Towards Verification of GVN-CSE on a Functional Intermediate Language with System Calls

Sigurd Schneider, Sebastian Hack, Gert Smolka

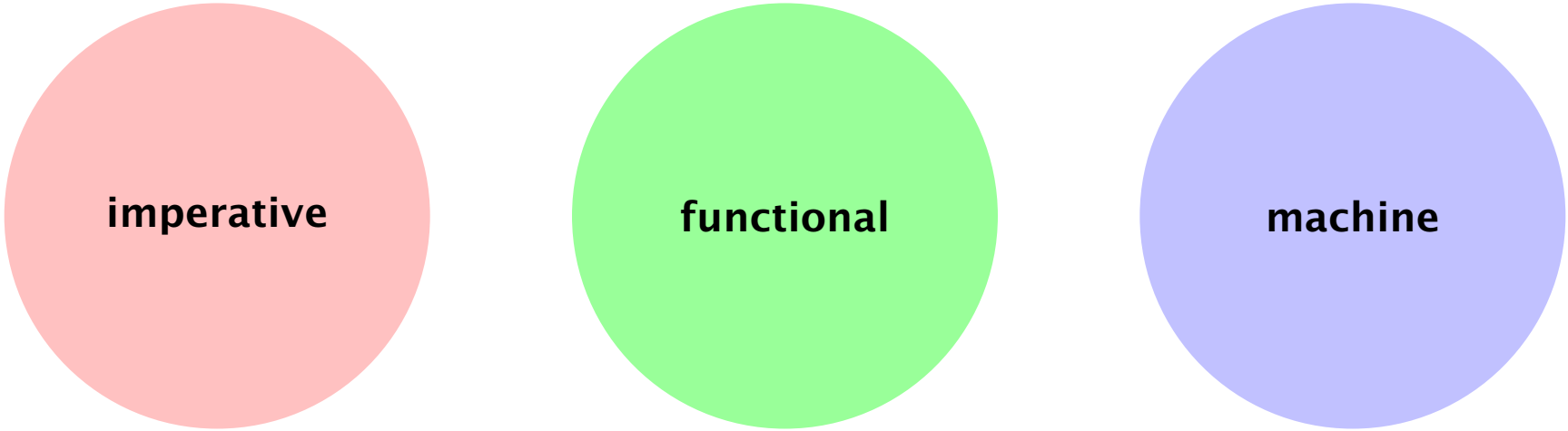
Work done during a visit to Xavier Leroy at Inria Paris-Rocquencourt

