Programming methodology

Proving correctness of algorithms

Gergely Feldhoffer

Overview of the course topic

- Correctness of algorithms
- Proof of correctness
- Method to prove correctness
- Proofs for basic algorithms
- Some practical tools at the end of the semester concerning software correctness (gdb, valgrind)

Course material

- This material is originated from Dijsktra and Hoare using advanced mathematical toolkit as temporal logic
- We will discuss the same terms defined with basic set theory
- This model was created by Ákos Fóthi using Hungarian definitions
- $\rightarrow$ so be prepared, there are no English sources but this presentation

What to do for your grade

- At end of October or at the beginning of November you will write a test. The possible outcomes are
  - You are awarded with the best grade without taking the exam
  - You can attend to the exam
  - You have to write the test again at the last week
- At the exam you have to convince me about the clarity of your understanding of the theory of algorithm proving.

Relations

- You should be familiar with the terms of set, element, $\emptyset$, direct multiplication $A \times B$, elementary tuple $(x, y) \in A \times B$, relation $F \subseteq A \times B$, set size $|A|$
- $\mathcal{D}_F = \{\forall x \colon F(x) \neq \emptyset\}$ is the **domain** of the relation F
- $\mathcal{R}_F = \{\forall y \colon \exists x \colon F(x) = y\}$ is the **range** of the relation F
- An F relation is function if $\forall x \in \mathcal{D}_F \colon \ |F(X)| = 1$

State space

Def: State space

Let $A_1, A_2, \ldots, A_n$ finite or countably infinite not empty sets. In this case $A = A_1 \times A_2 \times \ldots \times A_n$ is called **state space**, and A_i are type value sets.

Strings

- Let A be a set. We call all possible finite strings of A as A^*. This operator is the Kleene star.
- If $\alpha \in A^*$ then the elements of the string can be expressed like $\alpha_1, \alpha_2, \ldots, \alpha_{|\alpha|}$
- The last element of the string can be also expressed like $\tau(\alpha)$
- A^∞ is all possible infinite strings on set A.
- We will use $A^{**} = A^* \cup A^\infty$ for all finite and infinite string on set A
- The function τ is not defined on infinite strings.
- The reduction of a string is a process where every finite direct recurrences of the same elements are replaced with only one element, like $red(\langle 12333445 \rangle) = \langle 12345 \rangle$

Problem

Def.: Problem

We will call $F \subseteq A \times A$ relation a **problem** on state space A

For a starting state $a \in A$ the possible valid solutions are the set $F(a)$.

- A relation is **deterministic** if $\forall a \in A\colon\ |F(a)| \leq 1$
- If a problem does not assign anything to an $a \in A$ that means that there are no demands for this starting state. A possible misunderstanding would be to think that there are no valid answers for the state, but we will use this state in quite an opposite way, so $F(a) = \emptyset$ and $F(a) = A$ are equivalent in some way.

Program

Def.: Program

$S \subseteq A \times A^{**}$ is program if

1. $\mathcal{D}_S = A$
2. $\forall a \in A : \forall \alpha \in S(a) : \alpha_1 = a$
3. $\forall \alpha \in \mathcal{R}_S : \alpha = red(\alpha)$

- The strings of states are the temporal stages of the machine as the program runs. Different starting states can mean different input values.
- The program has to start from every state
- The program must start in the starting state
- In each step something has to happen

Program and variables

Def: Variable

Let $A = A_1 \times A_2 \times \ldots \times A_n$ be a state space. In this case for each $i \in [1 \ldots n] : f_i \subseteq A \times A_i = \{x \in A \colon f_i(x) = proj_{A_i}(x)\}$

So the variable is not a single one dimensional subspace of the state space but the projection function to it. The subspace is the set containing the possible elements eg. integer numbers. The variable is more than a set, it has history as the program runs. For each $S(a)$ a projection of this string is how the given variable changes its value.

