Completeness and Complexity of Reasoning about Call-by-Value in Hoare Logic

Hans-Dieter Hiep
joint work with: Krzysztof Apt, Frank de Boer

LIACS Theory group, June 9th, 2022

Program verification

> *"How can one check a routine in the sense of making sure that it is right?"*
>
> — *Alan Turing (1949)*

Background

Program verification

> *"How can one check a routine in the sense of making sure that it is right?"*
>
> *— Alan Turing (1949)*

Motivation

1950s —

> *If you don't have a working machine yet, better make sure your program is right!*

Background

Program verification

> *"How can one check a routine in the sense of making sure that it is right?"*
>
> *— Alan Turing (1949)*

Motivation

1950s —

> *If you don't have a working machine yet, better make sure your program is right!*

2000s —

> *Programming errors lead to computer insecurities.*

Background

Background material:

⇛ Origin of Recursive Procedures, G. van den Hove (2015)

⇛ Fifty years of Hoare's logic, K. Apt, E. Olderog (2019)

Basic Programs

Basic Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$
\begin{aligned}
S ::= \; & \bar{x} := \bar{t} \\
\mid \; & S;\ S \\
\mid \; & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\
\mid \; & \textbf{while } B \textbf{ do } S \textbf{ od}
\end{aligned}
$$

Basic Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$
\begin{aligned}
S ::= \; & \bar{x} := \bar{t} \\
\mid \; & S;\ S \\
\mid \; & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\
\mid \; & \textbf{while } B \textbf{ do } S \textbf{ od}
\end{aligned}
$$

- $\bar{x} := \bar{t}$: parallel assignment

Basic Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$\begin{aligned} S ::= \; & \bar{x} := \bar{t} \\ & \mid S;\ S \\ & \mid \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\ & \mid \textbf{while } B \textbf{ do } S \textbf{ od} \end{aligned}$$

- $\bar{x} := \bar{t}$: parallel assignment
- $S;\ S$: sequential composition

Basic Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$\begin{aligned} S ::= {} & \bar{x} := \bar{t} \\ & \mid S;\ S \\ & \mid \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\ & \mid \textbf{while } B \textbf{ do } S \textbf{ od} \end{aligned}$$

- $\bar{x} := \bar{t}$: parallel assignment
- $S;\ S$: sequential composition
- **if**: branching

Basic Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$\begin{aligned} S ::= \; & \bar{x} := \bar{t} \\ & \mid S;\ S \\ & \mid \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\ & \mid \textbf{while } B \textbf{ do } S \textbf{ od} \end{aligned}$$

- $\bar{x} := \bar{t}$: parallel assignment
- $S;\ S$: sequential composition
- **if**: branching
- **while**: looping

Basic Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

Basic Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ S\ \{q\}$: derivability

Basic Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ S\ \{q\}$: derivability
- $\models \{p\}\ S\ \{q\}$: truth

Basic Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ S\ \{q\}$: derivability
- $\models \{p\}\ S\ \{q\}$: truth

Claim

- Soundness: $\vdash \{p\}\ S\ \{q\}$ implies $\models \{p\}\ S\ \{q\}$

Basic Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ S\ \{q\}$: derivability
- $\models \{p\}\ S\ \{q\}$: truth

Claim

- Soundness: $\vdash \{p\}\ S\ \{q\}$ implies $\models \{p\}\ S\ \{q\}$
- Completeness*: $\models \{p\}\ S\ \{q\}$ implies $\vdash \{p\}\ S\ \{q\}$

Basic Program Logic

Definition (1/2: analytical rules and axiom)

PARALLEL ASSIGNMENT

$$\frac{}{\{p[\bar{x} := \bar{t}]\}\ \bar{x} := \bar{t}\ \{p\}}$$

COMPOSITION

$$\frac{\{p\}\ S_1\ \{r\} \quad \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$$

CONDITIONAL

$$\frac{\{p \wedge B\}\ S_1\ \{q\} \quad \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$$

WHILE

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$$

Definition (2/2: adaptation rule)

CONSEQUENCE (adaptation)

$$\frac{\models p \to p' \quad \{p'\}\, S\, \{q'\} \quad \models q' \to q}{\{p\}\, S\, \{q\}}$$

Definition (2/2: adaptation rule)

CONSEQUENCE (adaptation)

$$\frac{\models p \to p' \quad \{p'\}\ S\ \{q'\} \quad \models q' \to q}{\{p\}\ S\ \{q\}}$$

Now, $\vdash \{p\}\ S\ \{q\}$ means

there exists a proof with $\{p\}\ S\ \{q\}$ as conclusion.

