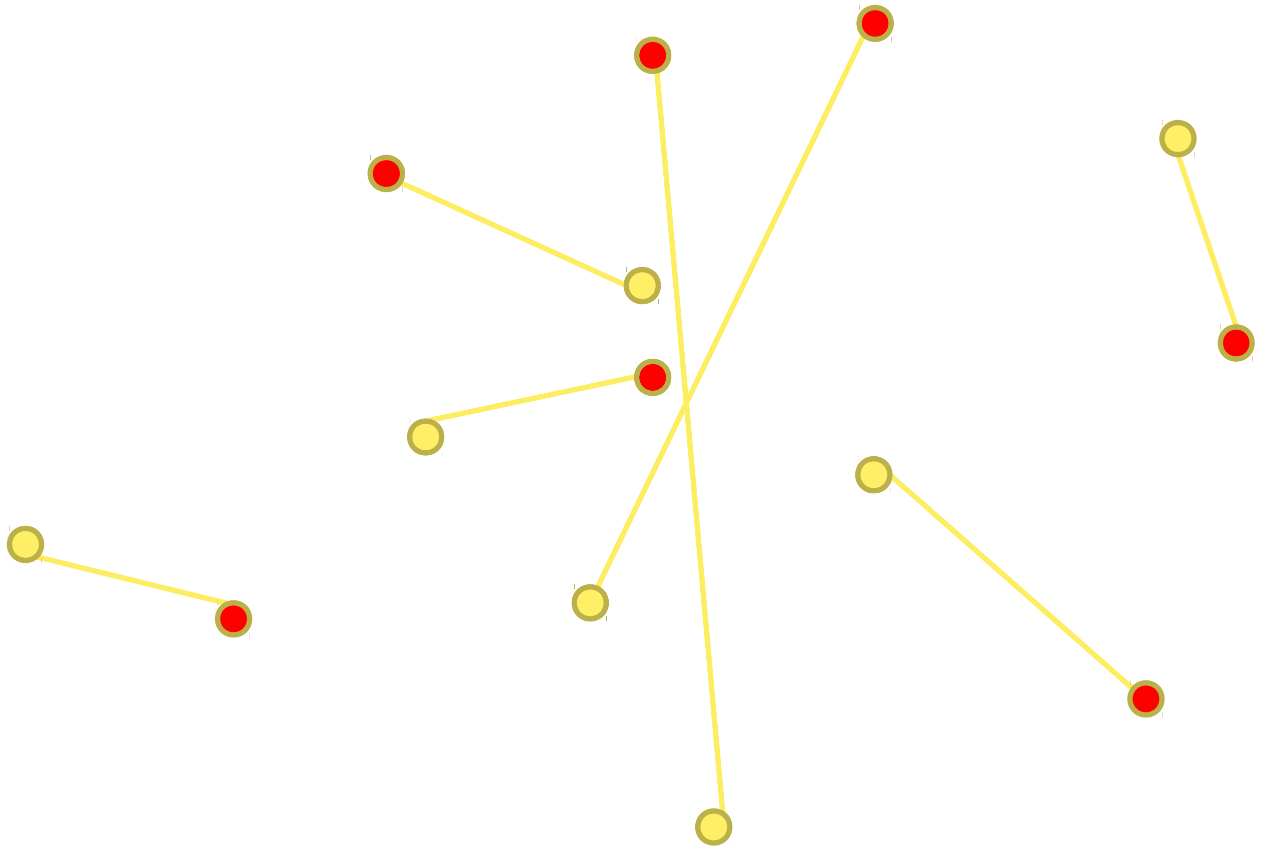
```
map = choose any one-to-one mapping

while ( exists crossing)
    uncross a pair of crossing lines
```