Program function

Def.: Program function

If $S \subseteq A \times A^{**}$ is program then we will define $p(S) \subseteq A \times A$ **program function** of S as

1. $\mathcal{D}_{p(S)} = \{a \in A \colon S(a) \subseteq A^*\}$
2. $\forall a \in \mathcal{D}_{p(S)} \colon p(S)(a) = \{b \in A \colon \exists a \in S(a) \colon b = \tau(a)\}$

- So the program function is defined only where the program certainly terminates,
- and assigns to starting state a the set of the possible outcomes of that state a at the end of the program run.

Solution

Def.: Solution

An $S \subseteq A \times A^{**}$ program **solves** the problem F if

1. $\mathcal{D}_F \subseteq \mathcal{D}_{p(S)}$
2. $\forall a \in \mathcal{D}_F : p(S)(a) \subseteq F(a)$

- So the program should certainly terminate where there are any demands for a starting state,
- and for each starting state the set of possible outcomes of the program run must be ones of the valid outcomes.
- if the program is deterministic then this can be expressed as $p(S)(a) \in F(a)$, and if both the program and the problem is deterministic, then $p(S)(a) = F(a)$

Truth set

If $R \subseteq A \times \mathbb{L}$ then

- **Truth set** of R :
  $\lceil R \rceil = \{a \in A \colon R(A) = \{true\}\}$
- **Weak truth set** of R :
  $\lfloor R \rfloor = \{a \in A \colon true \in R(A)\}$
- Note that if R is function then $\lceil R \rceil = \lfloor R \rfloor$
- $Q \to R \Rightarrow \lceil Q \rceil \subseteq \lceil R \rceil$

Precondition, postcondition

- We will use the term **precondition** as a condition which is true for all starting state of the state space where we have demand for the output, and the
- **postcondition** which is true for all valid input-output pairs and false for everything else.
- If $S \subseteq A \times A^{**}$ is a program any $R \subseteq A \times \mathbb{L}$ condition on state space A can be used as precondition or postcondition
- Our goal is to express the problem using only precondition and postcondition.

Weakest precondition

Def: Weakest precondition

$S \subseteq A \times A^{**}$ is a program, $R \subseteq A \times \mathbb{L}$ is a condition. R is the **weakest precondition** of S if
$\lceil wp(S,R) \rceil = \{a \in \mathcal{D}_{p(S)} : p(S)(a) \subseteq \lceil R \rceil\}$

So which is the section of the state space where if we run S program, we will certainly arrive in states where R is true.

Properties of weakest precondition

Let $S \subseteq A \times A^{**}$ be a program, and $Q, R \subseteq A \times \mathbb{L}$, then

- Principle 'Exclusion of miracles': $wp(S, False) = False$ where $False \subseteq A \times \mathbb{L} = \forall a \in A\colon False(a) = false$
- Monotonicity: If $Q \Rightarrow R$ then $wp(S, Q) \Rightarrow wp(S, R)$
- $wp(S, Q) \wedge wp(S, R) = wp(S, Q \wedge R)$
- $wp(S, Q) \vee wp(S, R) = wp(S, Q \vee R)$

Parameter space

Def: Parameter space

Let $F \subseteq A \times A$ problem. We name set B **parameter space** of F if $\exists F_1, F_2 \colon F_1 \subseteq A \times B \wedge F_2 \subseteq B \times A \wedge F = F_2 \circ F_1$

where $R_1 \circ R_2$ is composition of relations meaning if $a \in (R_1 \circ R_2)(b)$ then $a \in R_1(R_2(b))$
note that the order of arguments of expression $R_1 \circ R_2$ and the order of evaluation is opposite.
The parameter space can be subspace of the state space, restricting to the input variables. This will be our choice from now on, however, this definition allows to use even the whole state space.

