Design of Distributed Systems

Melinda Tóth, Zoltán Horváth

Design of Distributed Systems

Melinda Tóth, Zoltán Horváth

Publication date 2014

Supported by TÁMOP-4.1.2.A/1-11/1-2011-0052.

The proj
the Eurc
by the E

Table of Contents

Chapter 1. Lecture 1

1. Syllabus

1.1. Syllabus

- Dining/drinking philosophers
- Distributed problems
- Formal specification and properties of distributed systems
- Safety and progress properties of distributed programs
- Verification of safety critical properties
- Program compositions from components with proved properties
- Computing the value of an associative function
- Computing the value of an associative function

2. Motivation

2.1. Motivation

Motivation for using formal methods:

- safety critical applications
- safe application of software components
- primary goal: sound concepts about distributed and parallel programs

3. Literature

3.1. Literature 1.

- Chandy, K.M., Misra, J.: Parallel Program Design - A Foundation. Addison-Wesley, 1989.
- Misra, J.: A Discipline of Multiprogramming - Programming Theory for Distributed Applications. Springer, 2001.
- Horváth Z.: Parallel asynchronous computation of the values of an associative function. Acta Cybernetica, Vol.12, No. 1, Szeged (1995) 83-94.
- Horváth Z.: The Formal Specification of a Problem Solved by a Parallel Program – a Relational Model.
- Fóthi Á.- Horváth Z.- Kozsik T.: Parallel Elementwise Processing – A Novel Version. Annales Uni. Sci. Budapest de R. Eötvös Nom. Sectio Computatorica (1996).
- Horváth Z.- Kozsik T.- Venczel T.: On Composing Problems and Parallel Programs. In: Paakki J., ed., Proceedings of the Fifth Symposium on Programming Languages and Software Tools, Jyväskylä, Finland, June 7-8, 1997 (1997) Report C-1997-37, University of Helsinki, 1-12.
- Horváth Z.- Kozsik T.- Venczel T.: Parallel Programs Implementing Abstract Data Type Operations. Pure Mathematics and Applications (PU.M.A.)., Volume 11 (2000), Number 2. pp. 293-308.

4. Introduction

4.1. Properties of the formal model 1.

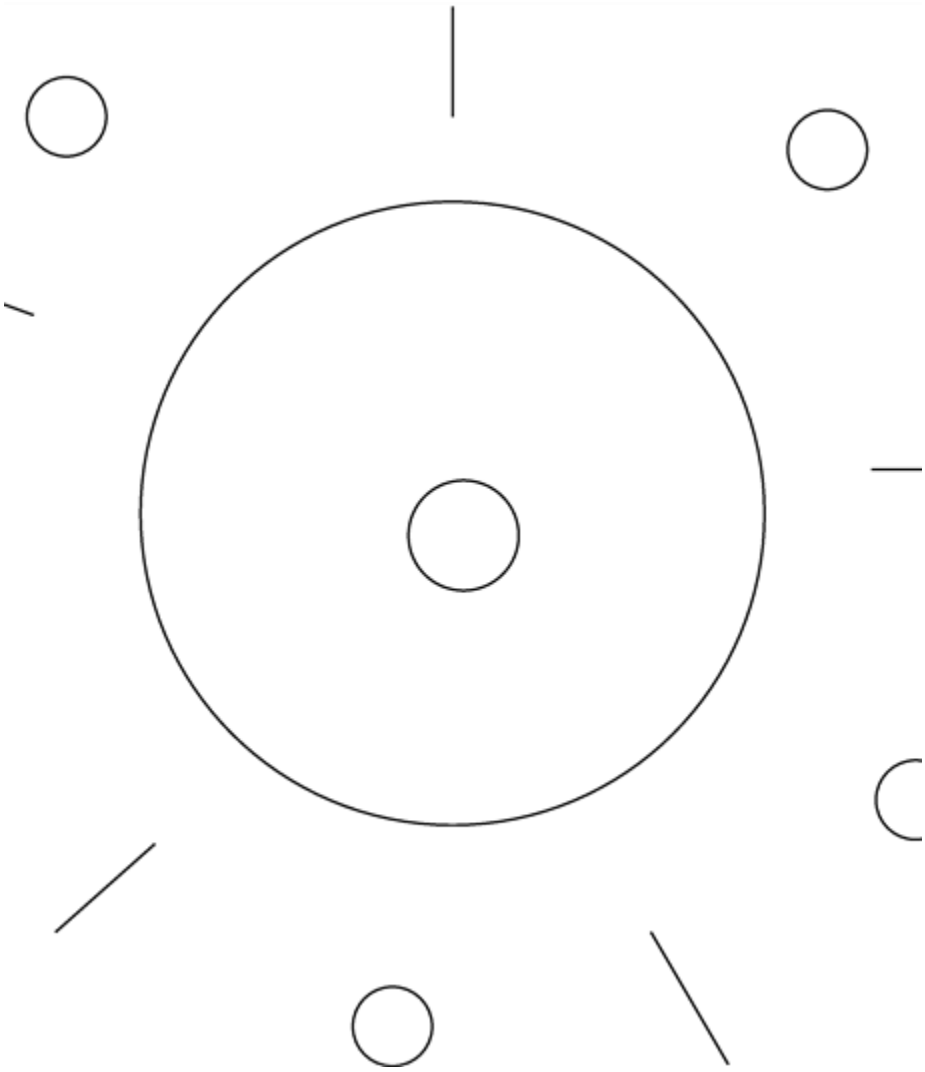
We need a formal model, which is appropriate for specification of problems and developing the solutions of problems in case of parallel and distributed systems.

4.2. Properties of the formal model 2.

The introduced model

- is an extension of a relational model of nondeterministic sequential programs,
- provides tools for stepwise refinement of problems, in a functional approach,
- uses the concept of iterative abstract program of UNITY,
- the concept of solution is based on the comparison of the problem as a relation and the behaviour relation of the program.

4.3. Dining philosophers

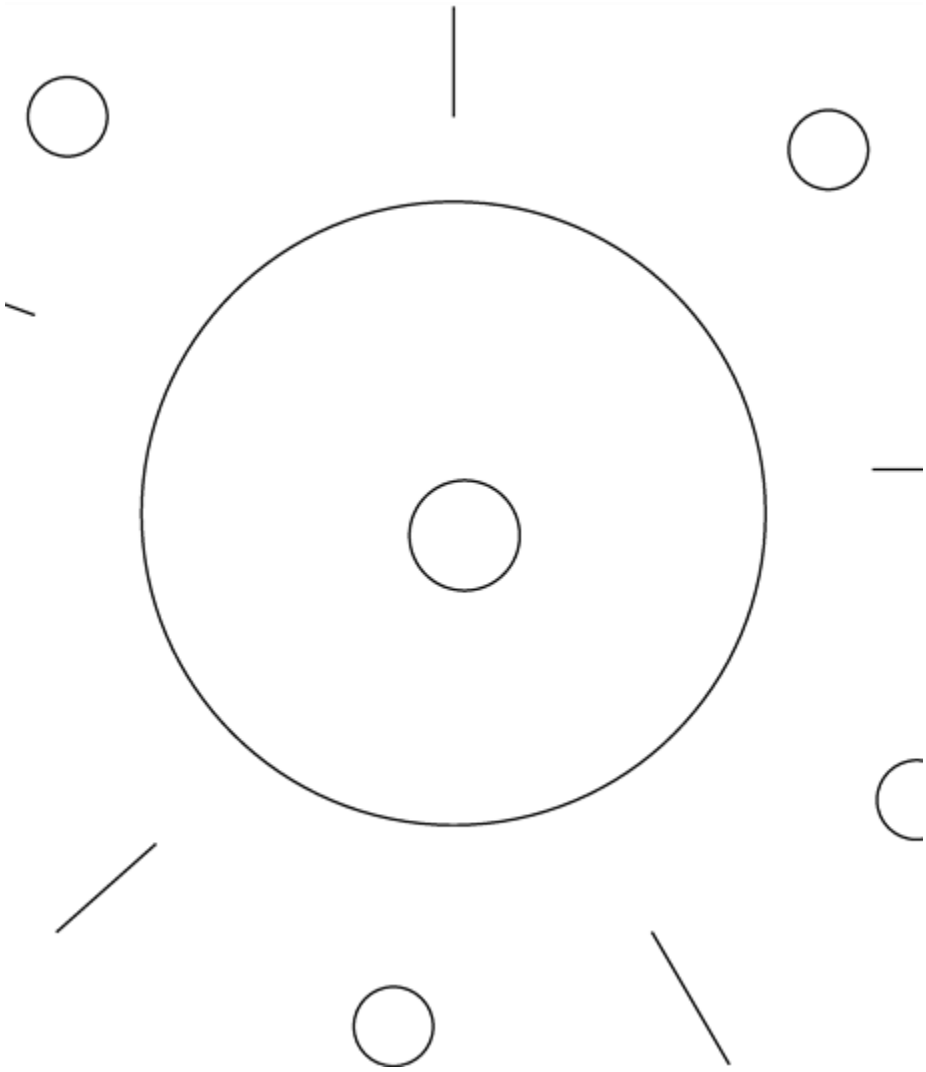


States:

- thinking: t

- forks in hands: f
- eating: e
- at home: h

4.4. Problem specification (requirements)



$\forall i$:

- unless: $f(i).t \rhd f(i).f \vee f(i).h$
- unless: $f(i).f \rhd f(i).e$
- ensures: $f(i).e \mapsto f(i).t$
- inevitable leads-to: $f(i).t \hookrightarrow f(i).h$
- invariant: $(f(i).e \rightarrow (\neg f(i+1).e \wedge \neg f(i-1).e)) \in inv$
- fixed point: $FP \Rightarrow f(i).h$
- termination: $\forall i : f(i).t \in TERM$

Help: thinking: t, forks in hands: f, eating: e, at home: h

4.5. Execution model

$$S ::= (SKIP,\ \{\ x := x + 1\ ,\ \text{if}\ x \leq y\ \|\ y := y + x\ \square\ \ z := x + y\ \}).$$

Abstract execution model

- No control flow, free processors select assignments asynchronously

4.6. Program, Solution

$$S ::= (SKIP,\ \{\ x := x + 1\ ,\ \text{if}\ x \leq y\ \|\ y := y + x\ \square\ \ z := x + y\ \}).$$

Program

- scheduling, processes, location, communication infrastructure, language

Solution

- Specification requirements are satisfied by program properties

4.7. Example

$$S ::= (SKIP,\ \{\ x := x + 1\ ,\ \text{if}\ x \leq y\ \|\ y := y + x\ \square\ \ z := x + y\ \}).$$

Example

- C/PVM PC-cluster (Parallel Virtual Machine)
- Erlang VM cluster

4.8.

- The notion of the state space makes it possible to define the semantical meaning of a problem independently of any program.
- The generalized concept of a problem is applicable for cases in which termination is not required but the behaviour of the specified system is restricted by safety and progress properties.
- The solution of a problem may be a sequential program, a parallel one, or even a program built up from both sequential and parallel components.

Chapter 2. Lecture 2

1. Example

1.1. An Example: sorting

$$S = (SKIP, \{ \Box_{i \in [1..n-1]} a(i), a(i+1) := a(i+1), a(i), \text{ if } a(i) > a(i+1) \})$$

1.2. An Implementation: Sorting

A valid implementation: the code for the i-th processor:

```
  loop
 < lock a(i) and a(i+1) >
 x := a(i);
 y := a(i+1);
 if x > y then
    a(i+1):=x;
    a(i):= y;
 end if;
 < unlock a(i) and a(i+1) >
end loop;
```

$n-1$ processes.

1.3. An Implementation: Sorting

A sequential program:

```
  loop
  for i=1 to n-1 do
    x := a(i);
    y := a(i+1);
    if x > y then
      a(i+1):=x;
      a(i):= y;
    end if;
  end for
end loop
```

2. Basic Concepts of the Relational Model

2.1. Concepts

A programming model defines

- the semantics of problems and programs
- operations for problem and program constructions
- when a program solves a program.

Relational model:

- the elements of the semantic domain are relations

2.2. Relations

- An arbitrary subset of a direct product of sets is called a relation.
- Let $R \subseteq A \times B$ where A and B are arbitrary sets. The domain of the relation R is defined by $\mathcal{D}_R ::= \{a \in A \mid \exists b \in B : (a,b) \in R\}$.

2.3. State Space

- Let $I \subset \mathcal{N}.\ \forall i \in I : A_i$ is a finite or numerable set.
- The set $A ::= \overset{\times}{_{i \in I}} A_i$ is called *state space*, the sets A_{i_j} are called *type value sets*.
- The projections $v_i : A \longmapsto A_i$ are called variables.
- A^* is the set of the finite sequences of the points of the state space and A^∞ the set of the infinite sequences.
- Let $A^{**} = A^* \cup A^\infty$.
- A *statement* is a subset of the direct product $A \times A^{**}$.

2.4. Statements and Effect Relation

- A *statement* is a subset of the direct product $A \times A^{**}$.
- The *effect relation* of a statement s is denoted by $p(s)$.
- The effect relation expresses the functionality of the statement.
- $p(s) \subseteq A \times A, s \subseteq A \times A^{**} : \quad \mathcal{D}_{p(s)} = \{a \in A | s(a) \subseteq A^*\}$.

2.5. Example

```
  var i,j : integer;
j:=2;
while i <> 5 loop
  i:=i+j
end loop
```

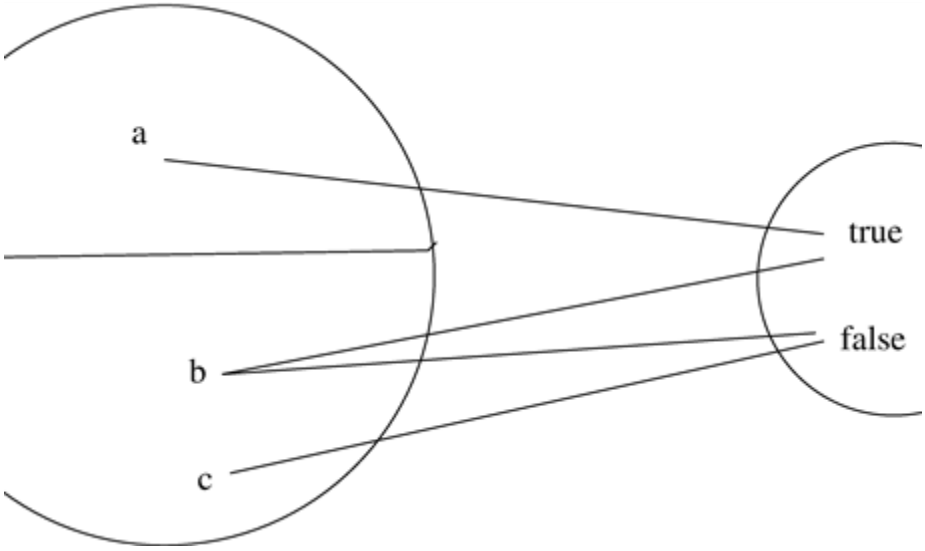
- State space: $A = i : Z \times j : Z$,
- variables: $i, j : Z,\ i((3,2)) = 3,\ j((3,2)) = 2$,
- seq. program:

  $s(1,1) = \{< (1,1),(1,2),(3,2),(5,2) >\}$,

  $s(0,0) = \{< (0,0),(0,2),(2,2),(4,2),(6,2), ... >\}$, etc.
- effect relation: $((1,1),(5,2)) \in p(S)$.

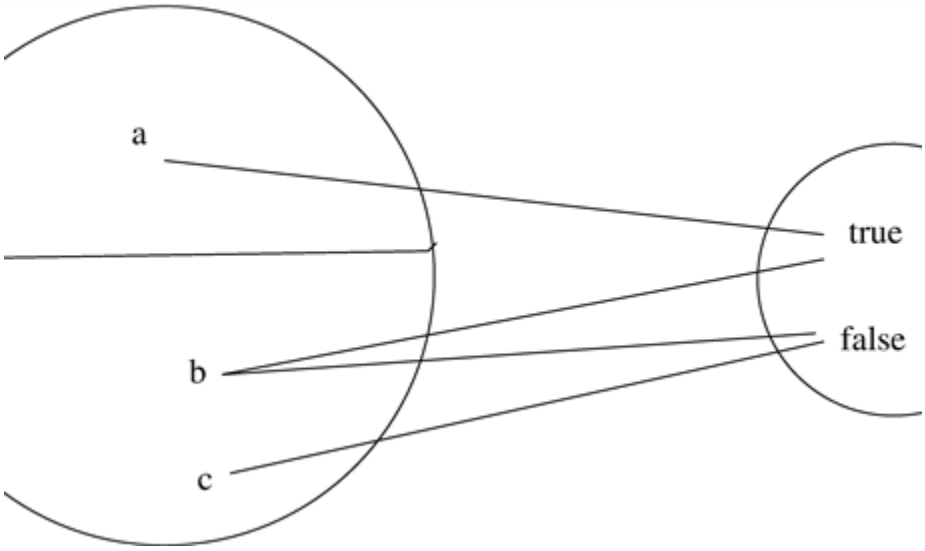
2.6. Partial Function and Logical Relation

- A relation R is called a *partial function*, if for all $a \in A$ the set $R(a)$ has at most one element. If $\forall a \in A: |R(a)| = 1$ then R is a function.
- If $R \subseteq A \times \mathcal{L}$ is a relation, where A is an arbitrary set and $\mathcal{L}$ is the set of the logical values, then R is called a *logical relation*.



2.7. Truth Set

- The *truth set of the logical function* $f : A \longmapsto \mathcal{L}$ is defined as $\lceil f \rceil ::= \{a \in A \mid f(a) = true\}$.
- The logical functions $True, False : A \longmapsto \mathcal{L}$ are defined by their truth sets. $\lceil True \rceil = A, \lceil False \rceil = \emptyset$.



2.8. Transitive Disjunctive Closure

- The *power-set* (set of subsets) of set A is denoted by $\mathcal{P}(A)$.
- $R^{tdc} \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ relation is the *transitive disjunctive closure of relation* $R \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$, if R^{tdc} is the smallest relation, for which holds:

- $R \subseteq R^{tdc}$
- if $(a, b) \in R^{tdc}$ and $(b, c) \in R^{tdc}$, then $(a, c) \in R^{tdc}$
- for any numerable set W : $(\forall m : m \in W :: (m, b) \in R^{tdc}) \implies ((\bigcup_{m \in W} m), b) \in R^{tdc}$.

2.9. Example TDC Relation

$A = \{1, 2, 3, 4\}, R \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$,

$R = \{(\{3\}, \{1\}), (\{2\}, \{1\}), (\{1\}, \{4\})\}$,

$R^{tdc} =$

$\{$

$(\{3\}, \{1\}), (\{2\}, \{1\}), (\{1\}, \{4\})$,

$(\{2, 3\}, \{1\}), (\{2\}, \{4\}), (\{3\}, \{4\}),$

$(\{2, 3\}, \{4\}), (\{1, 3\}, \{4\}), (\{1, 2\}, \{4\}), (\{1, 2, 3\}, \{4\})$

$\}$

Chapter 3. Lecture 3

1. Problem

1.1. Problem

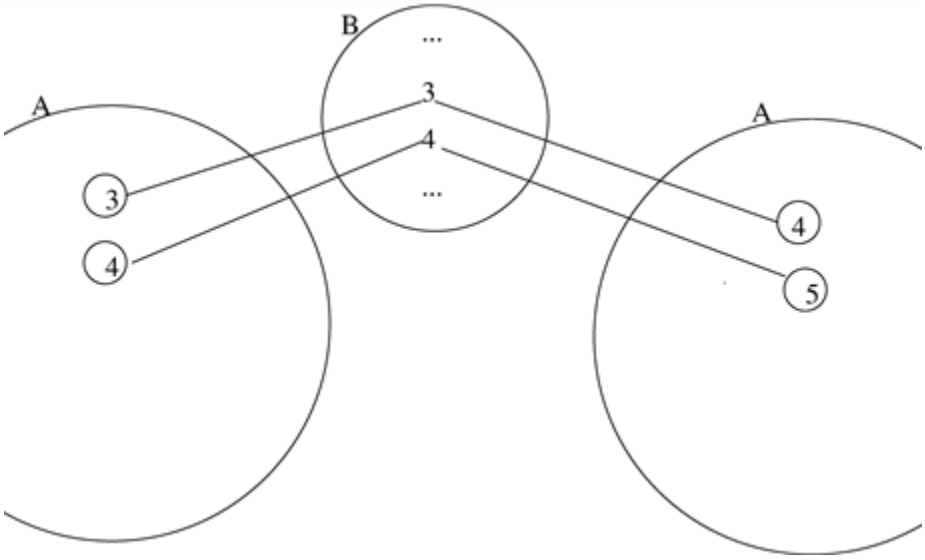
- The problem is defined as a set of specification relations.
- Every specification relation is defined over the powerset of the state space.
- Let $P, Q, R, U : A \longmapsto \mathcal{L}$ be logical functions.
- We define
  - $\triangleright, \mapsto, \hookrightarrow \in \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$, and
  - $FP, INIT, inv, TERM \subseteq \mathcal{P}(A)$

1.2. Specification Relations

- $P \triangleright Q$ - (P stable unless Q),
- $P \mapsto Q$ - (P ensures Q -t),
- $P \hookrightarrow Q$ - (Q is inevitable from P),
- $Q \hookrightarrow FP$, $Q \in TERM$ - (fixed point is inevitable from Q),
- $FP \Rightarrow R$ - (R holds in any fixed point),
- $invP$ - (P is invariant),
- $Q \in INIT$ (Q initially).

1.3. Example

- $A ::= \mathcal{N}$, $\triangleright ::= \{(\lceil i = k \rceil, \lceil i = k + 1 \rceil) \mid k \in \mathcal{N}\}$.
- According to $i = 5 \triangleright i = 6$ specification requirement the program is enabled to change state $a = (5)$ to state $a = (6)$ only.
- According to the specification relation the variable i is non-decreasing and can be increased one by one.



1.4. Problem Definition

- Let A be a state space and let B be a finite or numerable set.
- The relation $F \subseteq B \times H$, where

  $H = (\overset{\times}{_{i\in[1..3]}}\mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))_{i\in[1..4]}\overset{\times}{\mathcal{P}}(\mathcal{P}(A)))$ is called a problem defined over the state space A.
- B is called the parameter space of the problem.

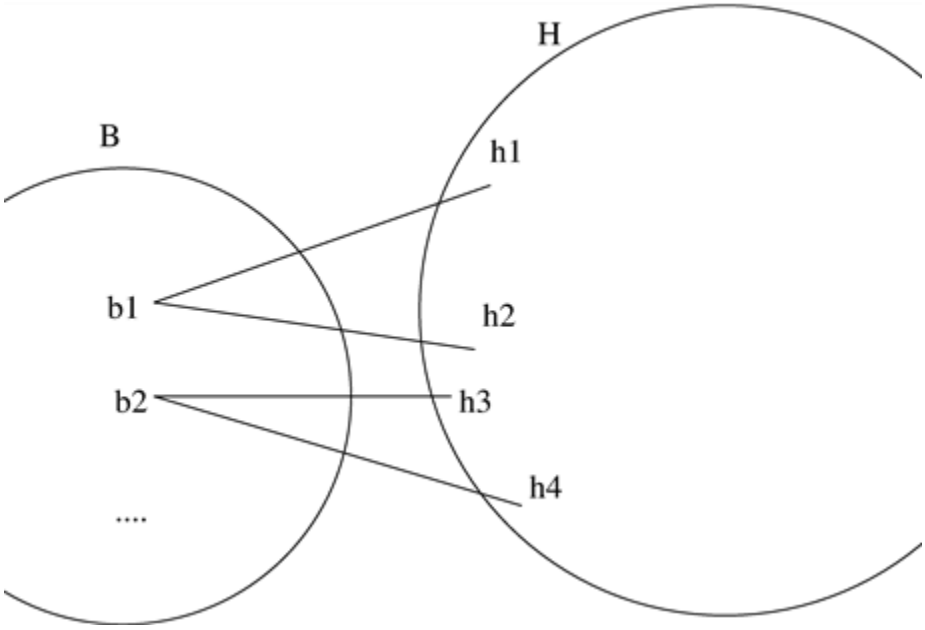
Two relations expressing boundary properties and five relations expressing transition properties are associated to every point of set B.