Talk at Inria Paris-Rocquencourt

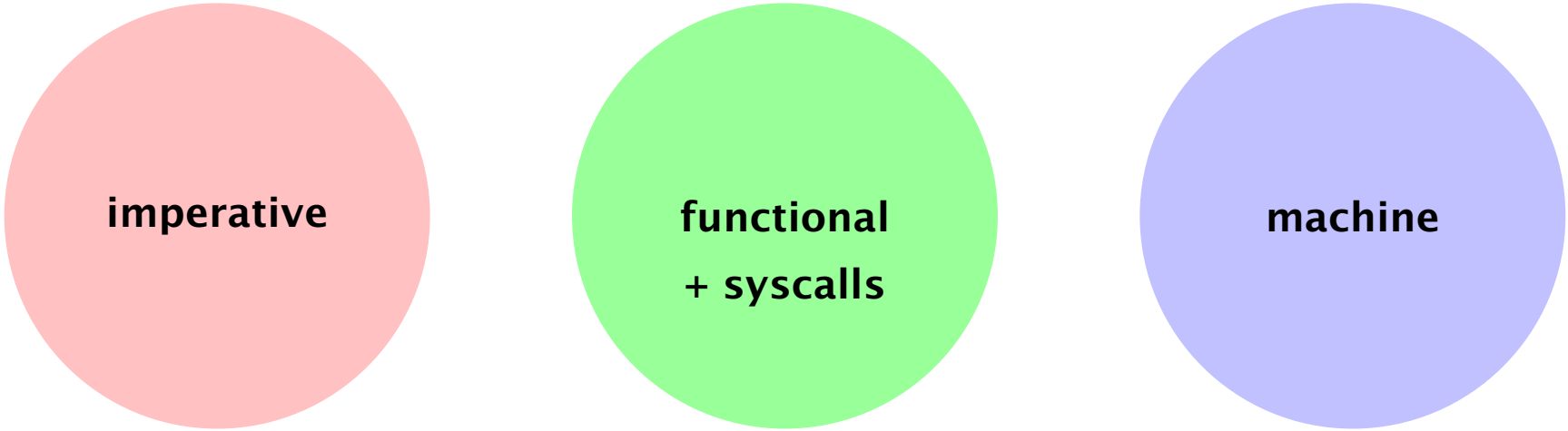
02.06.2014

SAARLAND UNIVERSITY
COMPUTER SCIENCE

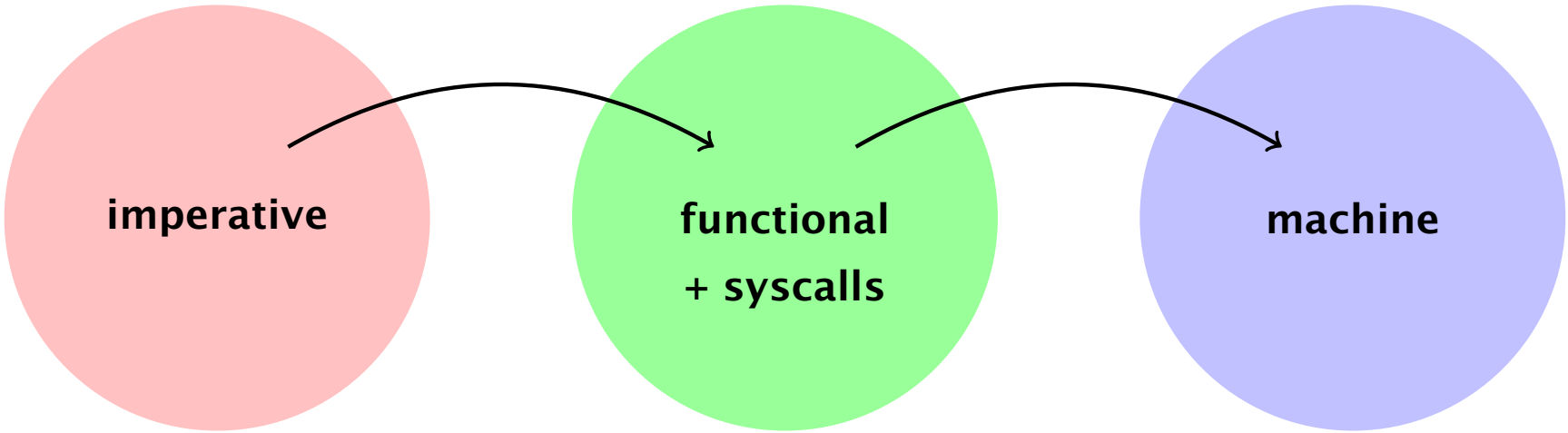
Overview



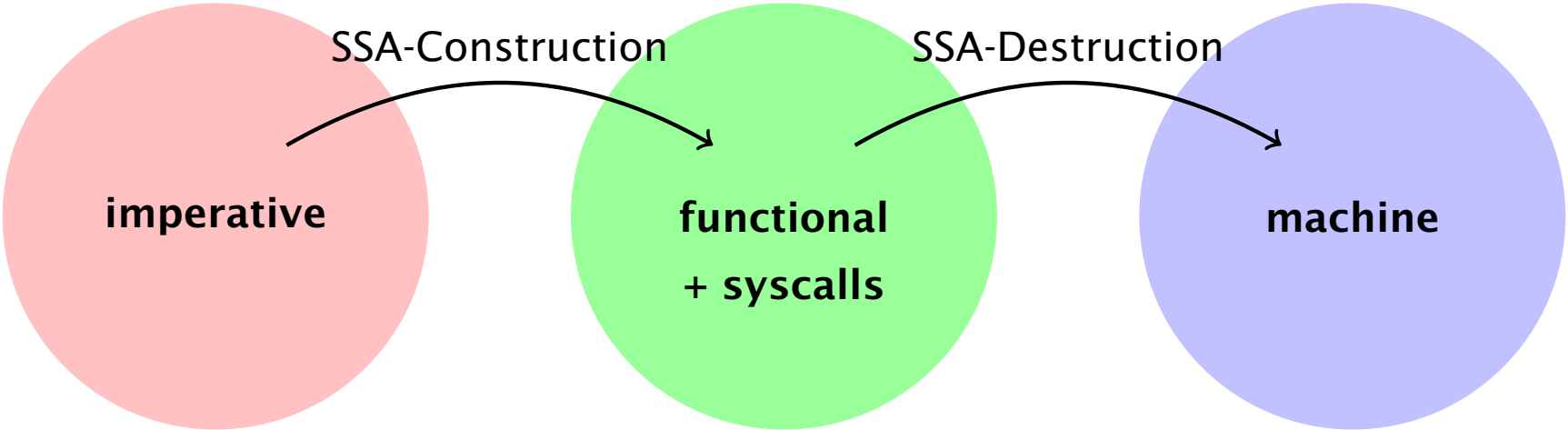
Overview



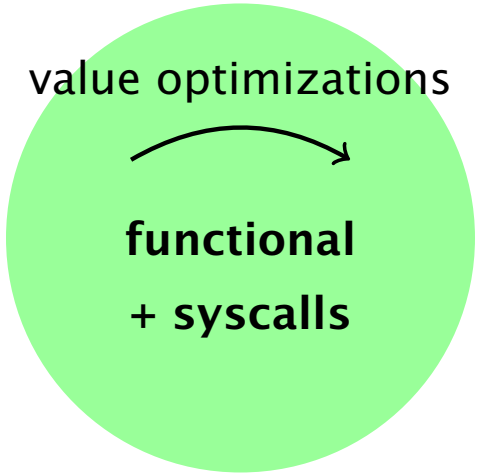
Overview



Overview


SSA-Construction
SSA-Destruction
imperative
functional
+ syscalls
machine

Overview



Overview

In this talk

1. Program equivalence
   - System calls
   - Inductive proofs
     - Allow modification of function signatures
2. Semantic specification of a value optimization
   - suitable for GVN-CSE
   - inductive correctness proof
3. Sparse Conditional Constant Propagation (SCCP) of Wegman and Zadeck (1991) in this framework

Syntax of IL

Adding system calls

$s, t ::= $	$let\ x\ =\ e\ in\ s$	let (base-types)
$\mid$	$if\ e\ then\ s\ else\ t$	conditional
$\mid$	x	value
$\mid$	$fun\ f\ \overline{x}\ =\ s\ in\ t$	recursive function
$\mid$	$f\ \overline{e}$	application
$\mid$	$let\ x\ =\ extern f\ \overline{e}\ in\ s$	extern call

- First-order functional language with tail-call restriction
- Expressions in conditionals an function applications
- Non-deterministically choose return value for extern functions

Semantics of IL/I and IL/F

Common Part

$$\text{Op}\ \frac{V \vdash e \Downarrow v}{\begin{array}{lll} & L \mid V \mid \mathit{let}\ x = e\ \mathit{in}\ s \\ \longrightarrow & L \mid V[x \mapsto v] \mid s \end{array}}$$

$$\text{If}\ \frac{\mathit{val2bool}(v) = i \qquad V \vdash e \Downarrow v}{\begin{array}{lll} & L \mid V \mid \mathit{if}\ x\ \mathit{then}\ s_0\ \mathit{else}\ s_1 \\ \longrightarrow & L \mid V \mid s_i \end{array}}$$

Semantics of IL/I and IL/F

