Program Verification Mechanics

Reduction of program verification to logic;

Three programs: Linear Search, Binary Search, Bubble Sort exhibiting loops, nested loops and recursion;

Basic paths; weakest precondition; verification conditions; termination using ranking functions to well-founded orderings

A decidable theory of arrays

```
@pre 0 ≤ ℓ ∧ u < |a|
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool LinearSearch(int[] a, int ℓ, int u, int e) {
  for @ ⊤
    (int i := ℓ; i ≤ u; i := i + 1) {
    if (a[i] = e) return true;
  }
  return false;
}
```

Fig. 5.6. LinearSearch with function specification

$x \text{ div } d = \lfloor \frac{x}{d} \rfloor$

```
@pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, ℓ, u)
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    int m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BinarySearch(a, m + 1, u, e);
    else return BinarySearch(a, ℓ, m − 1, e);
  }
}
```

Fig. 5.8. BinarySearch with function specification

$Sorted(a,l,u) \triangleq \forall i,j.\ (l \le i \le j \le u \Rightarrow a[i] \le a[j])$

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] BubbleSort(int[] a_0) {
  int[] a := a_0;
  for @ ⊤
    (int i := |a| − 1; i > 0; i := i − 1) {
    for @ ⊤
      (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Fig. 5.9. BubbleSort with function specification

```
@pre ⊤
@post ⊤
bool LinearSearch(int[] a, int ℓ, int u, int e) {|
  for @ ⊤
    (int i := ℓ; i ≤ u; i := i + 1) {
    @ 0 ≤ i < |a|;
    if (a[i] = e) return true;
  }
  return false;
}
```

Fig. 5.11. LinearSearch with runtime assertions

Q and $y \neq 0$

x/y

```
@pre ⊤
@post ⊤
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    @ 2 ≠ 0;
    int m := (ℓ + u) div 2;
    @ 0 ≤ m < |a|;
    if (a[m] = e) return true;
    else {
      @ 0 ≤ m < |a|;
      if (a[m] < e) return BinarySearch(a, m + 1, u, e);
      else return BinarySearch(a, ℓ, m − 1, e);
    }
  }
}
```

Fig. 5.12. BinarySearch with runtime assertions

```
@pre ⊤
@post ⊤
int[] BubbleSort(int[] a0) {
  int[] a := a0;
  for @ ⊤
    (int i := |a| − 1; i > 0; i := i − 1) {
    for @ ⊤
      (int j := 0; j < i; j := j + 1) {
      @ 0 ≤ j < |a|;
      @ 0 ≤ j + 1 < |a|;
      if (a[j] > a[j + 1]) {
        @ 0 ≤ j < |a|;
        int t := a[j];
        @ 0 ≤ j < |a|;
        @ 0 ≤ j + 1 < |a|;
        a[j] := a[j + 1];
        @ 0 ≤ j + 1 < |a|;
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Fig. 5.13. BubbleSort with runtime assertions

@L : $\ell \le i \wedge \forall j\ \ell \le j < i \Rightarrow a[j] \neq e$

assume not $(i \le u)$

rv := false

@post $rv \leftrightarrow \exists i\ \ell \le i \le u \wedge a[i] = e$

```
@pre 0 ≤ ℓ ∧ u < |a|
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool LinearSearch(int[] a, int ℓ, int u, int e) {
  for
    @L : ℓ ≤ i ∧ (∀j. ℓ ≤ j < i → a[j] ≠ e)
    (int i := ℓ; i ≤ u; i := i + 1) {
    if (a[i] = e) return true;
  }
  return false;
}
```

LI

@L : $\ell \le i \wedge \forall j \ldots$

assume $i \le u$

assume $a[i] = e$

rv := true

@post $rv \leftrightarrow \exists i\ \ell \le i \le u \wedge a[i] = e$

Fig. 5.15. LinearSearch with loop invariants

@pre: $0 \le \ell \wedge u < |a|$

$i := \ell$

@L : $\ell \le i \wedge \forall j (\ell \le j < i \Rightarrow a[j] \neq e)$

@L : $\ell \le i \wedge \forall j.(\ell \le j < i \Rightarrow a[j] \neq e)$

assume $i \le u$

assume $a[i] \neq e$

$i := i + 1$

@L : $\ell \le i \wedge \forall j (\ell \le j < i \Rightarrow a[j] \neq e)$

for loops vs while loops

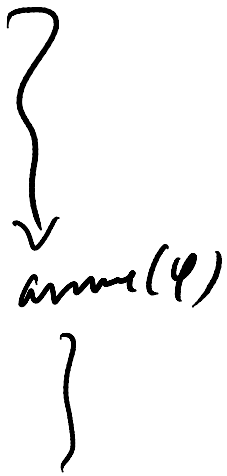
```
for ( init ; check ; update ) {
    body }
```

