

Integration of Software-Model-Checking into Functional Verification of Imperative Programs

Norbert Schirmer

Joint work with:

Amine Chaieb, Stefan Maus,
Martin Wildmoser

procedures $Fac(N|R) =$
IF $N = 0$ THEN $R := 1$
ELSE $R := CALL Fac(N - 1); R := N * R$ FI

lemma (in HOL)
shows
apply vcg

1. $\bigwedge N. (N = 0 \longrightarrow 1 = fac\ N) \wedge$
 $(N \neq 0 \longrightarrow N * fac\ (N - 1) = fac\ N)$



Motivation

- Full **functional** verification of imperative programs: inherently **interactive** task
- Recent advances in **automatic** program analysis, software-model-checking and termination analysis

Integrate software-model-checking into theorem prover
Isabelle/HOL to improve automation

Overview

- 1 The Verisoft Project
- 2 Verification Environment for Simpl
- 3 Integration of Automatic Tools
- 4 Summary/Outlook

- 1 The Verisoft Project
- 2 Verification Environment for Simpl
- 3 Integration of Automatic Tools
- 4 Summary/Outlook

Verisoft — Overview

Large scale verification project

Goal: **Pervasive formal verification of computer systems**

- Create *practically useful, sound* methods and tools
- Increase in industrial productivity and quality
- Prototypical realization

Founded by Federal Ministry of Education and Research

Verisoft — Partners

- University



UNIVERSITÄT
DES
SAARLANDES



TECHNISCHE
UNIVERSITÄT
DARMSTADT



UNIVERSITÄT
KOBLENZ · LANDAU



- Research Institutes



- Industry

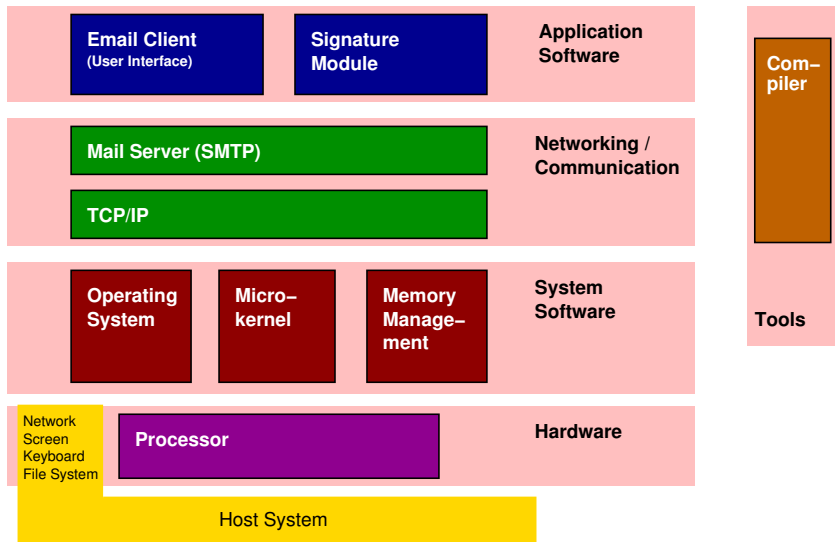


BMW Group

T·Systems

AbsInt
Angewandte Informatik

Academic System



Tools (Integration)

Interactive Theorem Provers

Isabelle/HOL



VSE

Tools (Integration)

Interactive Theorem Provers

Isabelle/HOL

Specifications

VSE

Subtasks on the
level of
assertions

Application Specific
Automatic Tools

Prog. Analysis

Termination

Exec. Time

Tools (Integration)