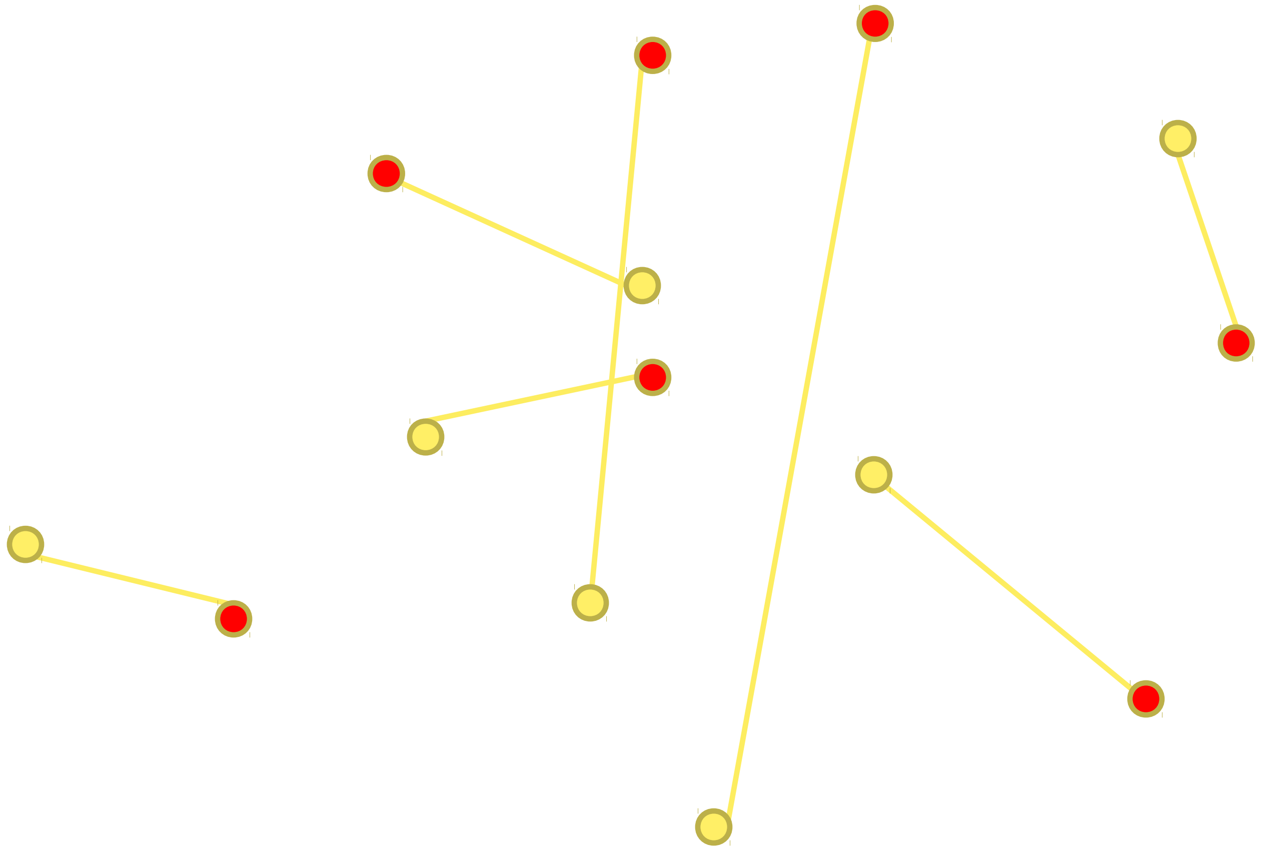


Correctness

- Is this algorithm correct when it terminates (partial correctness)?
- Does this algorithm terminate?