```
init;
while (check) {
    body
    update
}
```

Basic paths

A basic path is a sequence of instructions that begins at the function precondition or a loop invariant and ends at a loop invariant, an assertion, or the function postcondition.

Basic paths do not contain loops or conditionals, but use ***assume*** statements



assume semantics:

when the program sees an assume φ,
execution halts if φ does not hold,
and execution continues if φ holds

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] BubbleSort(int[] a0) {
  int[] a := a0;
  for @ ⊤
    (int i := |a| − 1; i > 0; i := i − 1) {
    for @ ⊤
      (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

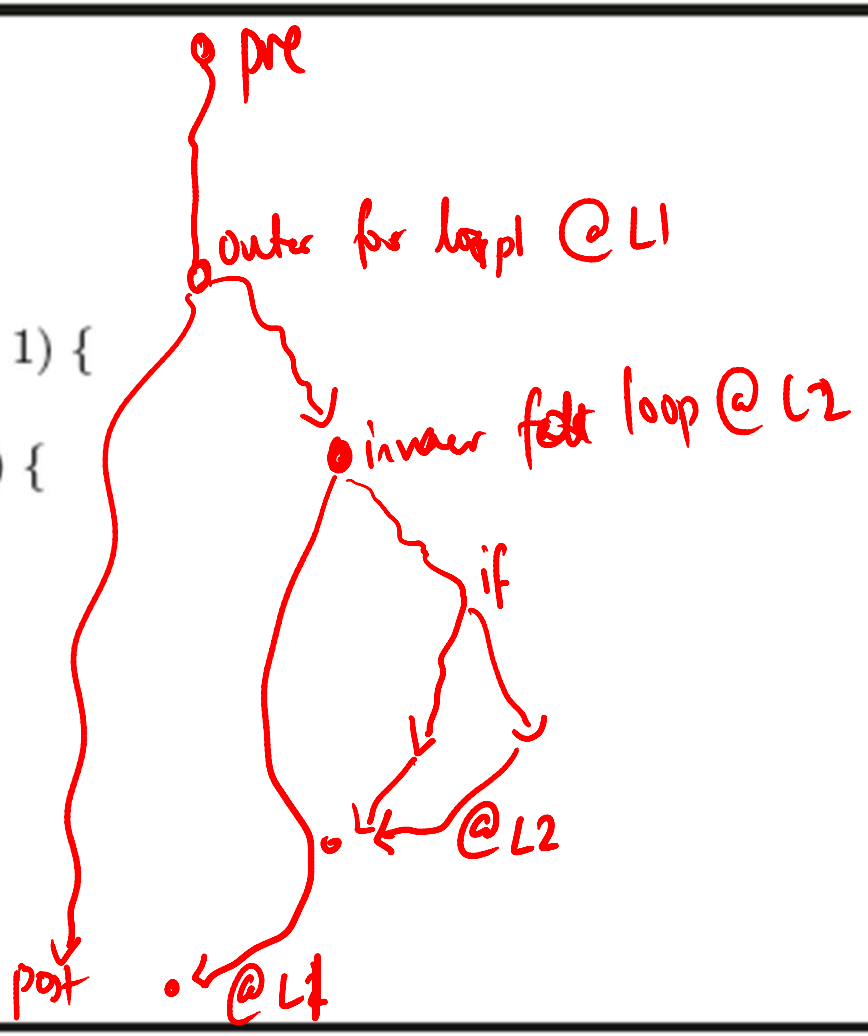


Fig. 5.9. BubbleSort with function specification

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] BubbleSort(int[] a0) {
  int[] a := a0;
  for
      @L1 : [ −1 ≤ i < |a|
            ∧ partitioned(a, 0, i, i + 1, |a| − 1)
            ∧ sorted(a, i, |a| − 1) ]
    (int i := |a| − 1; i > 0; i := i − 1) {
    for
        @L2 : [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i
              ∧ partitioned(a, 0, i, i + 1, |a| − 1)
              ∧ partitioned(a, 0, j − 1, j, j)
              ∧ sorted(a, i, |a| − 1) ]
      (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

partitioned(a, i, j, k, l)

∀ r, s

(i ≤ r ≤ j ∧ k ≤ s ≤ l

⇒ a[r] ≤ a[s])

theory of arrays and arithmetic and uns quantifiers.

{@L2} assume j < i ; assume a[j] > a[j+1] ; t := a[j] ; a[j] := a[j+1] ; a[j+1] := t

j := j+1 ; {@L2}

Fig. 5.17. BubbleSort with loop invariants

Adding assertions that check pre-conditions for method calls

{@pre} assume $\ell > u$; $m := (\ell+u)$ div 2; assume $a[m] \neq e$; assume $(a[m] < e)$;
assume $V_1 \leftrightarrow \exists i.\ m+1 \leq i < u \wedge a[i] = e$; $rv := V_1$ {@post}