Interactive Theorem Provers

Isabelle/HOL

Specifications

VSE

Subtasks on the
level of
assertions

Application Specific
Automatic Tools

Prog. Analysis

Termination

Exec. Time

Subgoals on the
level of
proofs

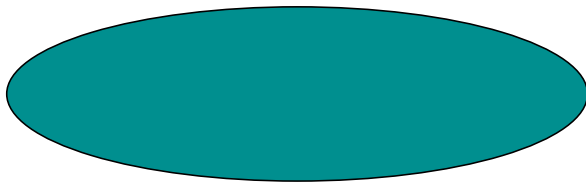
Generic Automatic
Theorem Provers

E-SETHEO

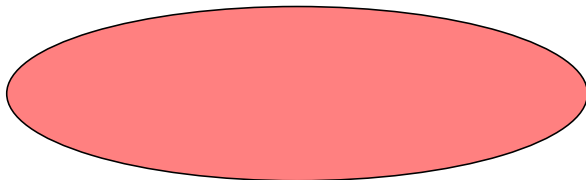
SPASS

Semantical Layers

Hoare



C0

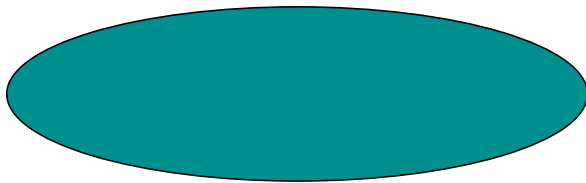


DLX

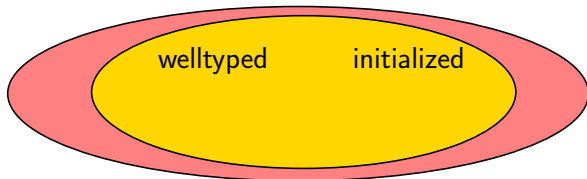


Semantical Layers

Hoare



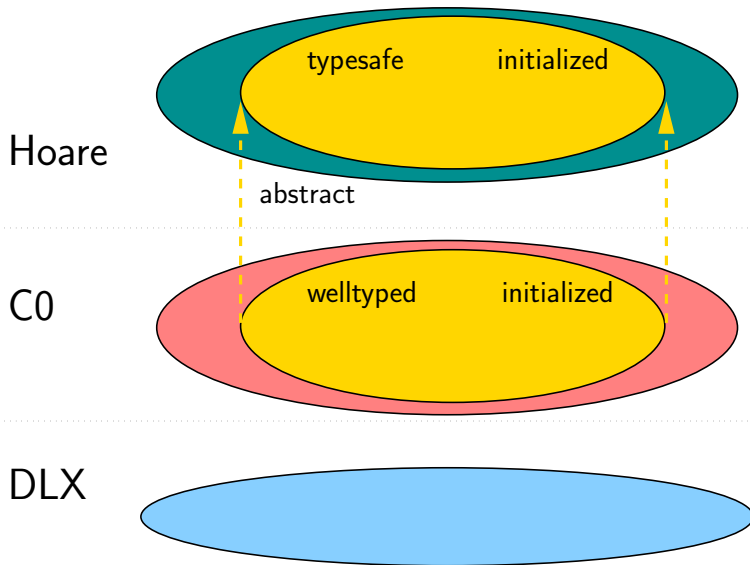
C0



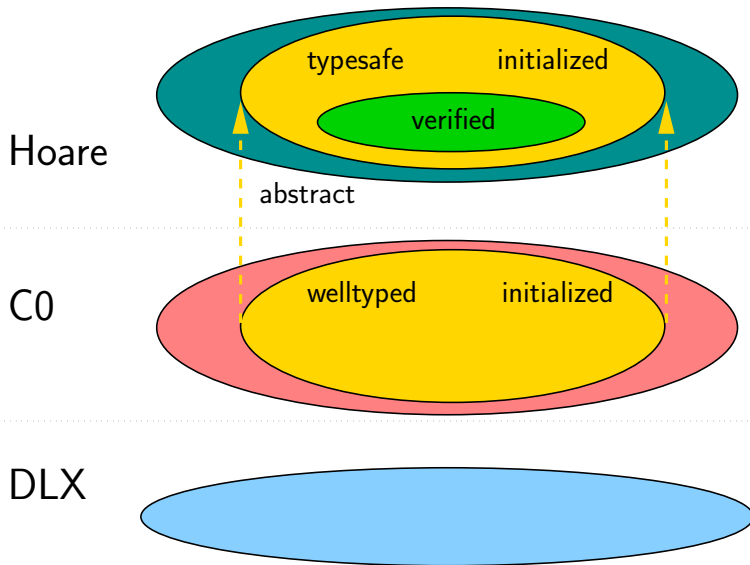
DLX



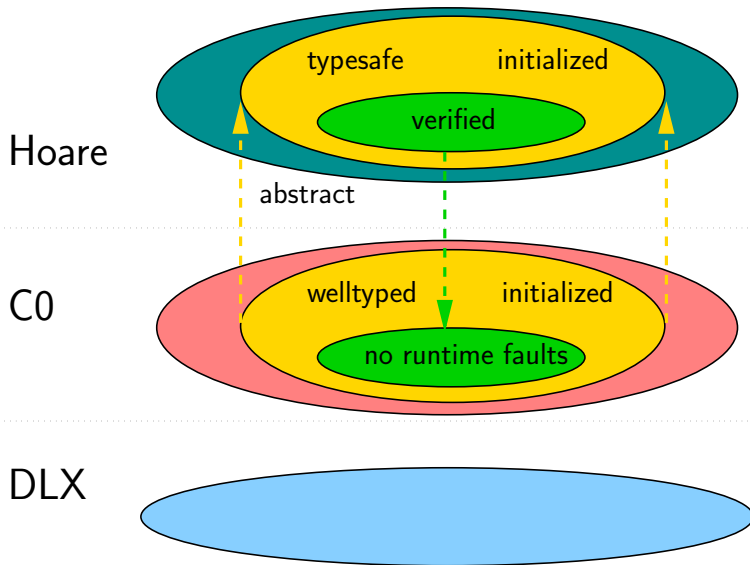
Semantical Layers



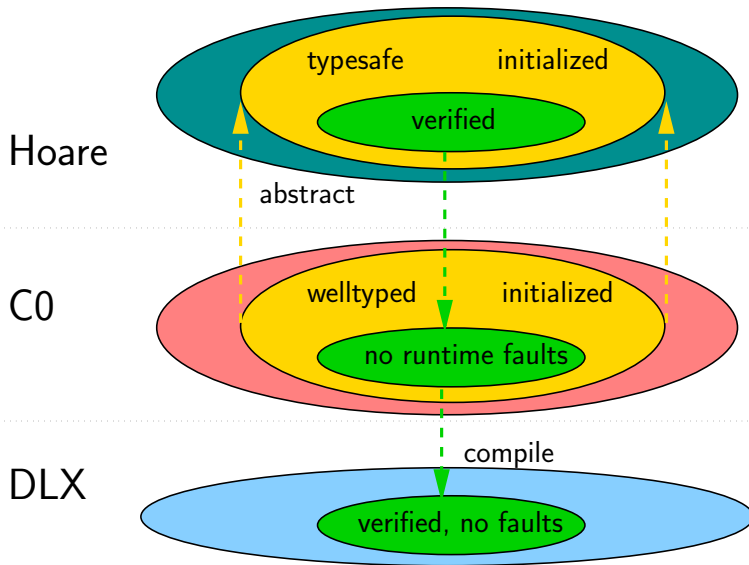
Semantical Layers



Semantical Layers



Semantical Layers



- 1 The Verisoft Project
- 2 Verification Environment for Simpl**
- 3 Integration of Automatic Tools
- 4 Summary/Outlook

Simpl in Isabelle/HOL

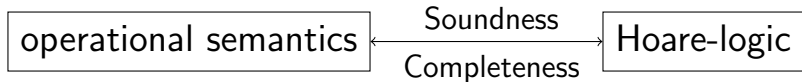
Simpl: Sequential Imperative Programming Language

- Syntax
 - Deep embedding of statements
 - Shallow embedding of expressions
 - Polymorphic state-space

Simpl in Isabelle/HOL

Simpl: Sequential Imperative Programming Language

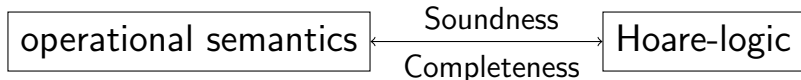
- Syntax
 - Deep embedding of statements
 - Shallow embedding of expressions