1.5. Notation

- Let $b \in B$ denote an arbitrary element of the domain of the problem.
- Let h denote an element of $F(b)$.
- The components of h are denoted by $\triangleright_h, \mapsto_h, \hookrightarrow_h$ and by $INIT_h, FP_h, inv_h, TERM_h$ respectively.
- If $|F(b)| = 1$ then we use b instead of h in the indices for the sake of simplicity.

1.6. Notation

- $h1 = (\triangleright_{h1}, \mapsto_{h1}, \hookrightarrow_{h1}, INIT_{h1}, FP_{h1}, inv_{h1}, TERM_{h1})$



1.7. Example – Value of a Function

$$A = X \times Y \quad x : X, y : Y,$$

$$B = X \quad x' : X.$$

$$(x = x') \in INIT_{x'} \tag{3.1}$$

$$True \hookrightarrow FP_{x'} \tag{3.2}$$

$$FP_{x'} \Rightarrow y = f(x') \tag{3.3}$$

2. Abstract Parallel Program

2.1. Abstract parallel program

The abstract program is a relation

- generated by a set of conditional assignments;
- assignments are selected nondeterministically,
- executions of different processors are fairly interleaved.
- a fixed point is said to be reached in a state, if any statement in that state leaves the state unchanged.

2.2. General Assignment

- A statement over the state space A is called empty and termed $SKIP$, if $\forall a \in A: \ SKIP(a) = \{< a >\}$.
- Let $A = A_1 \times ... \times A_n$, $F = (F_1, ..., F_n)$, where $F_i \subseteq A \times A_i$.
- The statement $s \subseteq A \times A^{**}$ is a general assignment defined by F, if
  - $s ::= \{(a, red(< a, b >)) \mid a, b \in A \wedge a \in \bigcap_{i \in [1,n]} \mathcal{D}_{F_i} \wedge$

$b \in F(a)\} \cup$

$\{(a, < a, a, a.... >) \mid a \in A \wedge a \notin \bigcap_{i \in [1,n]} \mathcal{D}_{F_i}\}$.

2.3. Example

- x,y : N,
- x,y := x+y, x-y,
- $F_1(x,y) = x + y,\ F_2(x,y) = x - y$,
- $F = (F_1, F_2),\ F(x,y) = (x + y, x - y)$,
- F(2,3)=?, F(3,2)=?

2.4. Extension

- We extend the domain of a relation for the whole state space in the following way:
  - $A = A_1 \times \cdots \times A_n$,
  - $F \subseteq A \times A$,
  - $F = (F_1, \ldots, F_n)$,
  - where $F_i \subseteq A \times A_i$.
- Let $\lceil \pi_i \rceil ::= \mathcal{D}_{F_i}$.
- The relation $F_i|_{True}$ is the extension of F_i for the truth set of condition $True$, i.e.,
  - $F_i|_{True}(a) ::= F_i(a)$, if $a \in \lceil \pi_i \rceil$ and
  - $F_i|_{True}(a) ::= a_i$, otherwise.
- $F||_{True} ::= (F_1|_{True}, \ldots, F_n|_{True})$.

2.5. Conditional Assignment

- Let be s_j an assignment, for which $((\mathcal{D}_{p(s_j)} = A) \wedge (\forall a \in A : p(s_j)(a) = F||_{\uparrow}(a)))$.
- This kind of (simultaneous, nondeterministic) assignment is called a conditional assignment, if $\forall a \in A : |p(s_j)(a)| < \omega$.
- We denote the conditional assignment s_j the following way: $(\|_{i \in [1,n]} (v_i :\in F_{j_i}(v_1, .., v_n)$, if $\pi_{j_i}))$.
- Simultaneous, nondeterministic, conditional assignment: $(v_i :\in F_i(v_1, .., v_n)$, if $\pi_i \parallel (v_k :\in F_k(v_1, .., v_n)$, if $\pi_k)$.
- Abbreviation: $\|_{i \in [1,n]} (...)$

2.6. Example – Abstract Program

$S ::= (SKIP, \{$

$(s_1 : \ x := x + 1$, if $x \leq y \parallel y := y + x)$,

$s_2 : \ z := x + y\}$

- Atomicity: $(2,3,0) - (s_1) - > (3,5,0), (3,5,0) - (s_2) - > (3,5,8)$
- if no atomicity: $(2,3,0) - - > (3,3,0), (3,3,0) - - > (3,3,6) - - > (3,5,6)$
- there is no state, when $x + y$ is 6.

Chapter 4. Lecture 4

1. Reminder

1.1. Problem

- The problem is defined as a set of specification relations.
- Every specification relation is defined over the powerset of the state space.
- Let $P, Q, R, U : A \longmapsto \mathcal{L}$ be logical functions.
- We define
  - $\triangleright, \mapsto, \hookrightarrow \in \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$, and
  - $FP, INIT, inv, TERM \subseteq \mathcal{P}(A)$

1.2. Abstract Parallel Program

The abstract program is a relation

- generated by a set of conditional assignments;
- assignments are selected nondeterministically,
- executions of different processors are fairly interleaved.
- a fixed point is said to be reached in a state, if any statement in that state leaves the state unchanged.

1.3. Example

$S ::= (SKIP, \{$

$(s_1 : \ x := x + 1$, if $x \leq y \parallel y := y + x)$,

$s_2 : \ z := x + y\}$

2. Semantics of the Abstract Program

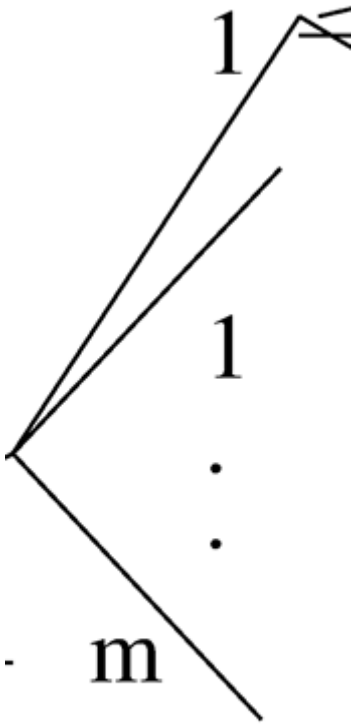
2.1. State Transition Trees

- Let S be an ordered pair of a conditional assignment and of a nonempty, finite set of conditional assignments, such that
  - $S = (s_0, \{s_j \mid j \in J\})$,
  - where $J = \{1..m\}$, $m \geq 1$.
- The semantics of the abstract program is defined as a binary relation which associates equivalence classes of correctly labeled state transition trees to the points of the state space.

2.2. State Transition Trees

- The labeled state transition trees are generated by the ordered pair

- of the effect relation of the initial assignment s_0 and
- of the UP(S) disjoint union of the effect relations of the $\{s_1, ..., s_m\}$ elements of the abstract program.



2.3. Abstract Parallel Program – Definition

- The relation $UPG(S) \subseteq A \times A^{***}$ is called an abstract parallel program, if
  - it associates equivalence classes of labelled transition trees to the element $b \in A$,
  - which trees are generated at b by the ordered pairs of relations $(p(s_0), UP(s))$ and
  - have a correct labelling.

2.4. Abstract Parallel Program – Notation

- The abstract parallel program $UPG(S)$ generated by $S = (s_0, \{s_1, ...s_m\})$ is abbreviated by S in the following.
  - The conditional assignment s_0 is called the initialization in S and
  - $s_j : j \in [1..m]$ is said to be an element of the program S .

2.5. Execution

- Any path of a representative of the equivalence class $UPG(a)$ is called an execution path of the abstract parallel program starting in the state a.
- Any concurrent execution of conditional assignments should satisfy the requirement of serializibility.
- Every execution path of the abstract parallel program represents a possible sequential execution sequence of the assignments.
- The introduced semantics is an interleaving semantics of parallel programs.

2.6. Reachable States

- The labels (states) along the execution paths of set $UPG(S)(a)$ is denoted by $E(S)(a)$.
- $E(S)(a)$ is the set of reachable states from state a.

2.7. Unconditionally Fair Scheduling

- An execution path corresponds to the requirement of unconditionally fair scheduling,
  - if every statement is selected infinitely times along the path, i.e.
  - every label from index set J is associated infinitely often to the vertices of the path.

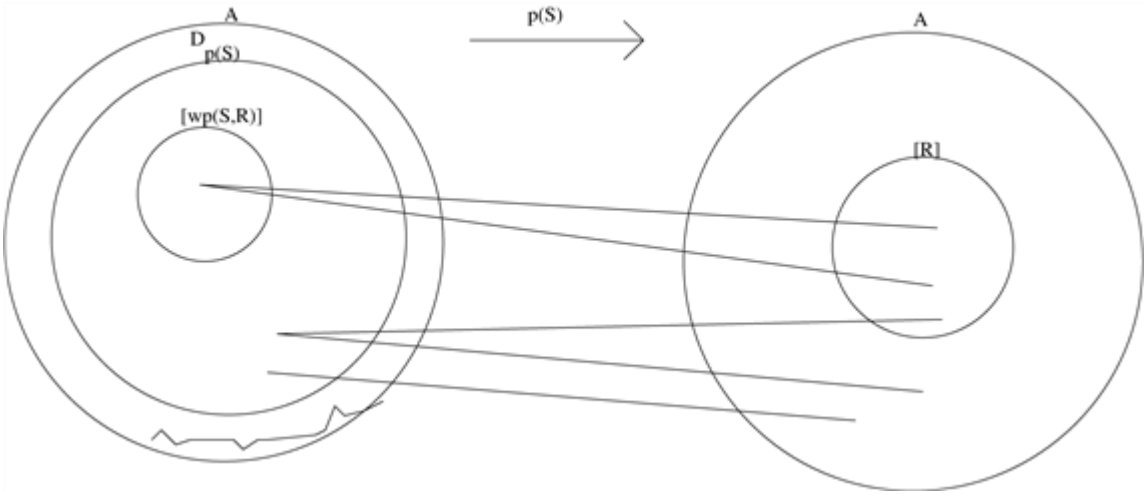
3. Program Properties of the Abstract Program

3.1. Weakest Precondition

- The program properties are defined in terms of the weakest precondition of the element statements of the abstract program.
- The logical function $wp(s, R) : A \longmapsto \mathcal{L}$ is called the weakest precondition of the postcondition R in respect to the statement s.
- We define $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.

3.2. Weakest Precondition

- $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- $wp(S, R) ::= \forall s \in S : wp(s, R)$.



3.3. Strongest Postcondition

- The logical function $sp(s, Q)$ is called the strongest postcondition of Q in respect to s .
- $\lceil sp(s, Q) \rceil ::= p(s)(\lceil Q \rceil)$.

Chapter 5. Lecture 5

1. Reminder

1.1. Abstract Parallel Program and Scheduling

- The abstract parallel program $UPG(S)$ generated by $S = (s_0, \{s_1, ...s_m\})$ is abbreviated by S in the following.
  - The conditional assignment s_0 is called the initialization in S and
  - $s_j : j \in [1..m]$ is said to be an element of the program S.
- An execution path corresponds to the requirement of unconditionally fair scheduling, if every statement is selected infinitely times along the path, i.e. every label from index set J is associated infinitely often to the vertices of the path.

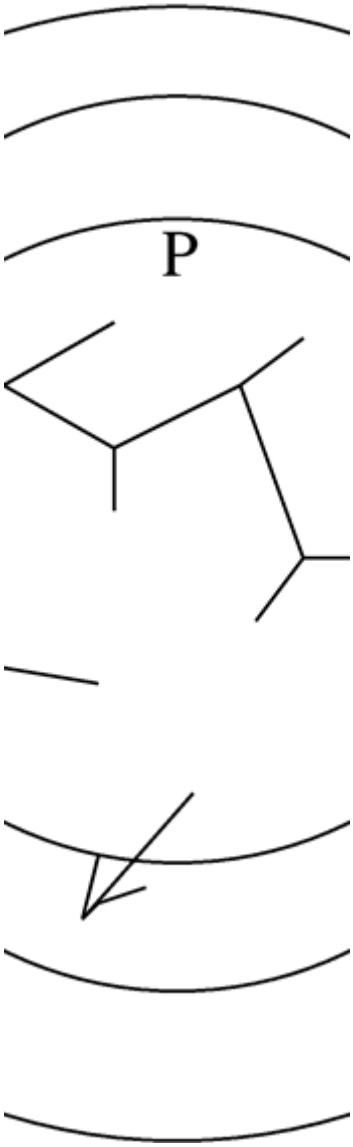
1.2. Weakest Precondition and Strongest Postcondition

- $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil \}$.
- $wp(S, R) ::= \forall s \in S : wp(s, R)$.
- $\lceil sp(s, Q) \rceil ::= p(s)(\lceil Q \rceil)$.

2. Program Properties of the Abstract Program

2.1. Invariant Properties, Definition

- $inv_S(Q)$ is the set of logical functions of which truth are preserved by the elements of S if the program is started from a state satisfying Q.
- $inv_S : \mathcal{P}(A) \longmapsto \mathcal{P}(\mathcal{P}(A))$.
- $inv_S(\lceil Q \rceil) \subseteq \mathcal{P}(A)$.
- $inv_S(\lceil Q \rceil) ::= \{ \lceil P \rceil \mid sp(s_0, Q) \Rightarrow P$ and $P \Rightarrow wp(S, P) \}$.



2.2. Strongest Invariant

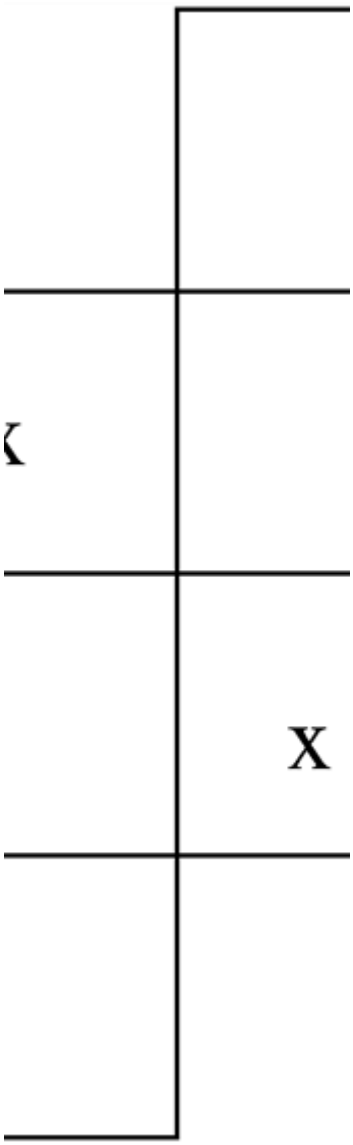
Lemma 1 (Conjunction of invariants). $inv_S(Q)$ is closed for the conjunction operation.

- $INV_S(Q)$ is the conjunction of the elements of the set $inv_S(Q)$
- $INV_S(Q)$ is the strongest invariant.

Theorem 1. The truth set of $INV_S(Q)$ is the set of reachable states from $\lceil Q \rceil$.

2.3. Always True Properties, Definition

- $true_S : \mathcal{P}(A) \longmapsto \mathcal{P}(\mathcal{P}(A))$.
- $true_S(\lceil Q \rceil) \subseteq \mathcal{P}(A)$.
- $true_S(\lceil Q \rceil) ::= \{\lceil P \rceil \mid INV_S(Q) \Rightarrow P\}$
- Always true is not invariant.



Chapter 6. Lecture 6

1. Reminder

1.1. Invariant Properties

- $inv_S(Q)$ is the set of logical functions of which truth are preserved by the elements of S if the program is started from a state satisfying Q.
- $INV_S(Q)$ is the conjunction of the elements of the set $inv_S(Q)$
- $INV_S(Q)$ is the strongest invariant.

2. Program Properties of the Abstract Program

2.1. Unless Properties, Definition

- P is stable while $\neg Q$.
- $\rhd_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.

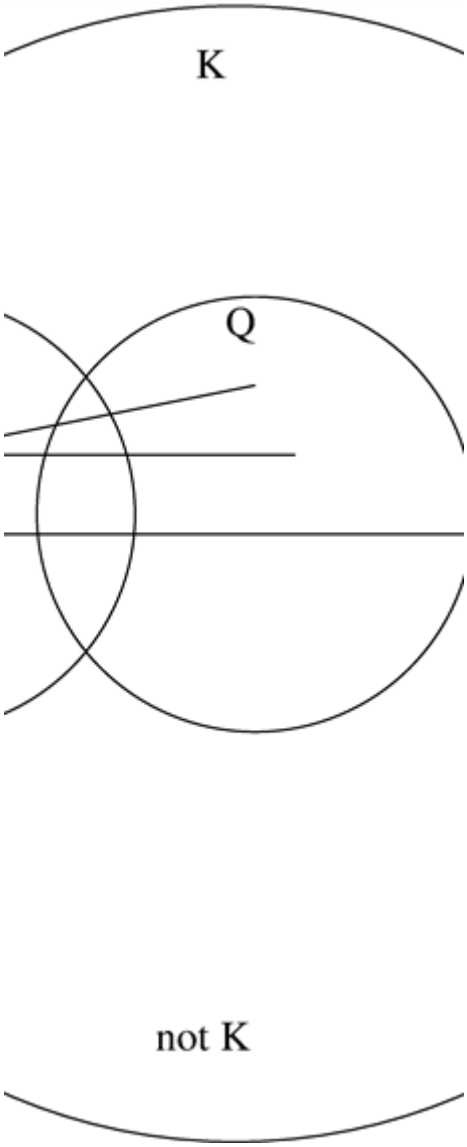
Unless.

$$\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$$

2.2. Unless and Invariant Property

Theorem 2. If $P \rhd_S Q$ and $K \in inv_S(V)$, then $P \wedge K \rhd_S Q \wedge K$.

Theorem 3. If $(P \wedge J) \rhd_S (R \wedge J)$ and $J \in inv_S(Q)$, then $P \wedge INV_S(Q) \rhd_S R \wedge INV_S(Q)$.



2.3. Ensures Property, Definition

- P is stable while $\neg Q$ in S and there is a conditional assignment s_j which ensures the transition from P to Q.
- $\mapsto_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.

Ensures.

$$\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \rhd_S \wedge$$

$$\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$$

Theorem 4. If $(P \wedge J) \mapsto_S (R \wedge J)$ and $J \in inv_S(Q)$, then $P \wedge INV_S(Q) \mapsto_S R \wedge INV_S(Q)$.

2.4. Leads-to Property, Definition

- $\hookrightarrow_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ is the transitive disjunctive closure of relation $\mapsto_S$.

$\hookrightarrow_S$ is the smallest binary relation satisfying the conditions:

- $\mapsto_S \subseteq \hookrightarrow_S$.
- if $(P, Q) \in \hookrightarrow_S$ and $(Q, R) \in \hookrightarrow_S$, then $(P, R) \in \hookrightarrow_S$.
- Let W denote a countable set. If $\forall m : (m \in W :: (P(m), Q) \in \hookrightarrow_S$, then $((\exists m : m \in W :: p(m)), Q) \in \hookrightarrow_S$.

Theorem 5. If $(P \wedge J) \hookrightarrow_S (R \wedge J)$ and $J \in inv_S(Q)$, then $P \wedge INV_S(Q) \hookrightarrow_S R \wedge INV_S(Q)$.

Chapter 7. Lecture 7

1. Reminder

1.1. Program Properties

- $\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$
- $\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \rhd_S \wedge$

  $\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$
- $\hookrightarrow_S$ is the smallest binary relation satisfying the conditions:
  - $\mapsto_S \subseteq \hookrightarrow_S$.
  - if $(P, Q) \in \hookrightarrow_S$ and $(Q, R) \in \hookrightarrow_S$, then $(P, R) \in \hookrightarrow_S$.
  - Let W denote an countable set. If $\forall m : (m \in W :: (P(m), Q) \in \hookrightarrow_S$, then $((\exists m : m \in W :: p(m)), Q) \in \hookrightarrow_S$.

2. Program Properties of the Abstract Program

2.1. Inevitability

Inevitability.

$(\{b\}, \lceil P \rceil) \in \rightsquigarrow_S$, if and only if when on all execution paths leading from b and satisfying the axiom of the unconditionally fair scheduling there is a node at a finite unbounded distance from b of which label is an element of the truth set of P, i.e., the program inevitable reaches the truth set of P started from b.

Theorem 6 ($\hookrightarrow_S$ sound and complete). $\hookrightarrow_S = \rightsquigarrow_S$

2.2. Fixed Point Properties

- A fixed point is said to be reached in a state of the state space A, if none of the statements changes the state.
- $S = (s_0, \{s_1, \ldots, s_m\})$ and $\forall s_j \in S$ is a simultaneous, non deterministic conditional assignment, i.e. : $s_j : \underset{i \in [1,n]}{\|} ((v_i :\in F_{j_i}(v_1, .., v_n)$, if $\pi_{j_i})$
- $\pi_{j_{id}}$ denotes the logical function, which characterizes the set of states over which the relation F_{j_i} is deterministic, i.e., $\pi_{j_{id}}(a) \Leftrightarrow (|F_{j_i}(a)| = 1)$.

2.3. Definitions

Set of fixed point.

$$fixpont_S ::= (\bigwedge_{j \in J, i \in [1..n]} (\neg \pi_{j_i} \vee (\pi_{j_{id}} \wedge v_i = F_{j_i}(v_1, \ldots, v_n))))$$

Set of fixed point with deterministic assignments.

$$fixpont_S ::= (\overset{\wedge}{_{j\in J, i\in[1..n]}}(\pi_{j_i} \to (v_i = F_{j_i}(a))))$$

Fixed point properties.

Let us denote by FP_S the set $\{\lceil R \rceil | fixpont_S \Rightarrow R\}$.

2.4. Example

- $S = (SKIP, \{k := k + 1$, ha $k < N\})$.
- $fixpont_S = (k < N \to k = k + 1) \equiv k \geq N$.

2.5. Weakening of fixed point property

Theorem 7. If $R \Rightarrow Q$ and $R \in FP_S$, then $Q \in FP_S$.

2.6. Termination properties

Termination properties.

$TERM_S$ denotes the set $\{\lceil Q \rceil | (Q, fixpoint_S) \in \hookrightarrow_S\}$.

2.7. Behaviour relation of abstract program

Behaviour relation.

Let S be a program over the state space A . The system of relations $(\triangleright_S, \mapsto_S, \hookrightarrow_S, FP_S, inv_S, TERM_S)$ is called the behaviour relation of the parallel program S .

Chapter 8. Lecture 8

1. Reminder

1.1. Program Properties

- Invariant
- Unless
- Ensures
- Leads-to
- Fixed point
- Termination

2. Solution

2.1. Solution

Definition.

The abstract parallel program $S \subseteq A \times A^{***}$ is a solution of the problem $F \subseteq B \times (\overset{\times}{_{i\in[1..3]}} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \overset{\times}{_{i\in[1..4]}} \mathcal{P}(\mathcal{P}(A)))$,

- if $\forall b \in B : \exists h \in F(b)$, such that
- the program S satisfies all the specification properties given in the inv_h, $\rhd_h$, $\mapsto_h$, $\hookrightarrow_h$, FP_h, $TERM_h$ components

  of h
- assuming that the program starts from a state satisfying all the elements of $INIT_h$.

2.2. Reachable states

- The truth set of an invariant property may be regarded as a characterization of a subset of reachable states.
- It is sufficient for us, if the program satisfies all properties over the truth set of an invariant property.

2.3. Satisfies a specification property

Definition.