```
@pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, ℓ, u)
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    int m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BinarySearch(a, m + 1, u, e);
    else return BinarySearch(a, ℓ, m − 1, e);
  }
}
```

@R1: $0 \leq m+1 \wedge u < |a| \wedge$ sorted$(a, m+1, u)$

@R2: $0 \leq \ell \wedge m-1 < |a| \wedge$ sorted$(a, \ell, m-1)$

Fig. 5.8. BinarySearch with function specification

{@pre} assume(not$(\ell > u)$; $m := (\ell+u)$ div 2; assume $a[m] \neq e$;
assume $a[m] < e$ {@R1}

```
@pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, ℓ, u)
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    int m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) {
      @R1 : 0 ≤ m + 1 ∧ u < |a| ∧ sorted(a, m + 1, u);
      return BinarySearch(a, m + 1, u, e);
    } else {
      @R2 : 0 ≤ ℓ ∧ m − 1 < |a| ∧ sorted(a, ℓ, m − 1);
      return BinarySearch(a, ℓ, m − 1, e);
    }
  }
}
```

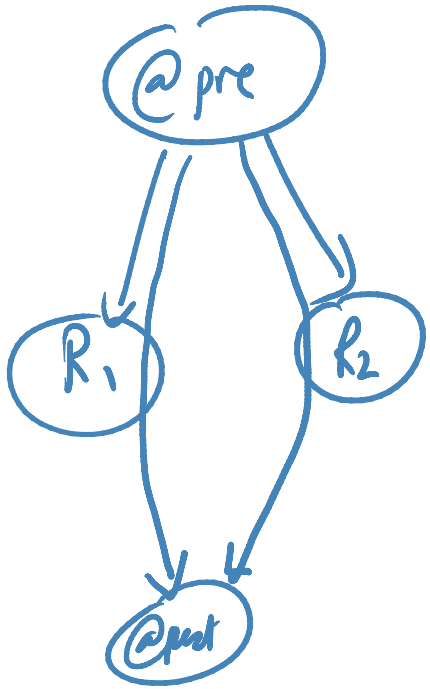


Fig. 5.20. BinarySearch with function call assertions

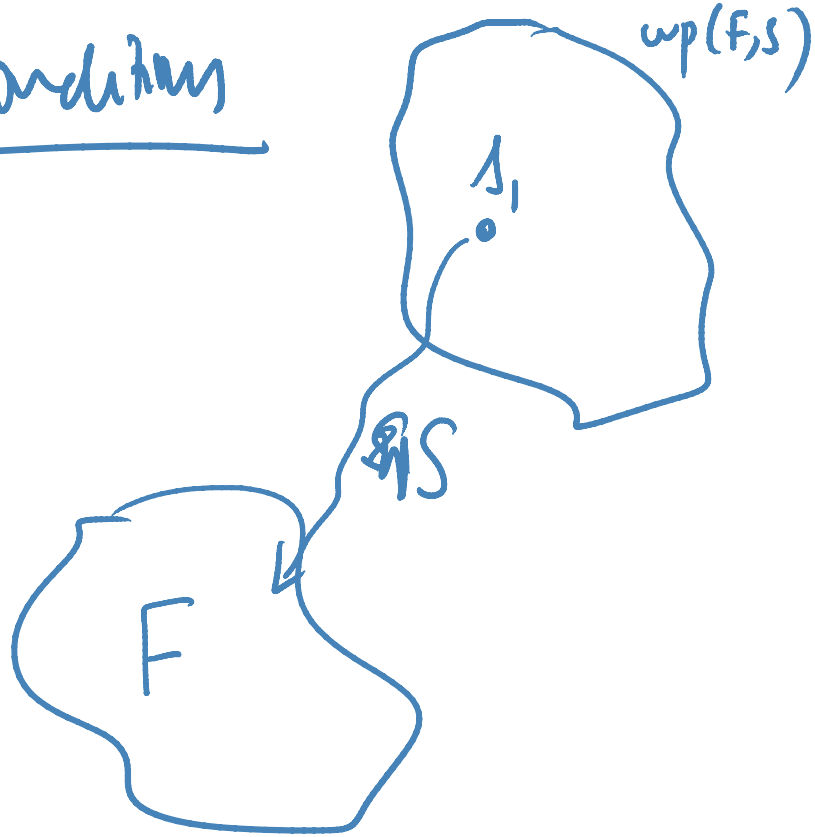
Program state:

$$(pc = L_1,\ x \mapsto 2,\ y \mapsto 3,\ a \mapsto [1, 19, 15, 23])$$

Weakest (liberal) pre-conditions

predicate ← program

$wp(F, S)$



$\{wp(F, S)\}\ S\ \{F\}$

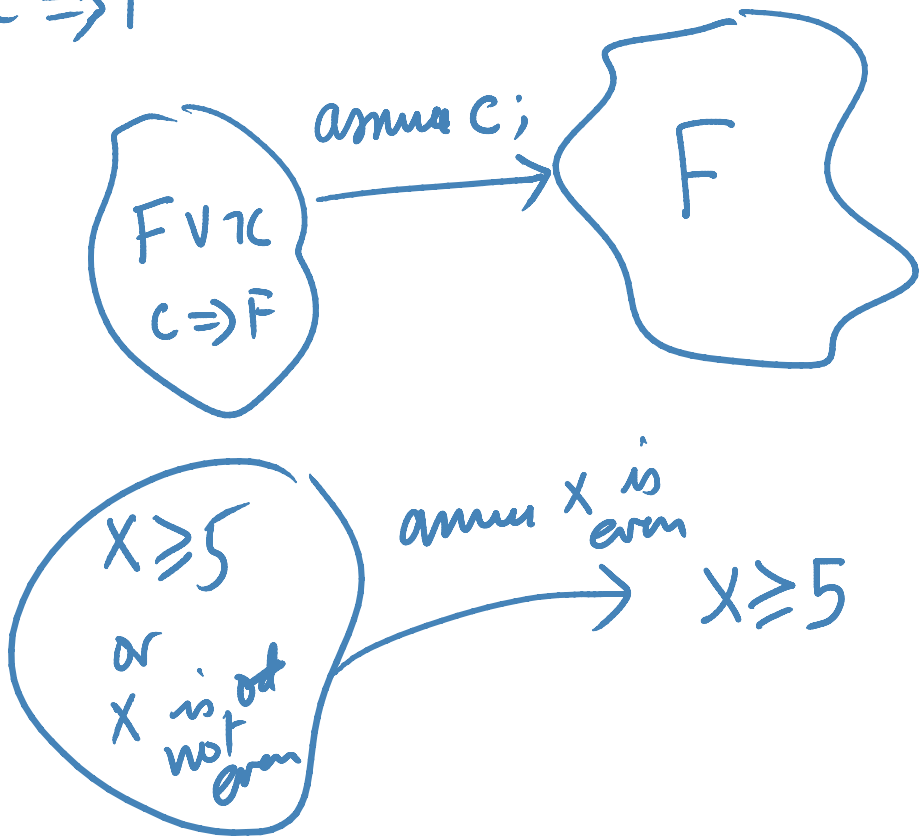
$\alpha \Rightarrow wp(F, S)$

$\{\alpha\}\ S\ \{F\}$

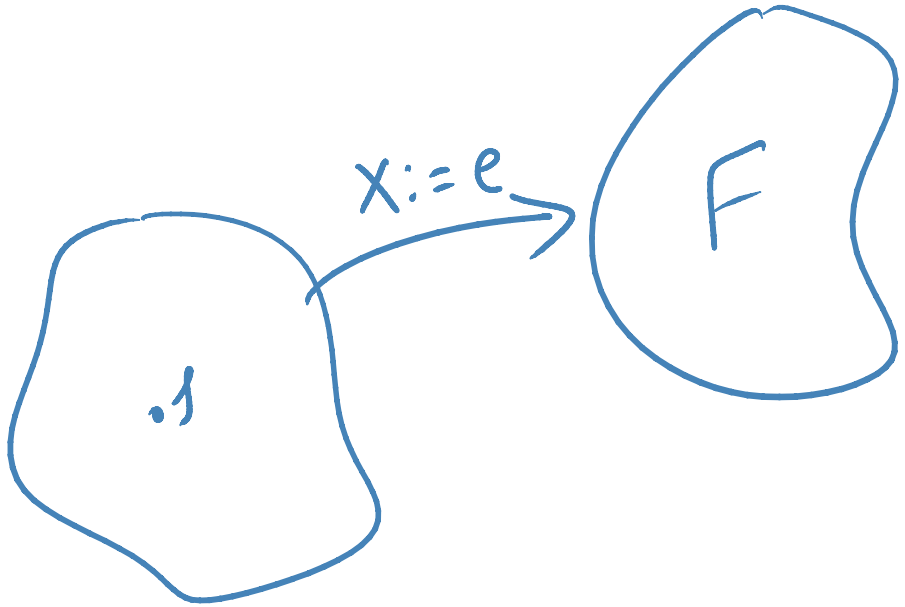
Basic paths : assume φ

- $x := e$
- $a[e_1] := e_2$

$wp(F, \text{assume } c) \equiv c \Rightarrow F$



$wp(F, x := e) = F[e/x]$

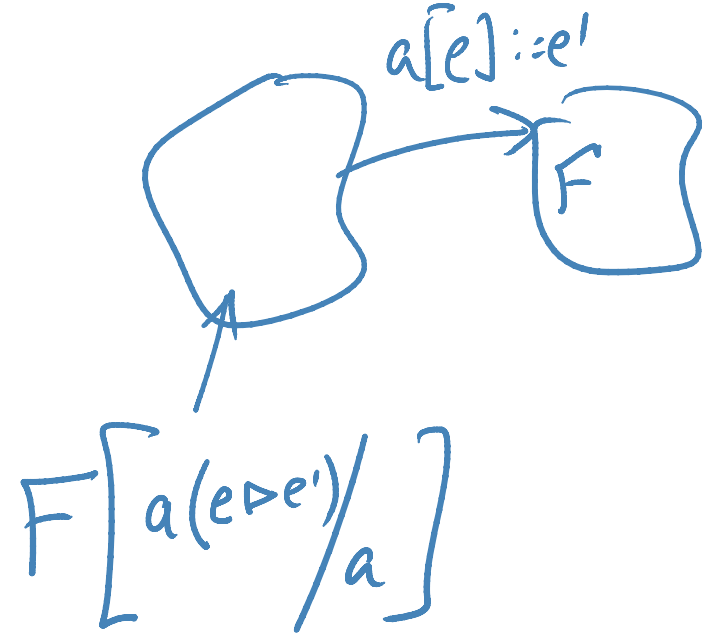


s

$wp(F, S_1 ; S_2) = wp(wp(F, S_2), S_1)$

$x := x+1;$ $x \geq 5$ (new) $\rightarrow$ $x \geq 4$ (old $x+1$)

$$wp\,(F, a[e] := e')$$