Specification

Theorem: Theorem of Specification

Let $F \subseteq A \times A$ problem, B is a parameter space of F,
$F_1 \subseteq A \times B \wedge F_2 \subseteq B \times A \wedge F = F_2 \circ F_1$, and $b \in B$
$\lceil Q_b \rceil = \{a \in A \colon (a, b) \in F_1\} = inv(F_1)(b)$
$\lceil R_b \rceil = \{a \in A \colon (b, a) \in F_2\} = F_2(b)$
In this case if $\forall b \in B \colon Q_b \Rightarrow wp(S, R_b)$ then program S solves problem F

Which means if we can express our demands with precondition and postcondition, then we defined a problem, and we also give a possible method to check whether a program is a solution or not.

Proof of theorem of specification

Let $F \subseteq A \times A$ problem, B is a parameter space of F,
$F_1 \subseteq A \times B \wedge F_2 \subseteq B \times A \wedge F = F_2 \circ F_1$, and $b \in B$
$\lceil Q_b \rceil = \{a \in A \colon (a, b) \in F_1\} = inv(F_1)(b)$
$\lceil R_b \rceil = \{a \in A \colon (b, a) \in F_2\} = F_2(b)$
In this case if $\forall b \in B \colon Q_b \Rightarrow wp(S, R_b)$ then program S solves problem F
Proof:

- $\mathcal{D}_F \subseteq \mathcal{D}_{p(S)}$ since every $a \in \mathcal{D}_F \colon \exists b \in B \colon a \in \lceil Q_b \rceil$ because of F_1. Since $Q_b \Rightarrow wp(S, R_b)$ so
  $a \in \lceil Q_b \rceil \subseteq \lceil wp(S, R_b) \rceil \subseteq (D)_{p(S)}$
- $\forall a \in \mathcal{D}_F \colon p(S)(a) \subseteq F(a)$ since if $a \in \mathcal{D}_F$ and $b \in B \wedge a \in \lceil Q_b \rceil$ then
  $p(S)(a) \subseteq \lceil R_b \rceil = F_2(b) \subseteq F_2(F_1(a)) = F(a)$

Example of specification

Example: addition of two numbers

$A = \mathbb{Z}_x \times \mathbb{Z}_y \times \mathbb{Z}_z$

$B = \mathbb{Z}_{x'} \times \mathbb{Z}_{y'}$

$Q_{x'(b),y'(b)}(a) = (x(a) = x'(b) \wedge y(a) = y'(b))$

$R_{x'(b),y'(b)}(a) = (x(a) = x'(b) \wedge y(a) = y'(b) \wedge z(a) = x'(b) + y'(b))$

We will use x' and y' are represent the starting value of the variable as it is in the parameter space.

This formalism can be simplified since each raw variable is defined on A and each parameter is defined on B ..

Example of specification

Example: addition of two numbers

$A = \mathbb{Z}_x \times \mathbb{Z}_y \times \mathbb{Z}_z$

$B = \mathbb{Z}_{x'} \times \mathbb{Z}_{y'}$

$Q_{x',y'}(a) = (x = x' \wedge y = y')$

$R_{x',y'}(a) = (x = x' \wedge y = y' \wedge z = x' + y')$

..and since Q and R has to be defined on the whole parameter space..

Example of specification

Example: addition of two numbers

$A = \mathbb{Z}_x \times \mathbb{Z}_y \times \mathbb{Z}_z$

$B = \mathbb{Z}_{x'} \times \mathbb{Z}_{y'}$

$Q : x = x' \wedge y = y'$

$R : x = x' \wedge y = y' \wedge z = x' + y'$

..we arrived to the formalism we will use for specifications.

Elementary program

Elementary program

$S \subseteq A \times A^{**}$ program is **elementary** if
$\forall a \in A\colon \forall \alpha \in S(a)\colon (\exists b \in A\colon \alpha = red(\langle ab \rangle)) \vee \alpha = \langle aa \ldots \rangle$

So a program is elementary if it does exactly one step or hangs.
We will use some special elementary programs such as SKIP, ABORT, and assignment.

Skip Abort

SKIP

$$SKIP = \{(a, \alpha) \in A \times A^{**} : \alpha = \langle a \rangle\}$$

ABORT

$$ABORT = \{(a, \alpha) \in A \times A^{**} : \alpha = \langle aaa \ldots \rangle\}$$

Assignment

Every other elementary programs are general assignments. To handle assignments, we will use a state transition function E to describe it: $E \subseteq A \times A$, so if we are in state a we step to (one of the) state(s) $E(a)$.