Basic Proof Outlines

Different notations for proofs in first-order logic:

1. Proof trees
2. Hilbert-style
3. Natural deduction
4. Gentzen sequent calculus

Basic Proof Outlines

Different notations for proofs in first-order logic:

1. Proof trees
2. Hilbert-style
3. Natural deduction
4. Gentzen sequent calculus

Different notations for proofs in program logic:

1. Proof trees
2. Hilbert-style
3. Proof outlines

Basic Proof Outlines

$\{p\}$
S_1
$\{r\}$
S_2
$\{q\}$

$$\frac{\{p\}\,S_1\,\{r\} \quad \{r\}\,S_2\,\{q\}}{\{p\}\,S_1\,;\,S_2\,\{q\}}$$

Basic Proof Outlines

$\{p\}$
$\{p'\}$
S
$\{q'\}$
$\{q\}$

$$\frac{p \to p' \quad \{p'\}S\{q'\} \quad q' \to q}{\{p\}S\{q\}}$$

Basic Proof Outlines

$\{p\}$
if B **then**
    $\{p \wedge B\}$
    S_1
    $\{q\}$
else
    $\{p \wedge \neg B\}$
    S_2
    $\{q\}$
fi
$\{q\}$

$$\frac{\{p \wedge B\}S_1\{q\} \quad \{p \wedge \neg B\}S_2\{q\}}{\{p\}\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}\{q\}}$$

Basic Proof Outlines

$\{\textbf{inv} : p\}$
while B **do**
$\{p \wedge B\}$
S
$\{p\}$
od
$\{p \wedge \neg B\}$

$$\frac{\{p \wedge B\} S \{p\}}{\{p\}\textbf{while } B \textbf{ do } S \textbf{ od}\{p \wedge \neg B\}}$$

Basic Proof Outlines

Example

$i,\ odd\ :=\ n,\ 0$

while $i > 0$ **do**

$odd,\ i\ :=\ 1 - odd,\ i - 1$

od

Basic Proof Outlines

Example

$\{n \geq 0\}$

$i,\ odd\ :=\ n,\ 0$

while $i > 0$ **do**

$odd,\ i\ :=\ 1 - odd,\ i - 1$

od

$\{odd = n\ \%\ 2\}$

Basic Proof Outlines

Example

$\{n \geq 0\}$

$i,\ odd\ :=\ n,\ 0$
$\{\textbf{inv} : n\,\%\,2 = (i + odd)\,\%\,2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
while $i > 0$ **do**

$odd,\ i\ :=\ 1 - odd,\ i - 1$

od

$\{odd = n\,\%\,2\}$

Basic Proof Outlines

Example

$\{n \geq 0\}$

$i,\ odd\ :=\ n,\ 0$
$\{\textbf{inv} : n\,\%\,2 = (i + odd)\,\%\,2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
while $i > 0$ **do**
$\{n\,\%\,2 = (i + odd)\,\%\,2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge i > 0\}$

$odd,\ i\ :=\ 1 - odd,\ i - 1$

od
$\{n\,\%\,2 = (i + odd)\,\%\,2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge \neg(i > 0)\}$
$\{odd = n\,\%\,2\}$

Basic Proof Outlines

Example

$\{n \geq 0\}$
$\{n \,\%\, 2 = (n+0) \,\%\, 2 \wedge 0 \leq 0 \leq 1 \wedge n \geq 0\}$
$i,\ odd\ :=\ n,\ 0$
$\{\mathbf{inv} : n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
while $i > 0$ **do**
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge i > 0\}$

$odd,\ i\ :=\ 1 - odd,\ i - 1$

od
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge \neg(i > 0)\}$
$\{odd = n \,\%\, 2\}$

Basic Proof Outlines

Example

$\{n \geq 0\}$
$\{n \,\%\, 2 = (n+0) \,\%\, 2 \wedge 0 \leq 0 \leq 1 \wedge n \geq 0\}$
$i,\ odd\ :=\ n,\ 0$
$\{\mathbf{inv} : n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
while $i > 0$ **do**
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge i > 0\}$

$odd,\ i\ :=\ 1 - odd,\ i - 1$
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
od
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge \neg(i > 0)\}$
$\{odd = n \,\%\, 2\}$

