4. Algorithms

4.1 Introduction

Definition *1 An* ***algorithm*** *is a finite set of instructions for performing a computation or solving a problem.*

precision

steps are pecisely stated

uniqueness

result of each step od execution are uniquely defined

depends on input and result of preceding step

determinancy vs. ***nondeterminancy***

finiteness

the algorithm stops after finitely many steps

Dijkstra's mini language

$S ::= x_1, \ldots, x_n := E_1, \ldots, E_n \mid S; S \mid \textbf{skip} \mid \textbf{abort}$

$\mid \textbf{if } B_1 \to SL_1 \text{ '|' } B_2 \to SL_2 \text{ '|' } \ldots \text{ '|' } B_n \to SL_n \textbf{ fi}$

$\mid \textbf{do } B_1 \to SL_1 \text{ '|' } B_2 \to SL_2 \text{ '|' } \ldots \text{ '|' } B_n \to SL_n \textbf{ od}$

concurrent assignment

$x, y := y, x$

sequence of statemets(S)

$S_1; S_2$

Two bases

skip

abort

Alternaternative construct

if $B_1 \to SL_1 \mid B_2 \to SL_2 \mid \ldots \mid B_n \to SL_n$ ***fi***

Exa. Find maximum *of two elements, x and y*

if $x \geq y$ ***then*** $m := x$ ***else*** $m := y$ ***fi***

if $x > y$ ***then*** $m := x$ ***else*** $m := y$ ***fi***

if $x \geq y \to m := x \mid x \leq y \to m := y$ ***fi***

if $x \geq y \to m := x$
$\mid x < y \to m := y$
fi

if $x > y \to m := x$
$\mid x \leq y \to m := y$
fi

if $x \geq y \to m := x$
$\mid x \leq y \to m := y$
fi

$(m=x \vee m=y) \wedge m \geq x \wedge m \geq y$

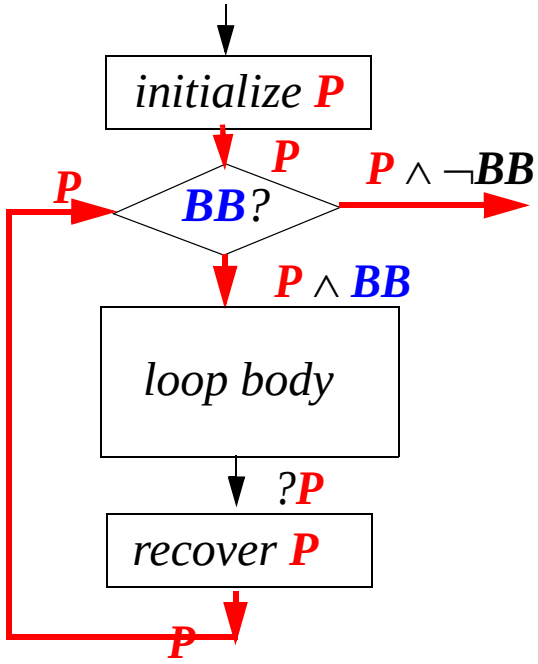
nondeterminancy

if fi ≡ ***abort***

do od ≡ ***skip***

Reapeatative construct

do $B_1 \rightarrow SL_1$ '|' … '|' $B_n \rightarrow SL_n$ ***od***



Let's assume that loop invariance is ***P****.*

and ***BB*** $= B_1 \vee B_2 \vee \ldots \vee B_n$.

$\therefore \neg$***BB*** $= \neg(B_1 \vee B_2 \vee \ldots \vee B_n)$

$= \neg B_1 \wedge \neg B_2 \wedge \ldots \neg B_n$.