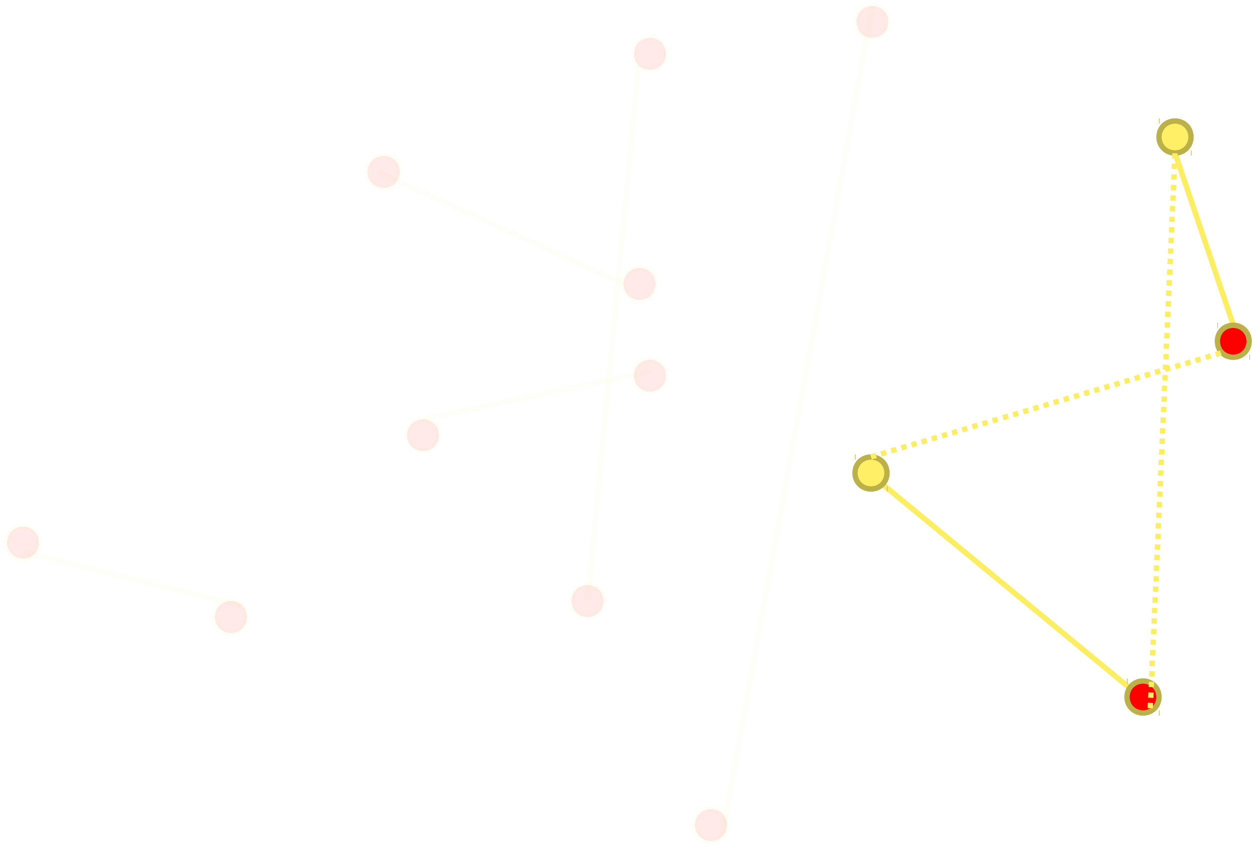
Correctness

- Is this algorithm correct when it terminates (partial correctness) ?
- Does this algorithm terminate?
  - May be the number of crossing diminishes at each iteration?

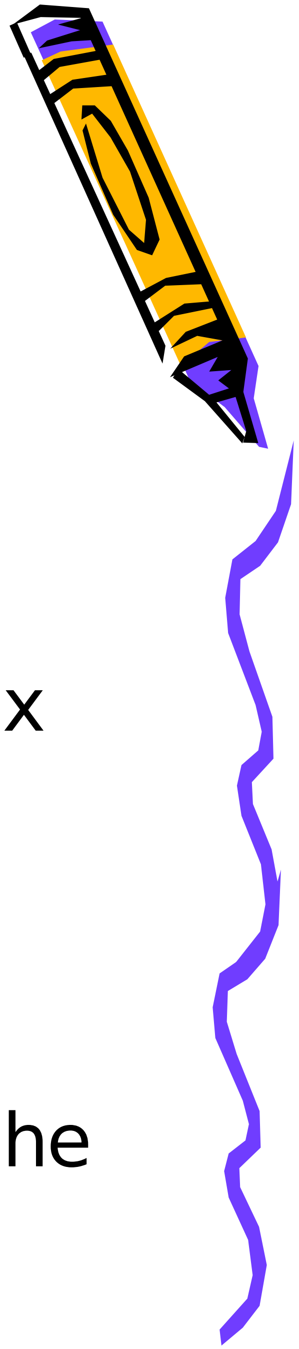


Correctness

- Is this algorithm correct when it terminates (partial correctness)?
- Does this algorithm terminate?
  - May be the number of crossing diminishes at each iteration? No
  - May be the total line length decreases



Other common invariants

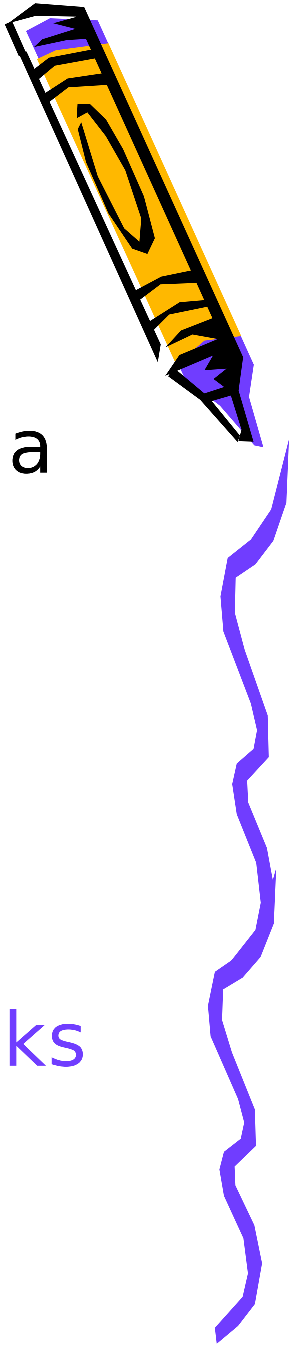


- k is the number of nodes traversed so far
- the current value of n does not exceed the initial value of n
- all array elements with an index less than j are smaller than x
- the number of processes whose program counter is inside the critical section is at most one
- the only principals that know the key K are A and B