 - Polymorphic state-space
- Semantics



Simpl in Isabelle/HOL

Simpl: Sequential Imperative Programming Language

- Syntax
 - Deep embedding of statements
 - Shallow embedding of expressions
 - Polymorphic state-space
- Semantics



- Integration into Isabelle infrastructure
 - Concrete syntax for Simpl
 - Verification-condition-generator (VCG)
 - Interface to automatic program-verification

Simpl statements (I)

(σ, γ) com

Simpl statements (I)

$(\sigma, \gamma) \text{ com} =$
Skip

Simpl statements (I)

$(\sigma, \gamma) \text{ com} =$

Skip

| *Basic upd* $\text{upd} :: \sigma \rightarrow \sigma$

Atomic state-updates

Simpl statements (I)

$(\sigma, \gamma) \text{ com} =$

Skip

| *Basic upd* $\text{upd} :: \sigma \rightarrow \sigma$

Atomic state-updates

| *Spec r* $r :: (\sigma \times \sigma) \text{ set}$

Nondeterministic choice

Simpl statements (I)

$(\sigma, \gamma) \text{ com} =$

Skip

| *Basic upd* $\text{upd} :: \sigma \rightarrow \sigma$

Atomic state-updates

| *Spec r* $r :: (\sigma \times \sigma) \text{ set}$

Nondeterministic choice

| *Seq* $c_1 c_2$ $c_1, c_2 :: (\sigma, \gamma) \text{ com}$

Sequential composition

| *Cond* $b c_1 c_2$ $b :: \sigma \text{ set}$

Conditional execution

| *While* $b c$

Loop

...

