UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics, and Science

School of Computer Science and Statistics

B.A. (Mod.) Computer Science
Junior Sophister SAMPLE

3BA31: Formal Methods (SAMPLE)

today right here hh:mm–hh+2:mm

Dr. Andrew Butterfield

Instructions to Candidates:

Attempt three questions.
(all questions carry equal marks)

Materials permitted for this examination:

Q1.

(a) What is an “Environment” and what is its purpose? Give two environments, one falsifying, the other “truthifying”, the following predicate,

\[ \forall x : \mathbb{N} \cdot x \geq y \lor y - z \leq x \]

given that \( y, z : \mathbb{N} \). \hspace{1cm} (9 marks)

(b) Prove the following law:

\[ (\exists x \bullet P) \Rightarrow N \equiv (\forall x \mid P \bullet N) \quad x \notin N \]

You may use any laws in the reference at the end of this paper, or in the Handbook of Mathematics. \hspace{1cm} (12 marks)

(c) Prove the following laws of programming:

\[ P; \text{skip} = P \]
\[ x := e \; ; \text{skip} ; x := f = x := f[e/x] \]

using the first to help in proving the second. \hspace{1cm} (12 marks)

Q2. The sum 1..n function \( \text{sum}(n) \) satisfies the following laws:

\[ \text{sum}(0) = 0 \]
\[ \text{sum}(n) = n + \text{sum}(n - 1), \quad n > 0 \]

We have the following specification (\( SSpec \)) and proposed program (\( SProg \)) and invariant \( inv \):

\[ SSpec \triangleq s, x : [s' = \text{sum}(n)] \]
\[ SProg \triangleq s, x := 0, n; (x > 0) * (s, x := s + x, x - 1) \]
\[ inv \triangleq s + \text{sum}(x) = \text{sum}(n) \]

both with alphabet \( \{n, n', s, s', x, x'\} \). Assume all variables are integer, except \( x \) which is natural.

(a) Prove: \( SSpec \subseteq s, x : [inv' \land x' \leq 0] \) \hspace{1cm} (9 marks)

(b) Prove: \( s, x : [inv'] \subseteq s, x := 0, n \) \hspace{1cm} (9 marks)

(c) Prove: \( s, x : [x > 0 \land inv \Rightarrow inv'] \subseteq s, x := s + x, x - 1 \) \hspace{1cm} (9 marks)

(d) The above is a proof of partial correctness. What does this term mean exactly? What also needs to be done to prove total correctness? \hspace{1cm} (6 marks)
Q3.

(a) Why are the properties of transitivity and monotonicity important for refinement?  
(7 marks)

(b) Which of the following predicate-transformer functions are monotonic?

\[ F(P) \equiv \exists x \cdot Q \land \neg(P \land \neg Q) \]
\[ G(P) \equiv Q \lor \neg\exists y \cdot (R \land \neg P) \]
\[ H(P) \equiv \neg P \land \neg(P \lor \neg R) \]
\[ K(P) \equiv P \lor (P \equiv Q) \]

Justify your answer in each case  
(8 marks)

(c) In UTP, why do healthiness predicate-transformers have to be idempotent?  
(6 marks)

(d) Given the following definitions of \(R3\) and \(\text{CSP}2\),

\[ R3(P) \equiv \text{II} \triangleleft \text{wait} \triangleright P \]
\[ \text{CSP}2(P) \equiv P ; (\text{ok} \Rightarrow \text{ok}') \land \text{wait}' = \text{wait} \land \text{tr}' = \text{tr} \land \text{ref}' = \text{ref} \]

and the fact that \(\text{II}\) is \(\text{CSP}2\)-healthy, prove that they commute:

\[ R3 \circ \text{CSP}2 = \text{CSP}2 \circ R3 \]

(12 marks)

Q4.

(a) Prove the following sequence property:

\[ s \neq \langle \rangle \Rightarrow (s - \text{front}(s)) = \langle \text{last}(s) \rangle \]

(hint: induction on non-nil \(s\) should work)  
(16 marks)