The program S satisfies the specification property $(inv_h P)$, if and only if

- there exists an invariant property K such that the program satisfies $(inv_h P)$ with respect to K, i.e.,
- $K \in inv_S(\overset{\wedge}{_{Q\in INIT_h}} Q)$ and $P \wedge K \in inv_S(\overset{\wedge}{_{Q\in INIT_h}} Q)$.

Theorem 8. The program S satisfies the specification property $P \in inv_h$, if it satisfies with respect to the strongest invariant, i.e. $P \in true_S(\bigwedge_{Q \in INIT_h} Q)$ is an always true program property: ( ).

2.4. Satisfies a specification property

Definition.

The program S satisfies the specification property $P \rhd_h Q$, if and only if

- there exists an invariant property K such that the program satisfies $P \rhd_h Q$ with respect to K, i.e.,
- $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ and $(P \wedge K, Q \wedge K) \in \rhd_S$.

Theorem 9. The program S satisfies the specification property $P \rhd_h Q$, if it satisfies with respect to the strongest invariant, i.e.

$$P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \rhd_S Q \wedge INV_S(\bigwedge_{Q \in INIT_h} Q).$$

2.5. Satisfies a specification property

Definition.

The program S satisfies the specification property $P \mapsto_h Q$, if and only if

- there exists an invariant K such that the program satisfies $P \mapsto_h Q$ with respect to K, i.e.,
- $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ and $(P \wedge K, Q \wedge K) \in \mapsto_S$

Theorem 10. The program S satisfies the specification property $P \mapsto_h Q$, if it satisfies with respect to the strongest invariant, i.e.

$$P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \mapsto_S Q \wedge INV_S(\bigwedge_{Q \in INIT_h} Q).$$

2.6. Satisfies a specification property

Definition.

The program S satisfies the specification property $P \hookrightarrow_h Q$, if and only if

- there exists an invariant K such that the program satisfies $P \hookrightarrow_h Q$ with respect to K, i.e.,
- $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ and $(P \wedge K, Q \wedge K) \in \hookrightarrow_S$

Theorem 11. The program S satisfies the specification property $P \hookrightarrow_h Q$, if it satisfies with respect to the strongest invariant, i.e.

$$P \wedge INV_S(\bigwedge_{Q \in INIT_h} Q) \hookrightarrow_S Q \wedge INV_S(\bigwedge_{Q \in INIT_h} Q).$$

2.7. Satisfies a specification property

Definition.

The program S satisfies the specification property $P \hookrightarrow FP_h$, if and only if

- there exists an invariant K such that the program satisfies $P \hookrightarrow FP_h$ with respect to K, i.e.,
- $K \in inv_S(\overset{\wedge}{_{Q \in INIT_h}} Q)$ and $(sp(s_0, P) \wedge K) \in TERM_S$.

Theorem 12. The program S satisfies the specification property $P \hookrightarrow FP_h$, if it satisfies with respect to the strongest invariant, i.e.

$$P \wedge INV_S(\overset{\wedge}{_{Q \in INIT_h}} Q) \hookrightarrow_S fixpont_S.$$

2.8. Satisfies a specification property

Definition.

The program S satisfies the specification property $FP_h \Rightarrow R$, if and only if

- there exists an invariant K such that the program satisfies ($FP_h \Rightarrow R$) with respect to K, i.e.,
- $K \in inv_S(\overset{\wedge}{_{Q \in INIT_h}} Q)$ and $fixpoint_S \wedge K \Rightarrow R$.

Theorem 13. The program S satisfies the specification property $FP_h \Rightarrow R$, if it satisfies with respect to the strongest invariant, i.e.

$$fixpont_S \wedge INV_S(\overset{\wedge}{_{Q \in INIT_h}} Q) \Rightarrow R.$$

2.9. Solved by a program

Definition.

The problem F is said to be solved by the program S with respect to an invariant property K, if $\forall b \in B : \exists h \in F(b)$ such that $K \in inv_S(\overset{\wedge}{_{Q \in INIT_h}} Q)$ and S satisfies all the specification properties given in h with respect to K and the initial conditions $Q \in INIT_h$.

2.10. Set of solutions

Definition.

We define $\Gamma(F)$ as the set of all abstract parallel programs that solve the problem F.

Chapter 9. Lecture 9

1. Reminder

1.1. Solution

Definition.

The abstract parallel program $S \subseteq A \times A^{***}$ is a solution of the problem $F \subseteq B \times (\overset{\times}{_{i\in[1..3]}} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \overset{\times}{_{i\in[1..4]}} \mathcal{P}(\mathcal{P}(A)))$,

- if $\forall b \in B : \exists h \in F(b)$, such that
- the program S satisfies all the specification properties given in the inv_h, $\rhd_h$, $\mapsto_h$, $\hookrightarrow_h$, FP_h, $TERM_h$ components

  of h
- assuming that the program starts from a state satisfying all the elements of $INIT_h$.

1.2. Solved by a Program

Definition.

The problem F is said to be solved by the program S with respect to an invariant property K, if $\forall b \in B : \exists h \in F(b)$ such that $K \in inv_S(\overset{\wedge}{_{Q\in INIT_h}} Q)$ and S satisfies all the specification properties given in h with respect to K and the initial conditions $Q \in INIT_h$.

2. Derivation Rules

2.1. Refinement of a Problem

Definition.

Let F_1, F_2 be problems defined over the state space A.

If $\forall S \in \mathcal{S}_A : S$ solves $F_2 \Rightarrow S$ solves F_1, then the problem F_2 is a refinement of the problem F_1.

2.2. Refinement of Invariant Specification Property

Theorem 14. If the abstract program S satisfies the specification properties $(inv_h P_1)$ and $(inv_h P_2)$, then satisfies the specification property $(inv_h P_1 \wedge P_2)$ too.

2.3. Refinement of Inevitable Specification Property in Finite Steps

Theorem 15. S satisfies to the specification property $P \hookrightarrow_h Q$, if it can be derived by finite number of application of the following rules:

1.

if S satisfies $(P \mapsto_h Q)$, then S satisfies $(P \hookrightarrow_h Q)$ too.

2.

Transitivity: if S satisfies $(P \hookrightarrow_h Q)$ and S satisfies $(Q \hookrightarrow_h R)$, then S satisfies $(P \hookrightarrow_h R)$ too.

3.

Disjunctivity: for all W numerable set: if $\forall m : m \in W :: S$ satisfies $(P(m) \hookrightarrow_h Q)$, then S satisfies $((\exists m : m \in W :: p(m)) \hookrightarrow_h Q)$ too.

2.4. Variant Function

Definition.

- $t : A \longmapsto \mathcal{Z}$ is a variant function.
- $m \in \mathcal{Z} : t = m, t > m : A \longmapsto \mathcal{Z}$ are logical functions:
  - $\lceil t = m \rceil ::= \{a \in A \mid t(a) = m\}$,
  - $\lceil t > m \rceil ::= \{a \in A \mid t(a) > m\}$.

2.5. Application of a Variant Function

Theorem 16. $P, Q : A \longmapsto \mathcal{L}$ logical functions, $t : A \longmapsto \mathcal{Z}$ is a variant function, for which $P \wedge \neg Q \Rightarrow t > 0$.

If $\forall m \in \mathcal{N} :: S$ satisfies $(P \wedge \neg Q \wedge t = m) \hookrightarrow_S ((P \wedge t < m) \vee Q)$, then S satisfies $P \hookrightarrow_S Q$ too.

2.6. $\hookrightarrow_h$ and Variant Function

Theorem 17. $P, Q : A \longmapsto \mathcal{L}$ logical functions, $t : A \longmapsto \mathcal{Z}$ is a variant function, for which $P \wedge \neg Q \Rightarrow t > 0$.

If $\forall m \in \mathcal{N} :: S$ satisfies $(P \wedge \neg Q \wedge t = m) \hookrightarrow_h ((P \wedge t < m) \vee Q)$, then S satisfies $P \hookrightarrow_h Q$ too.

2.7. Termination

Theorem 18. $K \in inv_S(\bigwedge_{Q \in INIT_h} Q)$ and t is a variant function, for which $K \wedge \neg\, fixpont_S \Rightarrow t > 0$. If S satisfies $\neg\, fixpont_S \wedge t = t' \hookrightarrow_h (t < t') \vee fixpont_S$ for all $t' \in \mathcal{N}$, then S satisfies $True \in TERM_h$.

2.8. Refinement of fixed point requirement

Theorem 19. If S satisfies $inv_h P$ and $FP_h \Rightarrow R$, and $P \wedge R \Rightarrow Q$, then S satisfies $FP_h \Rightarrow Q$.

Chapter 10. Lecture 10

1. Reminder

1.1. Reminder

- Problem
- Parallel Abstract Program
- Properties of the Programs
- Solution
- Derivation Rules

2. Program Constructions

2.1. Union

Definition.

- Let A_1 and A_2 be two subspaces of the state space A .
- Let B denote the largest common subspace of A_1 and A_2.
- Let $S_1 = (s_{1,0}, \{s_{1,1}, \ldots, s_{1,k}\})$ and $S_2 = (s_{2,0}, \{s_{2,1}, \ldots, s_{2,m}\})$ be the extensions to A of two programs on A_1 and A_2 respectively.
- If all v_i variables belonging to B get the same value in the assignments $s_{1,0}$ and $s_{2,0}$ (i.e. $F_{1,0_i} = F_{2,0_i}$), then the program

$S_1 \cup S_2 = (s_{1,0}||s_{2,0}, \{s_{1,1}, \ldots, s_{1,k}, s_{2,1}, \ldots, s_{2,m}\})$ that is defined on A , is called the union of S_1 and S_2.

2.2. Behaviour Relation of Union

Theorem 20. Let $S ::= (S_1 \cup S_2)$. Then:

1.

$$\rhd_S = \rhd_{S_1} \cap \rhd_{S_2}$$

2.

$$\mapsto_S = \rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2})$$

3.

$$(\rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2}))^{tdl} = \hookrightarrow_S$$

4.

$\forall Q$ for which $sp(s_{1,0}||s_{2,0}, Q) \Rightarrow sp(s_{1,0}, Q) \wedge sp(s_{2,0}, Q)$: $inv_{S_1}(Q) \cap inv_{S_2}(Q) \subseteq inv_S(Q)$

5.

$$fixpoint_S = fixpoint_{S_1} \wedge fixpoint_{S_2}$$

6.

$$FP_{S_1} \cap FP_{S_2} \subseteq FP_S$$

7.

$$((\rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2}))^{tdl})^{(-1)}(fixpoint_{S_1} \wedge fixpoint_{S_2}) = TERM_S.$$

2.3. Behaviour Relation of Union

Theorem 21. Let F_1 and F_2 be two problems over a common state space and parameter space $\forall b \in B : (F_1 \sqcup F_2)(b) ::=$

1.

$$\{((\rhd_{h_1} \cap \rhd_{h_2}),$$

2.

$$(\rhd_{h_1} \cap \rhd_{h_2} \cap (\mapsto_{h_1} \cup \mapsto_{h_2})),$$

3.

$$(\rhd_{h_1} \cap \rhd_{h_2} \cap (\mapsto_{h_1} \cup \mapsto_{h_2}))^{tdl},$$

4.

$$((\rhd_{h_1} \cap \rhd_{h_2} \cap (\mapsto_{h_1} \cup \mapsto_{h_2}))^{tdl})^{(-1)}(\bigwedge_{S \in \Gamma(F_1) \cup \Gamma(F_2)} fixpoint_S),$$

5.

$$(FP_{h_1} \cap FP_{h_2}),$$

6.

$$(inv_{h_1} \cap inv_{h_2}),$$

7.

$$(INIT_{h_1} \cup INITh_2))$$

$$\mid h_1 \in F_1(b), h_2 \in F_2(b)\}.$$

2.4. Derivation Rule of Union

Theorem 22.

1.

Let F_1 and F_2 be two problems over a common state space A and parameter space B.

2.

Let S_1 and S_2 be two programs extended to state space A, and let the union of this programs exist.

3. If S_1 is a solution of F_1 with respect to K and S_2 is a solution of F_2 with respect to K and

4. $\forall b \in B : \forall h_1 \in F_1(b) : \forall h_2 \in F_2(b) : sp(s_{1,0} \| s_{2,0}, (\overset{\wedge}{_{Q \in INIT_{h_1}}} Q) \wedge (\overset{\wedge}{_{Q \in INIT_{h_2}}} Q)) \Rightarrow sp(s_{1,0}, (\overset{\wedge}{_{Q \in INIT_{h_1}}} Q)) \wedge sp(s_{2,0}, (\overset{\wedge}{_{Q \in INIT_{h_2}}} Q))$,

5. then $S_1 \cup S_2$ is a solution of $F_1 \sqcup F_2$.

2.5. Union and Subset of the State Spaces (1)

Theorem 23. Let $S = S_1 \cup S_2$, π a logical function on state space A in such a way, that $\forall s \in S_2 : p(s) \cap (\lceil \pi \rceil \times A) = id_A \cap (\lceil \pi \rceil \times A)$ and $\pi \rhd_{S_1} False$. In this case:

- if $P \rhd_{S_1} Q$, then $P \wedge \pi \rhd_S Q \wedge \pi$,
- if $P \mapsto_{S_1} Q$, then $P \wedge \pi \mapsto_S Q \wedge \pi$,
- if $P \hookrightarrow_{S_1} Q$, then $P \wedge \pi \hookrightarrow_S Q \wedge \pi$.

2.6. Union and Subset of the State Spaces (2)

Theorem 24. Let $S = S_1 \cup S_2$, π a logical function on state space A in such a way that $\forall s \in S_2 : p(s) \cap (\lceil \pi \rceil \times A) = id_A \cap (\lceil \pi \rceil \times A)$, $\pi \rhd_{S_1} Q$. In this case

- if $P \rhd_{S_1} Q$, then $P \wedge \pi \rhd_S Q$,
- if $P \mapsto_{S_1} Q$, then $P \wedge \pi \mapsto_S Q$,
- if $\neg\pi \wedge \neg Q \rhd_{S_1} False$ and $P \hookrightarrow_{S_1} Q$, then $P \wedge \pi \hookrightarrow_S Q$.

2.7. General Locality Theorem

Theorem 25. S_1 and S_2 are programs on the same state space. $V(S)$ denotes the variables in abstract program S. $X ::= V(S_1) \cap V(S_2)$. If $VR(P) \subseteq V(S_1)$[1], then

- $P \rhd_{S_1} Q \Longrightarrow P \wedge (X = M) \rhd_{S_1 \cup S_2} Q \vee (X \neq M)$,
- $P \mapsto_{S_1} Q \Longrightarrow P \wedge (X = M) \mapsto_{S_1 \cup S_2} Q \vee (X \neq M)$
- $P \hookrightarrow_{S_1} Q \Longrightarrow P \wedge (X = M) \hookrightarrow_{S_1 \cup S_2} Q \vee (X \neq M)$
- $P \hookrightarrow_{S_1} Q$ és $P \wedge (X = M) \rhd_{S_1 \cup S_2} (P \wedge (X < M)) \vee Q \Longrightarrow P \hookrightarrow_{S_1 \cup S_2} Q$.

[1] $V(P) \cap V(S_2) \subseteq X$

Chapter 11. Lecture 11

1. Reminder

1.1. Union

Definition.

- Let A_1 and A_2 be two subspaces of the state space A .
- Let B denote the largest common subspace of A_1 and A_2.
- Let $S_1 = (s_{1,0}, \{s_{1,1}, \ldots, s_{1,k}\})$ and $S_2 = (s_{2,0}, \{s_{2,1}, \ldots, s_{2,m}\})$ be the extensions to A of two programs on A_1 and A_2 respectively.
- If all v_i variables belonging to B get the same value in the assignments $s_{1,0}$ and $s_{2,0}$ (i.e. $F_{1,0\,i} = F_{2,0\,i}$), then the program

$S_1 \cup S_2 = (s_{1,0}||s_{2,0}, \{s_{1,1}, \ldots, s_{1,k}, s_{2,1}, \ldots, s_{2,m}\})$ that is defined on A , is called the union of S_1 and S_2.

2. Program Constructions

2.1. Superposition

Definition.

- Let A_1 be a subspace of A and let $S = (s_0, \{s_1, \ldots, s_m\})$ be a program over A_1.
- Let s be a conditional assignment defined over A in such a way, that none of the variables of A_1 appear on the left hand side in s .
- Let $s_j||s$ denote the superposition of s_j and s .
- Let $S' = (s'_0, \{s'_1, \ldots, s'_m\})$ be the extension of S to A .

The

a) $(s'_0, \{s'_1, \ldots, s'_m, s\})$ and the

b) $(s'_0, \{s'_1, \ldots (s'_j||s), \ldots s'_m\})$, where $j \in [1, m]$

programs are called superpositions of the S program and the s assignment.

2.2. Behaviour Relation of Superposition

Theorem 26. Let the S'' program over state space A be a superposition of the program S and the statement $s ::= \|_{i\in[1,n]} (v_i :\in F_i(v_1, .., v_n)$, if $\pi_i)$, where S is a program over the A_1 subspace of A . Let P and Q be two logical functions over A_1 and let P' and Q' denote the extension of P and Q to A . $fixpoint'_S$ is the extension of the logical function $fixpoint_S$ and $fixpoint_s ::= (\wedge_{i\in[1..n]} (\neg\pi \vee (\pi_{id} \wedge v_i = F_i(v_1, \ldots, v_n))))$.

1. $P \triangleright_S Q \Longrightarrow P' \triangleright_{S''} Q'$,
2. $P \mapsto_S Q \Longrightarrow P' \mapsto_{S''} Q'$,
3. $P \hookrightarrow_S Q \Longrightarrow P' \hookrightarrow_{S''} Q'$,
4. $\forall Q : P \in inv_S(Q) \Longrightarrow P' \in inv_{S''}(Q')$,
5. $fixpoint_{S''} = fixpoint'_S \wedge fixpoint_s$,
6. $R \in FP_S \Longrightarrow R' \in FP_{S''}$,

2.3. Weak Extension of a Problem

Definition.

F'' is the weak extension of the problem F if it is derived from the extension of F, from F', by leaving out the "$Q \in TERM_h$ " type specification conditions.

2.4. Derivation Rule of Superposition

Theorem 27. Let F be a problem over the A_1 subspace of state space A and over the parameter space B. If S is a solution of F then any superposition of the S program and the s statement is a solution of the weak extension of F.

2.5. Sequence of Programs

Definition.

- Let $A_1 = \overset{\times}{_{i \in I_1}} A_{1i}$, $A_2 = \overset{\times}{_{i \in I_2}} A_{2i}$ be two subspaces of state space A.
- Let $S_1 = (s_{1,0}, \{s_{1,1}, \ldots, s_{1,k}\})$ be a program over A_1, $S_2 = (s_{2,0}, \{s_{2,1}, \ldots, s_{2,m}\})$ be a program over A_2.
- Let S'_1, S'_2 denote the extension of S_1, S_2 to A.
- Let u be a logical variable, where the state space component of u neither belongs to A_1 nor to A_2.

2.6. Sequence of Programs (cont.)

Definition (cont.)

- Let S^1 denote the $(s_0, \{s_1, \ldots, s_k\})$ program defined on state space $\overset{\times}{_{i \in I_1}} A_{1i} \times \mathcal{L}$, where
  - $s_0 = (s'_{1,0} \| u := false)$,
  - $\forall i \in [1..k] : s_i = ((s'_{1,i})$, if $\neg u$).
- Let S^2 denote the $(SKIP, \{s_{k+2}, \ldots, s_{k+m+1}\})$ program defined on state space $\mathcal{L} \overset{\times}{_{i \in I_2}} A_{2i}$,

  where
  - $\forall i \in [k+2..k+m+1] : s_i = ((s'_{2,i-(k+2)+1})$, if u).
  - $s_{k+1} = ((s'_{2,0} \| u := true)$, if $\neg u \wedge fixpoint_{S_1})$.

2.7. Sequence of Programs (cont.)

Definition (cont.)

The $S = (S^{1\prime} \cup S^{2\prime} \cup (SKIP, \{s_{k+1}\})')$ program is called the sequence of S_1, S_2 and is denoted as $S_1; S_2$.

2.8. Behaviour Relation of Sequence

Theorem 28. In the following we suppose that the predicates P, Q, etc. are independent of the variable u. P' and Q' are the extensions of the logical functions of P and Q respectively. Let $S = S_1; S_2$. Then:

1. if $P \rhd_{S_1} fixpoint_{S_1}$, then $P' \wedge \neg u \rhd_S fixpoint'_{S_1} \wedge \neg u$,
2. if $P \mapsto_{S_1} fixpoint_{S_1}$, then $P' \wedge \neg u \mapsto_S fixpoint'_{S_1} \wedge \neg u$,
3. if $P \hookrightarrow_{S_1} fixpoint_{S_1}$, then $P' \wedge \neg u \hookrightarrow_S fixpoint'_{S_1} \wedge \neg u$,
4. if $P \rhd_{S_2} Q$, then $P' \wedge u \rhd_S Q' \wedge u$,
5. if $P \mapsto_{S_2} Q$, then $P' \wedge u \mapsto_S Q' \wedge u$,
6. if $P \hookrightarrow_{S_2} Q$, then $P' \wedge u \hookrightarrow_S Q' \wedge u$,

7.

$u \wedge fixpoint'_{S_2} = fixpoint_S$,

8.

if $R \in FP_{S_2}$ then $R' \in FP_S$,

2.9. Behaviour Relation of Sequence (cont.)

Theorem 29. In the following we suppose that the predicates P, Q, etc. are independent of the variable u. P' and Q' are the extensions of the logical functions of P and Q respectively. Let $S = S_1; S_2$. Then:

1.

$P \in inv_{S_1}(Q)$ iff $(P' \vee u) \in inv_S(Q')$, $P \in inv_{S_2}(Q)$ iff $(P' \vee \neg u) \in inv_S(Q')$,

2.

$P \in inv_{S_1}(Q)$ and $P \in inv_{S_2}(P \wedge fixpoint_{S_1})$ iff $P' \in inv_S(Q')$,

3.

if $P \hookrightarrow_{S_1} Q$ then $(P' \wedge \neg u) \hookrightarrow_S (Q' \wedge \neg u)$,

4.

if $P \hookrightarrow_{S_1} Q$ and $P \hookrightarrow_{S_2} Q$ then $P' \hookrightarrow_S Q'$.

2.10. Derivation Rule of Program Sequencing

Theorem 30.

- Let A_1 and A_2 subspaces of state space A.
- Let F_1 and F_2 deterministic problems over A_1 and A_2 resp. and over parameter space B.
- Let $S_1;S_2$ be the sequence of S_1 (defined over A_1) and S_2 (defined over A_2).
- For any $b \in B$ we mark the components of $F_1(b)$ with $_{F_1}$, the components of $F_2(b)$ with $_{F_2}$.

2.11. Derivation Rule of Program Sequencing (cont.)

Theorem 31.

- If S_1 satisfies $P \in TERM_b^{F_1}$ and $R \in FP_b^{F_1}$ conditions under precondition $P \in INIT_b^{F_1}$,
- S_2 satisfies $Q \in TERM_b^{F_2}$ and $Z \in FP_b^{F_2}$ conditions under precondition $Q \in INIT_b^{F_2}$, and

- $R' \Rightarrow Q'$, then

  .

  $S_1; S_2$ satisfies $P' \in TERM_b$ and $Z' \in FP_b$ conditions under $P' \in INIT_b$ precondition.

Chapter 12. Lecture 12

1. Reminder

1.1. Program Constructions

- Union
- Superposition
- Sequence

2. Computation of the Value of an Associative Function

2.1. Notations

- Let H be a set.
- Let $\circ : H \times H \longmapsto H$ denote an arbitrary associative binary operator over H.
- $f : H^* \longmapsto H$ is a function describing the single or multiple application of the operator $\circ$.

