A Fully Compositional and Complete Program Logic for While

Reiner Hähnle
Joint work with Dilian Gurov
Department of Computer Science
Technische Universität Darmstadt

Part I

A While Language and its Semantics

While: A Standard Imperative Language

TECHNISCHE UNIVERSITÄT DARMSTADT

$$S \;::=\; \textbf{skip} \;\mid\; x := a \;\mid\; S_1; S_2 \;\mid\; \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \;\mid\; \textbf{while}\ b\ \textbf{do}\ S$$

S statement, $x \in \mathit{Var}$ variable, a/b arithmetic/boolean expression

While: A Standard Imperative Language With its Standard SOS Semantics

$$S \;::=\; \textbf{skip} \;\mid\; x := a \;\mid\; S_1; S_2 \;\mid\; \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \;\mid\; \textbf{while}\ b\ \textbf{do}\ S$$

$$\textsc{Skip}\quad \frac{-}{\langle \textbf{skip}, s\rangle \Rightarrow s} \qquad\qquad \textsc{Assign}\quad \frac{-}{\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]\,(s)]}$$

$s : \mathit{Var} \rightarrow \mathbb{Z}$ state

While: A Standard Imperative Language With its Standard SOS Semantics

$$S \;::=\; \textbf{skip} \;\mid\; x := a \;\mid\; S_1; S_2 \;\mid\; \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \;\mid\; \textbf{while}\ b\ \textbf{do}\ S$$

$$\textsc{Skip}\quad \frac{-}{\langle \textbf{skip}, s\rangle \Rightarrow s} \qquad\qquad \textsc{Assign}\quad \frac{-}{\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!](s)]}$$

$$\textsc{Seq-1}\quad \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle} \qquad\qquad \textsc{Seq-2}\quad \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$$

While: A Standard Imperative Language With its Standard SOS Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

$$S \;::=\; \textbf{skip} \;\mid\; x := a \;\mid\; S_1 ; S_2 \;\mid\; \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \;\mid\; \textbf{while}\ b\ \textbf{do}\ S$$

$$\textsc{Skip}\quad \frac{-}{\langle \textbf{skip}, s\rangle \Rightarrow s} \qquad\qquad \textsc{Assign}\quad \frac{-}{\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!](s)]}$$

$$\textsc{Seq-1}\quad \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle} \qquad\qquad \textsc{Seq-2}\quad \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_1' ; S_2, s'\rangle}$$

$$\textsc{If-1}\quad \frac{-}{\langle \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle} \quad \text{if } \mathcal{B}[\![b]\!](s) = \textbf{tt}$$

While: A Standard Imperative Language With its Standard SOS Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

$$S \;::=\; \textbf{skip} \mid x := a \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \mid \textbf{while } b \textbf{ do } S$$

SKIP $$\frac{-}{\langle \textbf{skip}, s\rangle \Rightarrow s}$$

ASSIGN $$\frac{-}{\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!](s)]}$$

SEQ-1 $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

SEQ-2 $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$$

IF-1 $$\frac{-}{\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle} \quad \text{if } \mathcal{B}[\![b]\!](s) = \textbf{tt}$$

WHILE-1 $$\frac{-}{\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } b \textbf{ do } S, s\rangle} \quad \text{if } \mathcal{B}[\![b]\!](s) = \textbf{tt}$$

WHILE-2 $$\frac{-}{\langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow s} \quad \text{if } \mathcal{B}[\![b]\!](s) = \textbf{ff}$$

Induced SOS Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Example (SOS Derivation)

$$\langle \textbf{skip}; x := x - 1, s\rangle \quad \Rightarrow \quad \langle x := x - 1, s\rangle \quad \Rightarrow \quad s[x \mapsto s(x) - 1]$$

Induced SOS Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Example (SOS Derivation)

$$\langle \textbf{skip}; x := x - 1, s\rangle \quad \Rightarrow \quad \langle x := x - 1, s\rangle \quad \Rightarrow \quad s[x \mapsto s(x) - 1]$$

Definition (Induced Finite-Trace Semantics)

$\mathcal{S}_{sos}[\![S]\!]$ is the set of finite sequences $s_0 \cdot s_1 \cdot \ldots \cdot s_n$ of states for which there are statements $S_0, S_1, \ldots, S_{n-1}$ such that $S_0 = S$, $\langle S_i, s_i\rangle \Rightarrow \langle S_{i+1}, s_{i+1}\rangle$ for all $0 \leq i \leq n - 2$, and $\langle S_{n-1}, s_{n-1}\rangle \Rightarrow s_n$.