(b) Sequence subtraction \((s - t)\) is partial, only being defined when its second argument is a prefix of its first \((t \leq s)\). Given the following definition of \(R2\),

\[ R2(P) \equiv \exists s \cdot P[s, s \leftarrow (\text{tr}' - \text{tr})/\text{tr}, \text{tr}'] \]

prove that it is idempotent (i.e., \(R2 \circ R2 = R2\)), and highlight how the partiality of sequence subtraction is considered in the proof.  
(17 marks)
Reference

\[ N \lor (\forall x \cdot P) \equiv (\forall x \cdot N \lor P), \quad x \not\equiv N \]
\[ N \land (\exists x \cdot P) \equiv (\exists x \cdot N \land P), \quad x \not\equiv N \]
\[ (\exists x \cdot P) \equiv \neg (\forall x \cdot \neg P) \]
\[ (\forall x \cdot x = e \land P) \equiv P[e/x] \]
\[ (\exists b : B \cdot P) \equiv P[False/b] \lor P[True/b] \]
\[ P \Rightarrow Q \lor R \equiv (P \Rightarrow Q) \lor (P \Rightarrow R) \]
\[ p \Rightarrow q \equiv \neg p \lor q \]
\[ P \land Q \Rightarrow P \]
\[ \neg -p \equiv \neg p \]
\[ x := A \equiv e \equiv x' = e \land \nu' = \nu, \quad A = \{x, x', \nu, \nu'\} \]
\[ skip_A \equiv \nu' = \nu, \quad A = \{\nu, \nu'\} \]
\[ (\forall x | R \cdot P) \equiv (\forall x \cdot R \Rightarrow P) \]
\[ P ; Q \equiv (\exists \nu_m \cdot P[\nu_m / \nu'] \land Q[\nu_m / \nu]) \]
\[ P[e/x][f/x] = P[e[f/x]/x] \]
\[ P[x/y][y/x] = P, \quad x \not\equiv P \]
\[ P[e_1, \ldots, e_n/x_1, \ldots, x_n][f/y] = P[e_1, \ldots, e_n, f/x_1, \ldots, x_n/y], \quad y \not\equiv e_1, \ldots, e_n \]
\[ (P \leftarrow c \triangleright Q)[e/x] = P[e/x] \leftarrow c[e/x] \triangleright Q[e/x] \]
\[ (P \leftarrow c \triangleright Q) ; R = (P ; R) \leftarrow c \triangleright (Q ; R) \]
\[ \bar{x} : [P] \equiv P \land \nu' = \nu \]
\[ P \sqsubseteq P \land Q \]
\[ \bar{x} := A \equiv e_1 \land \ldots \land e_n = e_n \land \nu' = \nu \]
\[ e = e \]
\[ x = e \land P \equiv x = e \land P[e/x] \]
\[ (x = e \land P \Rightarrow Q) \equiv (x = e \land P \Rightarrow Q[e/x]) \]
\[ (S \sqsubseteq P \land x = e) \equiv (S[e/x] \sqsubseteq P \land x = e) \]
\[ true \sqsubseteq P \]
\[ (S \sqsubseteq Q) \land (Q \sqsubseteq P) \Rightarrow (S \sqsubseteq P) \]
\[ \bar{v}[pre \Rightarrow inv' \land \neg c'] \sqsubseteq \bar{v}[pre \Rightarrow inv'] \land \bar{v}[c \land inv \Rightarrow inv'] \]
\[ front(x) \equiv \langle \rangle \]
\[ front(x_2s) \equiv x_2(front(s)) \]
\[ last(s) \equiv last(x) \]
\[ last(s) \equiv last(x) \]
\[ s - \langle \rangle \equiv \langle \rangle \]
\[ (x_2s) - (x_2t) \equiv s - t \]
\[ \#(\langle \rangle) \equiv 0 \]
\[ \#(x_2s) \equiv 1 + \#s \]
\[ (x_2s)(1) \equiv x \]
\[ (x_2s)(n) \equiv s(n - 1) \]
\[ s(1) \equiv head(s), \quad s \not\equiv \langle \rangle \]
\[ s \leq \langle \rangle \]
\[ s \leq s \land t, \quad \delta s \land \delta t \]
1 Marking Scheme: 3BA31 (Q1–4)

A1. (a) What is an “Environment" and what is its purpose? Give two environments, one falsifying, the other “truthifying”, the following predicate,

$$\forall x : \mathbb{N} \cdot x \geq y \lor y - z \leq x$$
given that $y, z : \mathbb{N}$. (9 marks)

An environment is a mapping from variable names to their values. It is used to assist in evaluating expressions to get their values, and predicates to see if they are true or false. In $\forall x : \mathbb{N} \cdot x \geq y \lor y - z \leq x$, $x$ is bound but both $y$ and $z$ are free, so we need to supply values for these two.