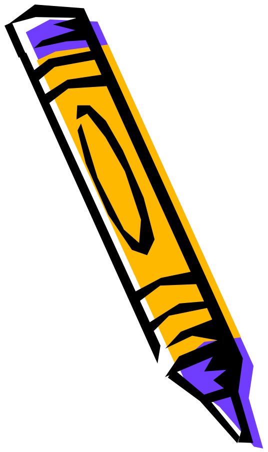
Belgian chocolate

- How many breaks do you need to make 50 individual pieces from a 10x5 Belgian chocolate bar?
- Note: Belgian chocolate is so thick that you can't break two pieces at once.

- Invariant: #pieces = 1 + #breaks



Loop proof obligations – a closer look



To prove

$\{P\}$ while B do S end $\{Q\}$

find invariant J and variant function vf such that:

- invariant initially: $P \Rightarrow J$

Are all of these conditions needed?

- invariant maintained: $\{J \land B\}\ S\ \{J\}$
- invariant sufficient: $J \land \neg B \Rightarrow Q$
- vf well-founded
- vf bounded: $J \land B \Rightarrow 0 \leqq vf$
- vf decreases: $\{J \land B \land vf = VF\}\ S\ \{vf < VF\}$

Loop p
obligations

– invariant holds initially

```
$\{0 \leqq N\}$ k := N; s := 0; $\{J\}$
while $k \neq N$ do
    $\{J \wedge k \neq N\}$ $\{0 \leqq vf\}$
    $\{J \wedge k \neq N \wedge vf = VF\}$
      s:=s+a[k]; k:=k+1
    $\{J \wedge vf < VF\}$
end
$\{J \wedge \neg(k \neq N)\}$ $\{s = (\Sigma i \mid 0 \leqq i < N \cdot a[i])\}$
```

J: $s = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \wedge 0 \leq k \leq N$

Loop obligations

– invariant is maintained

```
{0≤N} k := 0; s := 0; {J}
while k ≠ N do
    {J ∧ k≠N} {0 ≤ vf}
    {J ∧ k≠N ∧ vf=VF}
      s:=s+a[k] ; k:=k+2
    {J ∧ vf<VF}
end
{J ∧ ¬(k≠N)} {s = (Σi | 0≤i<N • a[i])}
```

J: $s = (\Sigma i \mid 0 \le i < k \cdot a[i]) \wedge 0 \le k \le N$

obligations

- invariant is sufficient

```
{0≤N} k := 0; s := 0; {J}
while k ≠ N do
    {J ∧ k≠N} {0 ≤ vf}
    {J ∧ k≠N ∧ vf=VF}
          k:=k+1
    {J ∧ vf<VF}
end
{J ∧ ¬(k≠N)} {s = (Σi | 0≤i<N · a[i])}
```

J : $0 \leq k \leq N$

vf : $N-k$

obligations

– variant function is well-founded

```
{0≤N} k := 0; s := 0; r := 1.0;
{J}
while k ≠ N do
    {J ∧ k≠N} {0 ≤ vf}
    {J ∧ k≠N ∧ vf=VF}
      r := r / 2.0;
    {J ∧ vf<VF}
end
{J ∧ ¬(k≠N)} {s = (Σi | 0≤i<N ·
a[i])}
J:   s=(Σi | 0≤i<k ·  a[i]) ∧ 0≤r
```

obligations

– variant function is bounded

```
$\{0 \le N\}$ k := 0; s := 0; $\{J\}$
while $k \ne N$ do
    $\{J \wedge k \ne N\}$ $\{0 \le vf\}$
    $\{J \wedge k \ne N \wedge vf = VF\}$
    k := k-1
    $\{J \wedge vf < VF\}$
end
$\{J \wedge \neg(k \ne N)\}$ $\{s = (\Sigma i \mid 0 \le i < N \cdot a[i])\}$
```

J: $s = (\Sigma i \mid 0 \le i < k \cdot a[i]) \wedge k \le N$

obligations

—variant function decreases

```
$\{0 \le N\}$ k := 0; s := 0; $\{J\}$
while k ≠ N do
    $\{J \land k \neq N\}$ $\{0 \le vf\}$
    $\{J \land k \neq N \land vf = VF\}$
    skip
    $\{J \land vf < VF\}$
end
$\{J \land \neg(k \neq N)\}$ $\{s = (\Sigma i \mid 0 \le i < N \cdot a[i])\}$
```