Induced SOS Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Example (SOS Derivation)

$$\langle \textbf{skip}; x := x - 1, s\rangle \quad\Rightarrow\quad \langle x := x - 1, s\rangle \quad\Rightarrow\quad s[x \mapsto s(x) - 1]$$

Definition (Induced Finite-Trace Semantics)

$\mathcal{S}_{sos}[\![S]\!]$ is the set of finite sequences $s_0 \cdot s_1 \cdot \ldots \cdot s_n$ of states for which there are statements $S_0, S_1, \ldots, S_{n-1}$ such that $S_0 = S$, $\langle S_i, s_i\rangle \Rightarrow \langle S_{i+1}, s_{i+1}\rangle$ for all $0 \leq i \leq n - 2$, and $\langle S_{n-1}, s_{n-1}\rangle \Rightarrow s_n$.

Example (Induced Trace)

$$\mathcal{S}_{sos}[\![\textbf{skip}; x := x - 1]\!] \quad=\quad \{s \cdot s \cdot s[x \mapsto s(x) - 1] \mid s \in \textbf{State}\}$$

Part II

A Denotational and Compositional Trace Semantics

A Denotational Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Compose traces directly from statement without executing them

A Denotational Finite-Trace Semantics

Compose traces directly from statement without executing them

Auxiliary definitions to describe sets of traces $\sigma \in A$:

$$A|_b \stackrel{\text{def}}{=} \{s \cdot \sigma \in A \mid \mathcal{B}[\![b]\!](s) = \mathbf{tt}\}$$

$$\sharp A \stackrel{\text{def}}{=} \{s \cdot s \cdot \sigma \mid s \cdot \sigma \in A\}$$

$$A \frown B \stackrel{\text{def}}{=} \{\sigma_A \cdot s \cdot \sigma_B \mid \sigma_A \cdot s \in A \ \wedge \ s \cdot \sigma_B \in B\}$$

A Denotational Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Compose traces directly from statement without executing them

Auxiliary definitions to describe sets of traces $\sigma \in A$:

$$
\begin{aligned}
A|_b &\stackrel{\text{def}}{=} \{s \cdot \sigma \in A \mid \mathcal{B}[\![b]\!](s) = \mathbf{tt}\} \\
\sharp A &\stackrel{\text{def}}{=} \{s \cdot s \cdot \sigma \mid s \cdot \sigma \in A\} \\
A \frown B &\stackrel{\text{def}}{=} \{\sigma_A \cdot s \cdot \sigma_B \mid \sigma_A \cdot s \in A \ \wedge \ s \cdot \sigma_B \in B\}
\end{aligned}
$$

$$\mathcal{S}_{tr}[\![\mathbf{skip}]\!] \stackrel{\text{def}}{=} \{s \cdot s \mid s \in \mathbf{State}\}$$

A Denotational Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Compose traces directly from statement without executing them

Auxiliary definitions to describe sets of traces $\sigma \in A$:

$$
\begin{aligned}
A|_b &\stackrel{\text{def}}{=} \{s \cdot \sigma \in A \mid \mathcal{B}[\![b]\!](s) = \mathbf{tt}\} \\
\sharp A &\stackrel{\text{def}}{=} \{s \cdot s \cdot \sigma \mid s \cdot \sigma \in A\} \\
A \frown B &\stackrel{\text{def}}{=} \{\sigma_A \cdot s \cdot \sigma_B \mid \sigma_A \cdot s \in A \ \wedge \ s \cdot \sigma_B \in B\}
\end{aligned}
$$

$$
\mathcal{S}_{tr}[\![x := a]\!] \stackrel{\text{def}}{=} \{s \cdot s[x \mapsto \mathcal{A}[\![a]\!](s)] \mid s \in \mathbf{State}\}
$$

A Denotational Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Compose traces directly from statement without executing them

