Example

{* P *} X,Y := x,y ; Pr(x,OUT y) {* Q *}

{* $P \wedge (Q \Rightarrow Q')$ [y'/y][x,y/X,Y] *} Pr(x,y) {* Q' *}

Given specification:

```
{∗ size > 0 ∗}
A := a;
sort(size, OUT a);
{∗ ∀i : 0 ≤ i < size : (∀j: i < j < size :a[i] ≤ a[j]) ∧
   ∀i : 0 ≤ i < size : (∃j: 0 ≤ j < size :A[i] = a[j] ∗}
```

Program:

```
{∗ size > 0 ∗}
A := a;
sort(size, OUT a);
{∗ a[size − 1] = MAX A[0..size) ∗}
```

$(Q \Rightarrow Q')$:

need to prove for all arrays a'

```
[A1]∀i : 0 ≤ i < size : (∀j: i < j < size :a'[i] ≤ a'[j])
[A2]∀i : 0 ≤ i < size : (∃j: 0 ≤ j < size : a[i] = a'[j]
[G] a'[size − 1] = MAX a[0..size)
```

Example

$$\frac{\{^*\ P\ ^*\}\ \texttt{X,Y := x,y ; Pr(x,OUT y)}\ \{^*\ Q\ ^*\}}{\{^*\ P \wedge (Q \Rightarrow Q')\ \texttt{[y'/y][x,y/X,Y]}\ ^*\}\ \texttt{Pr(x,y)}\ \{^*\ Q'\ ^*\}}$$

Given specification:

```
{* (∃k : k>i : a[k] ≠ a[i]) *}
Iold := i;
findOther(a, OUT i)
{* i>Iold ∧ a[i] ≠ a[Iold] *}
```

Calculating P2:

```
{* P2 *}
Iold := i;
findOther(a,i)
{*P3 : a[i] ≠ x*}
```

Program:

```
{* P0 : (∀i::(∃k:i<k:a[i]<a[k]))*}
{* P1 *}
x:=a[2i+1]; i:=2i+1;
{* P2 *}
findOther(a,i)
{* P3 : a[i] ≠ x*}
```

P

```
{* (∃k : k>i : a[k] ≠ a[i]) ∧
   (i>Iold ∧ a[i] ≠ a[Iold]) => a[i] ≠ x)  *}[i'/i][i/Iold]
```

Q   Q'

```
=
{* (∃k : k>i : a[k] ≠ a[i]) ∧
   (i'>Iold ∧ a[i'] ≠ a[Iold] => a[i'] ≠ x) *}[i/Iold]
=
{* (∃k : k>i : a[k] ≠ a[i]) ∧
   (i'>i ∧ a[i'] ≠ a[i] => a[i'] ≠ x) *}
```

Example

Given specification:

```
{* (∃k : k>i : a[k] ≠ a[i]) *}
Iold := i;
findOther(a, OUT i)
{* i>Iold ∧ a[i] ≠ a[Iold] *}
```

Program:

```
{*P0 : (∀i::(∃k:i<k:a[i]<a[k]))*}
{* P1 *}
x:=a[2i+1]; i:=2i+1;
{* P2 *}
findOther(a,i)
{* P3 : a[i] ≠ x*}
```

Calculating P_1:

```
{* (∃k : k>i : a[k] ≠ a[i]) ∧
   (i'>i ∧ a[i'] ≠ a[i] => a[i'] ≠ x) *}
```

```
{* (∃k : k>2i+1 : a[k] ≠ a[2i+1]) ∧
        (i'>2*i +1 ∧ a[i'] ≠ a[2*i +1] => a[i'] ≠ a[2i + 1]) *}
   x:=a[2i+1];
{* (∃k : k>2i+1 : a[k] ≠ a[2i+1]) ∧
        (i'>2*i +1 ∧ a[i'] ≠ a[2*i +1] => a[i'] ≠ x) *}
   i:=2i+1;
{* (∃k : k>i : a[k] ≠ a[i]) ∧ (i'>i ∧ a[i'] ≠ a[i] => a[i'] ≠ x) *}
```

Example

```
{*(∀i::(∃k:i<k:a[i]<a[k]))*}
```

implies

```
{* (∃k : k>2i+1 : a[k] ≠ a[2i+1]) ∧
        (i’>2*i +1 ∧ a[i’] ≠ a[2i +1] => a[i’] ≠ a[2i + 1]) *}
```

```
[A1] (∀i::(∃k:i<k:a[i]<a[k]))
[G1] (∃k : k>2i+1 : a[k] ≠ a[2i+1])

PROOF
1. {forall-elim} (∃k:2i+1<k:a[2i+1] < a[k]))
2. {1.} (∃k:2i+1<k:a[2i+1] ≠ a[k]))
```

```
[A1] (∀i::(∃k:i<k:a[i]<a[k]))
[A2] i’>2*i +1
[A3] a[i’] ≠ a[2*i +1]
[G2] a[i’] ≠ a[2i + 1]

PROOF
1. {A3}
```