Difference: Application Rule of IL/F

IL/F: Lexical scoping

```
1 let x = 7 in
2 fun f () = x in
3 let x = 5 in f ()
```

$$\text{IL/F-Let}\ \frac{}{L \mid V \mid \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t \longrightarrow L,\ f := (V, \overline{x}, s) \mid V \mid t}$$

$$\text{IL/F-App}\ \frac{}{L,\ f := \left(V', \overline{x}, s\right),\ L' \mid V \mid f\ \overline{y} \longrightarrow L,\ f := \left(V', \overline{x}, s\right) \mid V'^{\overline{x}}_{V\overline{y}} \mid s}$$

V': Closure Environment

V: Primary Environment

Semantics of IL/I and IL/F

Difference: Application Rule of IL/I

IL/I: Dynamic binding

```
1 x := 7;
2 fun f () = x in
3 x := 5; f ()
```

$$\text{IL/I-Let}\ \frac{}{L \mid V \mid \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t \longrightarrow L,\ f := (\overline{x}, s) \mid V \mid t}$$

$$\text{IL/I-App}\ \frac{}{L,\ f := (\overline{x}, s)\ ,\ L' \mid \boxed{V} \mid f\ \overline{y} \longrightarrow_I L,\ f := (\overline{x}, s) \mid \boxed{V}^{\overline{x}}_{V\overline{y}} \mid s}$$

V': Closure Environment

V: Primary Environment

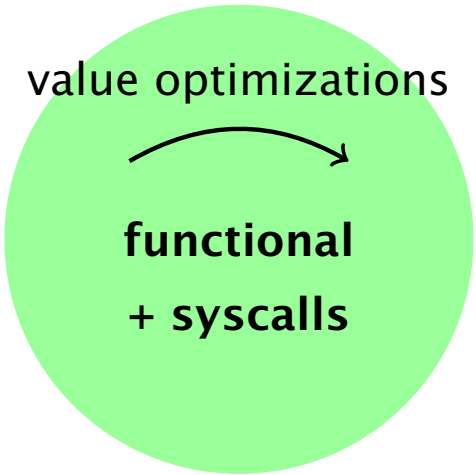
Semantics of IL/I and IL/F

External calls

$$\text{Extern}\ \frac{V \vdash \overline{e} \Downarrow \overline{v} \qquad w \in \mathbb{V}}{\begin{array}{lll} L & | \ V & | \ \mathit{let}\ x = \text{extern}\, f\, \overline{e}\ \mathit{in}\ s \\ L & | \ V_w^x & | \ s \end{array}}\ {}^{(f,\overline{v},w)}\!\longrightarrow$$

- Introduces non-determinism (any $w \in \mathbb{V}$ will do)
- *external* non-determinism: If observation know, deterministic
- *internal* non-determinism is not allowed
  - And our definitions exploit that