Simpl statements (II)

$(\sigma, \gamma) \text{ com} = \dots$

| *Guard f b c* $f :: \gamma$

Runtime-faults

Simpl statements (II)

$(\sigma, \gamma) \text{ com} = \dots$

| *Guard f b c* $f :: \gamma$

Runtime-faults

| *Call p* $p :: \text{string}$

Procedure call

Simpl statements (II)

$(\sigma, \gamma) \text{ com} = \dots$

| *Guard f b c* $f :: \gamma$
Runtime-faults

| *Call p* $p :: \text{string}$
Procedure call

| *DynCom d* $d :: \sigma \rightarrow (\sigma, \gamma) \text{ com}$
State-dependent statement

Simpl statements (II)

$(\sigma, \gamma) \text{ com} = \dots$

| *Guard* $f \ b \ c \quad f :: \gamma$
Runtime-faults

| *Call* $p \quad p :: \text{string}$
Procedure call

| *DynCom* $d \quad d :: \sigma \rightarrow (\sigma, \gamma) \text{ com}$
State-dependent statement

| *Throw* | *Catch* $c_1 \ c_2$
Abrupt termination

Hoare-logic

$$\vdash P \ c \ Q$$

- c : Statement
- P : Precondition
- Q : Postcondition

Hoare-logic

$$\vdash P \ c \ Q, A$$

- c : Statement
- P : Precondition
- Q : Postcondition for normal termination
- A : Postcondition for abrupt termination

Hoare-logic

$$\Gamma \vdash P \ c \ Q, A$$

- c : Statement
- P : Precondition
- Q : Postcondition for normal termination
- A : Postcondition for abrupt termination
- Γ : Mapping from procedure-name to body

Hoare-logic

$$\Gamma, \Theta \vdash P \ c \ Q, A$$

- c : Statement
- P : Precondition
- Q : Postcondition for normal termination
- A : Postcondition for abrupt termination
- Γ : Mapping from procedure-name to body
- Θ : Set of assumptions (Hoare-triple)

Hoare-logic

$$\Gamma, \Theta \vdash \text{ /}_F P \text{ c } Q, A$$

- c : Statement
- P : Precondition
- Q : Postcondition for normal termination
- A : Postcondition for abrupt termination
- Γ : Mapping from procedure-name to body
- Θ : Set of assumptions (Hoare-triple)
- F : Correctness modulo faults in set F

Hoare-logic

$$\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$$

- c : Statement
- P : Precondition
- Q : Postcondition for normal termination
- A : Postcondition for abrupt termination
- Γ : Mapping from procedure-name to body
- Θ : Set of assumptions (Hoare-triple)
- F : Correctness modulo faults in set F
- t : total Correctness (termination)

Usage (example)

```
procedures Fac (N|R) =  
  IF N = 0 THEN R := 1  
  ELSE R := CALL Fac(N - 1); R := N * R FI
```

Usage (example)

procedures $Fac(N|R) =$
 IF $N = 0$ **THEN** $R := 1$
 ELSE $R := \mathbf{CALL} Fac(N - 1); R := N * R$ **FI**

lemma $\forall n. \Gamma \vdash \{N = n\} R := \mathbf{CALL} Fac(N) \{R = n!\}$

Usage (example)

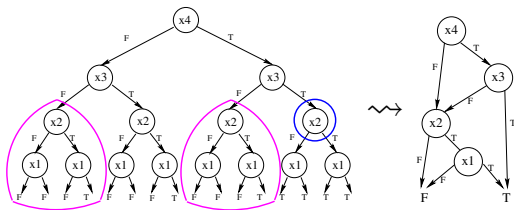
procedures $Fac(N|R) =$
IF $N = 0$ **THEN** $R := 1$
ELSE $R :=$ **CALL** $Fac(N - 1); R := N * R$ **FI**

lemma $\forall n. \Gamma \vdash \{N = n\} R :=$ **CALL** $Fac(N) \{R = n!\}$
apply v_{cg}

1. $\bigwedge N. (N = 0 \longrightarrow 1 = N!) \wedge$
 $(N \neq 0 \longrightarrow N * (N - 1)! = N!)$

Applications

- Case-study: BDD-normalisation [Ortner]



- Verisoft
 - Basic data-structures [Fischer, Starostin]
 - Operating system [Dörrenbächer]
 - Email-client [Beuster, Beckert]
 - C0-Compiler [Petrova]
- Memory-management of L4-kernel [Tuch, Klein]

- 1 The Verisoft Project
- 2 Verification Environment for Simpl
- 3 Integration of Automatic Tools**
- 4 Summary/Outlook

Integrating Automatic Tools

Program verification has three aspects:

- Termination
- Absence of runtime-faults
- Functional correctness

Integrating Automatic Tools

Program verification has three aspects:

- Termination
 - Reduction from total to partial correctness
- Absence of runtime-faults
- Functional correctness