Basic Proof Outlines

Example

$\{n \geq 0\}$
$\{n \,\%\, 2 = (n+0) \,\%\, 2 \wedge 0 \leq 0 \leq 1 \wedge n \geq 0\}$
$i,\ odd\ :=\ n,\ 0$
$\{\mathbf{inv} : n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
while $i > 0$ **do**
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge i > 0\}$
$\{n \,\%\, 2 = (i - 1 + 1 - odd) \,\%\, 2 \wedge$
$0 \leq 1 - odd \leq 1 \wedge i - 1 \geq 0\}$
$odd,\ i\ :=\ 1 - odd,\ i - 1$
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0\}$
od
$\{n \,\%\, 2 = (i + odd) \,\%\, 2 \wedge 0 \leq odd \leq 1 \wedge i \geq 0 \wedge \neg(i > 0)\}$
$\{odd = n \,\%\, 2\}$

Basic Proof Complexity

Fact

The proof system consists of:

- *Analytical rules and axiom*
- *Adaptation rule*

Basic Proof Complexity

Fact

The proof system consists of:

- *Analytical rules and axiom*
- *Adaptation rule*

Definition (Size of program)

- $\ell(\bar{x} := \bar{t}) = 1$
- $\ell(S_1;\ S_2) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{while } B \textbf{ do } S \textbf{ od}) = \ell(S) + 1$

Basic Proof Complexity

Fact

The proof system consists of:

- *Analytical rules and axiom*
- *Adaptation rule*

Definition (Size of program)

- $\ell(\bar{x} := \bar{t}) = 1$
- $\ell(S_1;\ S_2) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{while } B \textbf{ do } S \textbf{ od}) = \ell(S) + 1$

Lemma

If $\vdash \{p\}\ S\ \{q\}$ *then there exists a proof with at most* $2 \times \ell(S)$ *rule applications and* $\{p\}\ S\ \{q\}$ *as conclusion.*

Basic Linear Proofs

Lemma

If $\vdash \{p\}\ S\ \{q\}$ *then there exists a proof with at most* $2 \times \ell(S)$ *rule applications and* $\{p\}\ S\ \{q\}$ *as conclusion.*

Technique: **proof normalization**

Basic Linear Proofs

Lemma

If $\vdash \{p\}\ S\ \{q\}$ *then there exists a proof with at most* $2 \times \ell(S)$ *rule applications and* $\{p\}\ S\ \{q\}$ *as conclusion.*

Technique: **proof normalization**

$$\dfrac{\models p \to p' \qquad \dfrac{\mathscr{D}}{\{p'\}\ S\ \{q'\}}\ \dfrac{\models p' \to p'' \quad \{p''\}\ S\ \{q''\} \quad \models q'' \to q'}{} \qquad \models q' \to q}{\{p\}\ S\ \{q\}}$$

can be collapsed into

$$\dfrac{\models p \to p'' \quad \dfrac{\mathscr{D}}{\{p''\}\ S\ \{q''\}} \quad \models q'' \to q}{\{p\}\ S\ \{q\}}$$

Recursive Programs

Call-By-Name Procedures

Let $P_1, \ldots, P_n$ be procedures with formal parameters $\bar{u}_1, \ldots, \bar{u}_n$.

Call-By-Name Procedures

Let $P_1, \ldots, P_n$ be procedures with formal parameters $\bar{u}_1, \ldots, \bar{u}_n$.

Each procedure has an associated **body**.

Call-By-Name Procedures

Let $P_1, \ldots, P_n$ be procedures with formal parameters $\bar{u}_1, \ldots, \bar{u}_n$.

Each procedure has an associated **body**.

$P(\bar{t})$ is a procedure call

Call-By-Name Procedures

Let $P_1, \ldots, P_n$ be procedures with formal parameters $\bar{u}_1, \ldots, \bar{u}_n$.

Each procedure has an associated **body**.

$P(\bar{t})$ is a procedure call

Meaning: body replacement, but substitute the formal parameters ($\bar{u}$) by actual parameters ($\bar{t}$).

Call-By-Name Procedures

Let $P_1, \ldots, P_n$ be procedures with formal parameters $\bar{u}_1, \ldots, \bar{u}_n$.

Each procedure has an associated **body**.

$P(\bar{t})$ is a procedure call

Meaning: body replacement, but substitute the formal parameters ($\bar{u}$) by actual parameters ($\bar{t}$).