Loop invariance ***P***

Loop invariance ***P*** *should be* ***initialized before*** *the loop,*

Loop invariance ***P*** *must be* ***recovered in*** *the loop, if needed, and*

Loop invariance ***P*** *is still* ***valid after****(termination of) the loop.*

P $\wedge \neg$***BB*** *is the* ***final*** *condition that you want* ***after the loop****!*

$a, b, c, d := A, B, C, D;$
do $a > b \rightarrow a, b := b, a$
$\quad | \; b > c \rightarrow b, c := c, b$
$\quad | \; c > d \rightarrow c, d := d, c$
od

$\neg BB = \neg((a > b) \vee (b > c) \vee (c > d)) = (a \leq b) \wedge (b \leq c) \wedge (c \leq d)$
$\quad = a \leq b \leq c \leq d.$

What is the loop invariance of the above do-od loop?

P *= a, b, c, and d are* ***permutation*** *of A, B, C, and D.*

Loop invariance ***P*** *should be* ***strong*** *enough to get* ***final result***
with and $\neg BB$*;*

whereas ***weak*** *enough*
to be easily ***initialized*** *and be* ***still valid*** *inside the loop.*

$P \wedge \neg BB$ *is the* ***final condition*** *that you want.*

Alg. 4.1.2 Finding *the* **maximum value** *in a* **sequence**

function max($a_1, a_2, \ldots, a_n$: numbers): number $\quad n \geq 1$

$max := a_1;$ $\quad max = max(a_1)$

for $i := 2$ **to** n **do**

$\quad$ **if** $max \leq a_i \rightarrow max := a_i \mid max \geq a_i \rightarrow$ **skip fi** $\quad max = max(a_1, \ldots, a_i)$

$\quad$ **od** $\quad max = max(a_1, \ldots, a_n)$

Dijkstra's mini language

$max, i := a_1, 2;$ $\quad max = max(a_1, \ldots, a_{i-1}) \wedge (i=2) \equiv max = max(a_1) \equiv \mathbf{T}.$

do $i \leq n \rightarrow$

$\quad$ **if** $max \leq a_i \rightarrow max := a_i \mid max \geq a_i \rightarrow$ **skip**

$\quad$ **fi**; $\quad max = max(a_1, \ldots, a_i)$

$\quad i := i+1$ $\quad max = max(a_1, \ldots, a_{i-1})$

od $\quad max = max(a_1, \ldots, a_{i-1}) \wedge (i=n+1) \equiv max(a_1, \ldots, a_n)$

4.2 Examples of Algorithms

Alg. 4.2.1 *The* ***linear search*** *algorithm*

function ***linear search***$(s_1, s_2, \ldots, s_n$: *numbers with* $n \geq 1)$: $\mathbf{N}_{0,n}$;

$i := 1$; ***do*** $(i \leq n \wedge x \neq s_i) \rightarrow i := i+1$ ***od***;

$\mathbf{P} \wedge \neg \mathbf{BB} = (1 \leq^{\forall} k < i: x \neq s_k) \wedge ((i \leq n \wedge x = s_i) \vee (i = n+1))$

if $i \leq n \rightarrow loc := i \mid i = n+1 \rightarrow loc = 0$ ***fi***;

$((1 \leq^{\forall} k < loc \leq n: x \neq s_k) \wedge (x = s_{loc})) \vee ((loc = 0) \wedge (1 \leq^{\forall} k \leq n: x \neq s_k))$

return *loc*

Another implementation for the ***linear search***

$(i, loc := 1, 0);$ $(n \geq 0)$

do $((i \leq n) \wedge (loc = 0)) \rightarrow$

if $x = s_i \rightarrow loc := i \mid x \neq s_i \rightarrow i := i+1$ ***fi***

od

$((1 \leq^{\forall} k < loc \leq n: x \neq s_k) \wedge (x = s_{loc})) \vee ((loc = 0) \wedge (1 \leq^{\forall} k \leq n: x \neq s_k))$

Alg. 4.2.1' *The* ***binary search*** *algorithm*

procedure ***binary search****(x: number;* $s_1, s_2, \ldots, s_n$*:* ***increasing*** *numbers)*