So if we have an E as $\mathcal{D}_E = A$, then
$S_E = \{(a, \alpha) \in A \times A^{**} : \alpha = red(\langle ab \rangle) : b \in E(a)\}$

Assignment

In general $\mathcal{D}_E = A$ is not true, in these cases where the state transition function is not defined, the assignment should abort.

Assignment

$S_E = \{(a, \alpha) \in A \times A^{**} : a \in \mathcal{D}_E \wedge \alpha = red(\langle ab \rangle) \wedge b \in E(a)\} \cup$
$\cup \{(a, \alpha) \in A \times A^{**} : a \notin \mathcal{D}_E \wedge \alpha = \langle aaa \ldots \rangle\}$

Note that every state transition can be an assignment. Each machine support a limited possibilities of state transitions, that is why we can not solve NP complete tasks in one step.
Computer science calls esoteric imaginary state transitions as oracle. Interestingly it is proven that on a machine which supports every usual operation and includes one NP complete task, on that machine P=NP. The hard part is to prove that NP on this machine is not stronger.

Program functions of elementary programs

$$wp(SKIP, R) = R$$

$$wp(ABORT, R) = False$$

$$wp(S_E, R) = \{a \in \mathcal{D}_E \colon E(a) \subseteq \lceil R \rceil\}$$

$$\mathcal{D}_{p(SKIP)} = A$$
$$\forall a \in A \colon p(SKIP)(a) = a$$

$$\mathcal{D}_{p(ABORT)} = \emptyset$$

$$p(S_E) = E$$

Notations

Example: $A = \mathbb{Z}_x \times \mathbb{Z}_y$
$E = \{(a, b) \in A \times A \colon y(b) = 2 * x(a) \wedge x(b) = x(a)\}$
We will be noted as $y := 2x$, this is a simple assignment

$E = \{(a, b) \in A \times A \colon y(b) = 2 * x(a) \wedge x(b) = x(a) + 1\}$
as $x, y := x + 1, 2x$, this is a parallel assignment

$E = \{(a, b) \in A \times A \colon y(b) \in X\}$
as $y :\in X$, this is a value choice

Program structures

We will use three kind of program structure: sequence, branch, and loop.

The sequence means running two program after each other. The branch means state dependent conditional program execution. The loop is state dependent repeated execution of a program.

Note that every program can be used as part of program structure. Every usage of a program structure results a program.

Sequence

Def: Sequence

Let $S_1, S_2 \subseteq A \times A^{**}$ be programs. We will call $S \subseteq A \times A^{**}$ as **Sequence of S_1 and S_2** if $\forall a \in A$:

$S(a) = \{\alpha \in A^{\infty} : \alpha \in S_1(a)\} \cup$

$\cup \{red(concat(\alpha, \beta)) \in A^{**} : \alpha \in S_1(a) \cap A^{*} \wedge \beta \in S_2(\tau(\alpha))\}$

Branch

Def: Branch

Let $\pi_1, \ldots, \pi_n : A \to \mathbb{L}$ be conditions, $S_1, \ldots, S_n$ programs on A.
We will call $IF \subseteq A \times A^{**}$ program as **Branch** of
$(\pi_1 : S_1, \ldots, \pi_n : S_n)$ if $\forall a \in A$:

$$IF(a) = \bigcup_{i=1}^{n} w_i(a) \cup w_0(a)$$

$$\text{where } \forall i \in [1..n]\colon w_i(a) = \begin{cases} S_i(a) & \text{if } \pi_i(a), \\ \emptyset & \text{else} \end{cases}$$

$$w_0(a) = \begin{cases} \langle aaa \ldots \rangle & \text{if } \forall i \in [1..n]\colon \neg\pi_i(a), \\ \emptyset & \text{else} \end{cases}$$

Branch

- Note that a branch can be non-deterministic: if more than one condition are true in a state each of the routes can happen.
- In states where no branch condition is true, the branch must abort. This is not the usual approach, as C switch without default performs SKIP instead of ABORT.