Falsifying $\{y \mapsto 2, z \mapsto 1\}$

$$\forall x : \mathbb{N} \cdot x \geq 2 \lor 2 - 1 \leq x$$
$$= \forall x : \mathbb{N} \cdot x \geq 2 \lor 1 \leq x$$
$$= \text{false}, \text{ fails for } x = 0$$

Truthifying $\{y \mapsto 0, z \mapsto 0\}$

$$\forall x : \mathbb{N} \cdot x \geq 0 \lor 0 - 0 \leq x$$
$$= \forall x : \mathbb{N} \cdot x \geq 0 \lor 0 \leq x$$
$$= \text{true}, \text{ works for all } x$$

In fact the supplied predicate is equivalent to $y = 0 \lor y = z$.

(b) Prove the following law:

$$(\exists x \cdot P) \Rightarrow N \equiv (\forall x \mid P \cdot N) \quad x \notin N$$

You may use any laws in the reference at the end of this paper, or in the Handbook of Mathematics.

Strategy: transform rhs to lhs

$$(\exists x \cdot P) \Rightarrow N$$

$$= \text{“=} \Rightarrow\text{-def}$$
$$\neg(\exists x \cdot P) \lor N$$

$$= \text{“=} \neg\text{-dMorgan}$$
$$\neg\neg(\forall x \cdot \neg P) \lor N$$

$$= \text{“=} \neg\text{-invbl}$$
$$(\forall x \cdot \neg P) \lor N$$

$$= \text{“=} \lor\text{-distr, } x \notin N$$
$$\forall x \cdot \neg P \lor N$$

$$= \text{“=} \Rightarrow\text{-def}$$
$$\forall x \cdot P \Rightarrow N$$

$$= \text{“=} \forall\text{-trading}$$
$$(\forall x \mid P \cdot N)$$

(12 marks)
(c) **Prove the following laws of programming:**

\[
P; \text{skip} = P
\]

\[
x := e; \text{skip}; x := f = x := f[e/x]
\]

**using the first to help in proving the second.**

**Goal:** \(P; \text{skip} = P\)

**Strategy:** reduce lhs to rhs

\[
P; \text{skip}
\]

\[
= \quad " \langle \text{skip-def} \rangle "
\]

\[
P; \nu' = \nu
\]

\[
= \quad " \langle \nu \text{-def} \rangle "
\]

\[
\exists \nu_m \bullet P[\nu_m/\nu'] \land (\nu' = \nu)[\nu_m/\nu]
\]

\[
= \quad " \exists \text{-1pt}, \nu_m = \mu' "
\]

\[
P[\nu_m/\nu'][\nu_m/\nu']
\]

\[
= \quad " \langle \text{subst-inv}, \nu_m \not\in P \rangle "
\]

**Goal:** \(x := e; \text{skip}; x := f = x := f[e/x]\)

**Strategy:** reduce lhs to rhs

\[
x := e; \text{skip}; x := f
\]

\[
= \quad " \text{previous proof } "
\]

\[
x := e; x := f
\]

\[
= \quad " \langle ::= \text{-def} \rangle "
\]

\[
x' = e \land \nu' = \nu; x' = f \land \nu' = \nu
\]

\[
= \quad " \langle \nu \text{-def} \rangle "
\]

\[
\exists x_m, \nu_m \bullet (x' = e \land \nu' = \nu)[x_m, \nu_m/x', \nu] \land (x' = f \land \nu' = \nu)[x_m, \nu_m/x', \nu]
\]

\[
= \quad " \exists \text{-1pt}, x_m = e \land \nu_m = \nu "
\]

\[
x' = f[x_m, \nu_m/x, \nu][e, \nu/x_m, \nu_m] \land \nu' = \nu
\]

\[
= \quad " \langle \text{subst-comp} \rangle "
\]

\[
x' = f[e, \nu/x, \nu] \land \nu' = \nu
\]

\[
= \quad " \langle ::= \text{-def} \rangle "
\]

\[
x := f[e/x]
\]
A2. The sum 1..n function \( \text{sum}(n) \) satisfies the following laws:

\[
\begin{align*}
\text{sum}(0) &= 0 \\
\text{sum}(n) &= n + \text{sum}(n-1), \quad n > 0
\end{align*}
\]

We have the following specification (SSpec) and proposed program (SProg) and invarinat inv:

\[
\begin{align*}
\text{SSpec} & \cong s, x : \{s' = \text{sum}(n)\} \\
\text{SProg} & \cong s, x := 0, n; (x > 0) \ast (s, x := s + x, x - 1) \\
\text{inv} & \cong s + \text{sum}(x) = \text{sum}(n)
\end{align*}
\]

both with alphabet \( \{n, n', s, s', x, x'\} \). Assume all variables are integer, except \( x \) which is natural.