$i, j, loc := 1, n, 0;$

do $((i \le j) \wedge (loc = 0)) \rightarrow$

$m := \lfloor (i+j)/2 \rfloor ;$

if $x > s_m \rightarrow i := m+1$

$| \; x = s_m \rightarrow loc := m$

$| \; x < s_m \rightarrow j := m-1$

fi

od

$(1 \le^{\forall} k < n: s_k \le s_{k+1}) \wedge (n \ge 0)$ *is the given condition(constant)*

$(loc \le n) \Rightarrow (x = a_{loc}) \vee (loc = 0) \Rightarrow (1 \le^{\forall} k \le n: x \ne a_k)$

procedure ***bubble sort***$(s_1, s_2, \ldots, s_n$: *numbers with* $n \geq 2)$

for $i := 1$ *to* $n-1$ ***do***

$P1 \equiv (s_j = max(s_1, s_2, \ldots, s_j)$. *init* $P1$: $P1 \wedge (j=1) \equiv (s_1 = max(s_1)) \equiv \mathbf{T}$.

for $j := 1$ *to* $n-i$ ***do*** $\rightarrow$

if $s_j \geq s_{j+1} \rightarrow s_j, s_{j+1} := s_{j+1}, s_j$

$\mid\ s_j \leq s_{j+1} \rightarrow$ ***skip***

fi

$P1 \equiv (s_j = max(s_1, s_2, \ldots, s_j))$.

od

$P2 \equiv P1 \wedge (j=n-i+1) \equiv (s_{n-i+1} = max(s_1, s_2, \ldots, s_{n-i+1}))$

$P2 \wedge (i=1) \equiv (s_n = max(s_1, s_2, \ldots, s_n))$ *initialize P2*

od

$(1 \leq^{\forall} i \leq n: P2) \equiv 1 \leq^{\forall} i \leq n: (s_i = max(s_1, s_2, \ldots, s_i)$

$\equiv s_1 \leq s_2 \leq \ldots \leq s_n$. *sorted in increasing order*

$i = 1, j = 1$ $P1 = (s_1 = max(s_1)) \equiv$ **T**. *init. of P1(**before** loop)*

$, j = 2$ $P1 = (s_2 = max(s_1, s_2))$.

…

$i = 1, j = n–i+1 = n$ $P1 = (s_n = max(s_1, s_2, \ldots, s_n)) = P2$.*(n-1)-comp.*

$i = 2, j = 1$ $P1 = (s_1 = max(s_1))$.

…

$i = 2, j = n–2+1 = n–1$ $P1 = (s_{n-1} = max(s_1, s_2, \ldots, s_{n-1}))$. *(n-2)-com.*

…

$i = n\text{-}2, j=n-(n-2)+1 = 3$ $P1 = (s_3 = max(s_1, s_2, s_3)$

$i = n\text{-}1, j =1$ $P1= (s_1 = max(s_1))$.

$, j = n\text{-}i(=n-1)+1= 2$ $P1 = (s_2 = max(s_1, s_2)) = P2.$ *1 comp.*

$1 \le^{\forall} k \le n: s_k = max(s_1, s_2, \ldots, s_k)) \equiv s_1 \le s_2 \le \ldots \le s_n.$ *n(n-1)/2 comp.*

Greedy algorithm

Optimization problem

to find a solution to the given problem that

either ***minimizes*** *or* ***maximizes*** *the value of some parameters*

(global) optimal

to select the best choise considering all sequences of steps

local optimal(greddy algorithm)

to select the best choise at each step

Algorithm *6 Greedy Change-Making Algorithm*

procedure change(n_1, n_2, …, n_r: number of coins are r and

$c_1 > c_2 > \ldots > c_r$ is the value for each $coin_i$;

n: positive integer is the amount that you want to make change)

$n_1, n_2, \ldots, n_r := 0, 0, \ldots, 0;\ i := 1;$

do $(i \le r) \rightarrow$ ***do*** $(n \ge c_i) \rightarrow n_i := n_i + 1;\ n := n - c_i$ ***od***; $i := i+1$ ***od***

4.3 Analysis of Algorithms

Big-O Notation

Definition *1 Let f:* $\mathbf{N} \rightarrow \mathbf{R}^{+}$.