$\{\alpha\}\ S\ \{\beta\}$ is valid

iff

$\alpha \Rightarrow wp(\beta, S)$

$\{x \geq 0\}\ \ x := x+1\ \ \{x \geq 1\}$

$wp(x \geq 1,\ x := x+1) \equiv x+1 \geq 1$

$x \geq 0 \Rightarrow x+1 \geq 1$

@L: $l \le i \wedge \forall j (l \le j < i \Rightarrow a[j] \neq e)$

assume $i \le u$

assume $a[i] = e$

rv := true

@post rv $\leftrightarrow \exists j (l \le j \le u \wedge a[j] = e)$

F

~~true~~ $\leftrightarrow \exists j (l \le j \le u \wedge a[j] = e)$

$a[i] = e \Rightarrow \exists j (l \le j \le u \wedge a[j] = e)$

$i \le u \Rightarrow (a[i] = e \Rightarrow \exists j (l \le j \le u \wedge a[j] = e))$

$i \le u \wedge a[i] = e \Rightarrow \exists j (\ldots)$

$l \le u \wedge \forall j (l \le j < i \Rightarrow a[j] \neq e) \Rightarrow ((i \le u \wedge a[i] = e) \Rightarrow \exists j (l \le j \le u \wedge a[j] = e))$

Termination