In practice: think of macro expansion

Call-By-Name Procedures

Traditional approach to call-by-name verification:

- Correctness formula for each procedure call
- Add these formulas for calls as assumptions
- Prove each assumption correct w.r.t. assumptions

Call-By-Name Procedures

Traditional approach to call-by-name verification:

- Correctness formula for each procedure call
- Add these formulas for calls as assumptions
- Prove each assumption correct w.r.t. assumptions

Complexity argument (simplified):

1. A program may contain 1 procedure but k different calls
2. Body is verified for each call separately
3. Leading to quadratic proof complexity

Call-By-Name Procedures

Traditional approach to call-by-name verification:

- Correctness formula for each procedure call
- Add these formulas for calls as assumptions
- Prove each assumption correct w.r.t. assumptions

Complexity argument (simplified):

1. A program may contain 1 procedure but k different calls
2. Body is verified for each call separately
3. Leading to quadratic proof complexity

Necessary for call-by-name: actual parameters may be evaluated in different states for each call.

Call-By-Name Procedures

Traditional approach to call-by-name verification:

- Correctness formula for each procedure call
- Add these formulas for calls as assumptions
- Prove each assumption correct w.r.t. assumptions

Complexity argument (simplified):

1. A program may contain 1 procedure but k different calls
2. Body is verified for each call separately
3. Leading to quadratic proof complexity

Necessary for call-by-name: actual parameters may be evaluated in different states for each call.

What about call-by-value? Why is it most used in practice?

Recursive Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$
\begin{aligned}
S ::= {} & \bar{x} := \bar{t} \\
\mid {} & P(\bar{t}) \\
\mid {} & S;\ S \\
\mid {} & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\
\mid {} & \textbf{while } B \textbf{ do } S \textbf{ od} \\
\mid {} & \textbf{begin local } \bar{x} := \bar{t};\ S \textbf{ end}
\end{aligned}
$$

Recursive Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$
\begin{aligned}
S ::= \; & \bar{x} := \bar{t} \\
\mid \; & P(\bar{t}) \\
\mid \; & S;\ S \\
\mid \; & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\
\mid \; & \textbf{while } B \textbf{ do } S \textbf{ od} \\
\mid \; & \textbf{begin local } \bar{x} := \bar{t};\ S \textbf{ end}
\end{aligned}
$$

- $P(\bar{t})$: procedure call (call-by-value)

Recursive Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$
\begin{aligned}
S ::= {} & \bar{x} := \bar{t} \\
\mid {} & P(\bar{t}) \\
\mid {} & S;\ S \\
\mid {} & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\
\mid {} & \textbf{while } B \textbf{ do } S \textbf{ od} \\
\mid {} & \textbf{begin local } \bar{x} := \bar{t};\ S \textbf{ end}
\end{aligned}
$$

- $P(\bar{t})$: procedure call (call-by-value)
- **begin end**: block with local variable declaration

Recursive Programming Language

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

Our programming language in Backus-Naur form:

$$
\begin{aligned}
S ::= \; & \bar{x} := \bar{t} \\
\mid \; & P(\bar{t}) \\
\mid \; & S;\ S \\
\mid \; & \textbf{if } B \textbf{ then } S \textbf{ else } S \textbf{ fi} \\
\mid \; & \textbf{while } B \textbf{ do } S \textbf{ od} \\
\mid \; & \textbf{begin local } \bar{x} := \bar{t};\ S \textbf{ end}
\end{aligned}
$$

- $P(\bar{t})$: procedure call (call-by-value)
- **begin end**: block with local variable declaration

$$D = \{P_1(\bar{u}_1) :: S_1, \ldots, P_n(\bar{u}_n) :: S\} \qquad (D \mid S)$$

Recursive Odd Program

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
    if $n = 0$ **then** $even := 1$
    else $Odd(n-1);\ even := odd$ **fi**
$Odd(n)$ ::
    if $n = 0$ **then** $odd := 0$
    else $Even(n-1);\ odd := even$ **fi**

Recursive Odd Program

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Procedure inlining semantics:

$Even(n-1) \longrightarrow$
  begin local $n := n-1;$
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi end**

Recursive Odd Program

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Procedure inlining semantics:

$Even(n-1) \longrightarrow$
  begin local $n := n-1;$
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi end**

Recursive Odd Program

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Procedure inlining semantics:

$Even(n-1) \longrightarrow$
  begin local $n := n-1;$
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi end**

Recursive Odd Program

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Procedure inlining semantics:

$Even(n-1) \longrightarrow$
  begin local $n := n-1;$
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi end**

Recursive Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\, D \mid S\, \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

Recursive Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ D \mid S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ D \mid S\ \{q\}$: derivability

Recursive Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ D \mid S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ D \mid S\ \{q\}$: derivability
- $\models \{p\}\ D \mid S\ \{q\}$: truth

Recursive Program Logic

Let $\mathscr{L}$ be a first-order language (e.g. arithmetic).

$$\{p\}\ D \mid S\ \{q\}$$

is a **correctness formula**, where p and q are formula of $\mathscr{L}$.