(a) **Prove:** \( \text{SSpec} \sqsubseteq s, x : [\text{inv} \land x' \leq 0] \)  

Strategy: reduce to true

\[
\begin{align*}
\text{SSpec} \sqsubseteq s, x : [\text{inv} \land x' \leq 0] \\
&= \text{ " defns. "} \\
&= s, x : [s' = \text{sum}(n)] \sqsubseteq s, x : [s' + \text{sum}(x') = \text{sum}(n') \land x' \leq 0] \\
&= \text{ " frame-def "} \\
&= s' = \text{sum}(n) \land n' = n \sqsubseteq s' + \text{sum}(x') = \text{sum}(n') \land x' \leq 0 \land n' = n \\
&= \text{ " } x' \leq 0 \text{ and } x : \mathbb{N} \text{ means } x' = 0 \text{ "} \\
&= s' = \text{sum}(n) \land n' = n \sqsubseteq s' + \text{sum}(x') = \text{sum}(n') \land x' = 0 \land n' = n \\
&= \text{ " } \sqsubseteq\text{-subst } \text{"} \\
&= s' = \text{sum}(n') \land n' = n' \sqsubseteq s' + 0 = \text{sum}(n') \land x' = 0 \land n' = n \\
&= \text{ " } (=\sqsubseteq\text{-refl}, \text{ defn. sum} \text{"} \\
&= s' = \text{sum}(n') \sqsubseteq s' = \text{sum}(n') \land x' = 0 \land n' = n \\
&= \text{ " } \sqsubseteq\text{-progr-strengthen } \text{"} \\
&= \text{ true }
\end{align*}
\]

(b) **Prove:** \( s, x : [\text{inv'}] \sqsubseteq s, x := 0, n \)  

Strategy: reduce to true

\[
\begin{align*}
\text{false} \sqsubseteq s, x : [\text{inv}] & \sqsubseteq s, x := 0, n \\
&= \text{ " defin. inv, frame-def, sim-=-def "} \\
&= s' + \text{sum}(x') = \text{sum}(n') \land n' = n \sqsubseteq s' = 0 \land x' = n \land n' = n \\
&= \text{ " } (=\sqsubseteq\text{-refl}, \text{ do substitution} \text{"} \\
&= 0 + \text{sum}(n) = \text{sum}(n) \land n' = n \sqsubseteq s' = 0 \land x' = n \land n' = n \\
&= \text{ " } \sqsubseteq\text{-refl} \text{ "} \\
&= \text{ true } \sqsubseteq s' = 0 \land x' = n \land n' = n \\
&= \text{ " } \sqsubseteq\text{-abort } \text{"} \\
&= \text{ true }
\end{align*}
\]
(c) **Prove:** \( s, x : [x > 0 \land \text{inv} \Rightarrow \text{inv}'] \sqsubseteq s, x := s + x, x - 1 \) 

Strategy: reduce to true

\[
s, x : [x > 0 \land \text{inv} \Rightarrow \text{inv}'] \sqsubseteq s, x := s + x, x - 1
= \quad \text{"defn. } \text{inv, } \text{\langle frame-def \rangle}, \text{ } \text{\langle sim::-=def \rangle "}
\]
\[
(x > 0 \land s + \text{sum}(x) = \text{sum}(n) \Rightarrow s' + \text{sum}(x') = \text{sum}(n')) \land n' = n
\]
\[
\sqsubseteq s' = s + x \land x' = x - 1 \land n' = n
= \quad \text{"} \text{\langle } \sqsubseteq \text{-subst} \text{\rangle}, \text{ do substitution } \text{"}
\]
\[
(x > 0 \land s + \text{sum}(x) = \text{sum}(n) \Rightarrow s + x + \text{sum}(x - 1) = \text{sum}(n)) \land n = n
\]
\[
\sqsubseteq s' = s + x \land x' = x - 1 \land n' = n
= \quad \text{"} \text{\langle } \sqsubseteq \text{-refl} \text{\rangle, law for sum } \text{"}
\]
\[
(x > 0 \land s + \text{sum}(x) = \text{sum}(n) \Rightarrow s + \text{sum}(x) = \text{sum}(n))
\]
\[
\sqsubseteq s' = s + x \land x' = x - 1 \land n' = n
= \quad \text{"} \text{\langle } \text{strengthen-ante} \text{\rangle } \text{"}
\]
\[
\text{true} \sqsubseteq s' = s + x \land x' = x - 1 \land n' = n
= \quad \text{"} \text{\langle } \sqsubseteq \text{-abort} \text{\rangle } \text{"}
\]
\[
\text{true}
\]

(d) *The above is a proof of partial correctness. What does this term mean exactly? What also needs to be done to prove total correctness?* (6 marks)

It means we have shown that the program gives the right answer, if it terminates. A partial correctness proof does not guarantee termination, but only correctness if the program ends.