Overview



1. Program Equivalence
   - Bisimilarity
   - Similarity

Program Equivalence

Bisimilarity and Similarity

$$\text{BS}\ \frac{\sigma_1 \longrightarrow^+ \sigma_1' \qquad \sigma_2 \longrightarrow^+ \sigma_2' \qquad \sigma_1' \sim \sigma_2'}{\sigma_1 \sim \sigma_2}$$

$$\text{BC}\ \frac{v \in R \cup \{\bot\} \qquad \sigma_1 \Downarrow v \qquad \sigma_2 \Downarrow v}{\sigma_1 \sim \sigma_2}$$

$$\text{BN}\ \frac{\begin{array}{ll} \sigma_1, \sigma_2 \text{ activated} & \\ \sigma_1 \longrightarrow^+ \sigma_1' & \forall \sigma_1'', \sigma_1' \xrightarrow{o} \sigma_1'' \Rightarrow \exists \sigma_2'', \sigma_2' \xrightarrow{o} \sigma_2'' \wedge \sigma_1'' \sim \sigma_2'' \\ \sigma_2 \longrightarrow^+ \sigma_2' & \forall \sigma_2'', \sigma_2' \xrightarrow{o} \sigma_2'' \Rightarrow \exists \sigma_1'', \sigma_1' \xrightarrow{o} \sigma_1'' \wedge \sigma_1'' \sim \sigma_2'' \end{array}}{\sigma_1 \sim \sigma_2}$$

σ activated if there is a next step, and it is not silent

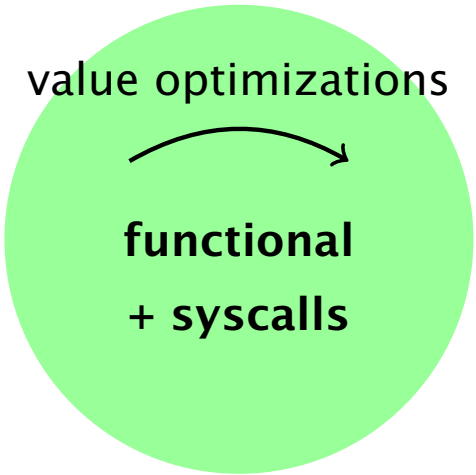
Program Equivalence

Bisimilarity and Similarity

$$\text{BE}\ \frac{\sigma_1 \Downarrow \bot}{\sigma_1 \sim \sigma_2}$$

- Bisimilarity (BS,BC,BN) preserves error
- Similarity (BS,BC,BN,BE) is an implementation relation
  - If first program errs, no further restriction on second
  - External events up to error must be preserved

Overview



1. Program Equivalence
   - Bisimilarity
   - Similarity
2. Inductive Proof Technique

Inductive Proof Technique

What programs do we want to prove similar?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
1 let x = 5 + 9 in
2 fun f (y) = y in
3 f (x)
```