We say f(n) is O(g(n)), if $\exists C, k > 0$.ə.$\forall n > k$: $|f(n)| \leq C|g(n)|$.

We also write f(n) = O(g(n)) or even $f(n) \in O(g(n))$.

The constant C and k are called the ***witnesses*** *of ... f(n) is O(g(n)).*

Assume C and k are ***smallest*** *witnesses, Then* $\forall C', k'$.ə. $C < C', n < n'$,

C' and k' are also ***witnesses***

g(n) is ***faster or equl(not slower)*** *glowing than f(n)*

when n becomes ***large enough***

and ***ignoring*** *multiplying some* ***constant*** *C.*

Theorem *1 Let* $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. *Then* $f(x)$ *is* $O(x^n)$.
proof *easy with witness* $k=1$ *and* $C = \Sigma |a_i|$.

Example 6 $n!$

$n! \le n^n$. $n!$ *is* $O(n^n)$ *with witness* $k=1$ *and* $C=1$.

$\log n! \le \log n^n = n \log n$

$1, \log n, n, n\log n, n^2, \ldots, 2^n, n!$

$O(1) \subset O(\log n) \subset O(n) \subset O(n\log n) \subset O(n^2) \subset O(n^3) \subset \ldots O(n^k) \subset \ldots$
$\subset O(2^n) \subset \ldots \subset O(k^n) \subset \ldots \subset O(n!) \subset O(n^n) \subset \ldots$

Theorem *2, 3 Let* $f_1 \in O(g_1)$, $f_2(x) \in O(g_2)$. *Then*

$f_1+f_2 \in O(g_1+g_2) = O(max(|g_1|, |g_2|)$ *and*

$f_1f_2 \in O(g_1g_2)$.

Cololllary *1 Let* $f_1, f_2 \in O(g)$. *Then* $f_1+f_2 \in O(g)$.

Big-Omega and Big-Theta Notation

Definition 2 $\Omega(g) = \{f: \mathbf{N} \to \mathbf{R}^+ | \exists C, k > 0 \ \forall n > k: |f(n)| \geq C|g(n)|\}$

*If $f \in \Omega(g)$, f glows **at least** order of g.*

Def. 3 $\Theta(g)=\{f:\mathbf{N}\to\mathbf{R}^+ | \exists C_1,C_2,k>0, \forall n>k: C_1|g(n)| \leq |f(n)| \leq C_2|g(n)|\}$

*If $f \in \Omega(g)$, f glows (exactly) **same** order of g.*

$f \in \Theta(g)$, *iff* $f \in O(g) \wedge f \in \Omega(g)$.

$|f(n)| \leq |O(g(n)|$ $|f(n)| \geq |\Omega(g(n))|$ $|\Theta(g(n))| \leq |f(n)| \leq |\Theta(g(n))|$

Example 1, 2, 3, 4

Theorem 4 *Let* $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ *with* $a_n \neq 0$. *Then*

$f(x) \in \Theta(x^n)$.

Computational complexity

time complexity

space complexity

Worst-case analysis	$O(g(n))$
Best-case Analysys	$\Omega(g(n))$
Average-case analysis	$\Theta(g(n))$

$\Theta(1)$	***constant*** *complexity*
$\Theta(\log n)$	***logarithmic*** *complexity*
$\Theta(n)$	***linear*** *complexity*
$\Theta(n^k)$	***polinomial*** *complexity*
$\Theta(k^n)$	***exponential*** *complexity*
$\Theta(n!)$	***factorial*** *complexity*

(totally) solvable

tractable

polynomial

intractable

exponential

***class* P**

polynomial, tractable

class* NP*(*P ⊆ NP*)

Nondeterministc polynomial *exponetial(upper bound)*

lower bound? *tractable or intractable*

NP-*complete* *problems*

subset of class NP

If any of these problems can be solved in polynomial time,

***all** the problems in class in* **NP** *also in class* **P*(*P = NP*)*.