Integrating Automatic Tools

Program verification has three aspects:

- Termination
 - Reduction from total to partial correctness
- Absence of runtime-faults
 - Discharging guards
- Functional correctness

Integrating Automatic Tools

Program verification has three aspects:

- Termination
 - Reduction from total to partial correctness
- Absence of runtime-faults
 - Discharging guards
- Functional correctness
 - Introduction of discharged guards

Termination

$$\frac{\Gamma, \Theta \vdash P \text{ c } Q \quad \forall s \in P. \Gamma \vdash c \downarrow s}{\Gamma, \Theta \vdash_t P \text{ c } Q}$$

- Termination provided by external tool
- Partial correctness remains for interactive proof

Guard against runtime faults

$$\frac{\Gamma, \Theta \vdash (g \cap P) \text{ c } Q}{\Gamma, \Theta \vdash (g \cap P) (\textit{Guard } f \textit{ g } \textit{ c}) Q}$$

Guard against runtime faults

$$\frac{\Gamma, \Theta \vdash (g \wedge P) \text{ c } Q}{\Gamma, \Theta \vdash (g \wedge P) (\text{Guard } f \text{ g c}) Q}$$

```
 $\Gamma \vdash \{P\}$   
 $l := 0;$   
WHILE ( $l < N$ ) DO  
   $A[l] :=_g p;$   
   $l :=_g l + 1;$   
   $p :=_g p \rightarrow \text{next}$   
OD  
 $\{Q\}$ 
```

Guard against runtime faults

$$\frac{\Gamma, \Theta \vdash (g \wedge P) \text{ c } Q}{\Gamma, \Theta \vdash (g \wedge P) (\textit{Guard } f \textit{ g c}) Q}$$

```
 $\Gamma \vdash \{P\}$   
 $l := 0;$   
WHILE ( $l < N$ ) DO  
   $\{0 \leq l \wedge l < \textit{length } A\} \vdash \rightarrow$   
   $A[l] := p;$   
   $l :=_g l + 1;$   
   $p :=_g p \rightarrow \textit{next}$   
OD  
 $\{Q\}$ 
```

Guards against:

- array bound violations

Guard against runtime faults

$$\frac{\Gamma, \Theta \vdash (g \wedge P) \text{ c } Q}{\Gamma, \Theta \vdash (g \wedge P) (\textit{Guard } f \textit{ g c}) Q}$$

```
 $\Gamma \vdash \{P\}$   
 $l := 0;$   
WHILE ( $l < N$ ) DO  
   $\{0 \leq l \wedge l < \textit{length } A\} \vdash \rightarrow$   
   $A[l] := p;$   
   $\{\textit{in-range}(l+1)\} \vdash \rightarrow l := l+1;$   
   $p :=_g p \rightarrow \textit{next}$   
OD  
 $\{Q\}$ 
```

Guards against:

- array bound violations
- over/underflows, division by 0

Guard against runtime faults

$$\frac{\Gamma, \Theta \vdash (g \wedge P) \text{ c } Q}{\Gamma, \Theta \vdash (g \wedge P) (\textit{Guard } f \textit{ g c}) Q}$$

```
 $\Gamma \vdash \{P\}$   
 $l := 0;$   
WHILE ( $l < N$ ) DO  
   $\{0 \leq l \wedge l < \textit{length } A\} \mapsto$   
   $A[l] := p;$   
   $\{\textit{in-range}(l+1)\} \mapsto l := l+1;$   
   $\{p \neq \textit{Null}\} \mapsto p := p \rightarrow \textit{next}$   
OD  
 $\{Q\}$ 
```

Guards against:

- array bound violations
- over/underflows, division by 0
- dereferencing null pointers

Guard against runtime faults

$$\frac{\Gamma, \Theta \vdash (g \wedge P) \text{ c } Q}{\Gamma, \Theta \vdash (g \wedge P) (\textit{Guard } f \textit{ g c}) Q}$$

```
 $\Gamma \vdash \{P\}$   
 $l := 0;$   
WHILE ( $l < N$ ) DO  
   $\{0 \leq l \wedge l < \textit{length } A\} \mapsto$   
   $A[l] := p;$   
   $\{\textit{in-range}(l+1)\} \mapsto l := l+1;$   
   $\{p \neq \textit{Null}\} \mapsto p := p \rightarrow \textit{next}$   
OD  
 $\{Q\}$ 
```

Guards against:

- array bound violations
- over/underflows, division by 0
- dereferencing null pointers