Auxiliary definitions to describe sets of traces $\sigma \in A$:

$$
\begin{aligned}
A|_b &\stackrel{\text{def}}{=} \{s \cdot \sigma \in A \mid \mathcal{B}[\![b]\!](s) = \mathbf{tt}\} \\
\sharp A &\stackrel{\text{def}}{=} \{s \cdot s \cdot \sigma \mid s \cdot \sigma \in A\} \\
A^\frown B &\stackrel{\text{def}}{=} \{\sigma_A \cdot s \cdot \sigma_B \mid \sigma_A \cdot s \in A \ \wedge \ s \cdot \sigma_B \in B\}
\end{aligned}
$$

$$\mathcal{S}_{tr}[\![S_1; S_2]\!] \stackrel{\text{def}}{=} \mathcal{S}_{tr}[\![S_1]\!]^\frown \mathcal{S}_{tr}[\![S_2]\!]$$

A Denotational Finite-Trace Semantics

Compose traces directly from statement without executing them

Auxiliary definitions to describe sets of traces $\sigma \in A$:

$$A|_b \stackrel{\text{def}}{=} \{s \cdot \sigma \in A \mid \mathcal{B}[\![b]\!](s) = \mathbf{tt}\}$$
$$\sharp A \stackrel{\text{def}}{=} \{s \cdot s \cdot \sigma \mid s \cdot \sigma \in A\}$$
$$A \frown B \stackrel{\text{def}}{=} \{\sigma_A \cdot s \cdot \sigma_B \mid \sigma_A \cdot s \in A \ \wedge \ s \cdot \sigma_B \in B\}$$

$$\mathcal{S}_{tr}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!] \stackrel{\text{def}}{=} (\sharp\mathcal{S}_{tr}[\![S_1]\!])|_b \ \cup \ (\sharp\mathcal{S}_{tr}[\![S_2]\!])|_{\neg b}$$

A Denotational Finite-Trace Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Compose traces directly from statement without executing them

Auxiliary definitions to describe sets of traces $\sigma \in A$:

$$
\begin{aligned}
A|_b &\stackrel{\text{def}}{=} \{s \cdot \sigma \in A \mid \mathcal{B}[\![b]\!](s) = \mathbf{tt}\} \\
\sharp A &\stackrel{\text{def}}{=} \{s \cdot s \cdot \sigma \mid s \cdot \sigma \in A\} \\
A \frown B &\stackrel{\text{def}}{=} \{\sigma_A \cdot s \cdot \sigma_B \mid \sigma_A \cdot s \in A \ \wedge \ s \cdot \sigma_B \in B\}
\end{aligned}
$$

$$\mathcal{S}_{tr}[\![\textbf{while}\ b\ \textbf{do}\ S]\!] \stackrel{\text{def}}{=} LFP\ H_{b,S}$$

$$H_{b,S}(\gamma) \stackrel{\text{def}}{=} (\sharp\mathcal{S}_{tr}[\![S]\!])|_b \frown \gamma \ \cup \ \{s \cdot s \mid \mathcal{B}[\![b]\!](s) = \mathbf{ff}\}$$

Equivalence of the Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem (Correctness)

For all statements S of **While***, we have:*

$$\mathcal{S}_{tr}[\![S]\!] = \mathcal{S}_{sos}[\![S]\!]$$

Equivalence of the Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem (Correctness)

For all statements S of **While***, we have:*

$$\mathcal{S}_{tr}[\![S]\!] \;=\; \mathcal{S}_{sos}[\![S]\!]$$

We can work with $\mathcal{S}_{tr}[\![S]\!]$ from now on!

But why would we do this?