A total correctness proof guarantees termination as well, as requires the specification of a variant - a natural number valued expression over program variables, that decreases each time around the loop.
A3.

(a) Why are the properties of transitivity and monotonicity important for refinement? (7 marks)

**Transitivity** \((S \sqsubseteq D) \land (D \sqsubseteq P) \Rightarrow (S \sqsubseteq P)\)

Transitivity allows us to refine a specification \(S\) into a program \(P\) by going through a number of intermediate steps (e.g. \(D\) above).

**Monotonicity** \((S \sqsubseteq P) \Rightarrow F(S) \sqsubseteq F(P)\)

Monotonicity of \(F\) allows a specification \(F(S)\) to be refined by refining part of it \((S \sqsubseteq P)\), with the overall result being obtained by replacing \(S\) in \(F\) by \(P\).

Both of these properties allow refinement to scale, by ensuring that we can do refinement by parts (monotonicity) in stages (transitivity).

(b) Which of the following predicate-transformer functions are monotonic? (8 marks)

\[
F(P) \equiv \exists x \bullet Q \land \neg (P \land \neg Q)
\]
\[
G(P) \equiv Q \lor \neg \exists y \bullet (R \land \neg P)
\]
\[
H(P) \equiv \neg P \land \neg (P \lor \neg R)
\]
\[
K(P) \equiv P \lor (P \equiv Q)
\]

**Justify your answer in each case**

- \(F(P)\) — non-monotonic: \(P\) inside single \(\neg\)
- \(G(P)\) — (anti-)monotonic: \(P\) inside two \textit{nots}
- \(H(P)\) — non-monotonic: both \(P\)'s inside a single \(\neg\).
- \(K(P)\) — not monotonic? as \(P\) occurs in mixed position (inside \(\equiv\))

However, let’s simplify:

\[
P \lor (P \equiv Q) \\
= P \lor (P \land Q) \lor (\neg P \land \neg Q) \\
= P \lor (\neg P \land \neg Q)
\]

Still not monotonic — \(P\) still mixed.

(c) In UTP, why do healthiness predicate-transformers have to be idempotent? (6 marks)

A healthiness predicate-transformer makes an un-healthy predicate into a healthy one. A predicate that is already healthy should not be changed further. We can test for healthiness by seeing if a predicate is so unchanged. If a healthiness predicate transformer was not idempotent then it would change healthy predicates, which is not what is wanted.
(d) Given the following definitions of $R_3$ and $CSP2$,

$$\begin{align*}
R_3(P) & \triangleq II \triangleleft \text{wait} \triangleright P \\
CSP2(P) & \triangleq P ; (ok \Rightarrow ok') \land wait' = wait \land tr' = tr \land ref' = ref
\end{align*}$$

and the fact that $II$ is $CSP2$-healthy, prove that they commute:

$$R_3 \circ CSP2 = CSP2 \circ R_3$$

First, introduce a shorthand:

$$J \triangleq (ok \Rightarrow ok') \land wait' = wait \land tr' = tr \land ref' = ref$$

Goal: $R_3(CSP2(P)) = CSP2(R_3(P))$

Strategy: reduce both to same

LHS:

$$\begin{align*}
R_3(CSP2(P)) & = \quad \text{" defn. CSP2 "} \\
R_3(P; J) & = \quad \text{" defn. R3 "} \\
II \triangleleft \text{wait} \triangleright (P; J)
\end{align*}$$