- $\vdash \{p\}\ D \mid S\ \{q\}$: derivability
- $\models \{p\}\ D \mid S\ \{q\}$: truth

$$\{p\}\ P(\bar{u})\ \{q\}$$

is a **contract**, and Γ is a set of contracts matching D.

- $\Gamma \vdash_D \{p\}\ S\ \{q\}$: derivability under assumptions Γ
- $\Gamma \models_D \{p\}\ S\ \{q\}$: truth under assumptions Γ

Recursive Program Logic

Definition (1/3)

BASIC PROGRAM LOGIC +

BLOCK

$$\frac{\{p\}\ \bar{x} := \bar{t};\ S\ \{q\}}{\{p\}\ \textbf{begin local}\ \bar{x} := \bar{t};\ S\ \textbf{end}\ \{q\}}$$

where $\{\bar{x}\} \cap \mathit{free}(q) = \emptyset$.

PROCEDURE CALL (INSTANTIATION)

$$\frac{\{p\}\ P(\bar{u})\ \{q\}}{\{p[\bar{u} := \bar{t}]\}\ P(\bar{t})\ \{q\}}$$

where $\{\bar{u}\} \cap \mathit{free}(q) = \emptyset$.

Definition (2/3)

SUBSTITUTION

$$\frac{\{p\}\ S\ \{q\}}{\{p[\bar{x} := \bar{y}]\}\ S\ \{q[\bar{x} := \bar{y}]\}}$$

where $\{\bar{x}\} \cap var(D \mid S) = \emptyset$ and $\{\bar{y}\} \cap change(D \mid S) = \emptyset$.

INVARIANCE

$$\frac{\{p\}\ S\ \{q\}}{\{p \wedge r\}\ S\ \{q \wedge r\}}$$

where $free(r) \cap change(D \mid S) = \emptyset$.

$\exists$-INTRODUCTION

$$\frac{\{p\}\ S\ \{q\}}{\{\exists \bar{x} : p\}\ S\ \{q\}}$$

where $\{\bar{x}\} \cap (var(D \mid S) \cup free(q)) = \emptyset$.

Recursive Program Logic

Definition (3/3)

RECURSION

$$\frac{\begin{array}{l}\{p_1\}\ P_1(\bar{u}_1)\ \{q_1\},\ldots,\{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash_D \{p\}\ S\ \{q\},\\ \{p_1\}\ P_1(\bar{u}_1)\ \{q_1\},\ldots,\{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash_D \{p_1\}\ S_1\ \{q_1\},\\ \vdots\\ \{p_1\}\ P_1(\bar{u}_1)\ \{q_1\},\ldots,\{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash_D \{p_n\}\ S_n\ \{q_n\}\end{array}}{\{p\}\ D \mid S\ \{q\}}$$

where $D = \{P_i(\bar{u}_i) :: S_i \mid i \in \{1,\ldots,n\}\}$ and $\{\bar{u}_i\} \cap \mathit{free}(q_i) = \emptyset$ for $i \in \{1,\ldots,n\}$.

Recursive Program Logic

Definition (3/3)

RECURSION

$$\frac{\begin{array}{l}\{p_1\}\ P_1(\bar{u}_1)\ \{q_1\},\ldots,\{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash_D \{p\}\ S\ \{q\},\\ \{p_1\}\ P_1(\bar{u}_1)\ \{q_1\},\ldots,\{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash_D \{p_1\}\ S_1\ \{q_1\},\\ \vdots\\ \{p_1\}\ P_1(\bar{u}_1)\ \{q_1\},\ldots,\{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash_D \{p_n\}\ S_n\ \{q_n\}\end{array}}{\{p\}\ D \mid S\ \{q\}}$$

where $D = \{P_i(\bar{u}_i) :: S_i \mid i \in \{1,\ldots,n\}\}$ and $\{\bar{u}_i\} \cap \mathit{free}(q_i) = \emptyset$ for $i \in \{1,\ldots,n\}$.

N.B. One contract per procedure, **generic** for every call.

Recursive Program Logic

We have established soundness and completeness.

Recursive Program Logic

We have established soundness and completeness.