- Next we define a logic to specify sets of traces
- Semantics of trace logic defined in denotational style: good match

Part III

A Logic to Specify Finite Traces

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi \; ::= \; p \;\mid\; R \;\mid\; X \;\mid\; \phi_1 \wedge \phi_2 \;\mid\; \phi_1 \vee \phi_2 \;\mid\; \phi_1 \frown \phi_2 \;\mid\; \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states,
  $X \in$ RVar recursion variable to be used in scope of smallest FP μX

A Logic over Finite Traces

$$\phi \; ::= \; p \mid R \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \frown \phi_2 \mid \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states, $X \in$ RVar recursion variable to be used in scope of smallest FP μX

Example (Express Transitive Closure of Binary Relation R)

$$R^+ \stackrel{\text{def}}{\iff} \mu X.\,(R \vee R \frown X)$$

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi \ ::= \ p \ \mid \ R \ \mid \ X \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \phi_1 \vee \phi_2 \ \mid \ \phi_1 \frown \phi_2 \ \mid \ \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states, $X \in$ RVar recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \overset{\text{def}}{\iff} s' = s$ and $Sb^a_x(s, s') \overset{\text{def}}{\iff} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi \ ::= \ p \mid R \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \frown \phi_2 \mid \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states,
  $X \in \mathrm{RVar}$ recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \overset{\text{def}}{\iff} s' = s$ and $Sb_x^a(s, s') \overset{\text{def}}{\iff} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$
- Semantics defined relative to valuation $\mathcal{V} : \mathrm{RVar} \to 2^{\textbf{State}^+}$ of $X \in \mathrm{RVar}$

Inductive definition of $\|\phi\|_{\mathcal{V}}$

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi ::= p \mid R \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \frown \phi_2 \mid \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states, $X \in \mathrm{RVar}$ recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \overset{\text{def}}{\Longleftrightarrow} s' = s$ and $Sb_x^a(s, s') \overset{\text{def}}{\Longleftrightarrow} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$
- Semantics defined relative to valuation $\mathcal{V} : \mathrm{RVar} \to 2^{\textbf{State}^+}$ of $X \in \mathrm{RVar}$

Inductive definition of $\|\phi\|_{\mathcal{V}}$

$$\|p\|_{\mathcal{V}} \overset{\text{def}}{=} \{s \cdot \sigma \mid s \models p\} = \textbf{State}^+|_p$$

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi ::= p \mid R \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 {}^\frown \phi_2 \mid \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states,
  $X \in \mathrm{RVar}$ recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \stackrel{\text{def}}{\iff} s' = s$ and $Sb_x^a(s, s') \stackrel{\text{def}}{\iff} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$
- Semantics defined relative to valuation $\mathcal{V} : \mathrm{RVar} \to 2^{\textbf{State}^+}$ of $X \in \mathrm{RVar}$

Inductive definition of $\|\phi\|_{\mathcal{V}}$

$$\|R\|_{\mathcal{V}} \stackrel{\text{def}}{=} \{s \cdot s' \mid R(s, s')\}$$

A Logic over Finite Traces

$$\phi \ ::= \ p \ | \ R \ | \ X \ | \ \phi_1 \wedge \phi_2 \ | \ \phi_1 \vee \phi_2 \ | \ \phi_1 \frown \phi_2 \ | \ \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states, $X \in \mathrm{RVar}$ recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \overset{\text{def}}{\Longleftrightarrow} s' = s$ and $Sb_x^a(s, s') \overset{\text{def}}{\Longleftrightarrow} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$
- Semantics defined relative to valuation $\mathcal{V} : \mathrm{RVar} \to 2^{\mathbf{State}^+}$ of $X \in \mathrm{RVar}$

Inductive definition of $\|\phi\|_{\mathcal{V}}$

$$\|X\|_{\mathcal{V}} \overset{\text{def}}{=} \mathcal{V}(X)$$

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi ::= p \mid R \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \frown \phi_2 \mid \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states, $X \in$ RVar recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \overset{\text{def}}{\Longleftrightarrow} s' = s$ and $Sb_x^a(s, s') \overset{\text{def}}{\Longleftrightarrow} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$
- Semantics defined relative to valuation $\mathcal{V} : \text{RVar} \to 2^{\textbf{State}^+}$ of $X \in$ RVar

Inductive definition of $\|\phi\|_{\mathcal{V}}$

$$\|\phi_1 \wedge \phi_2\|_{\mathcal{V}} \overset{\text{def}}{=} \|\phi_1\|_{\mathcal{V}} \cap \|\phi_2\|_{\mathcal{V}}\,, \text{ etc.}$$

A Logic over Finite Traces

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\phi ::= p \mid R \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \frown \phi_2 \mid \mu X.\phi$$