Revision

- Termination metrics with conditionals
  - how to find them
  - how to prove them correct

Finding a termination metric

```
{* 0 < x < y *}
while x < y do {
    if y-x=1
      then y:=y+1
      else y:=y-2
}
```

$m = (y\text{-}x\text{=}1) \to 5 \mid 2(y-x)$

x	y
10	18
	16
	14
	12
	10

x	y	(y-x)	2(y-x)
10	15	5	10
	13	3	6
	11	1	~~2~~ 5
	12	2	4
	10	0	0

Finding a termination metric

```
{* 0 < x < y *}
while x < y do {
    if y-x=1
       then y:=y+1
       else y:=y-2
}
```

m = (y−x=1) → 5 | 2(y−x)

Prove that

{* I ∧ g *} C:=m; S {* m < C *}

wp (if…) (m < C)

= (y−x=1 ⇒ ((y+1−x=1) → 5|2(y+1−x) < (y−x=1) → 5|2(y−x)) ∧

(y−x≠1 ⇒ ((y−2−x=1) → 5|2(y−2−x) < (y−x=1) → 5|2(y−x)))

= (2 ∗ 2 < 5) ∧

(y−x≠1 ⇒ (y−x=3) → 5 | 2(y−2−x) < 2(y−x))

= (y−x≠1 ⇒ (y−x=3) → 5 < 2(y−x) | 2(y−2−x) < 2(y−x)

= (y−x≠1 ⇒ (y−x=3) → 5 < 6 | −4 < 0

= true

Finding a termination metric

```
{* 0 < x < y *}
while x < y do {
    if y−x=1
      then y:=y+1
      else y:=y−2
}
```

m =(odd (y-x) → 2y | y)

Prove that

{* I ∧ g *} C:=m; S {* m < C *}

wp (if…) (m < C)

= (y−x=1 ⇒ (odd (y+1−x) →2(y+1) | y+1) < (odd (y−x) → 2y | y)) ∧

(y−x≠1 ⇒ (odd (y−2−x) →2(y−2) | y−2) < (odd (y−x) → 2y | y))

= y+1 < 2y ∧ we need y > 1 as invariant!

2(y−2) < 2y

∧

(y−2) < y ,

Exam

- (Almost) most importantly: don’t forget to bring a print out of the LN appendix!
- What to expect in the exam

Exam

- Part 1:
  - multiple choice questions
  - 40% of the overall marks
  - what do we test there?

- Do you understanding predicate logic proofs and know how to apply the proof rules?

Which of the following proofs is correct (according the the proof system of the LN)? Read the steps carefully.

(a)
```
PROOF
[A1:]   (∀x : P x : Q x ∨ R x)
[A2:]   P a
[G:]    P a ∧ Q a
1.  { ∀-elimination on A1 using A2 }   Q a ∨ R a
2.  { ∨ elimination on 1 }              Q a
3.  { conjunction of A2 and 2 }         P a ∧ Q a
END
```

(b)
```
PROOF
[A1:]   (∃x : P x : Q x)
[G:]    (∀x :: P x ∧ Q x)
1.  { ∃-elimination on A1 }    [SOME x] P x ∧ Q x
2.  { ∀-introduction on 1 }    (∀x :: P x ∧ Q x)
END
```

(c) PROOF

```
[A1:]   ¬(∀x :: P x)
[A2:]   ¬P a ⇒ Q a
[G:]    Q a
1.  { ∀-elimination on A1 }                ¬P a
2.  { Modus Ponens on A2 using 1 }          Q a
END
```

(d) PROOF

```
[A1:]   ¬(∃x :: P x)
[A2:]   P a ∨ Q a
[G:]    Q a
1.  { rewrite A1 with negate-∃ }   (∀x :: ¬P x)
2.  { ∀-elimination on 1 }          ¬P a
3.  { rewriting A2 with 2 }          false ∨ Q a
4.  { simplifying 3 }                Q a
END
```

- Do you understand the meaning of Hoare-triples, weakest preconditions, partia
correctness?

The specification $\{*\ Q\ *\}\ S\ \{*\ \mathbf{true}\ *\}$ is known to be valid under **total correctness**. Which of the following conclusions is correct?

(a) S will terminate when executed in any state.

(b) If the implication $P \Rightarrow Q$ is valid, then S will terminate when executed in any state satisfying P.

(c) The specification $\{*\ Q\ *\}\ S\ \{*\ Q \Rightarrow R\ *\}$ is also valid under partial correctness.

(d) The specification $\{*\ P \Rightarrow Q\ *\}\ S\ \{*\ \mathbf{true}\ *\}$ is also valid under total correctness.

Which of the following statements about weakest pre-condition is correct?

(a) $\{*\ \mathbf{wp}\ S\ Q\ *\}\ \ S\ \ \{*\ Q\ *\}$ is always a valid specification.