Lemma

- $\Gamma \vdash_D \{p\}\ S\ \{q\}$ *implies* $\Gamma \models_D \{p\}\ S\ \{q\}$
- $\models \{p\}\ D \mid S\ \{q\}$ *implies* $G(D) \vdash_D \{p\}\ S\ \{q\}$

Theorem

- $\vdash \{p\}\ D \mid S\ \{q\}$ *implies* $\models \{p\}\ D \mid S\ \{q\}$
- $\models \{p\}\ D \mid S\ \{q\}$ *implies* $\vdash \{p\}\ D \mid S\ \{q\}$

Recursive Program Logic

We have established soundness and completeness.

Lemma

- $\Gamma \vdash_D \{p\}\ S\ \{q\}$ *implies* $\Gamma \models_D \{p\}\ S\ \{q\}$
- $\models \{p\}\ D \mid S\ \{q\}$ *implies* $G(D) \vdash_D \{p\}\ S\ \{q\}$

Theorem

- $\vdash \{p\}\ D \mid S\ \{q\}$ *implies* $\models \{p\}\ D \mid S\ \{q\}$
- $\models \{p\}\ D \mid S\ \{q\}$ *implies* $\vdash \{p\}\ D \mid S\ \{q\}$

N.B. $G(D)$ is a set of most general correctness formulas (based on strongest postconditions).
N.B. Our notion of completeness is in the sense of Cook.

Recursive Program Example

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**,
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Recursive Program Example

Example

$(D \mid Odd(n))$ where D is defined as:

$$
\begin{array}{l}
Even(n) :: \\
\quad \textbf{if } n = 0 \textbf{ then } even := 1 \\
\quad \textbf{else } Odd(n-1);\ even := odd \textbf{ fi}, \\
Odd(n) :: \\
\quad \textbf{if } n = 0 \textbf{ then } odd := 0 \\
\quad \textbf{else } Even(n-1);\ odd := even \textbf{ fi}
\end{array}
$$

Contracts:

$\{?\}\ Even(n)\ \{?\}$

$\{?\}\ Odd(n)\ \{?\}$

Recursive Program Example

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**,
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Contracts:

$\{n \geq 0 \wedge z = n\}\ Even(n)\ \{even = (z+1)\ \%\ 2\}$
$\{n \geq 0 \wedge z = n\}\ Odd(n)\ \{odd = z\ \%\ 2\}$

Recursive Program Example

Example

$(D \mid Odd(n))$ where D is defined as:

$Even(n)$ ::
  if $n = 0$ **then** $even := 1$
  else $Odd(n-1);\ even := odd$ **fi**,
$Odd(n)$ ::
  if $n = 0$ **then** $odd := 0$
  else $Even(n-1);\ odd := even$ **fi**

Contracts:
$\{n \geq 0 \wedge z = n\}\ Even(n)\ \{even = (z+1)\ \%\ 2\}$
$\{n \geq 0 \wedge z = n\}\ Odd(n)\ \{odd = z\ \%\ 2\}$

N.B. Finding contracts as difficult as finding loop invariants.

Recursive Program Example

Example (Proof outline for $Even(n)$)

$\{n \geq 0 \wedge z = n\}$
if $n = 0$ **then**
    $even := 1$
else

    $Odd(n-1)$

    $\{odd = (z+1)\ \%\ 2\}$
    $even := odd$
fi
$\{even = (z+1)\ \%\ 2\}$

Recursive Program Example

Example (Proof outline for $Even(n)$)

$\{n \geq 0 \wedge z = n\}$
if $n = 0$ **then**
    $even := 1$
else

$\{n-1 \geq 0 \wedge z = n-1\}$
$Odd(n-1)$
$\{odd = z \,\%\, 2\}$

$\{odd = (z+1) \,\%\, 2\}$
$even := odd$
fi
$\{even = (z+1) \,\%\, 2\}$

Recursive Program Example

Example (Proof outline for $Even(n)$)

$\{n \geq 0 \wedge z = n\}$
if $n = 0$ **then**
    $even := 1$
else

$\{n-1 \geq 0 \wedge z = n-1\}$
$Odd(n-1)$
$\{odd = z \,\%\, 2\}$

$\{odd = (z+1) \,\%\, 2\}$
$even := odd$
fi
$\{even = (z+1) \,\%\, 2\}$

Recursive Program Example

Example (Proof outline for $Even(n)$)

$\{n \geq 0 \wedge z = n\}$
if $n = 0$ **then**
$even := 1$
else

$\{n-1 \geq 0 \wedge n-1 = n-1\}$
$\{n-1 \geq 0 \wedge z = n-1\}$
$Odd(n-1)$
$\{odd = z \,\%\, 2\}$
$\{odd = (n-1) \,\%\, 2\}$