- Semantics of ϕ is a set of finite traces
- p state formula such as **BExp**, R binary relation over states, $X \in \mathrm{RVar}$ recursion variable to be used in scope of smallest FP μX
- R: $Id(s, s') \overset{\text{def}}{\iff} s' = s$ and $Sb_x^a(s, s') \overset{\text{def}}{\iff} s' = s[x \mapsto \mathcal{A}[\![a]\!](s)]$
- Semantics defined relative to valuation $\mathcal{V} : \mathrm{RVar} \to 2^{\textbf{State}^+}$ of $X \in \mathrm{RVar}$

Inductive definition of $\|\phi\|_{\mathcal{V}}$

$$\|\mu X.\phi\|_{\mathcal{V}} \overset{\text{def}}{=} \bigcap \left\{ \gamma \subseteq \textbf{State}^+ \;\middle|\; \|\phi\|_{\mathcal{V}[X \mapsto \gamma]} \subseteq \gamma \right\}$$

Least pre-fixed point of $\|\phi\|_{\mathcal{V}[X \mapsto \gamma]}$ using Knaster-Tarski
(Omit $\mathcal{V}$ when ϕ closed)

Expressiveness of the Trace Logic

Trace logic expressive enough to characterize any **While** program

Expressiveness of the Trace Logic

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathrm{stf}(S)$:

Expressiveness of the Trace Logic

TECHNISCHE UNIVERSITÄT DARMSTADT

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathrm{stf}(S)$:

$$\mathrm{stf}(\mathbf{skip}) \stackrel{\text{def}}{=} Id$$

Expressiveness of the Trace Logic

TECHNISCHE UNIVERSITÄT DARMSTADT

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathrm{stf}(S)$:

$$\mathrm{stf}(x := a) \stackrel{\mathrm{def}}{=} Sb_x^a$$

Expressiveness of the Trace Logic

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathrm{stf}(S)$:

$$\mathrm{stf}(S_1;\, S_2) \stackrel{\text{def}}{=} \mathrm{stf}(S_1)^\frown \mathrm{stf}(S_2)$$

Expressiveness of the Trace Logic

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathrm{stf}(S)$:

$$\mathrm{stf}(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2) \stackrel{\text{def}}{=} (b \wedge \mathit{Id}^\frown \mathrm{stf}(S_1)) \vee (\neg b \wedge \mathit{Id}^\frown \mathrm{stf}(S_2))$$

Expressiveness of the Trace Logic

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathrm{stf}(S)$:

$$\mathrm{stf}(\textbf{while}\ b\ \textbf{do}\ S) \stackrel{\text{def}}{=} \mu X.\,\big((\neg b \wedge Id) \vee (b \wedge Id^\frown \mathrm{stf}(S)^\frown X)\big)$$

Expressiveness of the Trace Logic

TECHNISCHE UNIVERSITÄT DARMSTADT

Trace logic expressive enough to characterize any **While** program

Definition

For any program S define strongest trace formula $\mathsf{stf}(S)$:

$$\mathsf{stf}(\textbf{while } b \textbf{ do } S) \stackrel{\text{def}}{=} \mu X.\, ((\neg b \wedge Id) \vee (b \wedge Id ^\frown \mathsf{stf}(S) ^\frown X))$$

Theorem

Let S be a statement. Then the following holds:

$$\|\mathsf{stf}(S)\| = \mathcal{S}_{tr}[\![S]\!]$$

Part IV

A Calculus to Prove a Program is Valid for a Trace Formula

Statement Variables and Judgments

TECHNISCHE UNIVERSITÄT DARMSTADT

Statement Variable

Extend the syntax of **While** by new kind of atomic statement:
A statement variable $Y \in \mathrm{SVar}$ represents an arbitrary **While** statement

Semantics of programs $\mathcal{S}_{tr}[\![S]\!]_{\mathcal{I}}$ expressed relative to interpretation
$\mathcal{I} : \mathrm{SVar} \to 2^{\mathbf{State}^{+}}$

Statement Variable

Extend the syntax of **While** by new kind of atomic statement:
A statement variable $Y \in \mathrm{SVar}$ represents an arbitrary **While** statement

Semantics of programs $\mathcal{S}_{tr}[\![S]\!]_{\mathcal{I}}$ expressed relative to interpretation
$\mathcal{I} : \mathrm{SVar} \to 2^{\mathbf{State}^{+}}$

Definition (Judgment)

A judgment is of the form $S : \phi$, where S is a **While** statement, possibly containing statement variables, and ϕ a closed trace formula.

