

Gradual Verification

with Implicit Dynamic Frames

Master Thesis of

Johannes Bader

Karlsruhe Institute of Technology
Karlsruhe, Germany

Advised by

Assoc. Prof. Jonathan Aldrich
Carnegie Mellon University
Pittsburgh, USA

Assoc. Prof. Éric Tanter
University of Chile
Santiago, Chile



Gradual Verification

(with Implicit Dynamic Frames)

```
int getFour(int i)
  requires ?; // haven't figured that one out, yet
  ensures  result = 4;
{
  i := i + 1;
  return i;
}
```

Motivation

- Program verification (against some specification)
- Two flavors: static & dynamic

```
// spec: callable only if (this.balance >= amount)
void withdrawCoins(int amount)
{
    // business logic
    this.balance -= amount;
}
```

Dynamic Verification

- runtime checks
- testing techniques
- guarantee compliance **at runtime**

```
void withdrawCoins(int amount)
{
    assert this.balance >= amount;
    // business logic
    this.balance -= amount;
}
```

Dynamic Verification - Drawbacks

- runtime checks runtime overhead
- testing techniques additional efforts
- guarantee compliance **at runtime** pot. late detection

```
void withdrawCoins(int amount)
    requires this.balance >= amount;
{
    // business logic
    this.balance -= amount;
}
```

Static Verification

- declarative
- formal logic
- guarantee compliance **in advance**

```
void withdrawCoins(int amount)
    requires this.balance >= amount;
{
    // business logic
    this.balance -= amount;
}
```

Static Verification - Drawbacks

- declarative
 - formal logic
 - guarantee compliance **in advance**
- limited syntax
decidability
- annotation pressure

```
void withdrawCoins(int amount)
    requires this.balance >= amount;
    ensures  this.balance = old(this.balance) - amount;
{
    // business logic
    this.balance -= amount;
}
```

Solution? Static + Dynamic

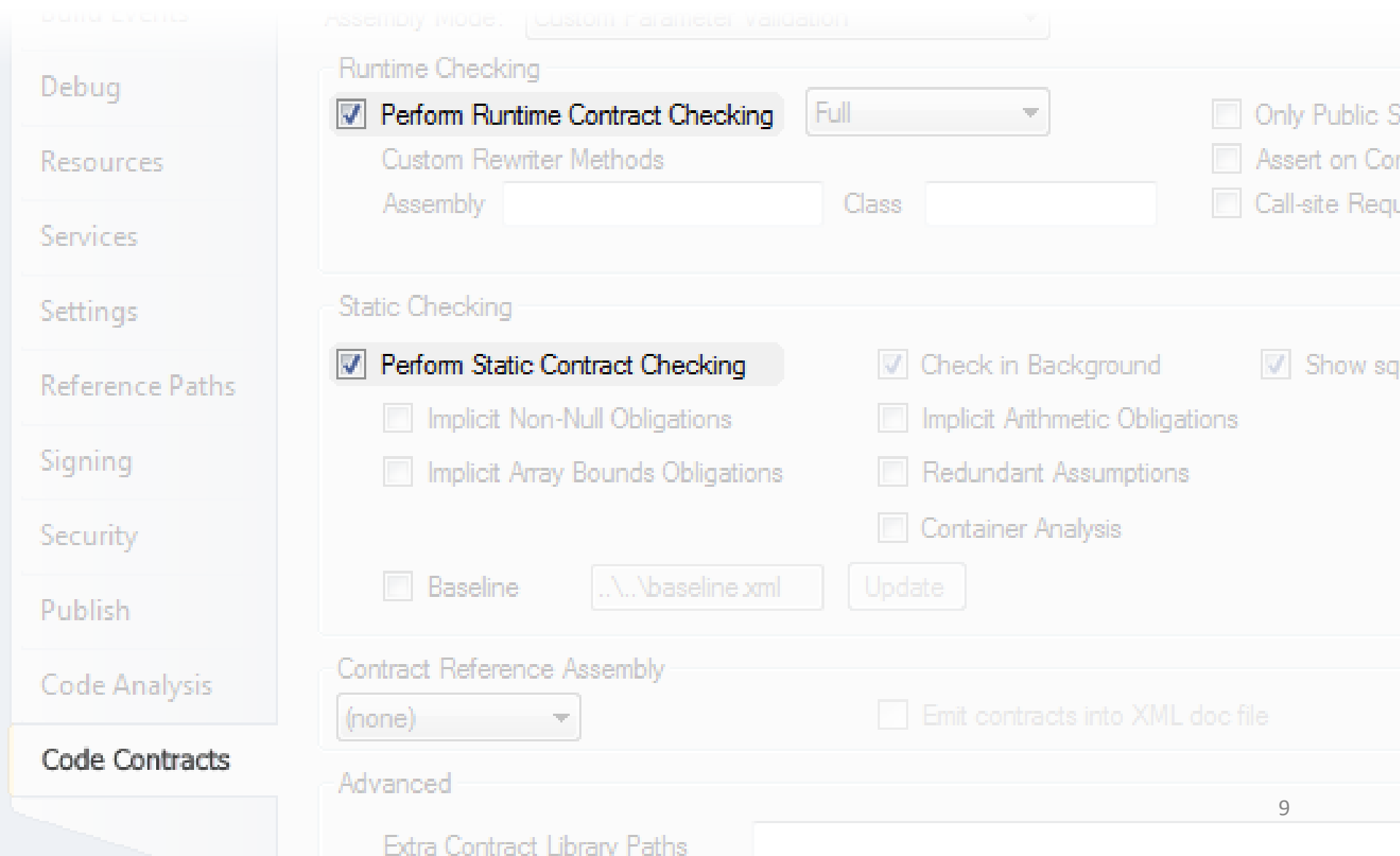
Means:

- “relaxed” static verification (warnings on failure)
- turn contracts into runtime assertions

Notable implementations:

- Java with JML annotations
 - “ESC/Java” for static verification
 - “JML4c” for dynamic verification
- Code Contracts for .NET (by RiSE, MSR)

Solution? Static + Dynamic



The image shows the 'Code Contracts' settings window in Visual Studio. The left sidebar contains a menu with the following items: 'Debug', 'Resources', 'Services', 'Settings', 'Reference Paths', 'Signing', 'Security', 'Publish', 'Code Analysis', and 'Code Contracts' (which is highlighted). The main panel is titled 'Assembly Mode: Custom Parameter Validation'. It is divided into three sections: 'Runtime Checking', 'Static Checking', and 'Contract Reference Assembly'. The 'Runtime Checking' section has a checked checkbox for 'Perform Runtime Contract Checking', a dropdown menu set to 'Full', and three unchecked checkboxes on the right: 'Only Public S...', 'Assert on Con...', and 'Call-site Requ...'. Below these are fields for 'Custom Rewriter Methods', 'Assembly', and 'Class'. The 'Static Checking' section has a checked checkbox for 'Perform Static Contract Checking' and several other unchecked checkboxes: 'Implicit Non-Null Obligations', 'Implicit Array Bounds Obligations', 'Baseline', 'Check in Background', 'Implicit Arithmetic Obligations', 'Redundant Assumptions', 'Container Analysis', and 'Show sq...'. At the bottom of this section are fields for 'Baseline' (containing '..\..\baseline.xml') and an 'Update' button. The 'Contract Reference Assembly' section has a dropdown menu set to '(none)' and an unchecked checkbox for 'Emit contracts into XML doc file'. The 'Advanced' section at the bottom is partially visible, showing a field for 'Extra Contract Library Paths'.

Assembly Mode: Custom Parameter Validation

Runtime Checking

- ☒ Perform Runtime Contract Checking
- Full
- ☐ Only Public S...
- ☐ Assert on Con...
- ☐ Call-site Requ...

Custom Rewriter Methods

Assembly: Class:

Static Checking

- ☒ Perform Static Contract Checking
- ☐ Implicit Non-Null Obligations
- ☐ Implicit Array Bounds Obligations
- ☐ Baseline
- ☒ Check in Background
- ☐ Implicit Arithmetic Obligations
- ☐ Redundant Assumptions
- ☐ Container Analysis
- ☒ Show sq...