```
1 fun f () = 14 in
2 f ()
```

- Constant folding + parameter elimination
- Suppose: proof by induction on the program
- Induction hypothesis must be strong enough to show that $f(x)$ (left) is equivalent to $f()$ (right)

Inductive Proof Technique

Extension Lemma Enables Inductive Proofs

- $\mathcal{A}$ relates arguments, $\mathcal{P}$ relates parameters each indexed by analysis information a
- $f := (\overline{x}, E, s), L \quad \approx_{a,A} \quad f' := (\overline{x'}, E', s'), L'$ if
  - parameters are related: $\mathcal{P}\, a\, \overline{x}\, \overline{x'}$
  - for $\mathcal{A}$ a-related arguments, f, f' equivalent (similar)
  - inductively, $L \approx_A L'$
- $\overline{x}, E, s \quad \sim_{a,A} \quad \overline{x'}, E', s'$ if $\mathcal{P}\, a\, \overline{x}\, \overline{x'}$ and for all $\overline{y}\, \overline{y'}\, L\, L'$

$$\mathcal{A}\, a\, \overline{y}\, \overline{y'} \Rightarrow L \approx_{a,A} L' \Rightarrow (L, E[\overline{x} \mapsto \overline{y}], s) \quad \sim \quad (L', E'[\overline{x'} \mapsto \overline{y'}], s')$$

Lemma (Extension)

If $L \approx_A L'$ *and* $\overline{x}, E, s \quad \sim_{a,A} \quad \overline{x'}, E', s'$ *then*

$$f := (\overline{x}, E, s), L \quad \approx_{a,A} \quad f' := (\overline{x'}, E', s'), L'$$

Inductive Proof Technique

Extension Lemma

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Suppose equivalence proof by induction on the syntax:
   $L \approx_A L' \Rightarrow (L,\ E,\ s) \sim (L',\ E',\ s')$

Inductive Proof Technique

Extension Lemma

1. Suppose equivalence proof by induction on the syntax:
   $L \approx_A L' \Rightarrow (L,\ E,\ s) \sim (L',\ E',\ s')$
2. Critical case is function definition:
   $(L,\ E,\ \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t) \sim (L',\ E',\ \mathit{fun}\ f\ \overline{x'}\ =\ s'\ \mathit{in}\ t')$

Inductive Proof Technique

Extension Lemma

1. Suppose equivalence proof by induction on the syntax:
   $L \approx_A L' \Rightarrow (L,\ E,\ s) \sim (L',\ E',\ s')$
2. Critical case is function definition:
   $(L,\ E,\ \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t) \sim (L',\ E',\ \mathit{fun}\ f\ \overline{x'}\ =\ s'\ \mathit{in}\ t')$
3. $\sim$ closed under expansion, reduce one step (left and right):
   $(f := (\overline{x}, E, s)\,;L,\ E,\ t) \sim (f := (\overline{x'}, E', s')\,;L',\ E',\ t')$

Inductive Proof Technique

Extension Lemma

1. Suppose equivalence proof by induction on the syntax:
   $L \approx_A L' \;\Rightarrow\; (L,\ E,\ s) \sim (L',\ E',\ s')$
2. Critical case is function definition:
   $(L,\ E,\ \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t) \sim (L',\ E',\ \mathit{fun}\ f\ \overline{x'}\ =\ s'\ \mathit{in}\ t')$
3. $\sim$ closed under expansion, reduce one step (left and right):
   $(f := (\overline{x}, E, s)\,;L,\ E,\ t) \sim (f := (\overline{x'}, E', s')\,;L',\ E',\ t')$
4. Apply IH for t, but now have to show
   $f := (\overline{x}, E, s)\,, L \quad \approx_{a,A} \quad f' := (\overline{x'}, E', s')\,, L'$

Inductive Proof Technique

Extension Lemma

1. Suppose equivalence proof by induction on the syntax:
   $L \approx_A L' \;\Rightarrow\; (L,\ E,\ s) \sim (L',\ E',\ s')$
2. Critical case is function definition:
   $(L,\ E,\ \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t) \sim (L',\ E',\ \mathit{fun}\ f\ \overline{x'}\ =\ s'\ \mathit{in}\ t')$
3. $\sim$ closed under expansion, reduce one step (left and right):
   $(f := (\overline{x}, E, s)\,; L,\ E,\ t) \sim (f := (\overline{x'}, E', s')\,; L',\ E',\ t')$
4. Apply IH for t, but now have to show
   $f := (\overline{x}, E, s)\,, L \quad \approx_{a,A} \quad f' := (\overline{x'}, E', s')\,, L'$
5. Apply extension lemma, reduces goal to: For all $y\ y'\ L_1\ L_1'$
   $\mathcal{A}\ a\ y\ y' \;\Rightarrow\; L_1 \approx_{a,A} L_1' \;\Rightarrow\; (L_1, E[\overline{x} \mapsto \overline{y}], s) \;\sim\; (L_1', E'[\overline{x'} \mapsto \overline{y'}], s')$

Inductive Proof Technique

Extension Lemma

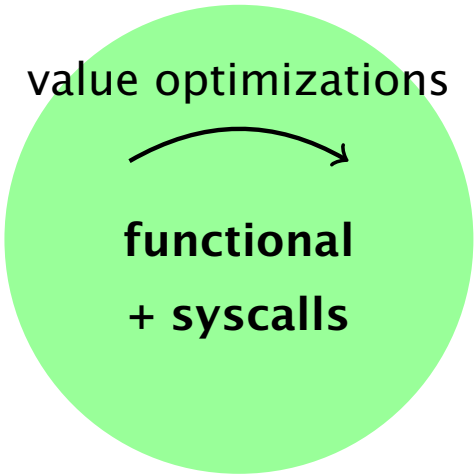
1. Suppose equivalence proof by induction on the syntax:
   $L \approx_A L' \;\Rightarrow\; (L,\ E,\ s) \sim (L',\ E',\ s')$
2. Critical case is function definition:
   $(L,\ E,\ \mathit{fun}\ f\ \overline{x}\ =\ s\ \mathit{in}\ t) \sim (L',\ E',\ \mathit{fun}\ f\ \overline{x'}\ =\ s'\ \mathit{in}\ t')$
3. $\sim$ closed under expansion, reduce one step (left and right):
   $(f := (\overline{x}, E, s)\,; L,\ E,\ t) \sim (f := (\overline{x'}, E', s')\,; L',\ E',\ t')$
4. Apply IH for t, but now have to show
   $f := (\overline{x}, E, s)\,, L \quad \approx_{a,A} \quad f' := (\overline{x'}, E', s')\,, L'$