Loop

Def: Loop

Let π be a condition and S_0 a program on A. We will call $DO \subseteq A \times A^{**}$ a **Loop** if

- $\forall a \notin \lceil \pi \rceil : DO(a) = \{\langle a \rangle\}$
- $\forall a \in \lceil \pi \rceil : DO(a) =$

$$
\begin{aligned}
& \{\alpha \in A^{**} : \exists \alpha^1, \ldots, \alpha^n \in A^{**} : && (1)\\
& \alpha = red(concat(\alpha^1, \ldots, \alpha^n)) \wedge \alpha^1 \in S_0(a) \wedge && (2)\\
& \wedge (\forall i \in [1..n-1] : \alpha^i \in A^* \wedge && (3)\\
& \wedge \alpha^{i+1} \in S_0(\tau(\alpha^i)) \wedge \pi(\tau(\alpha^i))) \wedge && (4)\\
& \wedge ((\alpha^n \in A^\infty) \vee (\alpha^n \in A^* \wedge \neg \pi(\tau(\alpha^n))))\} && (5)\\
\cup \quad & \{a \in A^\infty : \forall i \in \mathbb{N} : && (6)\\
& \exists \alpha^i \in A^* : \alpha = red(concat(\alpha^1, \alpha^2, \ldots)) \wedge && (7)\\
& \alpha^1 \in S_0(a) \wedge (\forall i \in \mathbb{N} : \alpha^i \in A^* \wedge && (8)\\
& \alpha^{i+1} \in S_0(\tau(\alpha^i)) \wedge \pi(\tau(\alpha^i)))\} && (9)
\end{aligned}
$$

Loop

- So the loop is the repetition of the loop body where the resulting program string is a concatenation of the iterations (2) where the iterations are finite in the first $n-1$ cases (3), all of the iterations starts where the previous one ended (4) and the loop condition is true(4), and the end of this line may be infinite or such kind of finite where the end of the last iteration goes outside of the loop conditions truth set(5),
- or, the string of the loop can be endless without aborting if there are infinite number of finite iterations where the loop condition is always true for the end of the strings of iterations (6-9).

Program function of sequence

Composition of relations $R_1 \subseteq A \times B, R_2 \subseteq B \times C$:
$R_2 \circ R_1 = \{(a, c) \in A \times C : \exists b \in B : (a, b) \in R_1 \wedge (b, c) \in R_2\}$

Strict composition of relations $R_1 \subseteq A \times B, R_2 \subseteq B \times C$:
$R_2 \odot R_1 = \{(a, c) \in A \times C : \exists b \in B : (a, b) \in R_1 \wedge (b, c) \in R_2 \wedge R_1(a) \subseteq \mathcal{D}_{R_2}\}$

Theorem: Program function of sequence

If $S = (S_1; S_2)$ sequence of S_1, S_2 programs
$p(S) = p(S_1) \odot p(S_2)$

Program function of branch

Theorem: Program function of branch

If $IF(\pi_1 : S_1, \ldots, \pi_n : S_n) \subseteq A \times A^{**}$ is a branch, then
$\mathcal{D}_{p(IF)} = \{a \in A \colon \bigvee_{i=1}^{n} \pi_i(a) \wedge \forall i \in [1 \ldots n] \colon \pi_i(a) \Rightarrow a \in \mathcal{D}_{p(S_i)}\}$

$$\forall a \in \mathcal{D}_{p(IF)} \colon p(IF)(a) = \bigcup_{i=1}^{n} pw_i(a)$$

$$\text{where } \forall i \in [1..n] \colon pw_i(a) = \begin{cases} p(S_i)(a) & \text{if } \pi_i(a), \\ \emptyset & \text{else} \end{cases}$$

So the program function of IF is defined in states where at least one condition is true, and no S_i hangs there, so it will certeanly terminate, and every possible pathway is included where the condition is true.