J: $s = (\Sigma i \mid 0 \le i < k \cdot a[i]) \land 0 \le k \le N$

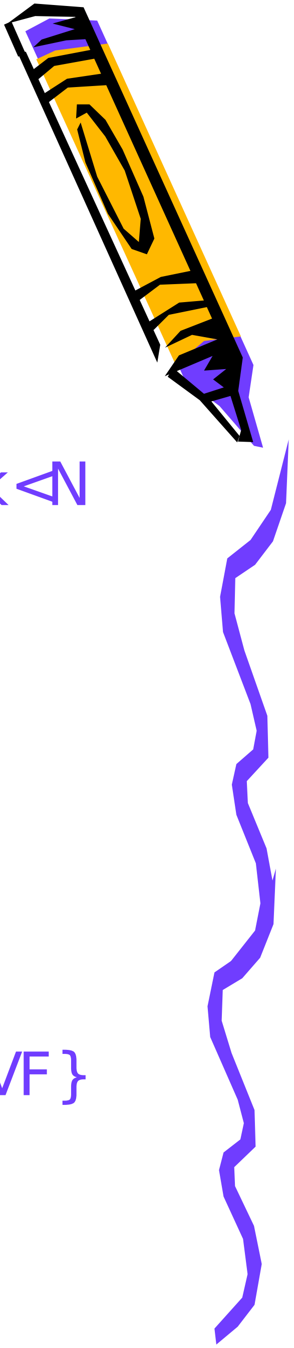
Ranges in invariants

```
{0≦N} k := 0; s := 0; {J}
while k ≠ N do
    {J ∧ k≠N} {0 ≦ vf}
    {J ∧ k≠N ∧ vf=VF}
      s:=s+a[k]; k:=k+1
    {J ∧ vf<VF}
end
{J ∧ ¬(k≠N)} {s = (Σi | 0≦i<N · a[i])}
```

Where are these used?

J: s = (Σi | 0≦i<k · a[i]) ∧ 0≤k≤N

Ranges: lower bound



$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \wedge 0 \leqq k \leqq N \wedge k \neq N$
$\wedge\ N-k=VF\}$

$\{s+a[k] = (\Sigma i \mid 0 \leqq i < k \cdot a[i])+a[k] \wedge 0 \leqq k < N$
$\wedge\ N-k-1<VF\}$

$s := s + a[k];$

$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i])+a[k] \wedge 0 \leqq k < N$
$\wedge\ N-k-1<VF\}$

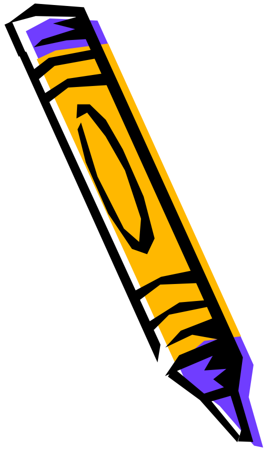
$\{s = (\Sigma i \mid 0 \leqq i < k+1 \cdot a[i]) \wedge 0 \leqq k+1 \leqq N$
$\wedge\ N-(k+1)<VF\}$

This step uses $0 \leqq k$

$k := k+1;$

$\{s = (\Sigma i \mid 0 \leqq i < k \cdot a[i]) \wedge 0 \leqq k \leqq N \wedge N-k<VF\}$

Ranges: upper bound



```
{0≤N} k := 0; s := 0; {J}
while k≠N do   This step uses k≤N
    {J ∧ k≠N} {0 ≤ vf}
    {J ∧ k≠N ∧ vf=VF}
      s:=s+a[k]; k:=k+1
    {J ∧ vf<VF}
end
{J ∧ ¬(k≠N)} {s = (Σi | 0≤i<N • a[i])}
```

J: $s = (\Sigma i \mid 0 \leq i < k \cdot a[i]) \wedge 0 \leq k \leq N$

Ranges: upper bound

```
{$0 \le N$} k := 0; s := 0; {J}
while k < N do
    {J $\land$ k<N} {$0 \le vf$}
    {J $\land$ k<N $\land$ vf=VF}
        s:=s+a[k] ; k:=k+1
    {J $\land$ vf<VF}
end
{J $\land \neg$(k<N)} {s = ($\Sigma$i | $0 \le i < N$ · a[i])}
```

Even with < instead of ≠

this step still needs k≤N

J: s = (Σi | $0 \le i < k$ · a[i]) $\land$ $0 \le k \le N$