5. Apply extension lemma, reduces goal to: For all $y\ y'\ L_1\ L_1'$
   $\mathcal{A}\ a\ y\ y' \;\Rightarrow\; L_1 \approx_{a,A} L_1' \;\Rightarrow\; (L_1, E[\overline{x} \mapsto \overline{y}], s) \;\sim\; (L_1', E'[\overline{x'} \mapsto \overline{y'}], s')$
6. Discharge by IH for s

Inductive Proof Technique

Extension Lemma

- Extension lemma works for
  - both bisimilarity and similarity
  - IL/F (shown before)
  - IL/I (analogous, but slightly different definition)
- Examples proven with extension lemma
  - Dead variable elimination
  - Value Optimizations (up next)
- Optimizations that remove code are no problem

Overview



1. Program Equivalence
   - Bisimilarity
   - Similarity
2. Inductive Proof Technique
3. Value Optimizations

Value Optimizations

Considerations for a correctness criterion for value optimizations

- When can an expression e be replaced by an expression e'?

  If e' evaluates in at least the environments occurring during program execution to the same value as e

- What if e does not evaluate to a value?

  Impose no requirement on e'

- How to characterize the environments that occur during program execution?

  Restrict to environments that satisfy a certain set of equations

Value Optimizations

Some semantic definitions

- Define $E \models e = e'$ to hold if e and e' evaluate to the same value or are both undefined under E
- Lift to sets of equations Γ in point-wise manner: $E \models \Gamma$
- Define $E \models e \sqsubseteq e'$ to hold if whenever e evaluates to a value under E, then e' evaluates to the same value
- Define approximation $\Gamma \models e \sqsubseteq e'$ as

$$\forall E, \quad E \models \Gamma \quad \Longrightarrow \quad E \models e \sqsubseteq e'$$

Value Optimization

A word on decidability

- $E \models \Gamma$ decidable (for finite Γ, simple enough expression language, and finite value domain)
  - Challenge: efficient decision procedure
  - In this talk: decidability not required
- Semantic specification is suitable for a wide range of value optimizations:
  - GVN-based common subexpression elimination (not done)
  - copy propagation (done)
  - conditional constant propagation (done)

Soundness for Value Optimizations

Judgement

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$\boxed{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ s\ /\ s'\ :\ \Gamma}$$

Λ	information about functions
Δ	set of defined variables
s	source program
s'	translated program
Γ	set of equations

Under assumptions Λ about the functions appearing in s and s',
and assuming all variables in Δ are defined
s' simulates s
in every environment that satisfies the equations Γ

- inductively defined
- requires s to be renamed apart

Soundness for Value Optimizations

Rules

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$\boxed{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ s\ /\ s'\ :\ \Gamma}$$

$$\text{Op}\ \frac{\Gamma \vDash e \sqsubseteq e' \qquad \mathcal{V}(e') \subseteq \Delta \qquad \Lambda \mid \Delta \cup \{x\} \vdash \textbf{\textit{vopt}}\ s\ /\ s'\ :\ \{x = e, x = e'\} \cup \Gamma}{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ \textit{let}\ x\ =\ e\ \textit{in}\ s\ /\ \textit{let}\ x\ =\ e'\ \textit{in}\ s'\ :\ \Gamma}$$

Soundness for Value Optimizations

Rules

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$\boxed{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ s\ /\ s'\ :\ \Gamma}$$

$$\text{Op}\ \frac{\Gamma \vDash e \sqsubseteq e' \qquad \mathcal{V}(e') \subseteq \Delta \qquad \Lambda \mid \Delta \cup \{x\} \vdash \textbf{\textit{vopt}}\ s\ /\ s'\ :\ \{x = e, x = e'\} \cup \Gamma}{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ \textit{let}\ x\ =\ e\ \textit{in}\ s\ /\ \textit{let}\ x\ =\ e'\ \textit{in}\ s'\ :\ \Gamma}$$

- Γ may become inconsistent
  - No E with $E\,y = 0$ will satisfy $\{x = y \cdot y/y, x = y\}$
  - But if $E\,y = 0$ behavior undefined after executing $y \cdot y/y$
  - Strengthens what $\Gamma \vDash e \sqsubseteq e'$ means here

Soundness for Value Optimizations

Rules

$$\textsf{Fun}\ \frac{\Lambda,\ f : (\overline{x}, \Delta, \Gamma_f, \Gamma') \mid \Delta \vdash \textbf{\textit{vopt}}\ t\ /\ t'\ :\ \Gamma \qquad \Gamma \Rightarrow \Gamma' \qquad \mathcal{V}(\Gamma_f) \subseteq \Delta \cup \overline{x} \qquad \Lambda,\ f : (\overline{x}, \Delta, \Gamma_f, \Gamma') \mid \Delta \cup \overline{x} \vdash \textbf{\textit{vopt}}\ s\ /\ s'\ :\ \Gamma' \cup \Gamma_f \qquad \mathcal{V}(\Gamma') \subseteq \Delta}{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ \textit{fun}\ f\ \overline{x}\ =\ s\ \textit{in}\ t\ /\ \textit{fun}\ f\ \overline{x}\ =\ s'\ \textit{in}\ t'\ :\ \Gamma}$$