Well founded ordering $(D, \prec)$

$rf : States \longrightarrow D$

compute / express

$\prec$ logically definable on D

$(\mathbb{N}, <)$

$(\mathbb{N} \times \mathbb{N}, <)$

$(a,b) \leq (c,d)$ if $a < c$
or $a = c$ and $b \leq d$

(D, τ)

δ : Program states $\to$ I)

@ $\bar{F}$

$\downarrow \delta[\bar{x}]$

S_1;

⋮

S_k;

$\downarrow K[\bar{x}]$

If WF ordering $(\mathbb{N}, <)$

@ $\bar{F}$

$rf := \delta[\bar{x}]$

S_1

⋮

S_k

@ $k(\bar{x}) < rf$

VC

$F \Rightarrow wp(K(\bar{x}) < \delta(\bar{x}_0),\ S_1; S_2; \ldots S_k; \bar{x}_0 := \bar{x})$

$F \Rightarrow wp(K(\bar{x}) < \delta(\bar{x}_0),\ S_1; \ldots S_k)[\bar{x}/\bar{x}_0]$

VC

@ $F \downarrow \delta(\bar{x})\ S_1; \ldots S_k \downarrow K(\bar{x})$

@L_1 : $i+1 \geq 0$

$\downarrow L_4$: $(i+1, i+1)$

assume $i > 0$

$j := 0$

$\downarrow L_2$: $(i+1, i-j)$