Baseline: ..\..\baseline.xml Update

Contract Reference Assembly

(none)

☐ Emit contracts into XML doc file

Advanced

Extra Contract Library Paths

Solution! Static \oplus Dynamic

“Static Typing Where Possible,
Dynamic Typing When Needed” (Erik Meijer)

```
void withdrawCoins(int amount)
    requires this.balance >= amount;
{
    ...
}
...
acc.balance = 100;
acc.withdrawCoins(50); // can prove acc.balance >= 50
acc.withdrawCoins(30); // can't prove acc.balance >= 30
acc.withdrawCoins(30); // can't prove acc.balance >= 30
```

Approach

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness



Gradualization

Syntax

$\tilde{s} \in \tilde{\text{STMT}}$

$\tilde{\phi} \in \tilde{\text{FORMULA}}$

Program State

$\tilde{\pi} \in \tilde{\text{PROGRAMSTATE}}$

Semantics

Static $\tilde{\vdash} \{\tilde{\phi}\} \tilde{s} \{\tilde{\phi}\}$

Dynamic $\tilde{\pi} \Longrightarrow \tilde{\pi}$

Formula $\tilde{\pi} \tilde{\models} \tilde{\phi}$

Soundness

Statically Verified Language

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness

$e ::= x \mid n \mid e_1 + e_2$

$s ::= x := e \mid \text{assert } \phi \mid s_1 ; s_2$

$\phi ::= \text{true} \mid (e_1 = e_2) \mid \phi_1 \wedge \phi_2$

$= (\text{VAR} \rightarrow \mathbb{N}_0) \times \text{STMT}$

Statically Verified Language

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness

$$\frac{x := e}{\vdash \{\phi[e/x]\} x := e \{\phi\}} \text{HASSIGN}$$

$$\frac{\phi \Rightarrow \phi_a}{\vdash \{\phi\} \text{assert } \phi_a \{\phi\}} \text{HASSERT}$$

$$\frac{\begin{array}{c} \phi_{q1} \Rightarrow \phi_{q2} \\ \vdash \{\phi_p\} s_1 \{\phi_{q1}\} \quad \vdash \{\phi_{q2}\} s_2 \{\phi_r\} \end{array}}{\vdash \{\phi_p\} s_1 ; s_2 \{\phi_r\}} \text{HSEQ}$$

Statically Verified Language

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness

$$\frac{\mathcal{N}_\sigma(e) = n}{\langle \sigma, x := e; s \rangle \longrightarrow \langle \sigma[x \mapsto n], s \rangle} \text{SsASSIGN}$$

$$\frac{\langle \sigma, \text{assert } \phi_a; s \rangle \models \phi_a}{\langle \sigma, \text{assert } \phi_a; s \rangle \longrightarrow \langle \sigma, s \rangle} \text{SsASSERT}$$

Statically Verified Language

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness

$$\frac{\vdash \{\phi\} s \{\phi'\}}{\models \{\phi\} s \{\phi'\}} \text{SOUNDNESS}$$

$$\models \{\phi\} s \{\phi'\}$$

$$\stackrel{\text{def}}{\iff}$$

$$\forall \pi, \pi'. \pi \xrightarrow{s} \pi' \wedge \pi \models \phi \implies \pi' \models \phi'$$

Approach

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness



Gradualization

Syntax

$\tilde{s} \in \tilde{\text{STMT}}$

$\tilde{\phi} \in \tilde{\text{FORMULA}}$

Program State

$\tilde{\pi} \in \tilde{\text{PROGRAMSTATE}}$

Semantics

Static $\tilde{\vdash} \{\tilde{\phi}\} \tilde{s} \{\tilde{\phi}\}$

Dynamic $\tilde{\pi} \Longrightarrow \tilde{\pi}$

Formula $\tilde{\pi} \tilde{\models} \tilde{\phi}$

Soundness

Gradualization – Goal 1/3

Introduction of wildcard formula ?

- placeholder for arbitrary (satisfiable) formula
- enables Hoare deduction despite incomplete information
- enables gradual annotation of programs (? as default)

Formula Precision

$$\widetilde{\phi}_1 \sqsubseteq \widetilde{\phi}_2 \stackrel{\text{def}}{\iff} \gamma(\widetilde{\phi}_1) \subseteq \gamma(\widetilde{\phi}_2)$$

Gradualization – Goal 2/3

Compatibility with static language

- don't reject source code that was accepted before
- observable behavior is not changed

static

dynamic

$$\text{FORMULA} \subseteq \tilde{\text{FORMULA}} \quad \text{STMT} \subseteq \tilde{\text{STMT}}$$

$$\vdash \{\phi\} \text{ } s \text{ } \{\phi'\} \implies \tilde{\vdash} \{\phi\} \text{ } s \text{ } \{\phi'\}$$

$$\pi \xrightarrow{s} \pi' \implies \exists \pi''. \pi \xrightarrow{\tilde{s}} \pi'' \wedge (\forall \phi. \pi' \models \phi \implies \pi'' \models \phi)$$