```
1 fun f (y) = y in
2 let x = ... in
3 f (x)
```

- $\mathcal{V}(\Gamma_f) \subseteq \Delta \cup \overline{x}$ required for scoping
- $\mathcal{V}(\Gamma') \subseteq \Delta$ because Γ could be inconsistent

Soundness for Value Optimizations

Rules

$$\textsf{App}\ \frac{\Gamma \vDash \overline{y} \sqsubseteq \overline{y'} \qquad \Gamma \Rightarrow \Gamma_f[\overline{x} \mapsto \overline{y'}]}{\Lambda,\ f : (\overline{x}, \Delta_f, \Gamma_f, \Gamma'_f),\ \Lambda' \mid \Delta \vdash \textbf{\textit{vopt}}\ f\ \overline{y}\ /\ f\ \overline{y'}\ :\ \Gamma}$$

- $\Gamma \vDash \overline{y} \sqsubseteq \overline{y'}$ means either both lists fully defined and equivalent, or first list contains at least one expression that is undefined
- $\Gamma_f[\overline{x} \mapsto \overline{y'}]$ does not mention parameters anymore

Soundness for Value Optimizations

Rules

$$\text{Cond}\ \frac{\Gamma \not\models e \neq 0 \Rightarrow \Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ s\ /\ s' \ :\ \Gamma \cup \{e \neq 0\} \qquad \Gamma \not\models e = 0 \Rightarrow \Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ t\ /\ t' \ :\ \Gamma \cup \{e = 0\} \qquad \Gamma \models e \sqsubseteq e'}{\Lambda \mid \Delta \vdash \textbf{\textit{vopt}}\ \textit{if}\ e\ \textit{then}\ s\ \textit{else}\ t\ /\ \textit{if}\ e'\ \textit{then}\ s'\ \textit{else}\ t' \ :\ \Gamma}$$

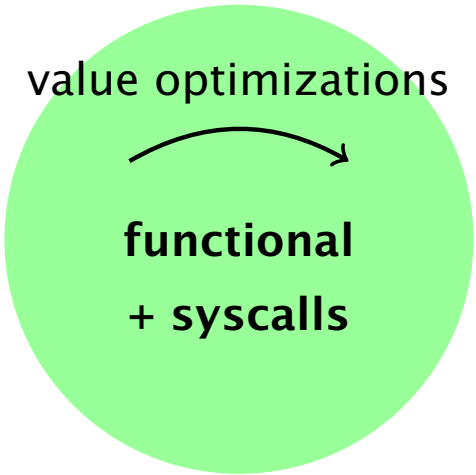
- Supports *conditional* analyses
  - If according to Γ, a branch is unreachable, no proof obligation
  - Exploit knowledge about value of condition inside branch

Value Optimizations

Correctness proofs for value optimization

- Proof without ***vopt***
  1. Specify sound analysis result (s.t. efficiently decidable)
  2. Write transformation
  3. Prove correctness semantically
- Proof with ***vopt***
  1. Specify sound analysis result (s.t. efficiently decidable)
  2. Write transformation
  3. Prove that ***vopt*** holds given sound analysis result
- Examples proven with ***vopt***
  - Copy Propagation
  - Sparse Conditional Constant Propagation (in this talk)
  - GVN-CSE (not finished yet)

Overview



1. Program Equivalence
   - Bisimilarity
   - Similarity
2. Inductive Proof Technique
3. Value Optimizations
4. Constant Propagation

Constant Propagation with Conditional Branches

MARK N. WEGMAN and F. KENNETH ZADECK
IBM T. J. Watson Research Center

- POPL 1991: Sparse Conditional Constant Propagation (SCCP)
- Sparse
  - Analysis tracks one global mapping $\mathcal{V} \to \mathbb{V} \cup \{\top\}$
  - Sparseness is one of the original SSA-promises (and we achieve it)
- Conditional
  - Use analysis information to identify unreachable branches in conditionals *during fixpoint computation*
  - Accommodated by Cond-rule
- More powerful than applying dead code elimination and constant propagation individually

Constant Propagation with Conditional Branches

MARK N. WEGMAN and F. KENNETH ZADECK
IBM T. J. Watson Research Center

Conditional