$i+1 \geq 0 \Rightarrow wp\left((i+1, i-j) < (i_0+1, i_0+1),\ \text{assume } i>0;\ j=0\right)\left[i, j / i_0, j_0\right]$

$\left(i+1 \geq 0 \Rightarrow \left(i > 0 \Rightarrow (i+1, i-0) < (i_0+1, i_0+1)\right)\right)\left[i, j / i_0, j_0\right]$

$\left(i+1 \geq 0 \Rightarrow \left(i > 0 \Rightarrow (i+1, i-0) < (i+1, i+1)\right)\right)$ — VC

VC for @F $\Phi_{\bar{x}} \downarrow \delta(\bar{x})\ S_1 \ldots S_k \downarrow K(\bar{x})$

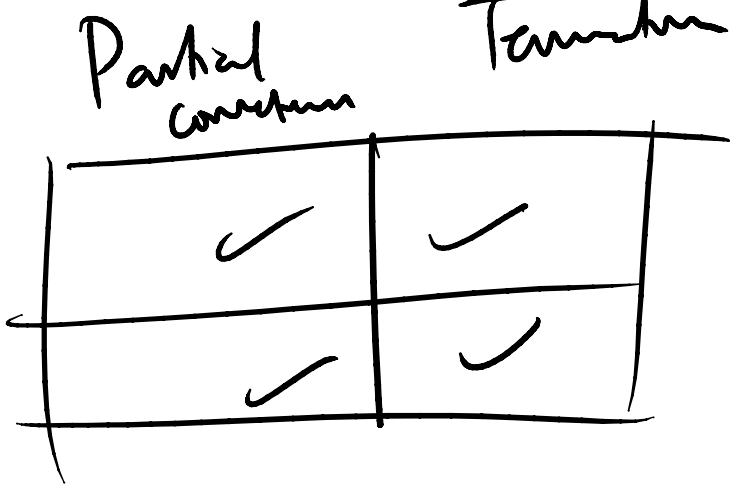
is

$F \Rightarrow wp(K < \delta(\bar{x}_0), S_1; \ldots S_k)[\bar{x}_0 \mapsto \bar{x}]$

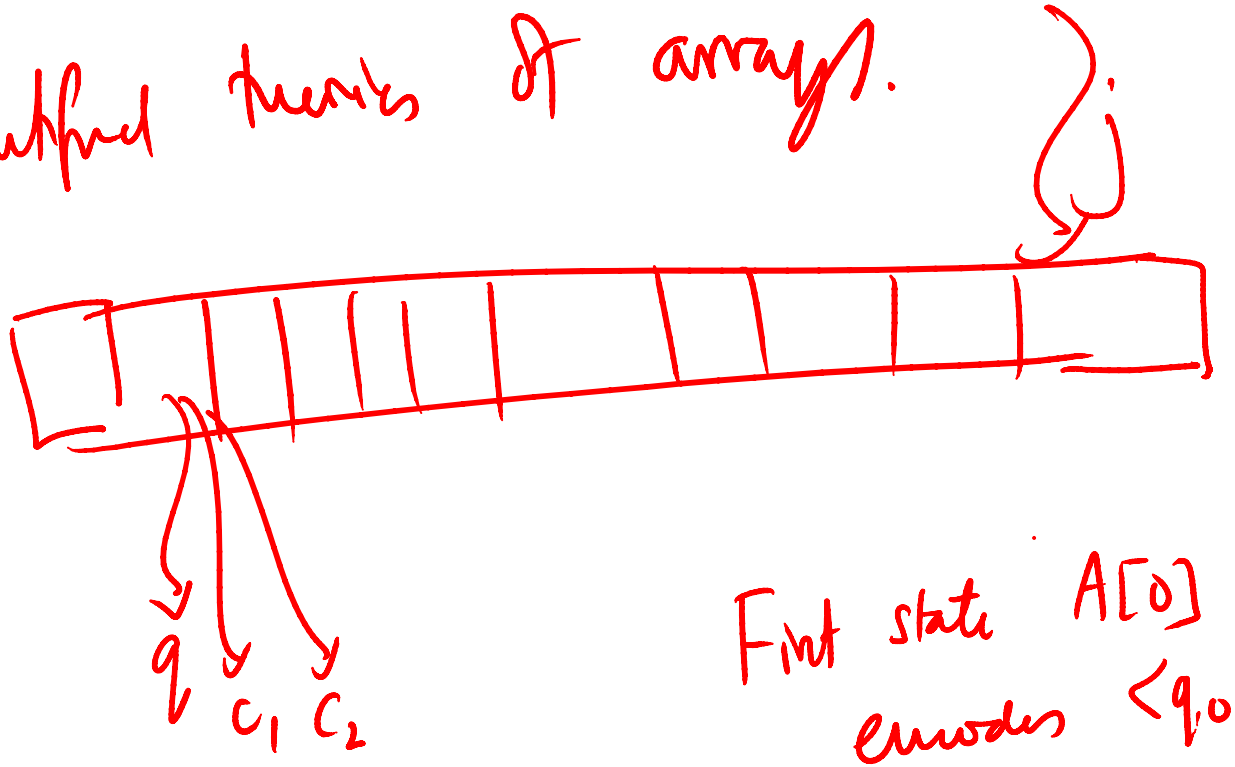
TT PL arrays loops functions etc

Basic patterns

Verification Continuity



Quantified theories of arrays.



First state A[0] encodes $\langle q_0, 0, 0 \rangle$

Two counter machines

A[j] encodes $\langle q_{halt}, \pm, - \rangle$

$\forall i$: $A[i] \xrightarrow[2CM]{} A[i+1]$