Employ automatic tool to discharge guards

Guards As Guarantees

$$\Gamma, \Theta \vdash_{/F} P \text{ c } Q$$

Correctness modulo faults in F

Guards As Guarantees

$$\Gamma, \Theta \vdash_{/F} P \text{ c } Q$$

Correctness modulo faults in F

$$\Gamma, \Theta \vdash_{/F} (g \cap P) \text{ c } Q$$

$$\frac{\Gamma, \Theta \vdash_{/F} (g \cap P) \text{ c } Q}{\Gamma, \Theta \vdash_{/F} (g \cap P) (\textit{Guard } f \textit{ } g \textit{ } c) Q}$$

Guard

Guards As Guarantees

$$\Gamma, \Theta \vdash_{/F} P \text{ c } Q$$

Correctness modulo faults in F

$$\frac{\Gamma, \Theta \vdash_{/F} (g \cap P) \text{ c } Q}{\Gamma, \Theta \vdash_{/F} (g \cap P) (\textit{Guard } f \textit{ } g \textit{ } c) Q}$$

Guard

$$\frac{f \in F \quad \Gamma, \Theta \vdash_{/F} (g \cap P) \text{ c } Q}{\Gamma, \Theta \vdash_{/F} P (\textit{Guard } f \textit{ } g \textit{ } c) Q}$$

Guarantee

Runtime Faults — Discharging Guards

$$\frac{\Gamma, \Theta \vdash_{/\{True\}} P \ c' \ Q \quad \Gamma, \Theta \vdash P \ c'' \ \{\{True\}\}}{c = \text{mark-guards } False \ c' \quad c'' = \text{strip-guards } \{False\} \ c' \quad \Gamma, \Theta \vdash P \ c \ Q}$$

User

$\Gamma \vdash \{P\}$

$l := 0;$

WHILE ($l < N$) **DO**

$\{0 \leq l\}, \{l < \text{length } A\} \vdash \longrightarrow$

$A[l] := p;$

$\{\text{in-range}(l+1)\} \vdash \longrightarrow l := l+1;$

$\{p \neq \text{Null}\} \vdash \longrightarrow p := p \rightarrow \text{next}$

OD

$\{Q\}$

Runtime Faults — Discharging Guards

$$\frac{\Gamma, \Theta \vdash_{/\{True\}} P \ c' \ Q \quad \Gamma, \Theta \vdash P \ c'' \ \{\{True\}\}}{c = \text{mark-guards } False \ c' \quad c'' = \text{strip-guards } \{False\} \ c' \quad \Gamma, \Theta \vdash P \ c \ Q}$$

User

Tool

$\Gamma \vdash \{P\} / \{True\}$

$l := 0;$

WHILE $(l < N)$ **DO**

$\{0 \leq l\}, \{l < \text{length } A\} \mapsto$

$A[l] := p;$

$\{\text{in-range}(l+1)\} \mapsto l := l+1;$

$\{p \neq \text{Null}\} \mapsto p := p \rightarrow \text{next}$

OD

$\{Q\}$

$\Gamma \vdash \{P\}$

$l := 0;$

WHILE $(l < N)$ **DO**

$\{l < \text{length } A\} \mapsto$

$A[l] := p;$

$l := l+1;$

$p := p \rightarrow \text{next}$

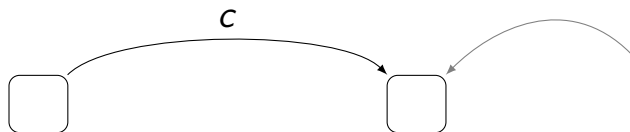
OD

$\{True\}$

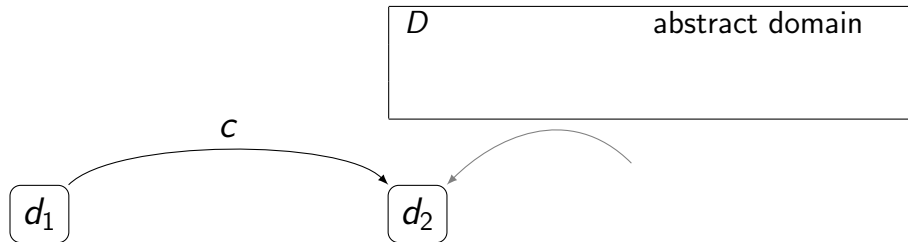
How do tools prove $\Gamma \vdash P \ c'' \ \{\{ True \}\}$?