```
1 fun f (x) =
2   if x then x
3   else f (x - 1)
4 in f (5)
```

↓ Optimization

```
1 fun f () = 5
2 in f ()
```

Constant Propagation with Conditional Branches

MARK N. WEGMAN and F. KENNETH ZADECK
IBM T. J. Watson Research Center

Conditional

```
1 fun f (x) =
2   if x then x
3   else f (x - 1)
4 in f (5)
```

↓ Optimization

```
1 fun f () = 5
2 in f ()
```

Uses conditions

```
1 if x = 5 then x
2 else y
```

↓ Optimization

```
1 if x = 5 then 5
2 else y
```

Sparse Conditional Constant Propagation

How do we do it?

- ***vopt*** does not support dead code elimination
- Two phases: optimization + dead variable elimination (DVE)
  - Analysis is conditional
  - Optimization leaves the conditional intact, but replaces the condition with constant
  - Run DVE afterwards (removes conditionals with constant condition)
  - Same principle for dead variables
- DVE is proven with extension lemma by induction (material for another talk)

Sparse Conditional Constant Propagation

Example

```
1 let x = 5 + 9 in
2 fun f (y) = y in
3 f (x)
```

↓ SCCP

```
1 let x = 14 in
2 fun f (y) = 14 in
3 f (14)
```

↓ DVE

```
1 fun f () = 14 in
2 f ()
```

Soundness for SCCP

Judgement

$$\boxed{\Lambda \vdash \textbf{\textit{sccp}}\ s : \kappa}$$

Λ	information about functions
κ	constants $\mathcal{V} \to \mathbb{V} \cup \{\top\}$
s	source program

Under assumptions Λ about the functions appearing in s
variables in s can be replaced according to κ
in every environment that refines κ on $\mathcal{V}(s)$

- inductively defined
- requires s to be renamed apart

Soundness for Value Optimizations

Rules

$$\text{Cond}\ \frac{\begin{array}{l} \mathit{val2bool}([\![e]\!]\kappa) \neq 0 \Rightarrow \Lambda \ \vdash \textbf{\textit{sccp}}\ s\ :\ \mathit{upd}\,\kappa\,e\,1 \\ \mathit{val2bool}([\![e]\!]\kappa) \neq 1 \Rightarrow \Lambda \ \vdash \textbf{\textit{sccp}}\ t\ :\ \mathit{upd}\,\kappa\,e\,0 \end{array}}{\Lambda \ \vdash \textbf{\textit{sccp}}\ \mathit{if}\ e\ \mathit{then}\ s\ \mathit{else}\ t\ :\ \kappa}$$

$$\mathit{upd}\,\kappa\,e\,b = \begin{cases} \kappa[x \mapsto 0] & b = 0 \wedge e \equiv x \\ \kappa[x \mapsto c] & b = 1 \wedge e \equiv x = c \\ \kappa[x \mapsto c] & b = 0 \wedge e \equiv x \neq c \end{cases}$$

Soundness for Value Optimizations

Rules

$$\text{Cond}\ \frac{\begin{array}{l} \mathit{val2bool}([\![e]\!]\kappa) \neq 0 \Rightarrow \Lambda \vdash \textbf{\textit{sccp}}\ s : \mathit{upd}\,\kappa\,e\,1 \\ \mathit{val2bool}([\![e]\!]\kappa) \neq 1 \Rightarrow \Lambda \vdash \textbf{\textit{sccp}}\ t : \mathit{upd}\,\kappa\,e\,0 \end{array}}{\Lambda \vdash \textbf{\textit{sccp}}\ \mathit{if}\ e\ \mathit{then}\ s\ \mathit{else}\ t : \kappa}$$

- κ can be turned into a set of equations $[\![\kappa]\!]_\Delta$ (restricted to variables from Δ)
- $[\![\kappa]\!]_\Delta \not\models e = 0 \Rightarrow \mathit{val2bool}([\![e]\!]\kappa) \neq 0$
- $[\![\kappa]\!]_\Delta \cup \{e \neq 0\} \Rightarrow [\![\mathit{upd}\,\kappa\,e\,1]\!]_\Delta$
- Similar obligations for alternative branch

Conclusion

1. Inductive method scales to interesting transformations
   - Dead variable elimination
   - Value optimizations
2. Constant Propagation (SCCP) in a functional setting
   - Sparse analysis as promised by SSA
   - Inductive Proof
3. Challenges
   - GVN fixpoint analysis
   - Performance

Thank you for listening! Questions?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Barthe, Gilles, Delphine Demange, and David Pichardie (2012). "A Formally Verified SSA-Based Middle-End - Static Single Assignment Meets CompCert". In: *Programming Languages and Systems - 21st European Symposium on Programming*. Ed. by Helmut Seidl. LNCS. Tallinn, Estonia.

Leroy, Xavier (2009). "A Formally Verified Compiler Back-end". In: *JAR* 4.

Wegman, Mark N. and F. Kenneth Zadeck (1991). "Constant Propagation with Conditional Branches". In: *ACM Trans. Program. Lang. Syst.* 2. issn: 0164-0925. doi: 10.1145/103135.103136.

Abstract

We present a semantic specification of a common subexpression elimination (CSE) based on global value numbering (GVN) in the setting of a functional intermediate language. The language includes system calls and expressions that may err (division by zero). Semantic preservation for GVN-CSE is shown via induction. So far, we have implemented and verified copy propagation and constant folding against the specification; the implementation of GVN analysis is in progress.