2.2. Notations

- Since $\circ$ is associative, for any arbitrary sequence $x \in H^*$ of length at least three:

  $f(\ll x_1, ..., x_{|x|} \gg) =$

  $f(\ll f(\ll x_1, .., x_{|x|-1} \gg), x_{|x|} \gg) =$

  $f(\ll x_1, f(\ll x_2, .., x_{|x|} \gg) \gg).$

- We write $f(\ll h_1, h_2 \gg)$ instead of the infix notation $(h_1 \circ h_2)$ in the following.
- We extend f for sequences of length one: $f(\ll h \gg) = h$.

2.3. Notations – The Problem

- Let a finite sequence $a \in H^*$ of the elements of H be given.
- $a = \ll a_1, ..., a_n \gg, \ (n \geq 1)$.
- Let us compute the value of the function $\mathcal{G} : [1..n] \longmapsto H$ for all $i \in [1..n]$, where $n \geq 1$ and $\mathcal{G}(i) = f(\ll a_1, ..., a_i \gg)$.

2.4. The Formal Specification of the Problem

- We represent the sequences a and the values of function $\mathcal{G}_a$ by arrays.
- We specify that the program inevitably reaches a fixed point and the array g contains the values of f in any fixed point.

- $G = vektor([1..n], H), \quad n \geq 1$.

2.5. The Formal Specification of the Problem

$$
\begin{array}{lll}
A = & G\times & G \\
 & g & a \\
B = & G & \\
 & a' &
\end{array}
$$

$$(a = a') \in INIT_{a'} \tag{12.1}$$

$$(a = a') \hookrightarrow FP_{a'} \tag{12.2}$$

$$FP_{a'} \Rightarrow (a = a' \wedge \forall i \in [1..n] : g(i) = f(\ll a_1, ..., a_i \gg)) \tag{12.3}$$

2.6. Properties of Associative Operators

- The computation of the values of $\mathcal{G}$ at place i is made easier with the knowledge of the value of f for subsequences $f(\ll a_u, ..., a_v \gg)$ indexed by the elements of an arbitrary $[u..v] \subseteq [1..i]$ interval.
- The result computed for a subsequence is useful in the computation of the value of f for any sequence which includes the subsequence.

2.7. Auxiliary Function

- Let us introduce the auxiliary function h .
- Let $h(i,k)$ denote the value of f for the sequence of which the first element is a_i and its length is 2^k or the last element is a_1, if $i < 2^k$.

Definition.

The precise definition of the partial function $h : [1..n] \times \mathcal{N}_0 \longrightarrow H$ is:

$$h(a,i,k) = \begin{cases} f(\ll a_1, ..., a_i \gg), \\ \quad \text{if } i - 2^k + 1 \leq 1 \\ f(\ll a_{(i-2^k+1)}, ..., a_i \gg), \\ \quad \text{if } i - 2^k + 1 \geq 1 \end{cases}$$

2.8. Auxiliary Function

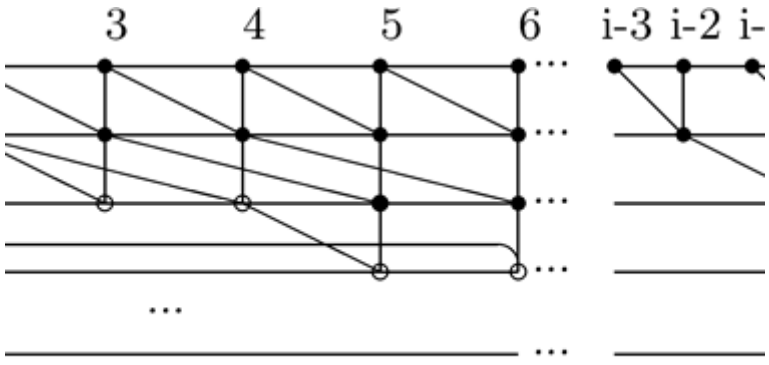
Lemma 2. If $(i - 2^k \geq 1)$, then $f(\ll h(a, i - 2^k, k), h(a,i,k) \gg) = h(a,i,k+1)$.

2.9. Substitution of a Function by a Variable

- The two-dimensional array gs is introduced to store the known values of h .
- This method is called the substitution of a function by a variable.
- The lines on the next Figure illustrate the connections among the elements of the matrix gs .
- In fixed points $gs(i, k(i)) = h(i, k(i))$ and $\lceil log(i) \rceil = k(i)$,

i.e. $gs(i,k(i))$ is the value of f for an at most length 2^k prefix.

2.10. Substitution of a Function by a Variable



$$\circ : k(i) = \lceil log(i) \rceil$$

2.11. Variant Function

- Let us choose the variant function $v : A \longmapsto \mathcal{N}_0$ in the following way:

.

$$v ::= 4 * n * n - \sum_{i=1}^{n} (k(i) + \chi(k(i) = \lceil \log(i) \rceil \wedge g(i) = gs(i, k(i))))$$

- The variant function depends on the number of elements of the matrix gs which elements are different from the value of function h at the corresponding place and on the number of places where the value of the array g is different from the value of function $\mathcal{G}$.

2.12. Refining the Specification of the Problem

- We extend the state space and refine the specification of the problem.

$$\begin{array}{lccccc} A' = & G\times & G\times & GS\times & K\times & T \\ & g & a & gs & k & t \end{array}$$

$$\begin{array}{ll} G & = vektor([1..n], H), \\ GS & = vektor([1..n, 0..(\lceil \log(n) \rceil)], H) \\ K & = vektor([1..n], \mathcal{N}_0), \\ T & = vektor([1..n], \mathcal{N}_0), \quad n \geq 1 \end{array}$$

2.13. Refining the Specification of the Problem

$$(a = a') \in INIT_{a'} \tag{12.4}$$

$$(a = a') \hookrightarrow FP_{a'} \tag{12.5}$$

$$FP_{a'} \Rightarrow \forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil) \wedge (g(i) = gs(i, \lceil \log(i) \rceil)) \tag{12.6}$$

$$inv_{a'}(\forall i \in [1..n] : k(i) \leq \lceil \log(i) \rceil \wedge \forall k : k \leq k(i) : gs(i,k) = h(a,i,k)) \tag{12.7}$$

$$inv_{a'}(\forall i \in [1..n] : t(i) = 2^{k(i)}) \tag{12.8}$$

$$inv_{a'}(a = a') \tag{12.9}$$

2.14. Refining the Specification of the Problem

- The connection between the variables gs, k, t and the function h is given by the invariants (6)-(8).

2.15. Refining the Specification of the Problem

Lemma 3. The given specification ((4)-(9)) is a refinement of the original specification ((1)-(3)).

Proof. $k(i) = \lceil \log(i) \rceil$ and $g(i) = gs(i, \lceil \log(i) \rceil)$ in fixed point according to (6).

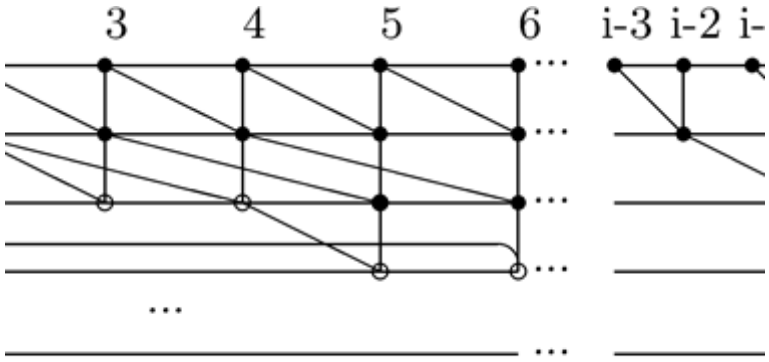
Using (7) it follows that the equation $g(i) = gs(i, \lceil \log(i) \rceil) = h(i, \lceil \log(i) \rceil)$ holds in fixed point.

Since $2^{\lceil \log(i) \rceil} \geq i$, after the application of the definition of h we get $h(i, \lceil \log(i) \rceil) = f(\ll a_i, ..., a_1 \gg)$, which is the same as property (3).

Chapter 13. Lecture 13

1. Reminder

1.1. Computation of the Value of an Associative Function



$$\circ : k(i) = \lceil log(i) \rceil$$

1.2. The Formal Specification of the Problem

$$\begin{array}{lll} A = & G\times & G \\ & g & a \\ B = & G & \\ & a' & \end{array}$$

$$(a = a') \in INIT_{a'} \tag{13.1}$$

$$(a = a') \hookrightarrow FP_{a'} \tag{13.2}$$

$$FP_{a'} \Rightarrow (a = a' \wedge \forall i \in [1..n] : g(i) = f(\ll a_1, ..., a_i \gg)) \tag{13.3}$$

1.3. Refined Specification of the Problem

$$\begin{array}{lllll} A' = & G\times & G\times & GS\times & K\times & T \\ & g & a & gs & k & t \end{array}$$

$$\begin{array}{ll} G & = vektor([1..n], H), \\ GS & = vektor([1..n, 0..(\lceil \log(n) \rceil)], H) \\ K & = vektor([1..n], \mathcal{N}_0), \\ T & = vektor([1..n], \mathcal{N}_0), \quad n \geq 1 \end{array}$$

1.4. Refined Specification of the Problem

$$(a = a') \in INIT_{a'} \tag{13.4}$$
$$(a = a') \hookrightarrow FP_{a'} \tag{13.5}$$
$$FP_{a'} \Rightarrow \forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil) \wedge (g(i) = gs(i, \lceil \log(i) \rceil)) \tag{13.6}$$
$$inv_{a'}(\forall i \in [1..n] : k(i) \leq \lceil \log(i) \rceil \wedge \forall k : k \leq k(i) : gs(i,k) = h(a,i,k)) \tag{13.7}$$
$$inv_{a'}(\forall i \in [1..n] : t(i) = 2^{k(i)}) \tag{13.8}$$
$$inv_{a'}(a = a') \tag{13.9}$$

2. Solution of the Problem

2.1. Solution of the Problem

$$s_0 : \quad \square_{i=[1..n]}\, gs(i,0), t(i), k(i) = f(\ll a_i \gg), 1, 0$$

$$S : \{ \quad \square_{i=[1..n]}\, gs(i, k(i)+1), t(i), k(i) := \begin{cases} f(\ll gs \quad ((i - t(i)), k(i)), gs(i, k(i)) \gg), \\ \qquad\qquad 2 * t(i), k(i) + 1, \\ \qquad\qquad \text{if } (i - 2 * t(i) + 1 \geq 1) \wedge \\ \qquad\qquad (k(i - t(i)) \geq k(i)) \\ f(\ll gs \quad (i - t(i), k(i - t(i))), gs(i, k(i)) \gg), \\ \qquad\qquad 2 * t(i), k(i) + 1, \\ \qquad\qquad \text{if } (i - t(i) \geq 1) \wedge \\ \qquad\qquad (i - 2 * t(i) + 1 < 1) \\ \qquad\qquad \wedge (k(i - t(i)) = \lceil \log(i - t(i)) \rceil \end{cases} \square_{i=[1..n]}\, g(i) := gs(i, k(i)), \text{if } (k(i) = \lceil \log(i) \rceil) \quad \}$$

2.2. Solution of the Problem

Theorem 32. The abstract program below is a solution for the problem specified by (4)-(9), i.e., a solution for the problem of the computation of the values of an associative function.

2.3. The Program Solves the Problem

Proof. (6): using the definition of FP_S :

$$\forall i \in [1..n]$$
$$(k(i) = \lceil \log(i) \rceil) \rightarrow g(i) = gs(i, k(i)) \wedge \tag{13.10}$$
$$((i - 2 * t(i) + 1 < 1) \vee (k(i - t(i)) < k(i)) \wedge \tag{13.11}$$
$$(i - t(i) < 1) \vee (i - 2 * t(i) + 1 \geq 1) \vee (k(i - t(i)) \neq \lceil \log(i - (t(i))) \rceil) \tag{13.12}$$

We use invariant properties and apply mathematical induction on i to prove that the program satisfies $\forall i \in [1..n] : (k(i) = \lceil \log(i) \rceil)$ in fixed points.

2.4. The Program Solves the Problem

Base Case. $i = 1$. From (7) and $sp(s_0, \uparrow)$ follows $(k(1) = \lceil \log(1) \rceil)$.

Inductive hypothesis. $\forall j < i : (k(j) = \lceil \log(j) \rceil)$.

2.5. The Program Solves the Problem

Proof.

- Since $t(i) \geq 1$, $(k(i - t(i)) \neq \lceil \log(i - (t(i)) \rceil)$ contradicts the hypothesis.
- This means (12) can be simplified to $(i - t(i) < 1) \vee (i - 2 * t(i) + 1 \geq 1)$.
- If $(i - 2 * t(i) + 1 \geq 1)$, then $k(i - t(i)) < k(i)$, else (11) does not hold.
- Using the inductive hypothesis and $t(i) \geq 1$ we get $k(i - t(i)) = \lceil \log(i - t(i)) \rceil$, i.e., $\lceil \log(i - t(i)) \rceil < k(i)$.

2.6. The Program Solves the Problem

Proof.

- The last statement contradicts the initial condition: $(i - 2 * t(i) + 1 \geq 1)) \Rightarrow (i - t(i) - t(i) + 1 \geq 1)) \Rightarrow \lceil \log(i - (t(i)) \rceil \geq k(i)$.
- This means $(i - 2 * t(i) + 1 < 1))$.
- $(i - 2 * t(i) + 1 < 1)) \Rightarrow (i - t(i) < 1)$, else (12) does not hold.
- $(i - t(i) < 1) \Rightarrow k(i) \geq \lceil \log(i) \rceil$.
- Using the invariant (7) we get $k(i) = \lceil \log(i) \rceil$.
- Based on (10) $g(i) = gs(i, k(i)) = gs(i, \lceil \log(i) \rceil)$.

2.7. The Program Solves the Problem

Proof. (5):

- Every statement of the program decreases the variant function by 1 or does not cause state transition.
- If the program is not in one of its fixed points, then there exists an $i \in [1..n]$ and a corresponding conditional assignment, which assignment increases the value of $k(i)$, or there exists an i for which $k(i) = \lceil \log(i) \rceil$ and the value of $g(i)$ is different from the value of $gs(i, (\lceil \log(i) \rceil))$

2.8. The Program Solves the Problem

Proof. (8):

- Since $sp(s_0, \uparrow)$ implies $t(i) = 1$ and $k(i) = 0$, the $t(i) = 2^{k(i)}$ equality holds initially.
- All the assignments change the value of $k(i)$ and $t(i)$ simultaneously.

2.9. The Program Solves the Problem

Proof. (7):

- Since $h(i,0) = f(\ll a(i) \gg)$, $sp(s_0, \uparrow) \Rightarrow gs(i,k(i)) = h(i,k(i))$.
- Since $k(i)$ is initially 0, $sp(s_0, \uparrow) \Rightarrow (k(i) \leq \lceil \log(i) \rceil)$.
- After calculating the weakest preconditions of the assignments it is sufficient to show that ...

2.10. The Program Solves the Problem

Proof.

- After calculating the weakest preconditions of the assignments it is sufficient to show that
  - $(i - 2 * t(i) + 1 \geq 1) \wedge (k(i - t(i)) \geq k(i))$ and $\forall k : k \leq k(i) : gs(i,k) = h(i,k)$ implies the equality for $k(i)+1$, i.e., $f(\ll gs(i,k(i)), gs(i - t(i), k(i)) \gg) = h(i, k(i)+1)$ and $k(i)+1 \leq \lceil \log(i) \rceil$,
  - $(i - t(i) \geq 1) \wedge (i - 2 * t(i) + 1 < 1) \wedge (k(i - t(i)) = \lceil \log(i) \rceil$ and $\forall k : k \leq k(i) : gs(i,k) = h(i,k)$ implies the equality for $k(i)+1$, i.e., $f(\ll gs(i,k(i)), gs(i - t(i), (\lceil \log(i - t(i)) \rceil)) \gg) = h(i, k(i)+1)$ and $k(i)+1 \leq \lceil \log(i) \rceil$.

2.11. The Program Solves the Problem

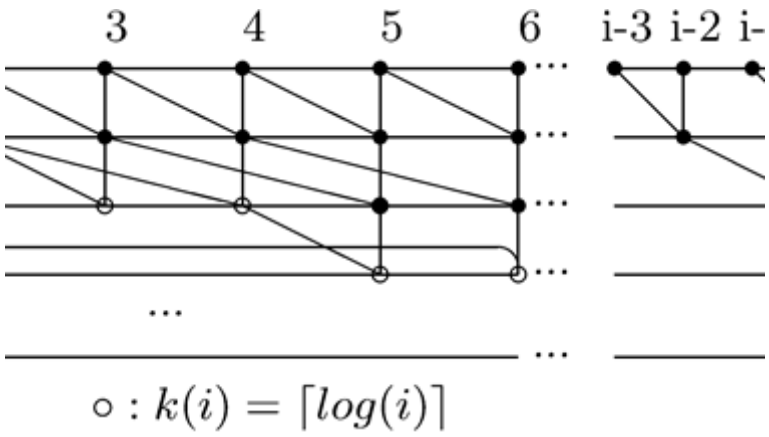
Proof.

- $(i - 2 * t(i) + 1 \geq 1) \wedge (t(i) \geq 1) \Rightarrow (i - t(i) \geq 1) \Rightarrow k \leq \log(i-1) < \log(i) \leq \lceil \log(i) \rceil)$.
- n the first case $k(i) \leq k(i)$ implies $gs(i,k(i)) = h(i,k(i))$ and $(k(i - t(i)) \geq k(i))$ implies $gs(i - t(i), k(i)) = h(i - t(i), k(i))$.
- In the second case $k(i) \leq k(i)$ implies $gs(i,k(i)) = h(i,k(i))$ and $k(i - t(i)) = \lceil \log(i - t(i)) \rceil$ implies $gs(i - t(i), (\lceil \log(i) \rceil)) = h(i - t(i), (\lceil \log(i) \rceil))$.
- We use the Lemma: If $(i - 2^k \geq 1)$, then $f(\ll h(a, i - 2^k, k), h(a,i,k) \gg) = h(a,i,k+1)$.
- In both of the cases the application of the Lemma leads to the statement.

Chapter 14. Lecture 14

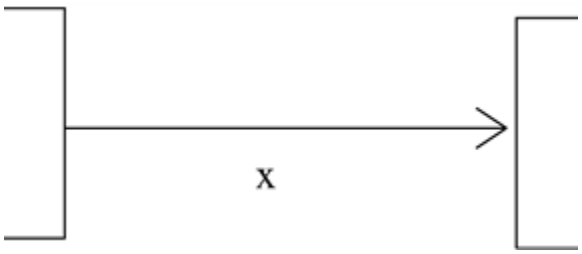
1. Reminder

1.1. Computation of the Value of an Associative Function



2. Channels

2.1. Channels



- $x : Ch(Int)$ – queue, buffer for one directional communication
- Error-free, unbounded or bounded
- $\overline{x}$ – the history of the channel
- Operations:
  - $x := x; e = hiext(x, e)$ (P1)
  - $x := lorem(x), \text{if } x \neq <>$ (P2)
  - $e := lov(x) = x.lov, \text{if } x \neq <>$
  - $x :=<>$
  - $n := length(x) = |x|$

2.2. Semantics of Operations

- $wp(x := x; e,\ R) = R^{x \leftarrow x;e,\ \overline{x} \leftarrow \overline{x};e}$
- $wp(x := lorem(x),\ \text{if } x \neq <>,\ R) =$

$(x \neq <> \rightarrow R^{x \leftarrow lorem(x)}) \wedge (x = <> \rightarrow R)$.

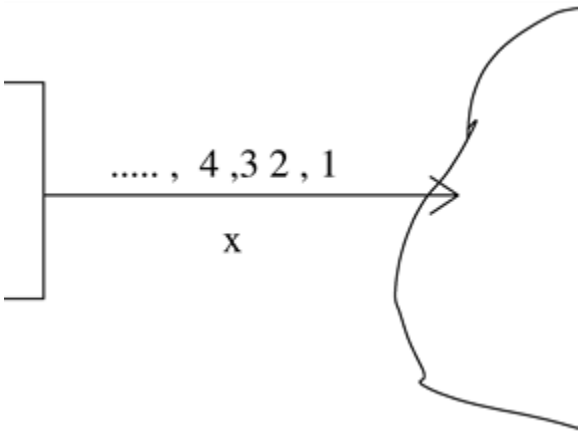
- $wp(x := <>,\ R) = R^{x \leftarrow <>, \overline{x} \leftarrow <>}$.

- Locality: any property P of P1 is stable in the other process(es), if $V(P)$ contains local variables and outgoing channels variables of P1 only.

- For any property P, if $P \Rightarrow P^{\overline{x} \leftarrow \overline{x}; e}$ and $V(P) = \{\overline{x}\}$, then P is stable in the system.

3. Natural Number Generator

3.1. Example – Natural Number Generator (NNG)



$$\begin{array}{lll} A = & Ch(Int) \times & Ch(Int) \\ & x & \overline{x} \\ B = & Ch(Int) \times & Ch(Int) \\ & x' & \overline{x}' \end{array}$$

$$(x = x' = <> \wedge \overline{x} = \overline{x}' = <>) \in INIT_{x', \overline{x}'} \quad (14.1)$$

$$\overline{x} \leq [1, 2, ..] \in inv_{x', \overline{x}'} \quad (14.2)$$

$$\forall k \in N_0 : |\overline{x}| = k \hookrightarrow_{x', \overline{x}'} |\overline{x}| = k + 1 \quad (14.3)$$

3.2. NNG –Refinement of the Problem

$$\begin{array}{lllll} A = & Ch(Int) \times & Ch(Int) \times & N_0 \\ & x & \overline{x} & i \\ B = & Ch(Int) \times & Ch(Int) \\ & x' & \overline{x}' \end{array}$$

$$(x = x' = <> \wedge \overline{x} = \overline{x}' = <>) \in INIT_{x', \overline{x}'} \quad (14.4)$$

$$i \in N_0 \in inv_{x', \overline{x}'} \quad (14.5)$$

$$((i = 0 \wedge \overline{x} = <>) \vee (i > 0 \wedge \overline{x} = [1, ..i])) \in inv_{x', \overline{x}'} \quad (14.6)$$

$$\forall k \in N_0 : |\overline{x}| = k \mapsto_{x', \overline{x}'} |\overline{x}| = k + 1 \quad (14.7)$$

3.3. NNG –Solution

$S:$
$($ $s_0 : i := 0,$

$\{s_1 : x, i := x; (i+1), i+1\}$
$)$

3.4. The Program Solves the Problem

Proof. (5):

- We show $i \in N_0 \in inv_S(x = x' =<> \wedge \overline{x} = \overline{x}' =<>)$
- $sp(i := 0, x = x' =<> \wedge \overline{x} = \overline{x}' =<>) = i = 0 \wedge x = x' =<> \wedge \overline{x} = \overline{x}' =<> \Rightarrow i \in N_0$
- $i \in N_0 \Rightarrow (wp(x, i := x; (i+1), i+1), i \in N_0) = i + 1 \in N_0$

3.5. The Program Solves the Problem

Proof. (6):

- $sp(i := 0, x = x' =<> \wedge \overline{x} = \overline{x}' =<>) = i = 0 \wedge x = x' =<> \wedge \overline{x} = \overline{x}' =<> \Rightarrow i = 0 \wedge x' =<>$
- $((i = 0 \wedge \overline{x} =<>) \vee (i > 0 \wedge \overline{x} = [1, .., i])) \Rightarrow (wp(x, i := x; (i+1), i+1), ((i = 0 \wedge \overline{x} =<>) \vee (i > 0 \wedge \overline{x} = [1, .., i]))) = ((i+1 = 0 \wedge \overline{x}; (i+1) =<>) \vee (i+1 > 0 \wedge \overline{x}; i = [1, .., i+1])) = \overline{x}; i = [1, .., i+1])$

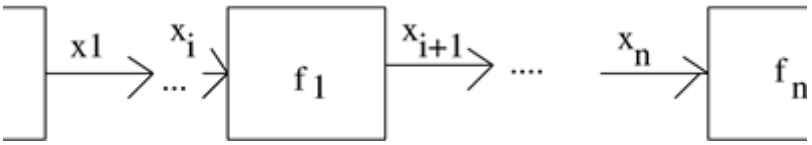
3.6. The Program Solves the Problem

Proof. (7):

- $\forall k \in N_0 :$
- $|\overline{x}| = k \rhd_{x', \overline{x}'} |\overline{x}| = k + 1 \Leftrightarrow |\overline{x}| = k \Rightarrow wp(S, |\overline{x}| = k \vee |\overline{x}| = k + 1) =$ $|\overline{x}; (i+1)| = k \vee |\overline{x}; (i+1)| = k + 1$ and
- $|\overline{x}| = k \Rightarrow wp(s, |\overline{x}| = k + 1) = |\overline{x}; (i+1)| = k + 1$

4. Pipeline

4.1. Pipeline



- $F = f_n \circ ... \circ f_0$.
- $D =< d_1, .., d_m >$.
- $m >> n$

4.2. Specification of Pipeline

A	$=$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	
		x_0		$\overline{x_0}$		x_{n+1}		$\overline{x_{n+1}}$	

B	$=$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	
		x_0'		$\overline{x_0}'$		x'_{n+1}		$\overline{x_{n+1}}'$	

$$Q ::= (x_0 = \overline{x_0} = x_0' = \overline{x_0}' = D \wedge$$
$$\wedge x_{n+1} = \overline{x_{n+1}} = x'_{n+1} = \overline{x_{n+1}}' = <>)$$
$$Q \in INIT_{\underbrace{x_0'\overline{x_0}'x'_{n+1}\overline{x_{n+1}}'}_{h}} \tag{14.8}$$
$$FP_h \Rightarrow \overline{x_{n+1}} = F(\overline{x_0}') = F(D) \tag{14.9}$$
$$Q \in TERM_h \tag{14.10}$$
$$(\overline{x_0} = \overline{x_0}' = D) \in inv_h \text{ for the whole system} \tag{14.11}$$

4.3. Refinement of the Problem

$$FP_h \Rightarrow \forall i \in [0..n] : x_i = <> \tag{14.12}$$
$$\forall i \in [0..n] : (f_i(\overline{x_i} - x_i) = \overline{x_{i+1}}) \in inv_h \tag{14.13}$$
$$\text{Variant function: } (|x_0|, ..., |x_n|) \tag{14.14}$$

4.4. Refinement of the Problem

Proof.