- Not at all: Oracle
- Only report invariants, proof in Isabelle
- Verified program analysis (e.g. Bytecodeverifier)
- Proof producing program analysis
 - Analyser discovers that $\vdash P \ c \ Q$ holds
 - Return a proof for verification condition $P \longrightarrow wp \ c \ Q$

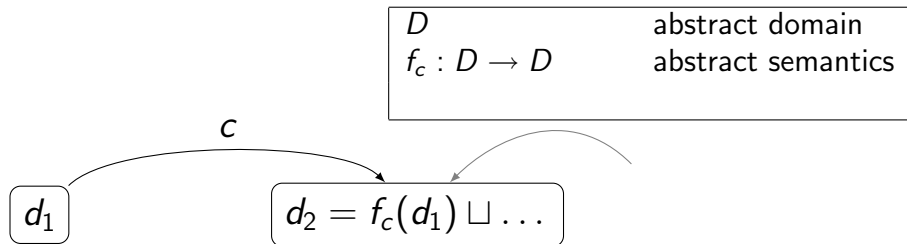
Proof producing program analysis



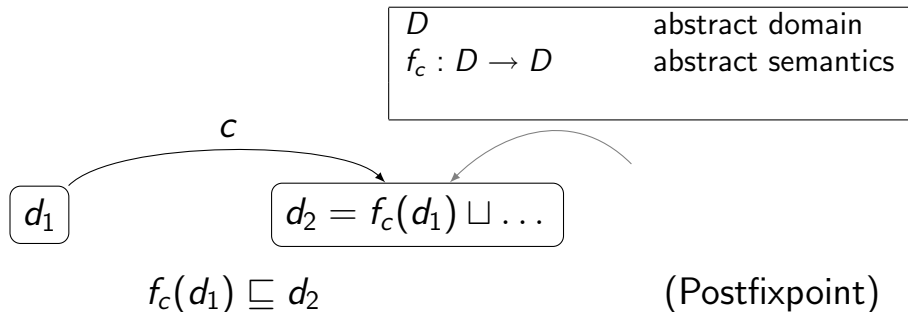
Proof producing program analysis



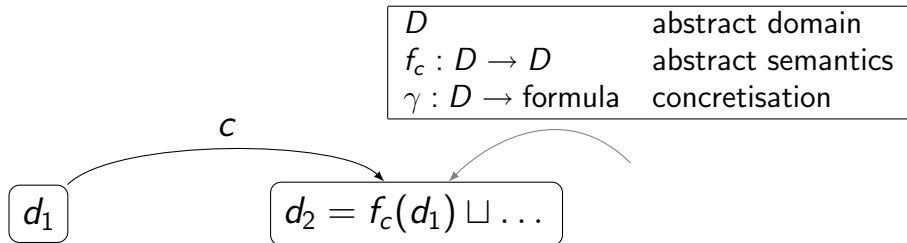
Proof producing program analysis



Proof producing program analysis

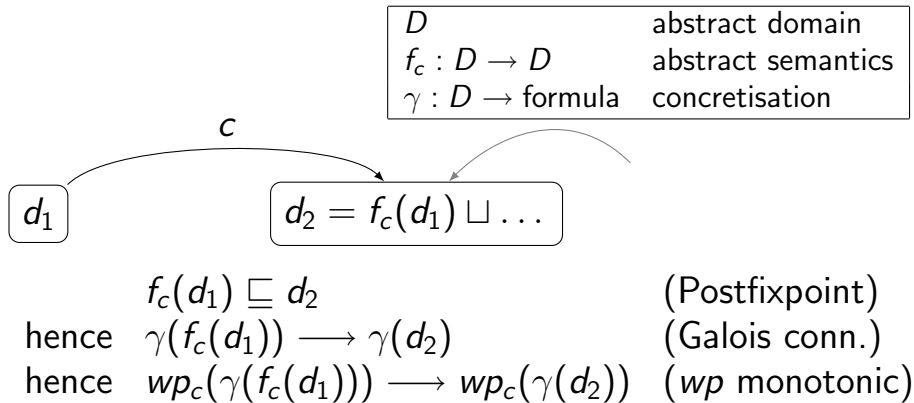


Proof producing program analysis

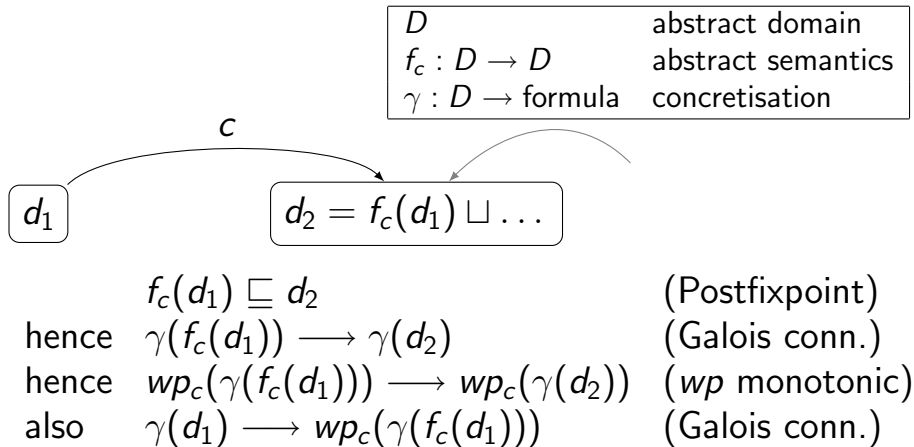


hence $f_c(d_1) \sqsubseteq d_2$ (Postfixpoint)
 $\gamma(f_c(d_1)) \longrightarrow \gamma(d_2)$ (Galois conn.)