Gradualization – Goal 3/3

Gradual guarantee (Siek et al.), adapted

Reducing precision will not

- introduce verification failure
- change observable behavior

static

dynamic

Given $\widetilde{\phi}_1 \sqsubseteq \widetilde{\phi}_2 \wedge \widetilde{\phi}'_1 \sqsubseteq \widetilde{\phi}'_2 \wedge \widetilde{s}_1 \sqsubseteq \widetilde{s}_2 \wedge \widetilde{\pi}_1 \sqsubseteq \widetilde{\pi}_2$

$\widetilde{\vdash} \{\widetilde{\phi}_1\} \widetilde{s}_1 \{\widetilde{\phi}'_1\} \implies \widetilde{\vdash} \{\widetilde{\phi}_2\} \widetilde{s}_2 \{\widetilde{\phi}'_2\}$

$\widetilde{\pi}_1 \xrightarrow{\widetilde{s}_1} \widetilde{\pi}'_1 \implies \exists \widetilde{\pi}'_2. \widetilde{\pi}_2 \xrightarrow{\widetilde{s}_2} \widetilde{\pi}'_2 \wedge \widetilde{\pi}'_1 \sqsubseteq \widetilde{\pi}'_2$

Gradual Predicate Lifting

Introduction

$$\forall \phi_1, \phi_2 \in \text{FORMULA}. P(\phi_1, \phi_2) \implies \tilde{P}(\phi_1, \phi_2)$$

Monotonicity

$$\forall \widetilde{\phi}_1, \widetilde{\phi}_2, \widetilde{\phi}'_1, \widetilde{\phi}'_2 \in \widetilde{\text{FORMULA}}. \widetilde{\phi}_1 \sqsubseteq \widetilde{\phi}'_1 \wedge \widetilde{\phi}_2 \sqsubseteq \widetilde{\phi}'_2 \wedge \tilde{P}(\widetilde{\phi}_1, \widetilde{\phi}_2) \implies \tilde{P}(\widetilde{\phi}'_1, \widetilde{\phi}'_2)$$

$$P(\phi_1, \phi_a, \phi_2) \stackrel{\text{def}}{=} \phi_1 = \phi_2 \wedge \phi_1 \Rightarrow \phi_a$$

$$\tilde{P}(\widetilde{\phi}_1, \widetilde{\phi}_a, \widetilde{\phi}_2) \stackrel{\text{def}}{=} \widetilde{\phi}_1 \approx \widetilde{\phi}_2 \wedge \widetilde{\phi}_1 \widetilde{\Rightarrow} \widetilde{\phi}_a$$

$$\widetilde{\phi}_1 \approx \widetilde{\phi}_2 \stackrel{\text{def}}{=} \widetilde{\phi}_1 = \widetilde{\phi}_2 \vee \widetilde{\phi}_1 = ? \vee \widetilde{\phi}_2 = ?$$

$$\widetilde{\phi}_1 \widetilde{\Rightarrow} \widetilde{\phi}_2 \stackrel{\text{def}}{=} \widetilde{\phi}_1 \Rightarrow \widetilde{\phi}_2 \vee \widetilde{\phi}_1 = ? \vee \widetilde{\phi}_2 = ?$$

$$\frac{\phi \Rightarrow \phi_a}{\vdash \{\phi\} \text{ assert } \phi_a \{\phi\}} \text{HASSERT}$$

$$\frac{}{\widetilde{\vdash} \{\widetilde{\phi}_1\} \text{ assert } \widetilde{\phi}_a \{\widetilde{\phi}_2\}} \widetilde{\text{HASSERT}}$$

Gradual Predicate Lifting

Introduction

$$\forall \phi_1, \phi_2 \in \text{FORMULA}. P(\phi_1, \phi_2) \implies \tilde{P}(\phi_1, \phi_2)$$

Monotonicity

$$\forall \widetilde{\phi}_1, \widetilde{\phi}_2, \widetilde{\phi}'_1, \widetilde{\phi}'_2 \in \widetilde{\text{FORMULA}}. \widetilde{\phi}_1 \sqsubseteq \widetilde{\phi}'_1 \wedge \widetilde{\phi}_2 \sqsubseteq \widetilde{\phi}'_2 \wedge \tilde{P}(\widetilde{\phi}_1, \widetilde{\phi}_2) \implies \tilde{P}(\widetilde{\phi}'_1, \widetilde{\phi}'_2)$$

(Optimality)