Definition (Semantics of Judgment)

A judgment $S : \phi$ is valid in $\mathcal{I}$, denoted $\models_{\mathcal{I}} S : \phi$, when $\mathcal{S}_{tr}[\![S]\!]_{\mathcal{I}} \subseteq \|\phi\|$.

Sequents

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition (Sequent)

A sequent has the form $\Gamma \vdash S : \phi$, where Γ is a possibly empty set of judgments.

Definition (Calculus Semantics)

A sequent $\Gamma \vdash S : \phi$ is valid, denoted $\Gamma \models S : \phi$, if for every interpretation $\mathcal{I}$, $S : \phi$ is valid in $\mathcal{I}$ whenever all judgments in Γ are valid in $\mathcal{I}$.

Sequent Calculus

$$\textsc{Skip}\quad \frac{-}{\Gamma \vdash \textbf{skip} : \mathit{Id}} \qquad\qquad \textsc{Assign}\quad \frac{-}{\Gamma \vdash x := a : Sb_x^a}$$

Sequent Calculus

$$\textsc{Skip} \quad \frac{-}{\Gamma \vdash \textbf{skip} : Id} \qquad \textsc{Assign} \quad \frac{-}{\Gamma \vdash x := a : Sb_x^a}$$

$$\textsc{Seq} \quad \frac{\Gamma \vdash S_1 : \phi_1 \qquad \Gamma \vdash S_2 : \phi_2}{\Gamma \vdash S_1; S_2 : \phi_1 \frown \phi_2}$$

Sequent Calculus

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\textsc{Skip}\quad \frac{-}{\Gamma \vdash \textbf{skip} : \mathit{Id}} \qquad \textsc{Assign}\quad \frac{-}{\Gamma \vdash x := a : Sb_x^a}$$

$$\textsc{Seq}\quad \frac{\Gamma \vdash S_1 : \phi_1 \qquad \Gamma \vdash S_2 : \phi_2}{\Gamma \vdash S_1; S_2 : \phi_1 \frown \phi_2}$$

$$\textsc{If}\quad \frac{\Gamma \vdash \textbf{skip}; S_1 : \neg b \vee \phi \qquad \Gamma \vdash \textbf{skip}; S_2 : b \vee \phi}{\Gamma \vdash \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 : \phi}$$

$\neg b \vee \phi$ trivially true when $\neg b$, only b-case must be checked

Sequent Calculus

TECHNISCHE UNIVERSITÄT DARMSTADT

SKIP $$\frac{-}{\Gamma \vdash \textbf{skip} : Id}$$

ASSIGN $$\frac{-}{\Gamma \vdash x := a : Sb_x^a}$$

SEQ $$\frac{\Gamma \vdash S_1 : \phi_1 \qquad \Gamma \vdash S_2 : \phi_2}{\Gamma \vdash S_1; S_2 : \phi_1 \frown \phi_2}$$

IF $$\frac{\Gamma \vdash \textbf{skip}; S_1 : \neg b \vee \phi \qquad \Gamma \vdash \textbf{skip}; S_2 : b \vee \phi}{\Gamma \vdash \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 : \phi}$$

WHILE $$\frac{\Gamma \vdash \textbf{skip} : b \vee \phi \qquad \Gamma, Y : \phi \vdash \textbf{skip}; S; Y : \neg b \vee \phi}{\Gamma \vdash \textbf{while}\ b\ \textbf{do}\ S : \phi}$$