Proof producing program analysis

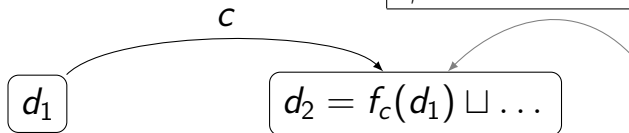


Proof producing program analysis



Proof producing program analysis

D	abstract domain
$f_c : D \rightarrow D$	abstract semantics
$\gamma : D \rightarrow \text{formula}$	concretisation



$$f_c(d_1) \sqsubseteq d_2$$


hence $\gamma(f_c(d_1)) \longrightarrow \gamma(d_2)$

hence $wp_c(\gamma(f_c(d_1))) \longrightarrow wp_c(\gamma(d_2))$ (Postfixpoint)

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$ (Galois conn.)

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$ (wp monotonic)

Proof producing interval analysis

$$c = x := y^2$$


$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$f_c(d_1) \sqsubseteq d_2$$


hence $\gamma(f_c(d_1)) \longrightarrow \gamma(d_2)$

hence $wp_c(\gamma(f_c(d_1))) \longrightarrow wp_c(\gamma(d_2))$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$


$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq d_2$$

hence $\gamma(f_c(d_1)) \longrightarrow \gamma(d_2)$

hence $wp_c(\gamma(f_c(d_1))) \longrightarrow wp_c(\gamma(d_2))$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $\gamma(f_c(d_1)) \longrightarrow \gamma(d_2)$

hence $wp_c(\gamma(f_c(d_1))) \longrightarrow wp_c(\gamma(d_2))$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow \gamma(d_2)$

hence $wp_c(\gamma(f_c(d_1))) \longrightarrow wp_c(\gamma(d_2))$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $wp_c(\gamma(f_c(d_1))) \longrightarrow wp_c(\gamma(d_2))$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow wp_c(\gamma(d_2))$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

also $\gamma(d_1) \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

also $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow wp_c(\gamma(f_c(d_1)))$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

also $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 49 \wedge y \leq 7$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

also $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 49 \wedge y \leq 7$

thus $\gamma(d_1) \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$

$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$


hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

also $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 49 \wedge y \leq 7$

thus $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow wp_c(\gamma(d_2))$

Proof producing interval analysis

$$c = x := y^2$$


$$d_1 = \begin{cases} x \in [0, 3] \\ y \in]-\infty, 7] \end{cases}$$

$$d_2 = \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

$$\begin{cases} x \in [0, 49] \\ y \in]-\infty, 7] \end{cases} \sqsubseteq \begin{cases} x \in [0, 50] \\ y \in]-\infty, 7] \end{cases}$$

hence $0 \leq x \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq x \leq 50 \wedge y \leq 7$

hence $0 \leq y^2 \leq 49 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

also $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 49 \wedge y \leq 7$

thus $0 \leq x \leq 3 \wedge y \leq 7 \longrightarrow 0 \leq y^2 \leq 50 \wedge y \leq 7$

Proof producing program analysis

$$PPPA = PA + wp + \text{decision-procedure}$$

Suggested modularisation:

- use standard program analysis
- wp implemented in verification environment
- decision-procedure independent of fixpoint calculation of program analysis

State of Integration

- Verisoft
 - Termination analysis: Terminator (Andrey Rybalchenko, MPI)
 - Software-model-checker ACSAR for array bound violations and arithmetic overflows (Mohamed Nassim Seghir, MPI)
 - Integration as trusted oracle
- VeryPcc (Chaieb, Wildmoser)
 - Verified Bytecode-Verifier
 - Proof producing interval analysis
 - Invariant report of pointer analysis (TVLA)

- 1 The Verisoft Project
- 2 Verification Environment for Simpl
- 3 Integration of Automatic Tools
- 4 Summary/Outlook

Summary/Outlook

- Language-model: universal, expressive and flexible
- Calculus: sound and complete
- Usable and field-tested
- Modular interface to automatic tools
- Integration of automatic tools promising

Thank you