\tilde{P} is smallest predicate closed under above rules



$$\tilde{P}(\widetilde{\phi}_1, \widetilde{\phi}_2) \iff \exists \phi_1 \in \gamma(\widetilde{\phi}_1), \phi_2 \in \gamma(\widetilde{\phi}_2). P(\phi_1, \phi_2)$$

Gradual Predicate Lifting

$$\begin{aligned}
 & \tilde{\pi} \tilde{\models} \tilde{\phi} \\
 \iff & \exists \pi \in \gamma(\tilde{\pi}), \phi \in \gamma(\tilde{\phi}). \pi \models \phi \\
 \iff & \exists \phi \in \gamma(\tilde{\phi}). \tilde{\pi} \models \phi \\
 \iff & \tilde{\pi} \models \tilde{\phi} \vee \tilde{\phi} = ?
 \end{aligned}$$

$$\frac{\tilde{\pi} \models \phi}{\tilde{\pi} \tilde{\models} \phi} \text{ EVALPHISTATIC}$$

$$\frac{}{\tilde{\pi} \tilde{\models} ?} \text{ EVALPHISTATIC}$$



$$\tilde{P}(\tilde{\phi}_1, \tilde{\phi}_2) \iff \exists \phi_1 \in \gamma(\tilde{\phi}_1), \phi_2 \in \gamma(\tilde{\phi}_2). P(\phi_1, \phi_2)$$

Gradual Function Lifting

Introduction

$$\forall \phi \in \text{FORMULA}. f(\phi) \sqsubseteq \tilde{f}(\phi)$$

Monotonicity

$$\forall \tilde{\phi}_1, \tilde{\phi}_2 \in \tilde{\text{FORMULA}}. \tilde{\phi}_1 \sqsubseteq \tilde{\phi}_2 \implies \tilde{f}(\tilde{\phi}_1) \sqsubseteq \tilde{f}(\tilde{\phi}_2)$$

(Optimality)

\tilde{f} has most precise return values among all liftings

$$\Leftrightarrow \tilde{f}(\tilde{\phi}) = \alpha(\overline{f}(\gamma(\tilde{\phi}))) \quad \text{where } \langle \alpha, \gamma \rangle \text{ is } \{ \overline{f} \}\text{-partial Galois connection}$$

Gradual Partial Function Lifting

Introduction

$$\forall \phi \in \text{FORMULA} \cap \text{dom}(f). f(\phi) \sqsubseteq \tilde{f}(\phi)$$

Monotonicity

$$\forall \tilde{\phi}_1, \tilde{\phi}_2 \in \tilde{\text{FORMULA}}. \tilde{\phi}_1 \sqsubseteq \tilde{\phi}_2 \wedge \tilde{\phi}_1 \in \text{dom}(\tilde{f}) \implies \tilde{f}(\tilde{\phi}_1) \sqsubseteq \tilde{f}(\tilde{\phi}_2)$$

(Optimality)

\tilde{f} has smallest domain and most precise return values among all liftings

$$\Leftrightarrow \tilde{f}(\tilde{\phi}) = \alpha(\overline{f}(\gamma(\tilde{\phi}))) \quad \text{where } \langle \alpha, \gamma \rangle \text{ is } \{ \overline{f} \}\text{-partial Galois connection}$$

Gradual Partial Function Lifting

$$\frac{\langle \sigma, \text{assert } \phi_a; s \rangle \models \phi_a}{\langle \sigma, \text{assert } \phi_a; s \rangle \longrightarrow \langle \sigma, s \rangle} \text{SSASSERT}$$

$$\frac{\langle \sigma, \text{assert } \widetilde{\phi}_a; s \rangle \widetilde{\models} \widetilde{\phi}_a}{\langle \sigma, \text{assert } \widetilde{\phi}_a; s \rangle \widetilde{\longrightarrow} \langle \sigma, s \rangle} \text{SSASSERT}$$

$$\Leftrightarrow \quad \widetilde{f}(\widetilde{\phi}) = \alpha(\overline{f}(\gamma(\widetilde{\phi}))) \quad \text{where } \langle \alpha, \gamma \rangle \text{ is } \{ \overline{f} \} \text{-partial Galois connection}$$

Bonus: $\{ F \}$ -partial Galois connection

Definition 28 (Partial Galois connection). *Let (C, \sqsubseteq_C) and (A, \sqsubseteq_A) be two posets, \mathcal{F} a set of operators on C , $\alpha : C \rightarrow A$ a partial function and $\gamma : A \rightarrow C$ a total function. The pair $\langle \alpha, \gamma \rangle$ is an \mathcal{F} -partial Galois connection if and only if:*

1. *If $\alpha(c)$ is defined, then $c \sqsubseteq_C \gamma(\alpha(c))$, and*
2. *If $\alpha(c)$ is defined, then $c \sqsubseteq_C \gamma(a)$ implies $\alpha(c) \sqsubseteq_A a$, and*
3. *For all $F \in \mathcal{F}$ and $c \in C$, $\alpha(F(\gamma(c)))$ is defined.*

This definition can be generalized for a set \mathcal{F} of arbitrary n -ary operators.