Program function of loop

Any homogeneous R relation can be composed with itself.
$R^n = R \circ R^{n-1}$ where R^0 is the identity.

Def: Exclosure of relation

If $R \subseteq A \times A$ is a relation, the exclosure of R is $\bar{R} \subseteq A \times A$ where
$\mathcal{D}_{\bar{R}} = \{a \in A : \nexists \alpha \in A^\infty : \alpha_1 \in R(A) \wedge \forall i \in \mathbb{N} : \alpha_{i+1} \in R(\alpha_i)\}$
$\forall a \in \mathcal{D}_{\bar{R}} : \bar{R}(a) = \{b \in A | \exists k \in \mathbb{N}_0 : b \in R^k(a) \wedge b \notin \mathcal{D}_R\}$

So the resulting relation contains every (a, b) pair where there is a path to reach b from a, and there is no step further from b available.

Program function of loop

If $R \subseteq A \times A$ and $B \subset A$ then $R|_B = \{(a, b) \in B \times A : (a, b) \in R\}$
This is the restriction of the relation to a subset.

Def: Restriction to condition

If $R \subseteq A \times A$ and $\pi \subseteq A \times \mathbb{L}$ then
$R|_\pi = (R \cap (\lceil \pi \rceil \times A)) \cup \{(a, a) \in A \times A : a \in \lceil \pi \rceil \setminus \mathcal{D}_R\}$

So the restriction to a condition differs from the restriction to the truth set of the condition as it is broadened with identities, making $\mathcal{D}_{R|_\pi} = \lceil \pi \rceil$

Program function of loop

Theorem: Program function of loop

If $DO(S_0, \pi)$ is a loop then
$p(DO) = \overline{p(S_0)|_\pi}$

Deduction rules

We will give a set of deduction rules for program structures. As every program can be expressed as a hierarchical structure where the nodes of the hierarchy are program structures and the leaves are elementary programs, rules for each structure is enough to deduce every program.

Deduction rules for sequence

Deduction rules for sequence

Let $S = (S_1; S_2)$ be a sequence, and $Q, R, Q' \subseteq A \times \mathbb{L}$ statements on A. If

1. $Q \Rightarrow wp(S_1, Q')$ and
2. $Q' \Rightarrow wp(S_2, R)$,

then $Q \Rightarrow wp(S, R)$.

Deduction rules for branch

Deduction rules for branch

Let $IF = (\pi_1 : S_1 \ldots \pi_n : S_n)$ be a branch and $Q, R \subseteq A \times \mathbb{L}$ statements on A. If

1. $Q \Rightarrow \bigvee_{i=1}^{n} \pi_i$, and
2. $\forall i \in [1 \ldots n]\colon Q \wedge \pi_i \Rightarrow wp(S_i, R)$,

then $Q \Rightarrow wp(IF, R)$.

Deduction rules for loop

Deduction rules for loop

Let $DO(\pi, S_0)$ be a loop, $P, Q, R \subseteq A \times \mathbb{L}$ statements and $t : A \to \mathbb{Z}$ function. If

1. $Q \Rightarrow P$
2. $P \wedge \neg\pi \Rightarrow R$
3. $P \wedge \pi \Rightarrow wp(S_0, P)$
4. $P \wedge \pi \Rightarrow t > 0$
5. $P \wedge \pi \wedge t = t_0 \Rightarrow wp(S_0, t < t_0)$

then $Q \Rightarrow wp(SO, R)$

Deduction rules for loop

P is the loop invariant, and t is the terminating function.
P must be true all along the loop and even after the last iteration.
P definitely not equals π, as the second role shows. P is the connection between the start and the end of the loop iteration chain.
The hard part of proving the correctness of a program is to guess P.
t should be used to confirm that the loop is not infinite. The value of $t(a)$ must be decreased along the loop body borders in each iteration, and must be positive. As an integer function infinite regression cannot happen.

behold, adventurer

no slides beyond

todo

todo

todo

todo