- By fixed point refinement it is sufficient: $(12) \wedge (13) \wedge (11) \Rightarrow (9)$.
- Proof by using the lemma: $(12) \wedge (13) \wedge (11) \Rightarrow (\overline{x_{i+1}}) = f^i(D)$.
- The lemma is proved by induction.

4.5. Solution

$$S: \quad (\|_{i=1}^{n}\ x_i := <>,$$
$$\{ \quad \square_{i=0}^{N} x_i, x_{i+1} := lorem(x_i), hiext(x_{i+1}, f_i(x_i.lov)),$$
$$\text{if } x_i \neq <>\})$$

Chapter 15. Practice 1

1. Definitions

1.1. Relations

- An arbitrary subset of a direct product of sets is called a relation.
- Let $R \subseteq A \times B$ where A and B are arbitrary sets. The domain of the relation R is defined by $\mathcal{D}_R ::= \{a \in A \mid \exists b \in B : (a,b) \in R\}$.

1.2. State Space

- Let $I \subset \mathcal{N}.\ \forall i \in I : A_i$ is a finite or numerable set.
- The set $A ::= \overset{\times}{_{i \in I}} A_i$ is called *state space*, the sets A_{i_j} are called *type value sets*.
- The projections $v_i : A \longmapsto A_i$ are called variables.
- A^* is the set of the finite sequences of the points of the state space and A^∞ the set of the infinite sequences.
- Let $A^{**} = A^* \cup A^\infty$.
- A *statement* is a subset of the direct product $A \times A^{**}$.

1.3. Statements and Effect Relation

- A *statement* is a subset of the direct product $A \times A^{**}$.
- The *effect relation* of a statement s is denoted by $p(s)$.
- The effect relation expresses the functionality of the statement.
- $p(s) \subseteq A \times A,\ \mathcal{D}_{p(s)} = \{a \in A | p(s)(a) \subseteq A^*\}$.

1.4. Partial Function and Logical Relation

- A relation R is called a *partial function*, if for all $a \in A$ the set $R(a)$ has at most one element. If $\forall a \in A : |R(a)| = 1$ then R is a function.
- If $R \subseteq A \times \mathcal{L}$ is a relation, where A is an arbitrary set and $\mathcal{L}$ is the set of the logical values, then R is called a *logical relation*.

1.5. Truth Set

- The *truth set of the logical function* $f : A \longmapsto \mathcal{L}$ is defined as $\lceil f \rceil ::= \{a \in A \mid f(a) = true\}$.
- The logical functions $True, False : A \longmapsto \mathcal{L}$ are defined by their truth sets. $\lceil True \rceil = A, \lceil False \rceil = \emptyset$.

1.6. General Assignment

- A statement over the state space A is called empty and termed $SKIP$, if $\forall a \in A : SKIP(a) = \{< a >\}$.
- Let $A = A_1 \times ... \times A_n$, $F = (F_1, ..., F_n)$, where $F_i \subseteq A \times A_i$.
- The statement $s \subseteq A \times A^{**}$ is a general assignment defined by F, if
  - $s ::= \{(a, red(< a, b >)) \mid a, b \in A \wedge a \in \bigcap_{i \in [1,n]} \mathcal{D}_{F_i} \wedge b \in F(a)\} \cup \{(a, < a, a, a.... >) \mid a \in A \wedge a \notin \bigcap_{i \in [1,n]} \mathcal{D}_{F_i}\}$.

1.7. Conditional Assignment

- Let be s_j an assignment, for which $((\mathcal{D}_{p(s_j)} = A) \wedge (\forall a \in A : p(s_j)(a) = F||_{\uparrow}(a)))$.
- This kind of (simultaneous, nondeterministic) assignment is called a conditional assignment, if $\forall a \in A : |p(s_j)(a)| < \omega$.
- We denote the conditional assignment s_j the following way: $(\parallel_{i \in [1,n]} (v_i :\in F_{j_i}(v_1, .., v_n)$, if $\pi_{j_i}))$.
- Simultaneous, nondeterministic, conditional assignment: $(v_i :\in F_i(v_1, .., v_n)$, if $\pi_i \parallel (v_k :\in F_k(v_1, .., v_n)$, if $\pi_k)$.
- Abbreviation: $\parallel_{i \in [1,n]} (...)$

1.8. Abstract Parallel Program

- $S = (s_0, \{s_1, ...s_m\})$
  - The conditional assignment s_0 is called the initialization in S and
  - $s_j : j \in [1..m]$ is said to be an element of the program S.

1.9. Weakest precondition

- The logical function $wp(s, R) : A \longmapsto \mathcal{L}$ is called the weakest precondition of the postcondition R in respect to the statement s.
- $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- $wp(S, R) ::= \forall s \in S : wp(s, R)$.

1.10. Strongest Postcondition

- The logical function $sp(s, Q)$ is called the strongest postcondition of Q in respect to s.
- $\lceil sp(s, Q) \rceil ::= p(s)(\lceil Q \rceil)$.

1.11. WP of the Abstract Parallel Program

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- $wp(S, R) ::= \forall s \in S : wp(s, R)$.
- $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
  - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.

1.12. Properties of WP

- $wp(s, \downarrow) = \downarrow$
- $wp(s, \uparrow) = \uparrow$, if $\mathcal{D}_{p(s)} = \mathcal{A}$
- $P \Rightarrow Q \Longrightarrow wp(s, P) \Rightarrow wp(s, Q)$
- $wp(s, P) \wedge wp(s, Q) \Longleftrightarrow wp(s, P \wedge Q)$
- $wp(s, P) \vee wp(s, Q) \Longrightarrow wp(s, P \vee Q)$
- $\lceil lf(s, P) \rceil = \lceil R \circ p(s) \rceil$
- $wp(SKIP, R) = R$

1.13. Properties of WP

- $wp(S, \downarrow) = \downarrow$,
- $wp(S, \uparrow) = \uparrow$,
- If $P \Rightarrow Q$, then $wp(S, P) \Rightarrow wp(S, Q)$,
- $wp(S, Q) \vee wp(S, R) \Rightarrow wp(S, Q \vee R)$,
- $wp(S, Q) \wedge wp(S, R) = wp(S, Q \wedge R)$.

1.14. Calculating the WP

- $v : \mathcal{A}$, $p(s)$ is a function and R is a logical relation then
  - $R \circ p(s) = R^{v \leftarrow p(s)(v))}$
- $wp(x_i := y_i, R) = R^{x_i \leftarrow y_i}$
- $wp(x_i := y_i \ if \ \pi, \ R) = (\pi \rightarrow R^{x_i \leftarrow y_i}) \wedge (\neg \pi \rightarrow R)$

2. Calculating the WP

2.1. Exercise 1.

$R = (x = 3 \vee x = 4)$

$s: \ x := x + 2$

$wp(s,R) = R^{x \leftarrow x+2} = (x+2 = 3 \vee x+2 = 4) = (x = 1 \vee x = 2)$

2.2. Exercise 1.(cont.)

$\lceil R \rceil = \{3,4\}$

$s: \ x := x+2$

$\lceil wp(s,R) \rceil = (x+2 \in \{3,4\}) = \{1,2\}$

2.3. Exercise 2.

$R = (2|x)$

$s: \ x := 0$

$wp(s,R) = R^{x \leftarrow 0} = (2|x)^{x \leftarrow 0} = (2|0) = \uparrow$

2.4. Exercise 3.

$R = (2|x+y)$

$s: \ x,y := x+1, x-1$

$wp(s,R) = R^{x \leftarrow x+1, y \leftarrow y-1} = (2|x+y)^{x \leftarrow x+1, y \leftarrow y-1} =$

$(2|x+1+y-1) = (2|x+y)$

2.5. Exercises

- $R = (x = 6), \ s: \ x := 0$
- $R = (x > 2), \ s: \ x := x+2$
- $R = (x > 0), \ s: \ x := -x, \ if \ x > 0$
- $R = (x > y), \ s: \ x := x+y$
- $R = (6|x), \ s: \ x := \begin{cases} x/2, \ if \ 2|x \\ x/3, \ if \ 3|x \end{cases}$
- $R = (0 < x+y < 7), \ s: \ x := \begin{cases} 5, \ if \ y < 2 \\ x, \ if \ y \geq 2 \end{cases}$

Chapter 16. Practice 2

1. Reminder

1.1. Effect Relation

- A *statement* is a subset of the direct product $A \times A^{**}$.
- The *effect relation* of a statement s is denoted by $p(s)$.
- The effect relation expresses the functionality of the statement.
- $p(s) \subseteq A \times A,\ \mathcal{D}_{p(s)} = \{a \in A | p(s)(a) \subseteq A^*\}$.

1.2. Weakest precondition

- The logical function $wp(s, R) : A \longmapsto \mathcal{L}$ is called the weakest precondition of the postcondition R in respect to the statement s.
- $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- $wp(S, R) ::= \forall s \in S : wp(s, R)$.

1.3. WP of the Abstract Parallel Program

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- $wp(S, R) ::= \forall s \in S : wp(s, R)$.
- $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
  - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.

1.4. Properties of WP

- $wp(s, \downarrow) = \downarrow$
- $wp(s, \uparrow) = \uparrow$, if $\mathcal{D}_{p(s)} = \mathcal{A}$
- $P \Rightarrow Q \Longrightarrow wp(s, P) \Rightarrow wp(s, Q)$
- $wp(s, P) \wedge wp(s, Q) \Longleftrightarrow wp(s, P \wedge Q)$
- $wp(s, P) \vee wp(s, Q) \Longrightarrow wp(s, P \vee Q)$
- $\lceil lf(s, P) \rceil = \lceil R \circ p(s) \rceil$
- $wp(SKIP, R) = R$

1.5. Properties of WP

- $wp(S, \downarrow) = \downarrow$,

- $wp(S, \uparrow) = \uparrow$,
- If $P \Rightarrow Q$, then $wp(S, P) \Rightarrow wp(S, Q)$,
- $wp(S, Q) \vee wp(S, R) \Rightarrow wp(S, Q \vee R)$,
- $wp(S, Q) \wedge wp(S, R) = wp(S, Q \wedge R)$.

1.6. Calculating the WP

- $v : \mathcal{A}$, $p(s)$ is a function and R is a logical relation then
  - $R \circ p(s) = R^{v \leftarrow p(s)(v))}$
- $wp(x_i := y_i, R) = R^{x_i \leftarrow y_i}$
- $wp(x_i := y_i \ if \ \pi, \ R) = (\pi \rightarrow R^{x_i \leftarrow y_i}) \wedge (\neg \pi \rightarrow R)$

2. Calculating WP(S, R)

2.1. Exercise 1.

$R = (x = 4)$

$S : (SKIP, \{x := x + 2, x := x * 2\})$

$wp(S, R) = wp(s_1, R) \wedge wp(s_2, R)$

$wp(s_1, R) = R^{x \leftarrow x+2} = (x + 2 = 4) = (x = 2)$

$wp(s_2, R) = R^{x \leftarrow x*2} = (x * 2 = 4) = (x = 2)$

$wp(S, R) = (x = 2) \wedge (x = 2) = (x = 2)$

2.2. Exercise 1.

$R = (x = 0)$

$S : (SKIP, \{x := x + 1, x := x - 1\})$

$wp(S, R) = wp(s_1, R) \wedge wp(s_2, R)$

$wp(s_1, R) = R^{x \leftarrow x+1} = (x + 1 = 0) = (x = -1)$

$wp(s_2, R) = R^{x \leftarrow x-1} = (x - 1 = 0) = (x = 1)$

$wp(S, R) = (x = -1) \wedge (x = 1) = \downarrow$

2.3. Exercises

- $R = (x = y)$, $S : (SKIP, \{x := x + 1, y := y - 1\})$
- $R = (x = 0)$, $S : (SKIP, \{x := 0, \ if \ y \neq 0;$

  $x := x * z, \ if \ z = 0\})$

- $R = (2|x)$, $S : (SKIP, \{x := x + 2,\ if\ x < 50;$

  $x := x + 1,\ if\ x \geq 50\})$

3. Unless Program Property

3.1. Definition

- P is stable while $\neg Q$.
- $\rhd_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
- $\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$

3.2. Properties

- $P \rhd_S P$
- $P \rhd_S \neg P$
- $\uparrow \rhd_S P$
- $\downarrow \rhd_S P$
- $P \rhd_S \uparrow$

3.3. Proof 1.

Theorem 33. $P \rhd_S P$

Proof. $P \rhd_S P$

$P \wedge \neg P \Rightarrow wp(S, P \vee P)$

$\downarrow \Rightarrow \ldots$

3.4. Proof 2.

Theorem 34. $P \rhd_S \neg P$

Proof. $P \rhd_S \neg P$

$P \wedge P \Rightarrow wp(S, P \vee \neg P)$

$\uparrow \Rightarrow wp(S, \uparrow)$

3.5. Stable Properties

- $P \rhd_S \downarrow$ does not always hold: $P \wedge \uparrow \Rightarrow wp(S, P \vee \downarrow)$
- If $P \Rightarrow wp(S, P)$, then P is stable

Counterexample. $P : (x = 1)$

$S : (SKIP, \{x := 5\})$

$$(x = 1) \Rightarrow wp(x := 5, (x = 1))$$

$$(x = 1) \Rightarrow (x = 1)^{x \leftarrow 5}$$

$$(x = 1) \Rightarrow (5 = 1)$$

$$(x = 1) \not\Rightarrow \downarrow$$

4. Calculating Unless

4.1. Exercise 1.

- $S : (SKIP, \{x := x + 1;$

  $x := x - 1,\ if\ 2|x\})$
- $P : (x = 3)$
- $Q : (X = 4)$
- $P \rhd_S Q$?

4.2. Exercise 1. (solution)

- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} wp(s_i, P \vee Q)$
- $\forall s_i \in S : P \wedge \neg Q \Rightarrow wp(s_i, P \vee Q)$
- s_1:
  - $((x = 3) \wedge (x \neq 4)) \Rightarrow ((x = 3) \vee (x = 4))^{x \leftarrow x+1}$
  - $(x = 3) \Rightarrow ((x + 1 = 3) \vee (x + 1 = 4))$
  - $(x = 3) \Rightarrow ((x = 2) \vee (x = 3))$
- s_2:
  - $((x = 3) \wedge (x \neq 4)) \Rightarrow$

    $[(2|x \rightarrow ((x = 3) \vee (x = 4))^{x \leftarrow x-1}) \wedge (\neg(2|x) \rightarrow ((x = 3) \vee (x = 4))^{SKIP})]$

4.3. Exercise 1. (solution)

s_2 :

- $((x = 3) \wedge (x \neq 4)) \Rightarrow$
  $[(2|x \rightarrow ((x = 3) \vee (x = 4))^{x \leftarrow x-1}) \wedge (\neg(2|x) \rightarrow ((x = 3) \vee (x = 4))^{SKIP})]$
- $(x = 3) \Rightarrow [(2 \not| x \vee (x - 1 = 3) \vee (x - 1 = 4)) \wedge (2|x \vee (x = 3) \vee (x = 4))]$
- $(x = 3) \Rightarrow [(2 \not| x \vee (x = 4) \vee (x = 5)) \wedge (2|x \vee (x = 3) \vee (x = 4))]$
- $(x = 3) \Rightarrow [(2 \not| x \vee (x = 4) \vee (x = 5)) \wedge (2|x \vee (x = 3) \vee (x = 4))]$

4.4. Simplified Solution

- $P \rhd_S Q$
- $P \wedge \neg Q \Rightarrow wp(S, P \vee Q)$
- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} wp(s_i, P \vee Q)$
- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} [(\pi \rightarrow (P \vee Q)^{s_i}) \wedge (\neg\pi \rightarrow (P \vee Q)^{SKIP})]$
- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} [(\neg\pi \vee (P \vee Q)^{s_i}) \wedge (\pi \vee (P \vee Q))]$

4.5. Simplified Solution

- $P \Rightarrow \pi \vee P \vee Q$
- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} [(\neg\pi \vee (P \vee Q)^{s_i}) \wedge (\pi \vee P \vee Q)]$
- $\Rightarrow$ SKIP execution paths can be omitted
- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} [(\neg\pi \vee (P \vee Q)^{s_i})]$

4.6. Simplified Solution

- $P \wedge \neg Q \Rightarrow \bigwedge_{i=1}^{n} (\neg\pi \vee (P \vee Q)^{s_i})$
- $\forall s_i \in S : P \wedge \neg Q \Rightarrow (\neg\pi \vee (P \vee Q)^{s_i})$
- Condition reordering $\Rightarrow$
- $\forall s_i \in S : P \wedge \neg Q \wedge \pi_i \Rightarrow (P \vee Q)^{s_i}$

4.7. Exercise 1. (simplified solution)

- $S : (SKIP, \{x := x + 1;$

  $x := x - 1,\ if\ 2|x\})$
- $P : (x = 3)$
- $Q : (X = 4)$
- $P \rhd_S Q$?

4.8. Exercise 1. (simplified solution)

- $\forall s_i \in S : P \wedge \neg Q \wedge \pi_i \Rightarrow (P \vee Q)^{s_i}$
- Omitting SKIP branches and reordering conditions
- s_1:

- $((x = 3) \wedge (x \neq 4)) \Rightarrow ((x = 3) \vee (x = 4))^{x \leftarrow x+1}$
- $(x = 3) \Rightarrow ((x + 1 = 3) \vee (x + 1 = 4))$
- $(x = 3) \Rightarrow ((x = 2) \vee (x = 3))$

- s_2:
  - $((x = 3) \wedge (x \neq 4)) \wedge 2|x \Rightarrow ((x = 3) \vee (x = 4))^{x \leftarrow x-1}$
  - $\downarrow \Rightarrow ((x - 1 = 3) \vee (x - 1 = 4))$

4.9. Exercise 2.

- $S : (SKIP, \{x := x + 2,\ if\ x < 50;$

  $x := x + 1,\ if\ x \geq 50\})$
- $P : (2|x)$
- $Q : (x \geq 50)$
- $P \rhd_S Q$?

Chapter 17. Practice 3

1. Reminder

1.1. Program Properties

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- Weakest Postcondition
  - $wp(S, R) ::= \forall s \in S : wp(s, R)$.
  - $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
    - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- Unless
  - P is stable while $\neg Q$.
  - $\rhd_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
  - $\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$

2. Properties of Unless

2.1. Unless and Stable Property

Theorem 35. If $P \rhd_S Q$ and $K \rhd_S \downarrow$, then $P \wedge K \rhd_S Q \wedge K$.

$$\frac{P \rhd_S Q,\ K \rhd_S \downarrow}{P \wedge K \rhd_S Q \wedge K}$$

Proof. What's needed?

$$\frac{P \wedge \neg Q \Rightarrow wp(S, P \vee Q), K \Rightarrow wp(S, K)}{P \wedge K \wedge \neg(Q \wedge K) \Rightarrow wp(S, (P \wedge K) \vee (Q \wedge K))}$$

$$P \wedge \neg Q \Rightarrow wp(S, P \vee Q), K \Rightarrow wp(S, K)$$

$$P \wedge \neg Q \wedge K \Rightarrow wp(S, P \vee Q) \wedge wp(S, K)$$

$$P \wedge \neg Q \wedge K \Rightarrow wp(S, (P \vee Q) \wedge K) \text{ (wp property)}$$

$$P \wedge \neg Q \wedge K \Rightarrow wp(S, (P \wedge K) \vee (Q \wedge K))$$

$$P \wedge \neg Q \wedge K \equiv P \wedge K \wedge \neg(Q \wedge K) \text{ (lemma)}$$

2.2. Unless and Stable Property

Lemma 4. $P \wedge \neg Q \wedge K \equiv P \wedge K \wedge \neg(Q \wedge K)$

Proof. $P \wedge K \wedge \neg(Q \wedge K) \equiv$

$P \wedge K \wedge (\neg Q \vee \neg K) \equiv$

$(P \wedge \neg Q \wedge K) \vee (P \wedge K \wedge \neg K) \equiv$

$(P \wedge \neg Q \wedge K) \vee \downarrow \equiv$

$P \wedge \neg Q \wedge K$

2.3. Unless Is Disjunctive and Conjunctive

Theorem 36.

$$\frac{P \triangleright_S Q,\ R \triangleright_S Q}{P \vee R \triangleright_S Q}$$

$$\frac{P \triangleright_S Q,\ R \triangleright_S Q}{P \wedge R \triangleright_S Q}$$

2.4. Unless Is NOT Transitive

.

$$\frac{P \triangleright_S Q,\ Q \triangleright_S R}{P \triangleright_S R}$$ does not always hold!

Counterexample. $A = \{1, 2, 3\}$

$P : (x = 1)$

$Q : (x = 2)$

$R : (x = 3)$

$S : (SKIP, \{x := x + 1,\ if x < 3\})$

2.5. Consequence Weakening

Theorem 37.

$$\frac{P \triangleright_S Q,\ Q \Rightarrow R}{P \triangleright_S R}$$

2.6. Condition Narrowing

.

$$\frac{P \Rightarrow Q,\ Q \triangleright_S R}{P \triangleright_S R}$$ does not always hold!