RHS:

$$\begin{align*}
CSP2(R_3(P)) & = \quad \text{" defn R3 "} \\
CSP2(II \triangleleft \text{wait} \triangleright P) & = \quad \text{" defn. CSP2 "} \\
(II \triangleleft \text{wait} \triangleright P); J & = \quad \text{" $\triangleleft$-seq"} \\
(II; J) \triangleleft \text{wait} \triangleright (P; J) & = \quad \text{" defn. CSP2 "} \\
CSP2(II) \triangleleft \text{wait} \triangleright (P; J) & = \quad \text{" Fact: II is CSP2 "} \\
II \triangleleft \text{wait} \triangleright (P; J)
\end{align*}$$
(a) **Prove the following sequence property:**

\[ s \neq \langle \rangle \Rightarrow (s - \text{front}(s)) = \langle \text{last}(s) \rangle \]

(*hint: induction on non-nil \(s\) should work*)

**Goal:** \( s \neq \langle \rangle \Rightarrow (s - \text{front}(s)) = \langle \text{last}(s) \rangle \)

**Strategy:** Assume \( s \neq \langle \rangle \), to show \((s - \text{front}(s)) = \langle \text{last}(s) \rangle\) using non-nil induction on \(s\).

**Base Case:** show true for \( s = \langle x \rangle \)

**Sub-Strategy:** reduce to true

\[
\langle x \rangle - \langle \rangle = \langle x \rangle = \langle \rangle = \langle \text{def-nil} \rangle = \langle x \rangle = \langle x \rangle = \langle \text{def-refl} \rangle = \text{true}
\]

**Inductive Case:** assume true for \( s \), show true for \( x \# s \)

**Sub-Strategy:** assume antecedent, reduce consequent to true:

\[
\langle x \# s \rangle - \text{front}(x \# s) = \langle \text{last}(x \# s) \rangle = \langle \text{def-cons} \rangle, \langle \text{last-def-cons} \rangle = \langle x \rangle - x \# \text{front}(s) = \langle \text{last}(s) \rangle = \langle \text{def-cons} \rangle = (s - \text{front}(s)) = \langle \text{last}(s) \rangle = \text{inductive hypothesis} = \text{true}
\]
(b) Sequence subtraction \((s - t)\) is partial, only being defined when its second argument is a prefix of its first \((t \leq s)\). Given the following definition of \(R_2\),

\[
R_2(P) \equiv \exists s \cdot P[s, s \leftarrow (tr' - tr)/tr, tr']
\]

prove that it is idempotent (i.e., \(R_2 \circ R_2 = R_2\)), and highlight how the partiality of sequence subtraction is considered in the proof. 

**Goal:** \(R_2(R_2(P)) = R_2(P)\)

**Strategy:** reduce lhs to rhs

\[
R_2(R_2(P))
\]

\[
= \text{ "defn. } R_2 \text{ "}
\]

\[
R_2(\exists s \cdot P[s, s \leftarrow (tr' - tr)/tr, tr'])
\]

\[
= \text{ "defn. } R_2 \text{, using different quantifier variable (t rather than s) "}
\]

\[
\exists t \cdot (\exists s \cdot P[s, s \leftarrow (tr' - tr)/tr, tr'])[t, t \leftarrow (tr' - tr)/tr, tr']
\]

\[
= \text{ "merge quantifiers, } \langle \langle \text{subst-comp} \rangle \rangle \text{ "}
\]

\[
\exists t, s \cdot P[s, s \leftarrow ((t \leftarrow (tr' - tr)) - t)/tr, tr']
\]

\[
= \text{ " } \langle \langle \text{pfx-\leftarrow-\dashwedge-\neg-inv} \rangle \rangle \text{, defd. information not lost (see below) "}
\]

\[
\exists t, s \cdot P[s, s \leftarrow (tr' - tr)/tr, tr']
\]

\[
= \text{ " } t \text{ not mentioned } \text{ "}
\]

\[
\exists s \cdot P[s, s \leftarrow (tr' - tr)/tr, tr']
\]

\[
= \text{ " defn. } R_2 \text{ "}
\]

\[
R_2(P)
\]

Before applying \(\langle \langle \text{pfx-\leftarrow-\dashwedge-\neg-inv} \rangle \rangle\) we have the following definedness information:

- \(tr' - tr\), requires that \(tr \leq tr'\)
- \(t \leftarrow (tr' - tr)\) - \(t\), requires that \(t \leq t \leftarrow (tr' - tr)\)
  
  this is true given the previous requirements, so we can ignore it.

After applying \(\langle \langle \text{pfx-\leftarrow-\dashwedge-\neg-inv} \rangle \rangle\) we have the following definedness information:

- \(tr' - tr\), requires that \(tr \leq tr'\)

There has been no change, so we don’t need to explicitly add in \(tr \leq tr'\) as a side-condition.