Gradual Verification - Approach

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness

Gradualization

Syntax

$\tilde{s} \in \tilde{\text{STMT}}$

$\tilde{\phi} \in \tilde{\text{FORMULA}}$

Program State

$\tilde{\pi} \in \tilde{\text{PROGRAMSTATE}}$

Semantics

Static $\tilde{\vdash} \{\tilde{\phi}\} \tilde{s} \{\tilde{\phi}\}$

Dynamic $\tilde{\pi} \widetilde{\longrightarrow} \tilde{\pi}$

Formula $\tilde{\pi} \tilde{\models} \tilde{\phi}$

Soundness

Gradual Verification - Approach

Syntax

$s \in \text{STMT}$

$\phi \in \text{FORMULA}$

Program State

$\pi \in \text{PROGRAMSTATE}$

Semantics

Static $\vdash \{\phi\} s \{\phi\}$

Dynamic $\pi \longrightarrow \pi$

Formula $\pi \models \phi$

Soundness



Syntax

$\tilde{s} \in \tilde{\text{STMT}}$

$\tilde{\phi} \in \tilde{\text{FORMULA}}$

Program State

$\tilde{\pi} \in \tilde{\text{PROGRAMSTATE}}$

Semantics

Static $\tilde{\vdash} \{\tilde{\phi}\} \tilde{s} \{\tilde{\phi}\}$

Dynamic $\tilde{\pi} \tilde{\longrightarrow} \tilde{\pi}$

Formula $\tilde{\pi} \tilde{\models} \tilde{\phi}$

Soundness

Gradual Soundness

$$\frac{\vdash \{\phi\} s \{\phi'\}}{\models \{\phi\} s \{\phi'\}} \text{SOUNDNESS}$$

$$\begin{aligned} & \models \{\phi\} s \{\phi'\} \\ & \xLeftrightarrow{\text{def}} \\ & \forall \pi, \pi'. \pi \xrightarrow{s} \pi' \wedge \pi \models \phi \implies \pi' \models \phi' \end{aligned}$$

$$\frac{\tilde{\vdash} \{\tilde{\phi}\} \tilde{s} \{\tilde{\phi}'\}}{\tilde{\models} \{\tilde{\phi}\} \tilde{s}; \text{assert } \tilde{\phi}' \{\tilde{\phi}'\}} \tilde{\text{SOUNDNESS}}$$

$$\begin{aligned} & \tilde{\models} \{\tilde{\phi}\} \tilde{s} \{\tilde{\phi}'\} \\ & \xLeftrightarrow{\text{def}} \\ & \forall \tilde{\pi}, \tilde{\pi}'. \tilde{\pi} \xrightarrow{\tilde{s}} \tilde{\pi}' \wedge \tilde{\pi} \tilde{\models} \tilde{\phi} \implies \tilde{\pi}' \tilde{\models} \tilde{\phi}' \end{aligned}$$

$$\begin{aligned} & \tilde{\vdash} \{?\} y := 4 \{ (x = 2) \wedge (y = 4) \} \\ & \tilde{\models} \{?\} y := 4; \text{assert } (x = 2) \{ (x = 2) \wedge (y = 4) \} \end{aligned}$$

Gradual Verification – Put to the Test

$$\frac{\begin{array}{c} \widetilde{\phi}_1 \widetilde{\Rightarrow} \widetilde{\phi}_2 \\ \widetilde{\vdash} \{?\} y := 2 \{ \widetilde{\phi}_1 \} \quad \widetilde{\vdash} \{ \widetilde{\phi}_2 \} x := 3 \{ (x = 3) \wedge (y = 2) \} \end{array}}{\widetilde{\vdash} \{?\} y := 2; x := 3 \{ (x = 3) \wedge (y = 2) \}} \widetilde{\text{HSEQ}}$$

Implicit Dynamic Frames

- Gradual Frame rule