Counterexample. $A = \mathbb{N}^+$

$P : (x = 1)$

$Q : ((x = 1) \vee (x = 2))$

$R : (x \geq 3)$

$S : (SKIP, \{x := x + 1\})$

2.7. Cancellation

Theorem 38.

$$\frac{P \rhd_S Q,\ Q \rhd_S R}{P \vee Q \rhd_S R}$$

3. Exercises

3.1. Exercise 1.

.

$$\frac{P \vee Q \rhd_S Q,\ Q \rhd_S R}{P \rhd_S Q \vee R}$$

3.2. Exercise 2.

.

$$\frac{P \vee Q \rhd_S R}{P \rhd_S Q \vee R}$$

Chapter 18. Practice 4

1. Reminder

1.1. Program Properties

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- Weakest Postcondition
  - $wp(S, R) ::= \forall s \in S : wp(s, R)$.
  - $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
    - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- Unless
  - P is stable while $\neg Q$.
  - $\rhd_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
  - $\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$

2. Ensures

2.1. Ensures Property, Definition

- P is stable while $\neg Q$ in S and there is a conditional assignment s_j which ensures the transition from P to Q.
- $\mapsto_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.

Ensures.

$$\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \rhd_S \wedge$$

$$\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$$

2.2. Properties

- $P \mapsto_S P$
- $\downarrow \mapsto_S P$
- $P \mapsto_S \uparrow$

2.3. Proof 1.

Theorem 39. $P \mapsto_S P$

Proof. $P \rhd_S P$ and $\exists\, j \in J : (P \wedge \neg P \Rightarrow wp(s_j, P))\}$

$P \rhd_S P$ is true (see Lecture 2) and

$\exists\ j \in J : (P \wedge \neg P \Rightarrow wp(s_j, P))$

$P \wedge \neg P = \downarrow$

$\downarrow \Rightarrow \ldots$

2.4. Properties

- $P \mapsto_S \downarrow$ does not always hold

  Counterexample. $P : (x = 1)$

  $S : (SKIP, \{x := 5\})$

  $P \rhd_S \downarrow$ and $\exists\ j \in J : (P \wedge \neg \downarrow \Rightarrow wp(s_j, \downarrow))$

  $(x = 1) \Rightarrow wp(x := 5, (x = 1))$

  $(x = 1) \Rightarrow (x = 1)^{x \leftarrow 5}$

  $(x = 1) \Rightarrow (5 = 1)$

  $(x = 1) \not\Rightarrow \downarrow$

2.5. Properties

- $P \mapsto_S \neg P$ does not always hold

  Counterexample. $A = \{1, 2\}$

  $P : (x = 1)$

  $S : (SKIP, \{x := 1\})$

2.6. Properties

- $\uparrow \mapsto_S P$ does not always hold

  Counterexample. $A = \{0, 1\}$

  $P : (x = 1)$

  $S : (SKIP, \{x := 0\})$

3. Calculating Ensures

3.1. Exercise 1.

- $S : (SKIP, \{x := x + 1\ ;$

  $x := x - 1,\ if\ 2|x\})$

- $P : (x = 3)$

- $Q : (x = 4)$
- $P \mapsto_S Q$?

3.2. Exercise 1. (solution)

- $P \rhd_S Q$ (see Lecture 2)
- $\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))$
- $j := 1$
- s_1:
  - $((x = 3) \wedge (x \neq 4)) \Rightarrow ((x = 4))^{x \leftarrow x+1}$
  - $(x = 3) \Rightarrow ((x + 1 = 4))$
  - $(x = 3) \Rightarrow ((x = 3))$

4. Properties

4.1. Ensures and Stable Property

Theorem 40. If $P \mapsto_S Q$ and $K \rhd_S \downarrow$, then $P \wedge K \mapsto_S Q \wedge K$.

$$\frac{P \mapsto_S Q,\ K \rhd_S \downarrow}{P \wedge K \mapsto_S Q \wedge K}$$

Proof. What's needed?

$$\frac{P \rhd_S Q \ \textit{and}\ \exists\ j \in J:\ P \wedge \neg Q \Rightarrow wp(s_j, Q)),\ K \rhd_S \downarrow}{P \wedge K \rhd_S Q \wedge K \ \textit{and}\ \exists\ i \in J:\ P \wedge K \wedge \neg(Q \wedge K) \Rightarrow wp(s_i, Q \wedge K)}$$

$P \wedge K \rhd_S Q \wedge K$ is true (Unless and Stable property)

$K \rhd_S \downarrow$, therefore $\forall j \in J : K \Rightarrow wp(s_j, K)$

Needed:

$\exists\, i \in J :\ P \wedge K \wedge \neg(Q \wedge K) \Rightarrow wp(s_i, Q \wedge K)$

4.2. Ensures and Stable Property

Proof. $i := j$

$P \wedge \neg Q \Rightarrow wp(s_j, Q)$ and $K \Rightarrow wp(s_j, K)$, then

$P \wedge \neg Q \wedge K \Rightarrow wp(s_j, Q) \wedge wp(s_j, K)$ (wp property)

$P \wedge \neg Q \wedge K \Rightarrow wp(s_j, Q \wedge K)$

$P \wedge K \wedge \neg(Q \wedge K) = P \wedge \neg Q \wedge K$
$\exists\, i \in J :\ P \wedge K \wedge \neg(Q \wedge K) \Rightarrow wp(s_i, Q \wedge K)$, therefore

4.3. Ensures Is NOT Transitive

.

$$\frac{P \mapsto_S Q,\ Q \mapsto_S R}{P \mapsto_S R}$$ does not always hold!

Counterexample. $A = \mathbb{N}$

$P : (x = 1)$

$Q : (x = 2)$

$R : (x = 3)$

$S : (SKIP, \{x := x + 1\})$

4.4. Ensures Is NOT Disjunctive

.

$$\frac{P \mapsto_S R,\ Q \mapsto_S R}{P \vee Q \mapsto_S R}$$ does not always hold!

Counterexample. $A = \{1, 2, 3\}$

$P : (x = 1)$

$Q : (x = 2)$

$R : (x = 3)$

$S : (SKIP, \{x := 3,\ if\ x = 1;$

$x := x + 1,\ if\ x = 2\})$

4.5. Consequence Weakening

Theorem 41.

$$\frac{P \mapsto_S Q,\ Q \Rightarrow R}{P \mapsto_S R}$$

4.6. Corollario

Theorem 42.

$$\frac{P \Rightarrow Q}{P \mapsto_S Q}$$

4.7. Impossibility

Theorem 43.

$$\frac{P \mapsto_S \downarrow}{\neg P}$$

Chapter 19. Practice 5

1. Reminder

1.1. Program Properties

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- Weakest Postcondition
  - $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
    - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- Unless
  - $\rhd_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
  - $\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$
- Ensures
  - $\mapsto_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
  - $\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \rhd_S \wedge$
    $\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$

2. Ensures

2.1. Exercise

- $P = (b = false)$
- $Q = (a = false)$
- $S = (SKIP, \{a := \ a \ and \ b$
  $a := \ true\})$
- $P \mapsto_S Q$?

3. Leads-to

3.1. Leads-to Property, Definition

- $\hookrightarrow_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ is the transitive disjunctive closure of relation $\mapsto_S$.

  $\hookrightarrow_S$ **is the smallest binary relation satisfying the conditions:**

  - $\mapsto_S \subseteq \hookrightarrow_S$.
  - if $(P, Q) \in \hookrightarrow_S$ and $(Q, R) \in \hookrightarrow_S$, then $(P, R) \in \hookrightarrow_S$.

- Let W denote a countable set. If $\forall m : (m \in W :: (P(m), Q) \in \hookrightarrow_S$, then $(\exists m : m \in W :: P(m)), Q) \in \hookrightarrow_S$.

3.2. Exercise

- $A = \{1, 2, 3, 4, 5\}$
- $S = (SKIP, \{x := \ x + 2, \ if \ x < 4;$
  $x := x + 1, \ if \ x = 4\})$

4. Properties

4.1. Basic Properties

- $P \hookrightarrow_S P$
- $\downarrow \hookrightarrow_S P$
- $P \hookrightarrow_S \uparrow$
- $P \hookrightarrow_S \downarrow$ does not always hold
- $P \hookrightarrow_S \neg P$ does not always hold
- $\uparrow \hookrightarrow_S P$ does not always hold

4.2. Implication Property

Theorem 44.

$$\frac{P \Rightarrow Q}{P \hookrightarrow_S Q}$$

4.3. Consequence Weakening

Theorem 45.

$$\frac{P \hookrightarrow_S Q, \ Q \Rightarrow R}{P \hookrightarrow_S R}$$

4.4. Condition Narrowing

.

$$\frac{P \Rightarrow Q, \ Q \hookrightarrow_S R}{P \hookrightarrow_S R}$$

5. Proof Strategy

5.1. Structural Induction

- Induction on the structure of the proof
- Applied when $\hookrightarrow_s$ appears in the premise of the theorem

- Strategy:
  - Base case: prove the theorem for $P \mapsto_S Q$
  - Inductive step 1 (transitivity): prove the theorem for $P \hookrightarrow_S Q$, where $P \hookrightarrow_S R$ and $R \hookrightarrow_S Q$ for a given R
  - Inductive step 2 (disjunction): prove the theorem for $P \hookrightarrow_S Q$, where $P' \hookrightarrow_S Q$ and $P'' \hookrightarrow_S Q$ and $P = P' \vee P''$

5.2. Impossibility

Theorem 46.

$$\frac{P \hookrightarrow_S \downarrow}{\neg P}$$

Proof. Structural induction:

1. Base case:

$P \mapsto_S \downarrow$ (Impossibility of $\mapsto_S$)

$\neg P$

5.3. Impossibility

Proof. Structural induction:

2. Induction on transitivity:

$P \hookrightarrow_S \downarrow$, where $P \hookrightarrow_S R$ and $R \hookrightarrow_S \downarrow$

Inductive hypothesis: the theorem holds for $P \hookrightarrow_S R$ and $R \hookrightarrow_S \downarrow$

$R \hookrightarrow_S \downarrow$ (Inductive hyp.)

$\neg R$

$P \hookrightarrow_S R$

$P \hookrightarrow_S \downarrow$ (Inductive hyp.)

$\neg P$

5.4. Impossibility

Proof. Structural induction:

3. Induction on disjunction:

$P \hookrightarrow_S \downarrow$, where $P' \hookrightarrow_S \downarrow$ and $P'' \hookrightarrow_S \downarrow$

Inductive hypothesis: the theorem holds for $P' \hookrightarrow_S \downarrow$ and $P'' \hookrightarrow_S \downarrow$

$P' \hookrightarrow_S false$ (Inductive hyp.)

$\neg P'$

$P'' \hookrightarrow_S \downarrow$ (Inductive hyp.)

$\neg P''$

$P = P' \wedge P'' = \downarrow$

$\neg P$

Chapter 20. Practice 6

1. Reminder

1.1. Program Properties

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- Weakest Postcondition
  - $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
    - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- Unless
  - $\triangleright_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
  - $\triangleright_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$
- Ensures
  - $\mapsto_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
  - $\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \triangleright_S \wedge$
    $\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$

1.2. Program Properties

- $\hookrightarrow_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ is the transitive disjunctive closure of relation $\mapsto_S$.

  $\hookrightarrow_S$ **is the smallest binary relation satisfying the conditions:**

  - $\mapsto_S \subseteq \hookrightarrow_S$.
  - if $(P, Q) \in \hookrightarrow_S$ and $(Q, R) \in \hookrightarrow_S$, then $(P, R) \in \hookrightarrow_S$.
  - Let W denote a countable set. If $\forall m : (m \in W :: (P(m), Q) \in \hookrightarrow_S$, then $((\exists m : m \in W :: p(m)), Q) \in \hookrightarrow_S$.

1.3. Structural Induction

- Induction on the structure of the proof
- Applied when $\hookrightarrow_s$ appears in the premise of the theorem
- Strategy:
  - Base case: prove the theorem for $P \mapsto_S Q$
  - Inductive step 1 (transitivity): prove the theorem for $P \hookrightarrow_S Q$, where $P \hookrightarrow_S R$ and $R \hookrightarrow_S Q$ for a given R

- Inductive step 2 (disjunction): prove the theorem for $P \hookrightarrow_S Q$, where $P' \hookrightarrow_S Q$ and $P'' \hookrightarrow_S Q$ and $P = P' \vee P''$.

2. Leads-to Properties

2.1. Leads-to and Stable Property

Theorem 47. If $P \hookrightarrow_S Q$ and $K \triangleright_S \downarrow$, then $P \wedge K \hookrightarrow_S Q \wedge K$.

$$\frac{P \hookrightarrow_S Q,\ K \triangleright_S \downarrow}{P \wedge K \hookrightarrow_S Q \wedge K}$$

Proof. Structural induction

1. Base case

2. Induction on transitivity

3. Induction on disjunction

2.2. PSP Theorem

Theorem 48. Progress-Safety-Progress Theorem:

$$\frac{P \hookrightarrow_S Q,\ B \triangleright_S R}{P \wedge B \hookrightarrow_S (Q \wedge B) \vee R}$$

Proof. Structural induction

1. Base case

2. Induction on transitivity

3. Induction on disjunction

3. Exercises

3.1. Exercise 1.

.

$$\frac{C \wedge A \hookrightarrow_S B \wedge C,\ B \hookrightarrow_S A \wedge \neg C,\ C \triangleright_S \downarrow}{\neg (A \wedge C)}$$

3.2. Exercise 2.

.

$$\frac{P_1 \hookrightarrow_S Q_1,\ P_2 \hookrightarrow_S Q_2}{P_1 \vee P_2 \hookrightarrow_S Q_1 \vee Q_2}$$

3.3. Exercise 3.

.

$$\frac{R \hookrightarrow_S Q,\ P \rhd_S R \wedge Q}{P \hookrightarrow_S Q}$$

3.4. Exercise 3.

.

$$\frac{R \hookrightarrow_S Q,\ P \rhd_S R \wedge Q}{P \hookrightarrow_S Q}$$

Counterexample. $A = \{1, 2, 3\}$

$P = (x = 1)$

$Q = (x = 2 \vee x = 3)$

$R = (x = 2)$

$S = (SKIP, \{SKIP\})$

How can we prove that $P \not\hookrightarrow_S Q$?

4. Inevitability

4.1. Inevitability

Inevitability.

$(\{b\}, \lceil P \rceil) \in \rightsquigarrow_S$, if and only if when on all execution paths leading from b and satisfying the axiom of the unconditionally fair scheduling there is a node at a finite unbounded distance from b of which label is an element of the truth set of P, i.e., the program inevitable reaches the truth set of P started from b .

Theorem 49 ($\hookrightarrow_S$ sound and complete). $\hookrightarrow_S = \rightsquigarrow_S$

4.2. Inevitability

- $\hookrightarrow_S = \rightsquigarrow_S$
- Confuting $\hookrightarrow_S$ is the same as confuting $leadsto_S$
- Give an unconditionally fair scheduling starting from b that does not reach the truth set of P

5. Exercises

5.1. Exercise 3. (cont.)

.

$$\frac{R \hookrightarrow_S Q,\ P \rhd_S R \wedge Q}{P \hookrightarrow_S Q}$$

Counterexample. $A = \{1, 2, 3\}$

$P = (x = 1)$

$Q = (x = 2 \vee x = 3)$

$R = (x = 2)$

$S = (SKIP, \{SKIP\})$

$(x = 1) \overset{SKIP}{\rightarrow} (x = 1) \overset{SKIP}{\rightarrow} (x = 1) \overset{SKIP}{\rightarrow} \ldots \not\rightarrow_S (x = 2 \vee x = 3)$

5.2. Exercise 4.

.

$$\frac{P \rhd_S Q,\ Q \rhd_S R,\ P \hookrightarrow_S R}{P \hookrightarrow_S Q}$$

5.3. Exercise 4.

.

$$\frac{P \rhd_S Q,\ Q \rhd_S R,\ P \hookrightarrow_S R}{P \hookrightarrow_S Q}$$

Counterexample. $A = \{1, 2\}$

$P = R = (x = 1)$

$Q = (x = 2)$

$S = (SKIP, \{SKIP\})$

$(x = 1) \overset{SKIP}{\rightarrow} (x = 1) \overset{SKIP}{\rightarrow} (x = 1) \overset{SKIP}{\rightarrow} \ldots \not\rightarrow_S (x = 2)$

5.4. Exercise 5.

.

$$\frac{P \hookrightarrow_S Q \vee B,\ B \hookrightarrow_S R}{P \hookrightarrow_S Q \vee R}$$

5.5. Exercise 6.

.

$$\frac{A \hookrightarrow_S B,\ B \hookrightarrow_S A \vee C}{A \hookrightarrow_S C}$$

5.6. Exercise 6.

.

$$\frac{A \hookrightarrow_S B,\ B \hookrightarrow_S A \vee C}{A \hookrightarrow_S C}$$

Counterexample. $A = \{1, 2\}$

$$A = B = (x = 1)$$

$$C = (x = 2)$$

$$S = (SKIP, \{SKIP\})$$

$$(x = 1) \overset{SKIP}{\rightarrow} (x = 1) \overset{SKIP}{\rightarrow} (x = 1) \overset{SKIP}{\rightarrow} \ldots \not\rightarrow_S (x = 2)$$

Chapter 21. Practice 7

1. Reminder

1.1. Program Properties

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- $wp(S, R) = \bigwedge_{i=1}^{n} wp(s_i, R)$,
  - where $\lceil wp(s, R) \rceil ::= \{a \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R \rceil\}$.
- $\rhd_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q))\}$
- $\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \rhd_S \wedge$
  $\exists\, j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$
- $\hookrightarrow_S$ is the smallest binary relation satisfying the conditions:
  - $\mapsto_S \subseteq \hookrightarrow_S$.
  - if $(P, Q) \in \hookrightarrow_S$ and $(Q, R) \in \hookrightarrow_S$, then $(P, R) \in \hookrightarrow_S$.
  - Let W denote a countable set. If $\forall m : (m \in W :: (P(m), Q) \in \hookrightarrow_S$, then $((\exists m : m \in W :: p(m)), Q) \in \hookrightarrow_S$.

1.2. Program Properties

- Inevitability:
  - $(\{b\}, \lceil P \rceil) \in \rightsquigarrow_S$, if and only if when on all execution paths leading from b and satisfying the axiom of the unconditionally fair scheduling there is a node at a finite unbounded distance from b of which label is an element of the truth set of P, i.e., the program inevitable reaches the truth set of P started from b.

2. Fixed Point Properties

2.1. Fixed Point Properties

- A fixed point is said to be reached in a state of the state space A, if none of the statements changes the state.
- $S = (s_0, \{s_1, \ldots, s_m\})$ and $\forall s_j \in S$ is a simultaneous, non deterministic conditional assignment, i.e. : $s_j : \|_{i \in [1,n]} ((v_i :\in F_{j_i}(v_1, .., v_n)$, if $\pi_{j_i})$
- $\pi_{j_{id}}$ denotes the logical function, which characterizes the set of states over which the relation F_{j_i} is deterministic, i.e., $\pi_{j_{id}}(a) \Leftrightarrow (|F_{j_i}(a)| = 1)$.

2.2. Definitions

Set of fixed point.

$$fixpont_S ::= (\bigwedge_{j\in J, i\in[1..n]} (\neg\pi_{ji} \vee (\pi_{jid} \wedge v_i = F_{ji}(v_1, \ldots, v_n))))$$

Set of fixed point with deterministic assignments.

$$fixpont_S ::= (\bigwedge_{j\in J, i\in[1..n]} (\pi_{ji} \rightarrow (v_i = F_{ji}(a))))$$

Fixed point properties.

Let us denote by FP_S the set $\{\lceil R\rceil | fixpont_S \Rightarrow R\}$.

2.3. Exercise 1.

.

$$A = \mathbb{Z}$$
$$S = (SKIP, \{x := |x|;$$
$$x := x - 1,\ if\ x > 10\})$$

2.4. Exercise 1.

.

$$A = \mathbb{Z}$$
$$S = (SKIP, \{x := |x|;$$
$$x := x - 1,\ if\ x > 10\})$$

.

$$\varphi_s = ((\uparrow \rightarrow x = |x|) \wedge (x > 10 \rightarrow x = x - 1))$$

$$\varphi_s = ((\downarrow \vee x = |x|) \wedge (x \leq 10 \vee x = x - 1)) \quad \varphi_s = ((x = |x|) \wedge (x \leq 10))$$
$$\varphi_s = ((x \geq 0) \wedge (x \leq 10)) \ \varphi_s = (0 \leq x \leq 10)$$

3. Invariant

3.1. Invariant Properties, Definition

- $inv_S(Q)$ is the set of logical functions of which truth are preserved by the elements of S if the program is started from a state satisfying Q .
- $inv_S : \mathcal{P}(A) \longmapsto \mathcal{P}(\mathcal{P}(A))$.
- $inv_S(\lceil Q\rceil) \subseteq \mathcal{P}(A)$.
- $inv_S(\lceil Q\rceil) ::= \{\lceil P\rceil \mid sp(s_0, Q) \Rightarrow P$ and $P \Rightarrow wp(S, P)\ \}$.
- $inv_S(\lceil Q\rceil) ::= \{\lceil P\rceil \mid Q \Rightarrow wp(s_0, P)$ and $P \Rightarrow wp(S, P)\ \}$.

3.2. Exercise 2.

.

$$A = \mathbb{Z}$$

$$S = (x = 0, \{x := x + 1\})$$

$$(x \geq 0) \in inv_S(\uparrow)$$

.

1)

$$\uparrow \Rightarrow wp(x := 0, x \geq 0)$$

$$\uparrow \Rightarrow (0 \geq 0)$$

$$\uparrow \Rightarrow \uparrow$$

2)

$$x \geq 0 \Rightarrow wp(x := x + 1, x \geq 0)$$

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

4. Exercises

4.1. Calculate the Properties of the Program 1.

.

$$A = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$$

$$f : \mathbb{Z} \to \mathbb{Z}$$

$$S = (s_0 : x, y, u, v := 0, 0, f(0), f(0),$$

$$\{s_1 : y, v := x, u, \ if \ v < u;$$

$$s_2 : x, u := y - 1, f(y - 1), \ if \ u < f(y - 1);$$

$$s_3 : x, u := y + 1, f(y + 1), \ if \ u < f(y + 1)\}$$

4.2. Calculate the Properties of the Program 1.

1.

$$(u = f(x)) \in inv_S(\uparrow)$$

2.

$$(v = k) \rhd_S (v > k)$$

3.

$$(v < u) \wedge (v = k) \mapsto_S (v > k)$$

4.

$$\varphi_S \Rightarrow (f(y + 1) \leq v) \wedge (f(y - 1) \leq v)$$

4.3. Calculate the Properties of the Program 2.

.

$$A = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$$

$$n \in \mathbb{N},\ n > 1$$

$$S = (s_0 : r, x, y := 0, 1, n,$$

$$\{s_1 : r, x := r + x, x + 1,\ if\ x \geq y;$$

$$s_2 : r, y := r + y, y - 1,\ if\ x \geq y\}$$

4.4. Calculate the Properties of the Program 2.

1.

$$((x \geq 1) \wedge (y \geq 0)) \in inv_S(\uparrow)$$

2.

$$(r = k) \wedge (x \leq y) \mapsto_S (r > k)$$

3.

$$\varphi_S \Rightarrow (x > y)$$

Chapter 22. Practice 8

1. Reminder

1.1. Program Properties

- $S = (s_0, \{s_1, \ldots, s_n\})$.
- $wp(S, R)$
- $\rhd_S$
- $\mapsto_S$
- $\hookrightarrow_S$
- φ_S
- inv_S
- $TERM_S = \{\lceil Q \rceil | (Q, \varphi_S) \in \hookrightarrow_S\}$.