(b) $\{*\ P \Rightarrow (\mathbf{wp}\ S\ Q)\ *\}\ \ S\ \ \{*\ Q\ *\}$ is always a valid specification.

(c) $\{*\ Q\ *\}\ \ S\ \ \{*\ \mathbf{wp}\ S\ Q\ *\}$ is always a valid specification.

(d) $\{*\ P\ *\}\ \ S\ \ \{*\ Q\ *\}$ is valid if and only if the predicate $\mathbf{wp}\ S\ (P \Rightarrow Q)$ is valid.

Do you know how to calculate weakest preconditions?

What is the **weakest** pre-condition of the following statement with respect to the given post-condition?

$$\{*\ ?\ *\}\ \ \{\ \textbf{if}\ x{=}y\ \textbf{then}\ x := x{+}1\ \textbf{else skip}\ \}\ ;\ x := x{+}y \quad \{*\ x = y\ *\}$$

(a) $((x{=}y) \land x{=}{-}1)\ \lor\ ((x{\neq}y) \land x{=}0)$

(b) $x{=}{-}1 \land y{=}{-}1$

(c) $(x{=}y)\ \rightarrow\ x{+}1{+}x{+}y{=}y \mid x{+}y{=}y$

(d) $(x{=}y\ \Rightarrow\ x{+}1{+}y{=}y)\ \lor\ (x{\neq}\ y\ \Rightarrow\ x{+}y{=}y)$

What is the **weakest** pre-condition of the following statement with respect to the given post-condition?

$$\{*\ ?\ *\}\quad a[k] := a[0]{+}a[k]\ \ \{*\ a[0]{=}a[k]\ *\}$$

(a) $a[0]\ =\ a[0] + a[k]$

(b) $a(0\ \textbf{repby}\ a[0]{+}a[k])[0]\ =\ a[0] + a[k]$

(c) $a(k\ \textbf{repby}\ a[0]{+}a[k])[0]\ =\ a[0] + a[k]$

(d) $a[0]\ \textbf{repby}\ a[0]{+}a[k]\ =\ a[k]\ \textbf{repby}\ a[0]{+}a[k]$

Do you understand loop invariants and termination metrics?

Consider the following program, with the given specification.

$\{*\ \mathbf{true}\ *\}$

$\mathbf{while}\ x{>}y\ \mathbf{do}\{$
$\quad \mathbf{if}\ (\mathtt{odd}(x))\ \mathbf{then}\ \ x := x{+}1\ \mathbf{else}\ y := y{+}2$
$\}$

$\{*\ \mathbf{true}\ *\}$

Which pair of invariant I and termination metric m is consistent and good enough to prove that the program above terminates?

(a) invariant: **true**, termination metric: $x - y + 2 * (x\ \mathbf{mod}\ 2)$

(b) invariant: $\mathtt{even}(x)$, termination metric: $x - y$

(c) invariant: **true**, termination metric: $x - 2 * y$

(d) invariant: **true**, termination metric: $x - y - (x\ \mathbf{mod}\ 2)$

- Do you understand how to verify imperative programs against a functional specification?

$$\begin{aligned} \mathsf{val}\ v\ [] &= v \\ \mathsf{val}\ v\ (x:z) &= \mathsf{val}\ (10{*}v + x)\ z \end{aligned}$$

Below is an implementation of the function. The specification is given.

```
{* true *}
v := 0 ; t := s ;
while t≠[ ] do {
    v := 10*v + head(t) ;
    t := tail(t)
}

{* v = val 0 s *}
```

Which of the following is a consistent and good enough invariant to prove the correctness of the above specification?

(a) $v = \mathsf{val}\ 0\ t$

(b) $\mathsf{val}\ v\ t = \mathsf{val}\ v\ s$

(c) $\mathsf{val}\ v\ t = \mathsf{val}\ 0\ s$

(d) $v = \mathsf{sum}[s_i{*}10^i \mid 0 \le i < \mathsf{length}(s)]$

Exam

- Part 2:
  - 60% of the overall marks
  - three subquestions
  - what do we test there?
    - can you prove program properties for functional programs and/or imperative programs
    - for functional programs: equational reasoning, induction
    - for imperative programs:
      - can you find invariants for partial and total correctness, metrics for termination
      - can you prove correctness with given invariant
      - can you apply rules like black box,
  - `findOther` and conditional termination metric were from Part II of exams

Example

1. [4 pt] **Loop**

   Consider the following program and its specification. The program checks if every k-th element of an integer array **a** does not exceed x^k.

```
{*  0≤N ∧ 0<x  *}     // pre-condition

   y := 1 ;
   k := 0 ;
   ok := true ;
   while k≠N ∧ ok do {
      ok := ok ∧ a[k]≤y ;
      y := y*x ;
      k := k + 1 ;
   } ;

{*  ok  =  (∀j : 0≤j<N : a[j] ≤ x^j)  *}     // post-condition
```

   Give a **formal proof** that the program satisfies its specification, under *partial correctness.*