$\{odd = (z+1) \,\%\, 2\}$
$even := odd$
fi
$\{even = (z+1) \,\%\, 2\}$

Recursive Program Example

Example (Proof outline for $Even(n)$)

$\{n \geq 0 \wedge z = n\}$
if $n = 0$ **then**
    $even := 1$
else

$\{n-1 \geq 0 \wedge n-1 = n-1\}$
$\{n-1 \geq 0 \wedge z = n-1\}$
$Odd(n-1)$
$\{odd = z \,\%\, 2\}$
$\{odd = (n-1) \,\%\, 2\}$

    $\{odd = (z+1) \,\%\, 2\}$
    $even := odd$
fi
$\{even = (z+1) \,\%\, 2\}$

Recursive Program Example

Example (Proof outline for $Even(n)$)

$\{n \geq 0 \wedge z = n\}$
if $n = 0$ **then**
    $even := 1$
else
    $\{n-1 \geq 0 \wedge n-1 = n-1 \wedge z = n\}$
    $\{n-1 \geq 0 \wedge n-1 = n-1\}$
    $\{n-1 \geq 0 \wedge z = n-1\}$
    $Odd(n-1)$
    $\{odd = z \,\%\, 2\}$
    $\{odd = (n-1) \,\%\, 2\}$
    $\{odd = (n-1) \,\%\, 2 \wedge z = n\}$
    $\{odd = (z+1) \,\%\, 2\}$
    $even := odd$
fi
$\{even = (z+1) \,\%\, 2\}$

Recursive Proof Complexity

Definition (Size of program)

- $\ell(\bar{x} := \bar{t}) = 1$
- $\ell(P(\bar{t})) = 1$
- $\ell(S_1;\ S_2) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{while } B \textbf{ do } S \textbf{ od}) = \ell(S) + 1$
- $\ell(\textbf{begin local } \bar{x} := \bar{t};\ S \textbf{ end}) = \ell(\bar{x} := \bar{t};\ S) + 1$

Recursive Proof Complexity

Definition (Size of program)

- $\ell(\bar{x} := \bar{t}) = 1$
- $\ell(P(\bar{t})) = 1$
- $\ell(S_1;\ S_2) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \ell(S_1) + \ell(S_2) + 1$
- $\ell(\textbf{while } B \textbf{ do } S \textbf{ od}) = \ell(S) + 1$
- $\ell(\textbf{begin local } \bar{x} := \bar{t};\ S \textbf{ end}) = \ell(\bar{x} := \bar{t};\ S) + 1$

Lemma

If $\Gamma \vdash_D \{p\}\ S\ \{q\}$ *then there exists a proof with at most* $6 \times \ell(S)$ *rule applications and* $\{p\}\ S\ \{q\}$ *as conclusion.*

Recursive Proof Complexity

Lemma

If $\Gamma \vdash_D \{p\}\ S\ \{q\}$ *then there exists a proof with at most* $6 \times \ell(S)$ *rule applications and* $\{p\}\ S\ \{q\}$ *as conclusion.*

Technique: **proof normalization**

Recursive Proof Complexity

Lemma

If $\Gamma \vdash_D \{p\}\ S\ \{q\}$ *then there exists a proof with at most* $6 \times \ell(S)$ *rule applications and* $\{p\}\ S\ \{q\}$ *as conclusion.*

Technique: **proof normalization**

Between every analytical rule we may have adaptation rules:

- C for CONSEQUENCE
- S for SUBSTITUTION
- I for INVARIANCE
- E for $\exists$-INTRODUCTION

Can we reduce arbitrary sequence $(C \mid S \mid I \mid E)^*$ to a fixed **normal** form?