2. Problem

2.1. Problem

- The problem is defined as a set of specification relations.
- Every specification relation is defined over the powerset of the state space.
- Let $P, Q, R, U : A \longmapsto \mathcal{L}$ be logical functions.
- We define
  - $\rhd, \mapsto, \hookrightarrow \in \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$, and
  - $FP, INIT, inv, TERM \subseteq \mathcal{P}(A)$

2.2. Specification Relations

- $P \rhd Q$ - (P stable unless Q),
- $P \mapsto Q$ - (P ensures Q-t),
- $P \hookrightarrow Q$ - (Q is inevitable from P),
- $Q \hookrightarrow FP$, $Q \in TERM$ - (fixed point is inevitable from Q),
- $FP \Rightarrow R$ - (R holds in any fixed point),
- $invP$ - (P is invariant),
- $Q \in INIT$ (Q initially).

2.3. Problem Definition

- Let A be a state space and let B be a finite or numerable set.
- The relation $F \subseteq B \times H$, where

  $H = (\overset{\times}{_{i\in[1..3]}}\mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \overset{\times}{_{i\in[1..4]}}\mathcal{P}(\mathcal{P}(A)))$ is called a problem defined over the state space A.
- B is called the parameter space of the problem.

Two relations expressing boundary properties and five relations expressing transition properties are associated to every point of set B.

2.4. Notation

- Let $b \in B$ denote an arbitrary element of the domain of the problem.
- Let h denote an element of $F(b)$.
- The components of h are denoted by $\triangleright_h, \mapsto_h, \hookrightarrow_h$ and by $INIT_h, FP_h, inv_h, TERM_h$ respectively.
- If $|F(b)| = 1$ then we use b instead of h in the indices for the sake of simplicity.

2.5. Example: Greatest Common Divisor – GCD

.

$A = \mathbb{Z}^n$

$a_1, \ldots, a_n$

$B = \mathbb{Z}^n$

$a'_1, \ldots, a'_n$

1.

$Q = (\forall i \in [1..n] : a_i = a'_i \wedge a_i > 0)$

$Q \in INIT_h$

2.

$R = (a_1 = gcd(a'_1, \ldots, a'_n))$

$R \in FP_h$

3.

$Q \in TERM_h$

3. Solution

3.1. Solution

Definition.

The abstract parallel program $S \subseteq A \times A^{***}$ is a solution of the problem $F \subseteq B \times (\underset{i\in[1..3]}{\times} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))) \underset{i\in[1..4]}{\times} \mathcal{P}(\mathcal{P}(A)))$,

- if $\forall b \in B : \exists h \in F(b)$, such that
- the program S satisfies all the specification properties given in the inv_h, $\rhd_h$, $\mapsto_h$, $\hookrightarrow_h$, FP_h, $TERM_h$ components

  of h
- assuming that the program starts from a state satisfying all the elements of $INIT_h$.

3.2. Solved by a Program

Definition.

The problem F is said to be solved by the program S with respect to an invariant property K, if $\forall b \in B : \exists h \in F(b)$ such that $K \in inv_S(\overset{\wedge}{_{Q\in INIT_h}} Q)$ and S satisfies all the specification properties given in h with respect to K and the initial conditions $Q \in INIT_h$.

3.3. Solution

.

The program S satisfies the specification property $*$, if and only if there exists an invariant property K such that the program satisfies $*$ with respect to K, i.e., $K \in inv_S(\overset{\wedge}{_{Q\in INIT_h}} Q)$ and

- $(inv_h P)$
- $P \rhd_h Q$
- $P \mapsto_h Q$
- $P \hookrightarrow_h Q$
- $P \hookrightarrow FP_h$
- $FP_h \Rightarrow R$
- $P \wedge K \in inv_S(\overset{\wedge}{_{Q\in INIT_h}} Q)$
- $(P \wedge K, Q \wedge K) \in \rhd_S$
- $(P \wedge K, Q \wedge K) \in \mapsto_S$
- $(P \wedge K, Q \wedge K) \in \hookrightarrow_S$
- $(sp(s_0, P) \wedge K) \in TERM_S$
- $\varphi_S \wedge K \Rightarrow R$

3.4. Refinement of fixed point requirement

Theorem 50. If S satisfies $inv_h P$ and $FP_h \Rightarrow R$, and $P \wedge R \Rightarrow Q$, then

S satisfies $FP_h \Rightarrow Q$.

4. Exercise

4.1. Greatest Common Divisor – GCD

.

$A = \mathbb{Z}^n$

$a_1, \ldots, a_n$

$B = \mathbb{Z}^n$

$a'_1, \ldots, a'_n$

1.

$Q = (\forall i \in [1..n] : a_i = a'_i \wedge a_i > 0)$

$Q \in INIT_h$

2.

$R = (a_1 = gcd(a'_1, \ldots, a'_n))$

$R \in FP_h$

3.

$Q \in TERM_h$

4.2. Refinement of fixed point requirement

.

$A = \mathbb{Z}^n$

$a_1, \ldots, a_n$

$B = \mathbb{Z}^n$

$a'_1, \ldots, a'_n$

1.

$Q = (\forall i \in [1..n] : a_i = a'_i \wedge a_i > 0)$

$Q \in INIT_h$

2.

$R' = (a_1 = a_2 = \ldots = a_n)$

$$R' \in FP_h$$

3.

$$P' = gcd(a'_1, \ldots, a'_n) = gcd(a_1, \ldots, a_n)$$

$$inv_h(P')$$

4.

$$Q \in TERM_h$$

4.3. Solution

.

$$S = (SKIP, \{ \Box_{i,j \in [1..n]} a_i := a_i - a_j,\ if\ a_i > a_j \})$$

4.4. Refinement of fixed point requirement

- If S satisfies $inv_h P'$ and $FP_h \Rightarrow R'$, and
- $P' \wedge R' \Rightarrow R$, then
- S satisfies $FP_h \Rightarrow R$.

4.5. S Solves the Problem

We have to check:

1.

$$\exists K \in inv_S(Q) :\ P' \wedge K \in inv_S(Q)$$

2.

$$\exists K \in inv_S(Q) :\ \varphi_S \wedge K \Rightarrow R'$$

3.

$$P' \wedge R' \Rightarrow R$$

4.

$$\exists K \in inv_S(Q) :\ sp(s_0, Q) \wedge K \hookrightarrow_S \varphi_S$$

4.6. Step 1.

.

$$\exists K \in inv_S(Q) :\ P' \wedge K \in inv_S(Q)$$

$$K :=\uparrow$$

Check: $P' \in inv_S(Q)$

$Q \Rightarrow wp(s_0, P')$ and $P' \Rightarrow wp(S, P')$

4.7. Step 2.

.

$$\exists K \in inv_S(Q): \ \varphi_S \wedge K \Rightarrow R' \ \ K := \forall i \in [1..n] : a_i > 0$$

Check: $K \in inv_S(Q)$ and $\varphi_S \wedge K \Rightarrow R'$

4.8. Step 3.

.

$$P' \wedge R' \Rightarrow R$$

4.9. Step 4.

.

$$\exists K \in inv_S(Q): \ sp(s_0, Q) \wedge K \hookrightarrow_S \varphi_S$$

Use the Theorem of Variant Function

> **Theorem 51.** $P, Q : A \longmapsto \mathcal{L}$ logical functions, $t : A \longmapsto \mathcal{Z}$ is a variant function, for which $P \wedge \neg Q \Rightarrow t > 0$.
>
> If $\forall m \in \mathcal{N} :: (P \wedge \neg Q \wedge t = m) \hookrightarrow_S ((P \wedge t < m) \vee Q)$ S satisfies , then S satisfies $P \hookrightarrow_S Q$ too.

4.10. Step 4.

.

Check:

$$K \wedge \neg\varphi_S \Rightarrow t > 0$$
$$\forall m \in \mathcal{N} :: (K \wedge \neg\varphi_S \wedge t = m) \hookrightarrow_S ((K \wedge t < m) \vee \varphi_S)$$

and

Then:

$$K \hookrightarrow_S \varphi_S$$

$$sp(s_0, Q) \wedge K \hookrightarrow_S \varphi_S$$

Use the variant function: $t := \sum_{i=1}^{n} a_i$

4.11. Sorting

.

$$A = V$$

$$v$$

$$B = V$$

$$v'$$

$V := vector([1..n], \mathbb{Z})$

1.

$Q = (v = v')$

$Q \in INIT_h$

2.

$R = (\forall i \in [1..n-1] : v_i \leq v_{i+1}) \wedge v \in Perm(v')$

$R \in FP_h$

3.

$Q \in TERM_h$

4.12. Refinement of fixed point requirement

.

$A = V$

v

$B = V$

v'

$V := vector([1..n], \mathbb{Z})$

1.

$Q = (v = v')$

$Q \in INIT_h$

2.

$R' = (\forall i \in [1..n-1] : v_i \leq v_{i+1})$

$R \in FP_h$

3.

$P' = v \in Perm(v')$

$inv_h(P')$

4.

$Q \in TERM_h$

4.13. Solution

.

$$S = (SKIP, \{ \underset{i \in [1..n-1]}{\square} v_i, v_{i+1} := v_{i+1}, v_i, \ if \ v_i > v_{i+1} \})$$

Chapter 23. Practice 9

1. Reminder

1.1. Test Scope

- Program Properties
- Checking Program Properties
- Problem
- Solution

2. Test Examples

2.1. Does it hold?

A.

$$\frac{A \mapsto_S B,\ C \rhd_S D, E \Rightarrow \neg\ (B \vee D)}{(A \wedge C) \hookrightarrow_S \neg E}$$

B.

$$\frac{P \mapsto_S C,\ R \in inv_S(C), A \Rightarrow C}{(P \wedge R) \vee A \hookrightarrow_S C}$$

2.2. Check the Properties!

A.

$A =\ V \times\ \mathbb{N}_0$, where $V = vektor([1..n], \mathbb{N}_0)$

$x\ p$

$S = (\ p := 1,\ \{\ \square_{i \in [1..n]}\ x_i, p\ :=\ 1, p * x_i\})$

1.

$(x_k \neq 1) \mapsto_S (x_k = 1),\ k \in \mathbb{N}$

2.

$\varphi_S \Rightarrow ((\forall i \in [1..n] : x_i = 1) \wedge (p > 0))$

2.3. Check the Properties!

B.

$A =\ V \times \mathbb{Z}$, where $V = vektor([1..n], \mathbb{Z})$

$a\ d$

$$S = (\ d := 0,\ \{ \square_{i \in [1..n]} a_i, d\ :=\ 0, d + a_i \})$$

1.

$$(a_k \neq 0) \mapsto_S (a_k = 0)\ ,\ k \in \mathbb{N}$$

2.

$$\varphi_S \Rightarrow ((\forall i \in [1..n] : a_i = 0) \wedge (d > 0))$$

2.4. Does S Satisfy the Properties?

A.

$A = \mathbb{N}_0 \times \mathbb{N}_0$

$a\ b$

$B = \mathbb{N}_0$

a' (1) $(a = a') \in INIT_{a'}$

(2) $(b + \sum_{i=0}^{a} i = \sum_{i=0}^{a'} i) \in inv_{a'}$

(3) $(b = \sum_{i=0}^{a'} i) \in FP_{a'}$

(4) If the program terminate, give a variant function which

can be used to proof that S satisfies the

$(a = a') \in TERM_{a'}$ property.

$S = (b := 0, \{a, b := a - 1, b + a, \text{ if } a > 0\})$

2.5. Does S Satisfy the Properties?

B.

$A = \mathbb{N}_0 \times \mathbb{N}_0$

$x\ y$

$B = \mathbb{N}_0$

x' (1) $(x = x') \in INIT_{x'}$

(2) $(y * \prod_{i=1}^{x} i = \prod_{i=1}^{x'} i) \in inv_{x'}$

(3) $(y = \prod_{i=1}^{x'} i) \in FP_{x'}$

(4) If the program terminate, give a variant function which

can be used to proof that S satisfies the

$(x = x') \in TERM_{x'}$ property.

$$S = (y := 1, \{x, y := x - 1, y * x, \text{if } x > 0\})$$

Chapter 24. Practice 10

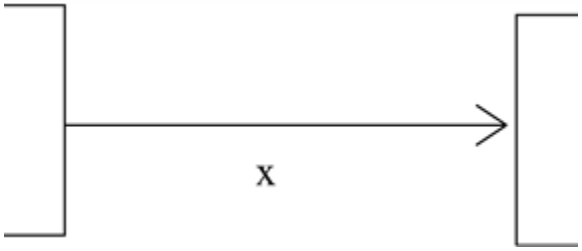
1. Reminder

1.1. Where we are now?

- Problem
- Parallel Program
- Solution

2. Channels

2.1. Channels



- $x : Ch(Int)$ – queue, buffer for one directional communication
- Error-free, unbounded or bounded
- $\overline{x}$ – the history of the channel
- Operations:
  - $x := x; e = hiext(x, e)$ (P1)
  - $x := lorem(x), \text{if } x \neq <>$ (P2)
  - $e := lov(x) = x.lov, \text{if } x \neq <>$
  - $x :=<>$
  - $n := length(x) = |x|$

2.2. Semantics of Operations

- $wp(x := x; e,\ R) = R^{x \leftarrow x;e,\ \overline{x} \leftarrow \overline{x};e}$
- $wp(x := lorem(x),\ \text{if } x \neq <>,\ R) =$

  $(x \neq <> \rightarrow R^{x \leftarrow lorem(x)}) \wedge (x =<> \rightarrow R)$.
- $wp(x :=<>,\ R) = R^{x \leftarrow <>, \overline{x} \leftarrow <>}$.

3. FORK

3.1. FORK

Requirements:

- Data must not be lost.
- New data must not be produced.
- The scheduling must be fair.
- FORK must do something ($SKIP$ is not a good solution).

3.2. The function “split”

A helper function:

- $split : Ch \times Ch \times Ch \longrightarrow L$
- $split(<>, <>, <>) = true$
- $split(a, b, c) \longrightarrow split(a; x, b; x, c) \wedge split(a; x, b, c; x)$
- Take the smallest from these functions.

3.3. Specification

$$\begin{array}{llllllll} A = & Ch\times & Ch\times & Ch\times & Ch\times & Ch\times & Ch \\ & x & \overline{x} & y & \overline{y} & z & \overline{z} \\ B = & Ch\times & Ch\times & Ch\times & Ch\times & Ch\times & Ch \\ & x' & \overline{x}' & y' & \overline{y}' & z' & \overline{z}' \end{array}$$

$$Q = (x = y = z = \overline{x} = \overline{y} = \overline{z} = x' = y' = z' = \overline{x}' = \overline{y}' = \overline{z}' =<>)$$

$$Q \in INIT_h \tag{24.1}$$

$$P = (split(\overline{x} - x, \overline{y}, \overline{z})) \in inv_h \tag{24.2}$$

$$\forall k \in N : |\overline{x}| \geq k \hookrightarrow_S |\overline{y}| + |\overline{z}| \geq k \tag{24.3}$$

3.4. Solution

$$
\begin{array}{l}
S: \\
\quad (\quad s_0: \quad SKIP, \\
\qquad \{ \qquad s_1 : x, y := lorem(x), hiext(y, x.lov), \text{ if } x \neq <> \\
\qquad \boxdot \\
\qquad\qquad\quad s_2 : x, z := lorem(x), hiext(z, x.lov), \text{ if } x \neq <> \\
\qquad \} \\
\quad)
\end{array}
$$

3.5. The Program Solves the Problem

Proof. (2): $\exists K \in inv_S(Q) : P \wedge K \in inv_S(Q)$

- $K :=\uparrow$
- $Q \Rightarrow wp(s_0, P)$

  $split(<> - <>, <>, <>) = split(<>, <>, <>) = true$
- $P \Rightarrow wp(S, P) = wp(s_1, P) \wedge wp(s_2, P)$
- Lets see: $P \Rightarrow wp(s_1, P)$ ($P \Rightarrow wp(s_2, P)$ is similar)
- $P \Rightarrow wp(s_1, P) = (x \neq <> \rightarrow P^{y \leftarrow hiext(y, x.lov), \overline{y} \leftarrow \overline{y}; x.lov, x \leftarrow lorem(x)}) \wedge (x = <> \rightarrow P) = (x \neq <> \rightarrow split(\overline{x} - lorem(x), \overline{y}; x.lov, \overline{z})) \wedge (x = <> \rightarrow P)$

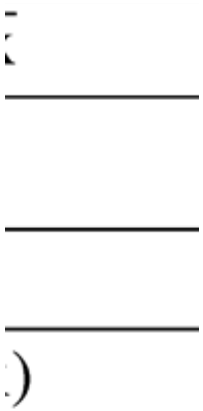
  $P \wedge x \neq <> \Rightarrow split(\overline{x} - lorem(x), \overline{y}; x.lov, \overline{z})$

3.6. The Program Solves the Problem

Proof. (2):

- We have to proof that:

  $$\overline{x} - lorem(x) = (\overline{x} - x); x.lov$$
- Lets see the following figure:



3.7. The Program Solves the Problem

Proof. (2):

- $P \wedge x \neq <> \Rightarrow split((\overline{x} - x); x.lov, \overline{y}; x.lov, \overline{z})$
- $P = (split(\overline{x} - x, \overline{y}, \overline{z})) \wedge x \neq <> \Rightarrow split((\overline{x} - x); x.lov, \overline{y}; x.lov, \overline{z})$
- (2) holds based on the definition of the function $split$.

3.8. The Program Solves the Problem

Proof. (3): $\exists K \in inv_S(Q) : K \wedge |\overline{x}| \geq k \hookrightarrow_S K \wedge (|\overline{y}| + |\overline{z}|) \geq k$, $k \in \mathbb{N}$

- $K := split(\overline{x} - x, \overline{y}, \overline{z})$
- (*) There are two cases:
  - a.) $|\overline{x}| \geq k$ and $|\overline{y}| + |\overline{z}| \geq k$
  - b.) $|\overline{x}| \geq k$ and $|\overline{y}| + |\overline{z}| < k$
- In case of a): we are ready
- In case of b): we can assume that $x \neq <>$ (based on K)

3.9. The Program Solves the Problem

Proof. (3) b):

- We have to proof that: $|\overline{y}| + |\overline{z}| = l \wedge x \neq <> \mapsto_S |\overline{y}| + |\overline{z}| = l + 1$
- Then go back to step (*)
- That results:

$$\begin{array}{lll} |\overline{y}| + |\overline{z}| = l & & \\ |\overline{y}| + |\overline{z}| = l + N & \mapsto_S \quad |\overline{y}| + |\overline{z}| = l + 1 \quad \ldots \quad \mapsto_S & \\ \geq k & & \end{array}$$

- we can use $\hookrightarrow_S$ instead of $\mapsto_S$
- $\hookrightarrow_S$ is transitive:

$$|\overline{x}| \geq k \hookrightarrow_S |\overline{y}| + |\overline{z}| \geq k$$

3.10. The Program Solves the Problem

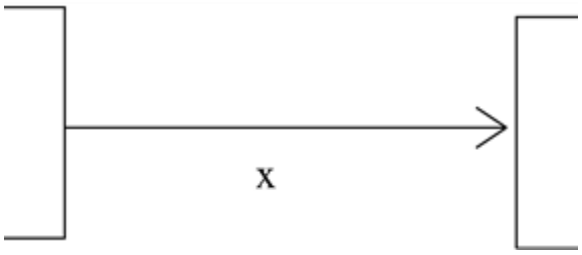
Proof. (3): $\exists K \in inv_S(Q) : K \wedge |\overline{x}| \geq k \hookrightarrow_S K \wedge (|\overline{y}| + |\overline{z}|) \geq k$, $k \in \mathbb{N}$

- $K := split(\overline{x} - x, \overline{y}, \overline{z})$
- we can use the variant function theorem to proof (3)
- $t = k - (|\overline{y}| + |\overline{z}|)$

Chapter 25. Practice 11

1. Reminder

1.1. Channels



- $x : Ch(Int)$ – queue, buffer for one directional communication
- Error-free, unbounded or bounded
- $\overline{x}$ – the history of the channel
- Operations:
  - $x := x; e = hiext(x, e)$ (P1)
  - $x := lorem(x), \text{if } x \neq <>$ (P2)
  - $e := lov(x) = x.lov, \text{if } x \neq <>$
  - $x :=<>$
  - $n := length(x) = |x|$

1.2. The function “split”

- $split : Ch \times Ch \times Ch \longrightarrow L$
- $split(<>, <>, <>) = true$
- $split(a, b, c) \longrightarrow split(a; x, b; x, c) \wedge split(a; x, b, c; x)$
- Take the smallest from these functions.

2. Multiplexer

2.1. MUX

MUX

Requirements:

- Data must not be lost.
- New data must not be produced.
- The scheduling must be fair.
- MUX must do something ($SKIP$ is not a good solution).

2.2. Specification

$$\begin{array}{lllllll} A = & Ch\times & Ch\times & Ch\times & Ch\times & Ch\times & Ch \\ & x & \overline{x} & y & \overline{y} & z & \overline{z} \\ B = & Ch\times & Ch\times & Ch\times & Ch\times & Ch\times & Ch \\ & x' & \overline{x}' & y' & \overline{y}' & z' & \overline{z}' \end{array}$$

$$Q = (x = y = z = \overline{x} = \overline{y} = \overline{z} = x' = y' = z' = \overline{x}' = \overline{y}' = \overline{z}' =<>)$$

$$Q \in INIT_h \tag{25.1}$$

$$P = (split(\overline{z}, \overline{x} - x, \overline{y} - y)) \in inv_h \tag{25.2}$$

$$\forall k \in N : |\overline{x}| \geq k \hookrightarrow_S |\overline{x} - x| \geq k$$

$$\forall l \in N : |\overline{y}| \geq l \hookrightarrow_S |\overline{y} - y| \geq l \tag{25.3}$$

2.3. Solution

$$\begin{array}{l} S: \\ \quad (\quad s_0 : \quad SKIP, \\ \qquad \{ \qquad s_1 : x, z := lorem(x), hiext(z, x.lov), \text{ if } x \neq <> \\ \qquad \boxdot \\ \qquad\qquad\quad s_2 : y, z := lorem(y), hiext(z, y.lov), \text{ if } y \neq <> \\ \qquad \} \\ \quad) \end{array}$$

2.4. The Program Solves the Problem

Proof. (2): $\exists K \in inv_S(Q) : P \wedge K \in inv_S(Q)$

- $K :=\uparrow$
- $Q \Rightarrow lf(s_0, P)$

- $split(<>, <> - <>, <> - <>) = split(<>, <>, <>) = true$
- $P \Rightarrow wp(S, P) = wp(s_1, P) \wedge wp(s_2, P)$
- Lets see: $P \Rightarrow wp(s_1, P)$ ($P \Rightarrow wp(s_2, P)$ is similar)
- $P \Rightarrow lf(s_1, P) = (x \neq <> \rightarrow P^{x:=lorem(x), z:=z.x.lov, \overline{x}:=\overline{z}.x.lov}) \wedge (x = <> \rightarrow P) = (x \neq <> \rightarrow split(\overline{z}; x.lov, \overline{x} - lorem(x), \overline{y} - y)) \wedge (x = <> \rightarrow P)$

  $P \wedge x \neq <> \Rightarrow split(\overline{z}; x.lov, \overline{x} - lorem(x), \overline{y} - y)$

2.5. The Program Solves the Problem

Proof. (2):

- We can use the lemma from the previous lecture:

$$\overline{x} - lorem(x) = (\overline{x} - x); x.lov$$

- $P \wedge x \neq <> \Rightarrow split(\overline{z}; x.lov, (\overline{x} - x); x.lov, \overline{y} - y)$
- $split(\overline{z}, \overline{x} - x, \overline{y} - y) \wedge x \neq <> \Rightarrow split(\overline{z}; x.lov, (\overline{x} - x); x.lov, \overline{y} - y)$
- (2) holds based on the definition of the function $split$.