Read Y as an arbitrary continuation of rule body

Sequent Calculus

TECHNISCHE UNIVERSITÄT DARMSTADT

$$\textsc{Skip}\quad \frac{-}{\Gamma \vdash \textbf{skip} : Id} \qquad \textsc{Assign}\quad \frac{-}{\Gamma \vdash x := a : Sb_x^a}$$

$$\textsc{Seq}\quad \frac{\Gamma \vdash S_1 : \phi_1 \qquad \Gamma \vdash S_2 : \phi_2}{\Gamma \vdash S_1; S_2 : \phi_1 \frown \phi_2}$$

$$\textsc{If}\quad \frac{\Gamma \vdash \textbf{skip}; S_1 : \neg b \vee \phi \qquad \Gamma \vdash \textbf{skip}; S_2 : b \vee \phi}{\Gamma \vdash \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 : \phi}$$

$$\textsc{While}\quad \frac{\Gamma \vdash \textbf{skip} : b \vee \phi \qquad \Gamma, Y : \phi \vdash \textbf{skip}; S; Y : \neg b \vee \phi}{\Gamma \vdash \textbf{while}\ b\ \textbf{do}\ S : \phi}$$

Read Y as an arbitrary continuation of rule body

Rules for Fixed-point unfolding and weakening not shown

Soundness & Completeness

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem (Soundness)

Every derivable sequent is valid.

Soundness & Completeness

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem (Soundness)

Every derivable sequent is valid.

Theorem (Relative Completeness)

Every valid sequent is derivable with an oracle for $\phi \models \psi$.

Proof Sketch.

$\vdash S : \mathrm{stf}(S)$ is derivable by structural induction on S.

For any valid judgment $S : \phi$, formula ϕ must imply $\mathrm{stf}(S)$.
Use weakening. □

Part V

Closing

Highlights

TECHNISCHE UNIVERSITÄT DARMSTADT

- Semantics of programs, formulas, and calculus is fully compositional: no context information needed
- Invariant rule, to best of our knowledge, is new and uses symbolic continuations
  - Modelled after fixed-point definition in semantics
  - Like abstract execution, uses abstract programs
- Trace logic is sufficiently expressive to characterize any program: Strongest trace formula leads to direct completeness proof

Future / Ongoing Work

From **While** to **Rec**

Language **Rec** obtained by replacing loops with parameterless, void procedure calls $m()$ (only global variables)

Future / Ongoing Work

TECHNISCHE UNIVERSITÄT DARMSTADT

From **While** to **Rec**

Language **Rec** obtained by replacing loops with parameterless, void procedure calls $m()$ (only global variables)

- We are confident that previous results generalize to **Rec**
  - The trace logic stays unchanged
  - Most complex proof cases completed
  - Two calculus rules needed to handle calls (first call / recursive call)

Future / Ongoing Work

TECHNISCHE UNIVERSITÄT DARMSTADT

From **While** to **Rec**

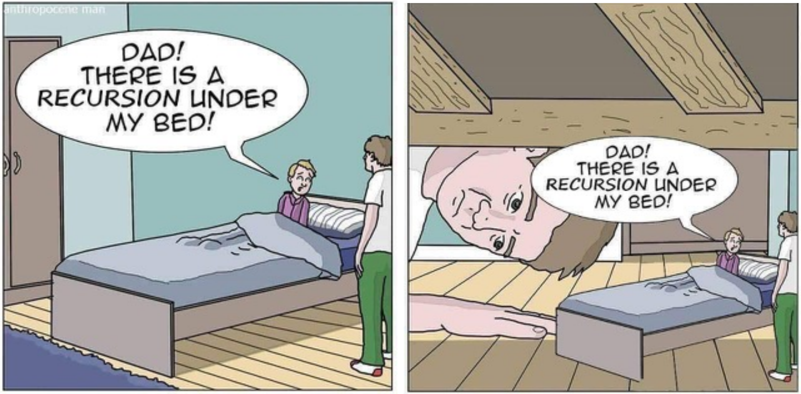
Language **Rec** obtained by replacing loops with parameterless, void procedure calls *m*() (only global variables)

- We are confident that previous results generalize to **Rec**
  - The trace logic stays unchanged
  - Most complex proof cases completed
  - Two calculus rules needed to handle calls (first call / recursive call)

Conjecture

Every trace formula can be translated into a **Rec** *program that has exactly the same semantics (modulo stuttering):*
Our trace logic is "the logic of programs with recursive procedures"

Thank you!


anthropocene man
DAD! THERE IS A RECURSION UNDER MY BED!
DAD! THERE IS A RECURSION UNDER MY BED!