Recursive Proof Complexity

We introduce a proof rewrite system:

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse
2. For any in $(C \mid S \mid E)^*$, move S to the front and collapse

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse
2. For any in $(C \mid S \mid E)^*$, move S to the front and collapse
3. For any in $(C \mid E)^*$, collapse and analyze $(CE)^*C$

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse
2. For any in $(C \mid S \mid E)^*$, move S to the front and collapse
3. For any in $(C \mid E)^*$, collapse and analyze $(CE)^*C$
4. For any in $(CE)^*C$, reduce $CECE$ to CEC and collapse

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse
2. For any in $(C \mid S \mid E)^*$, move S to the front and collapse
3. For any in $(C \mid E)^*$, collapse and analyze $(CE)^*C$
4. For any in $(CE)^*C$, reduce $CECE$ to CEC and collapse

We end up with a normal form: $ISCEC$

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse
2. For any in $(C \mid S \mid E)^*$, move S to the front and collapse
3. For any in $(C \mid E)^*$, collapse and analyze $(CE)^*C$
4. For any in $(CE)^*C$, reduce $CECE$ to CEC and collapse

We end up with a normal form: $ISCEC$

We need to show:

1. $II \longrightarrow IC$, $CI \longrightarrow IC$, $SI \longrightarrow IS$, $EI \longrightarrow IECS$,
2. $SS \longrightarrow S$, $CS \longrightarrow SC$, $ES \longrightarrow SE$,
3. $EE \longrightarrow E$, $CC \longrightarrow C$,
4. $CECE \longrightarrow CEC$

Recursive Proof Complexity

We introduce a proof rewrite system:

1. For any in $(C \mid S \mid I \mid E)^*$, move I to the front and collapse
2. For any in $(C \mid S \mid E)^*$, move S to the front and collapse
3. For any in $(C \mid E)^*$, collapse and analyze $(CE)^*C$
4. For any in $(CE)^*C$, reduce $CECE$ to CEC and collapse

We end up with a normal form: $ISCEC$

We need to show:

1. $II \longrightarrow IC$, $CI \longrightarrow IC$, $SI \longrightarrow IS$, $EI \longrightarrow IECS$,
2. $SS \longrightarrow S$, $CS \longrightarrow SC$, $ES \longrightarrow SE$,
3. $EE \longrightarrow E$, $CC \longrightarrow C$,
4. $CECE \longrightarrow CEC$

The rewrite system is **terminating** and **confluent**.

Recursive Proof Complexity

For example, $EI \longrightarrow IECS$:

$$\dfrac{\dfrac{\dfrac{\mathscr{D}}{\{p\}\ S\ \{q\}}}{\{\exists \bar{x} : p\}\ S\ \{q\}}}{\{(\exists \bar{x} : p) \wedge r\}\ S\ \{q \wedge r\}}$$

into

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathscr{D}}{\{p\}\ S\ \{q\}}}{\{p \wedge r[\bar{x} := \bar{z}]\}\ S\ \{q \wedge r[\bar{x} := \bar{z}]\}}}{\{\exists \bar{x} : (p \wedge r[\bar{x} := \bar{z}])\}\ S\ \{q \wedge r[\bar{x} := \bar{z}]\}} \quad \models \ldots}{\{(\exists \bar{x} : p) \wedge r[\bar{x} := \bar{z}]\}\ S\ \{q \wedge r[\bar{x} := \bar{z}]\}} \quad \models \ldots$$
$$\dfrac{\{(\exists \bar{x} : p) \wedge r[\bar{x} := \bar{z}]\}\ S\ \{q \wedge r[\bar{x} := \bar{z}]\}}{\{(\exists \bar{x} : p) \wedge r\}\ S\ \{q \wedge r\}}$$

Recursive Proof Complexity

For example, $CECE \longrightarrow CEC$:

$$\dfrac{\models p \to \exists \bar{x} : p' \qquad \dfrac{\dfrac{\dfrac{\mathscr{D}}{\models p' \to p'' \quad \{p''\}\, S\, \{q''\} \quad \models q'' \to q'}}{\{p'\}\, S\, \{q'\}}}{\{\exists \bar{x} : p'\}\, S\, \{q'\}} \qquad \models q' \to q}{\dfrac{\{p\}\, S\, \{q\}}{\{\exists \bar{y} : p\}\, S\, \{q\}}}$$

into

$$\dfrac{\models \ldots \quad \dfrac{\dfrac{\mathscr{D}}{\models \ldots \quad \{p''\}\, S\, \{q''\} \quad \models \ldots}}{\{p'\}\, S\, \{q_0\}} \quad \models \ldots}{\{\exists \bar{y}, \bar{x} : p'\}\, S\, \{q_0\}}$$

Conclusion

Overview

1. Basic Hoare Logic
2. Recursive procedures (call-by-name):
   quadratic proof complexity
3. Recursive procedures (call-by-value):
   linear proof complexity
4. Proof normalization of adaptation rules: *ISCEC*

On-going work

- Improve proof system's modularity (subscript D)
- Invariance rule (*change*): semantic footprints
- Formalization in dependent type theory (Coq)
- Higher-order Hoare logic and separation logic