2.6. The Program Solves the Problem

Proof. (3): $\exists K \in inv_S(Q) : K \wedge |\overline{x}| \geq k \hookrightarrow_S K \wedge |\overline{x} - x| \geq k$, $k \in \mathbb{N}$ and $K \wedge |\overline{y}| \geq l \hookrightarrow_S K \wedge |\overline{y} - y| \geq l$, $l \in \mathbb{N}$

- $K :=\uparrow$
- (*) There are two cases:
  - a.) $|\overline{x}| \geq k$ and $|\overline{x} - x| \geq k$
  - b.) $|\overline{x}| \geq k$ and $|\overline{x} - x| < k$
- In case of a): we are ready
- In case of b): we can assume that $x \neq <>$

2.7. The Program Solves the Problem

Proof. (3) b):

- We have to proof that: $|\overline{x} - x| = l \wedge x \neq <> \mapsto_S |\overline{x} - x| = l + 1$
- Then go back to step (*)
- That results:

$$|\overline{x} - x| = l \mapsto_S |\overline{x} - x| = l + 1 \ ... \mapsto_S |\overline{x} - x| = l + N$$

$$\geq k$$

- we can use $\hookrightarrow_S$ instead of $\mapsto_S$
- $\hookrightarrow_S$ is transitive:

$$|\overline{x}| \geq k \hookrightarrow_S |\overline{x} - x| \geq k$$

2.8. The Program Solves the Problem

Proof. (3) $K \wedge |\overline{y}| \geq l \hookrightarrow_S K \wedge |\overline{y} - y| \geq l$, $l \in \mathbb{N}$ is similar

3. Exercise

3.1. Specification

$$\begin{array}{lllllll} A = & Ch\times & Ch\times & Ch\times & Ch\times & Ch\times & Ch \\ & x & \overline{x} & y & \overline{y} & z & \overline{z} \\ B = & Ch\times & Ch\times & Ch\times & Ch\times & & \\ & x' & \overline{x}' & y' & \overline{y}' & & \end{array}$$

$$Q = (x = x' = \overline{x} = \overline{x}' \wedge y = y' = \overline{y} = \overline{y}') \in INIT_h \tag{25.4}$$

$$P = (|\overline{x} - x| = |\overline{y} - y|) \in inv_h \tag{25.5}$$

3.2. Solution

$$\begin{array}{l} S: \\ \quad (\quad s_0: \quad z := <>, \\ \quad\quad \{ \quad\quad s_1 : x, y, z := lorem(x), lorem(y), hiext(z, 2 * x.lov), \\ \quad\quad\quad\quad if\ x \neq <> \wedge y \neq <> \\ \quad\quad \boxdot \\ \quad\quad\quad\quad s_2 : y, z := lorem(x), lorem(y), hiext(z, 2 * y.lov), \\ \quad\quad\quad\quad if\ x \neq <> \wedge y \neq <> \\ \quad\quad \} \\ \quad) \end{array}$$

Does this program solve the specified problem?

3.3. Check the properties of the program!

$$A = Ch(\mathbb{Z})\times\ Ch(\mathbb{Z})\times\ Ch(\mathbb{Z})\times Ch(\mathbb{Z})\times\ Ch(\mathbb{Z})\times Ch(\mathbb{Z})\times \mathbb{Z}$$

$$x\,\overline{x}\ y\,\overline{y}\ z\,\overline{z}\ s$$

$$s_0: \quad z,\ s := <>,\ 0$$

$$S: \{\ x, y,\ z,\ s\ :=$$

$$lorem(x),\ lorem(y),\ hiext(z, x.lov + y.lov),\ s - x.lov,$$

$$if\ x \neq <>,\ y \neq <>,\ x.lov \leq 0$$

$$\boxdot$$

$$x,\ y,\ z,\ s\ :=$$

$$lorem(x),\ lorem(y),\ hiext(z, x.lov * y.lov),\ s + x.lov,$$

$$if\ x \neq <>,\ y \neq <>,\ x.lov > 0\}$$

3.4. Check the properties of the program!

1.

$$\varphi_S \Rightarrow (x = <> \vee y = <>)$$

2.

$$(s = (\sum_{i=1}^{|\overline{x}-x|} |(\overline{x} - x)_i|)) \in inv_S(x = \overline{x})$$

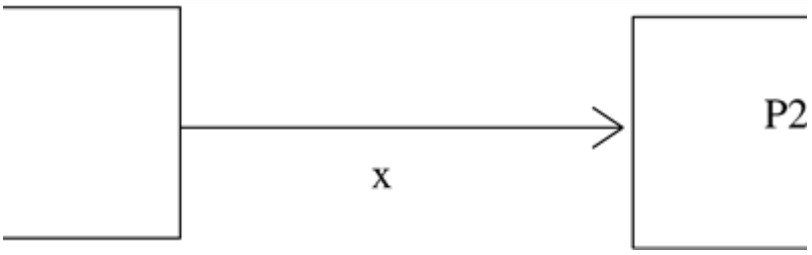
3.

$$\forall k \in \mathcal{N} : \ (|\overline{z}| = k, \ x \neq <>, \ y \neq <> \hookrightarrow_S |\overline{z}| > k)$$

Chapter 26. Practice 12

1. Reminder

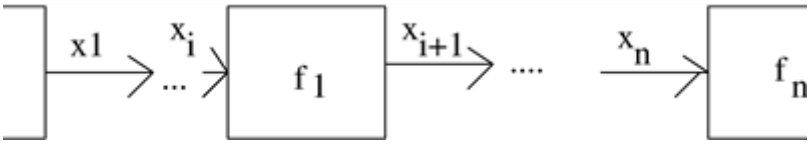
1.1. Channels



- $x : Ch(Int)$ – queue, buffer for one directional communication
- Error-free, unbounded or bounded
- $\overline{x}$ – the history of the channel
- Special problems: FORK, MUX

2. Pipeline

2.1. Pipeline



- $F = f_n \circ ... \circ f_0$.
- $D =< d_1, .., d_m >$.
- $m >> n$

2.2. Specification of Pipeline

A	$=$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	
		x_0		$\overline{x_0}$		x_{n+1}		$\overline{x_{n+1}}$	
B	$=$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	$\times$	$Ch(a)$	
		x'_0		$\overline{x_0}'$		x'_{n+1}		$\overline{x_{n+1}}'$	

$$Q ::= (x_0 = \overline{x_0} = x_0' = \overline{x_0}' = D \wedge$$
$$\wedge x_{n+1} = \overline{x_{n+1}} = x_{n+1}' = \overline{x_{n+1}}' = <>)$$
$$Q \in INIT_{\underbrace{x_0' \overline{x_0}' x_{n+1}' \overline{x_{n+1}}'}_{h}} \quad (26.1)$$
$$FP_h \Rightarrow \overline{x_{n+1}} = F(\overline{x_0}') = F(D) \quad (26.2)$$
$$Q \in TERM_h \quad (26.3)$$
$$(\overline{x_0} = \overline{x_0}' = D) \in inv_h \text{ for the whole system} \quad (26.4)$$

2.3. Refinement of the Problem

$$FP_h \Rightarrow \forall i \in [0..n] : x_i = <> \quad (26.5)$$
$$\forall i \in [0..n] : (f_i(\overline{x_i} - x_i) = \overline{x_{i+1}}) \in inv_h \quad (26.6)$$
$$\text{Variant function: } (|x_0|, ..., |x_n|) \quad (26.7)$$

2.4. Solution

$$S: \quad (\|_{i=1}^{n} x_i := <>,$$
$$\{ \quad \square_{i=0}^{N} x_i, x_{i+1} := lorem(x_i), hiext(x_{i+1}, f_i(x_i.lov)),$$
$$\text{if } x_i \neq <> \})$$

3. Exercise

3.1. Reduction to Pipeline Theorem

- Given the Pipeline Theorem and a similar problem to solve
- The specification of the problem corresponds to the specification of pipeline
- Use the solution of pipeline (S) and transform it according to the correspondence (S')
- If S solves pipeline, than S' solves the similar problem

3.2. Example: Approximation of Square Root

- Given m numbers: $< a_1, \ldots, a_m >= D$
- Calculate the square root of the numbers: $< \sqrt{a_1}, \ldots, \sqrt{a_m} >= F(D)$
- Use the following iteration:
  - $x_0 := \frac{a}{2}$
  - $x_{i+1} := \frac{1}{2} * (\frac{a}{x_i} + x_i)$

3.3. Specification of the Problem

A	$=$	$Ch(\mathbb{N})$	$\times$	$Ch(\mathbb{N})$	$\times$	$Ch(\mathbb{N})$	$\times$	$Ch(\mathbb{N})$	
		x_0		$\overline{x_0}$		x_{n+1}		$\overline{x_{n+1}}$	

B	$=$	$Ch(\mathbb{N})$	$\times$	$Ch(\mathbb{N})$	$\times$	$Ch(\mathbb{N})$	$\times$	$Ch(\mathbb{N})$	
		x_0'		$\overline{x_0}'$		x'_{n+1}		$\overline{x_{n+1}}'$	

$$Q ::= (x_0 = \overline{x_0} = x_0' = \overline{x_0}' = D \wedge$$
$$\wedge x_{n+1} = \overline{x_{n+1}} = x'_{n+1} = \overline{x_{n+1}}' = <>)$$
$$Q \in INIT_{\underbrace{x_0'\overline{x_0}'x'_{n+1}\overline{x_{n+1}}'}_{h}} \quad (26.8)$$
$$FP_h \Rightarrow \overline{x_{n+1}} = F(\overline{x_0}') = F(D) \quad (26.9)$$
$$Q \in TERM_h \quad (26.10)$$
$$(\overline{x_0} = \overline{x_0}' = D) \in inv_h \quad (26.11)$$

3.4. Refinement of the Problem

- $f_0(a) = (\frac{a}{2}, a)$
- $f_i(prev, a) = (\frac{1}{2} * (\frac{a}{prev} + prev), a)$, $i \in [1..n]$
- $f_{n+1}(prev, a) = prev$
- $F(D) =< \sqrt{a_1}, \ldots, \sqrt{a_m} >= f_{n+1} \circ f_n \circ \ldots \circ f_0(D)$

3.5. Refinement of the Problem

$$FP_h \Rightarrow \forall i \in [0..n] : x_i =<> \quad (26.12)$$
$$\forall i \in [0..n] : (f_i(\overline{x_i} - x_i) = \overline{x_{i+1}}) \in inv_h \quad (26.13)$$

3.6. Solution

$S:$ $(\|_{i=1}^{n} x_i :=<>,$

$\{$ $x_0, x_1 := lorem(x_0), hiext(x_1, (\frac{x_0.lov}{2}, x_0.lov))$, if $x_0 \neq <>$;

$\square_{i=1}^{n} x_i, x_{i+1} := lorem(x_i), hiext(x_{i+1}, (\frac{1}{2} * (\frac{x_i.lov_2}{x_i.lov_1} + x_i.lov_1))),$
$x_i.lov_2))$, if $x_i \neq <>$;

$\square x_n, x_{n+1} := lorem(x_n), hiext(x_{n+1}, x_n.lov_1)$, if $x_n \neq <>\})$

3.7. Exercise 1.

- Given thousands of e-mails: $< e_1, \ldots, e_m >$, $e_i \in E$ and
- ten different spam filters: $spam_i : E \to [0..100]$.
- Calculate the average of the spam filters for every e-mails: $< \frac{\sum_{i=1}^{10} spam_i(x_1)}{10}, \ldots, \frac{\sum_{i=1}^{10} spam_i(x_m)}{10} >$!

3.8. Exercise 2.

- Given m values: $< x_1, \ldots, x_m >$
- Calculate the "cosine" of every value: $< cos(x_1), \ldots, cos(x_m) >$
- Use the following rule: $cos(x) = \sum_{k=0}^{\infty} \frac{(-1)^k * x^{2k}}{(2k)!}$

Chapter 27. Practice 13

1. Reminder

1.1. Reminder

- Program Properties: $\triangleright_S, \mapsto_S, \hookrightarrow_S, inv_S, \varphi_S, FP_S, TERM_S$
- Program Construction

2. Union

2.1. Union

Definition.

- Let A_1 and A_2 be two subspaces of the state space A .
- Let B denote the largest common subspace of A_1 and A_2.
- Let $S_1 = (s_{1,0}, \{s_{1,1}, \ldots, s_{1,k}\})$ and $S_2 = (s_{2,0}, \{s_{2,1}, \ldots, s_{2,m}\})$ be the extensions to A of two programs on A_1 and A_2 respectively.
- If all v_i variables belonging to B get the same value in the assignments $s_{1,0}$ and $s_{2,0}$ (i.e. $F_{1,0_i} = F_{2,0_i}$), then the program

$S_1 \cup S_2 = (s_{1,0}||s_{2,0}, \{s_{1,1}, \ldots, s_{1,k}, s_{2,1}, \ldots, s_{2,m}\})$ that is defined on A , is called the union of S_1 and S_2.

2.2. Behaviour Relation of Union

Theorem 52. Let $S ::= (S_1 \cup S_2)$. Then:

1. $$\triangleright_S = \triangleright_{S_1} \cap \triangleright_{S_2}$$
2. $$\mapsto_S = \triangleright_{S_1} \cap \triangleright_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2})$$
3. $$(\triangleright_{S_1} \cap \triangleright_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2}))^{tdl} = \hookrightarrow_S$$
4. $\forall Q$ for which $sp(s_{1,0}||s_{2,0}, Q) \Rightarrow sp(s_{1,0}, Q) \wedge sp(s_{2,0}, Q)$: $inv_{S_1}(Q) \cap inv_{S_2}(Q) \subseteq inv_S(Q)$
5. $$fixpoint_S = fixpoint_{S_1} \wedge fixpoint_{S_2}$$

6.

$$FP_{S_1} \cap FP_{S_2} \subseteq FP_S$$

7.

$$((\rhd_{S_1} \cap \rhd_{S_2} \cap (\mapsto_{S_1} \cup \mapsto_{S_2}))^{tdl})^{(-1)}(fixpoint_{S_1} \wedge fixpoint_{S_2}) = TERM_S.$$

2.3. Properties Based on the Definition

.

$$\frac{P \rhd_{S_1} Q,\ P \rhd_{S_2} Q}{P \rhd_{S_1 \bigcup S_2} Q}$$

.

$$\frac{P \rhd_{S_1 \bigcup S_2} Q}{P \rhd_{S_1} Q,\ P \rhd_{S_2} Q}$$

.

$$\frac{P \rhd_{S_1} Q,\ P \mapsto_{S_2} Q}{P \mapsto_{S_1 \bigcup S_2} Q}$$

.

$$\frac{P \mapsto_{S_1} Q,\ P \rhd_{S_2} Q}{P \mapsto_{S_1 \bigcup S_2} Q}$$

2.4. Counterexample of $\hookrightarrow_S$

.

$A = \mathbb{N}$

$P:\ (x = 0)$

$R:\ (x < -5 \vee x > -5)$

$S_1:\ (x = 0, \{x := x + 1\})$

$S_2:\ (x = 0, \{x := x - 1\})$

$$\frac{P \hookrightarrow_{S_1} R,\ P \hookrightarrow_{S_2} R}{P \not\hookrightarrow_{S_1 \bigcup S_2} R}$$

2.5. Counterexample of $\hookrightarrow_S$

.

$S = S_1 \bigcup S_2:\ (x = 0, \{s_1 : x := x + 1;$

$s_2 : x := x - 1\})$

$(x = 0) \not\hookrightarrow_S (x < -5 \vee x > -5)$

$(x = 0) \xrightarrow{s_1} (x = 1) \xrightarrow{s_2} (x = 0) \xrightarrow{s_1} (x = 1) \xrightarrow{s_2} (x = 0) \xrightarrow{s_1} \dots$

$\not\models_S (x < 5 \vee x > 5)$

3. Exercises

3.1. Check the property! (1)

.

$$\frac{P \mapsto_{S_1} \neg P}{\forall S_2 : P \mapsto_{S_1 \bigcup S_2} \neg P}$$

3.2. Check the property!(1)

.

$$\frac{P \mapsto_{S_1} \neg P}{\forall S_2 : P \mapsto_{S_1 \bigcup S_2} \neg P}$$

Proof. $P \mapsto_S \neg P$ holds for every program, so it holds for S_2:

$$\frac{P \mapsto_{S_1} \neg P, P \mapsto_{S_2} \neg P}{P \mapsto_{S_1 \bigcup S_2} \neg P}$$

3.3. Check the property! (2)

.

$$\frac{P \mapsto_{S_1} Q, Q \Rightarrow R, P \rhd_{S_2} Q}{P \vee Q \hookrightarrow_{S_1 \bigcup S_2} R}$$

3.4. Check the property! (2)

Proof.

$$\frac{P \mapsto_{S_1} Q, P \rhd_{S_2} Q}{P \mapsto_{S_1 \bigcup S_2} Q}$$

$$\frac{P \mapsto_{S_1 \bigcup S_2} Q}{P \hookrightarrow_{S_1 \bigcup S_2} Q}$$

$$\frac{P \hookrightarrow_{S_1 \bigcup S_2} Q, Q \Rightarrow R}{P \hookrightarrow_{S_1 \bigcup S_2} R}$$

$$\frac{Q \Rightarrow R}{Q \hookrightarrow_{S_1 \bigcup S_2} R}$$

$$\frac{P \hookrightarrow_{S_1 \bigcup S_2} R, Q \hookrightarrow_{S_1 \bigcup S_2} R}{P \vee Q \hookrightarrow_{S_1 \bigcup S_2} R}$$

3.5. Check the property! (3)

.

$$\frac{P \mapsto_{S_1} Q, R \rhd_{S_1 \bigcup S_2} \downarrow, P \rhd_{S_2} Q \wedge R}{P \wedge R \mapsto_{S_1 \bigcup S_2} Q \wedge R}$$

3.6. Check the property! (3)

Proof.

$$\frac{P \rhd_{S_2} Q \wedge R, Q \wedge R \Rightarrow Q}{P \rhd_{S_2} Q}$$

$$\frac{P \rhd_{S_2} Q, P \mapsto_{S_1} Q}{P \mapsto_{S_1 \bigcup S_2} Q}$$

$$\frac{P \mapsto_{S_1 \bigcup S_2} Q, R \rhd_{S_1 \bigcup S_2} \downarrow}{P \wedge R \mapsto_{S_1 \bigcup S_2} Q \wedge R}$$

3.7. Check the property! (4)

.

$$\frac{P \hookrightarrow_{S_1} Q, Q \mapsto_{S_1} R, R \in inv_{S_2}(Q)}{P \hookrightarrow_{S_1 \bigcup S_2} R}$$

3.8. Check the property! (4)

Counterexample. $A = \mathbb{N}$

$P: \ (x = 1)$

$Q: \ (x = 2)$

$R: \ (x > 2)$

$S_1: \ (x = 3, \{x := x + 1\})$

$S_2: \ (x = 3, \{x := 1, \ if \ x \leq 2\})$

3.9. Check the property! (5)

.

$$\frac{A \mapsto_{S_2} B, B \mapsto_{S_1} C, A \vee B \rhd_{S_2} C}{A \vee B \hookrightarrow_{S_1 \bigcup S_2} C}$$

3.10. Check the property! (5)

Counterexample. $A = \{1, 2, 3\}$

$A: \ (x = 1)$

$B: \ (x = 2)$

$C: \ (x = 3)$

$S_1: \ (SKIP, \{x := x + 1, \ if \ x \neq 3\})$

$$S_2 : \ (SKIP, \{x := 1\})$$

3.11. Check the property! (6)

.

$$\frac{P \mapsto_{S_1} R, R \rhd_{S_1} Q, Q \mapsto_{S_2} R}{P \vee Q \hookrightarrow_{S_1 \bigcup S_2} R}$$

3.12. Check the property! (7)

.

$$\frac{P \hookrightarrow_{S_1} Q, P \mapsto_{S_2} Q, P \in inv_{S_1} Q}{P \hookrightarrow_{S_1 \bigcup S_2} Q}$$

Chapter 28. Practice 14

1. Reminder

1.1. Test Scope

- Program Properties
- Program Constructions, Union
- Channels
  - Checking Program Properties
  - Solution
- Reduction to Pipeline Theorem

2. Test Examples

2.1. Does it hold?

A.

$$\frac{Q \rhd_{S_1} (Q \wedge R),\ P \rhd_{S_1} \downarrow,\ (P \wedge Q) \mapsto_{S_2} (Q \wedge R)}{(P \wedge Q) \hookrightarrow_{S_1 \bigcup S_2} R}$$

B.

$$\frac{Q \in inv_{S_1}(R),\ Q \mapsto_{S_2} R,\ P \rhd_{S_1 \bigcup S_2} \downarrow}{(Q \wedge P) \hookrightarrow_{S_1 \bigcup S_2} (R \wedge P)}$$

2.2. Check the Properties!

A.

$spl : Ch(T) \times Ch(T) \times Ch(T) \to L$ is a function defined by the following rules::

- $spl(<>, <>, <>) = true$
- $spl(x, y, z) \Rightarrow spl(x; a, y; b, z; f(a, b))$, where $(f : T \times T \to T)$
- spl has the smallest truth set from these functions

$$A = Ch(\mathbb{Z}) \times Ch(\mathbb{Z}) \times Ch(\mathbb{Z}) \times Ch(\mathbb{Z}) \times Ch(\mathbb{Z}) \times Ch(\mathbb{Z})$$

$$x\,\overline{x}\ y\,\overline{y}\ z\,\overline{z}$$

$$s_0 :\ z := <>$$

$$S : \{\ x,\ y,\ z,\ := lorem(x),\ lorem(y),\ hiext(z, x.lov * y.lov)$$

$$if\ x \neq <>,\ y \neq <>\}$$

2.3. Check the Properties!

A.

- $\varphi_S \Rightarrow (x =<> \vee\ y =<>)$
- $(spl(\overline{x} - x, \overline{y} - y, \overline{z})) \in inv_S(x = \overline{x} \wedge y = \overline{y})$, if $f(a,b) = a * b$
- $\forall k \in N, k \geq 0 :\ \left(|\overline{z}| = k \mapsto_S |\overline{z}| = k + 1\right)$

2.4. Check the Properties!

B.

$spl : Ch(T) \times Ch(T) \times Ch(T) \to L$ is a function defined by the following rules::

- $spl(<>, <>, <>) = true$
- $spl(x, y, z) \Rightarrow spl(x; a, y; b, z; f(a,b))$, where $(f : T \times T \to T)$
- spl has the smallest truth set from these functions

$A\ = Ch(\mathbb{Z}) \times\ Ch(\mathbb{Z}) \times\ Ch(\mathbb{Z}) \times Ch(\mathbb{Z}) \times\ Ch(\mathbb{Z}) \times\ Ch(\mathbb{Z})$

$x\,\overline{x}\ y\,\overline{y}\ z\,\overline{z}$

$s_0 :\ z := <>$

$S : \{\ x,\ y,\ z,\ := lorem(x),\ lorem(y),\ hiext(z, x.lov + y.lov)$

$if\ x \neq <>,\ y \neq <>\}$

2.5. Check the Properties!

B.

- $\varphi_S \Rightarrow (x =<> \wedge\ y =<>)$
- $(spl(\overline{x} - x, \overline{y} - y, \overline{z})) \in inv_S(x = \overline{x} \wedge y = \overline{y})$, if $f(a,b) = a + b$
- $\forall k \in [0..|\overline{y} - 1|] :\ \left(|\overline{z}| = k \wedge x \neq <> \mapsto_S |\overline{z}| = k + 1\right)$

2.6. Reduction

A.

- Given m values: $< x_1, \ldots, x_m >$
- Calculate the value of the function g for every value: $< g(x_1), \ldots, g(x_m) >$
- Where $g(x)$ is:

$$g(x) = \sum_{k=0}^{n} \frac{(-1)^k x^{(2k+1)}}{k!}$$

- The power of x and the factorial must not be recalculated in every step!

2.7. Reduction

B.

- Given m values: $< x_1, \ldots, x_m >$
- Calculate the value of the function g for every value: $< g(x_1), \ldots, g(x_m) >$
- Where $g(x)$ is:

$$g(x) = \sum_{k=0}^{n} \frac{(-1)^k x^k}{(2k+1)!}$$

- The power of x and the factorial must not be recalculated in every step!