

Simple Relational Correctness Proofs for Static Analyses and Program Transformations

By Nick Benton

Enrico Steffinlongo

Università Ca' Foscari - Computer science

May 29, 2015

Soundness of Program optimization

Lot of work on functional languages especially in

- formalization
- validation

Few work on imperative programming languages

- seems trivial
- ... but it's not

This work is to propose three systems (one type based, and two Hoare-logics based) to prove correctness of optimization transformations.

Optimization transformations

Transformation of a program to a semantically equivalent one in order to reduce the time used, or to decrease the resources used. Typical imperative program optimization includes:

- constant propagation
- dead-code elimination
- program slicing
- loop unrolling

X := 7;		X := 7;		X := 7;
Y := Y + 1;		Y := Y + 1;		Y := Y + 1;
X := 7;	\Rightarrow	X := 7;	\Rightarrow	
Z := X;		Z := 7;		Z := 7;

Optimization transformations: examples

<hr/>	
X := 3	
if X = 3 then	
X := 7;	
else	==>
skip;	X := 7;
Z := X + 1;	Z := 8;
<hr/>	
if X = 3 then	
Y := X;	
else	==>
Y := 3;	Y := 3
<hr/>	
X := -Y	X := Y
Z := Z - X	==> Z := Z + X
X := -X	
<hr/>	

Table : Transformation examples

The while-language: syntax

In this work we will use the while-language.

\mathbb{V}	$=$	$\{X, Y, \dots\}$	a set of variables
n	\in	\mathbb{Z}	a number,
b	\in	\mathbb{B}	a boolean literal
iop	\in	$\{+, -, \times, \dots\} \subseteq \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$	an integer operation
bop	\in	$\{<, =, \dots\} \subseteq \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$	an integer to boolean operation
lop	\in	$\{\wedge, \vee, \dots\} \subseteq \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$	a logical operation
E	$:=$	$n X E \ iop \ E$	integer expressions
B	$:=$	$b E \ bop \ E \text{not } B B \ lop \ B$	boolean expressions
C	$:=$	$\text{skip} X := E C; C$ $ \text{if } B \text{ then } C \text{ else } C$ $ \text{while } B \text{ do } C$	commands
S	\in	$\mathbb{S} = \mathbb{V} \rightarrow \mathbb{Z}$	A valid State

- Denotational semantics of Integer expression (similar to the one for Boolean expression)

$$\begin{aligned}\llbracket E \rrbracket &\in \mathbb{S} \rightarrow \llbracket \text{int} \rrbracket = \mathbb{S} \rightarrow \mathbb{Z} \\ \llbracket n \rrbracket S &= n \\ \llbracket X \rrbracket S &= S(X) \\ \llbracket E_1 \text{ iop } E_2 \rrbracket S &= (\llbracket E_1 \rrbracket S) \text{ iop } (\llbracket E_2 \rrbracket S) \\ \llbracket n \rrbracket S &= n\end{aligned}$$

- Denotational semantics of commands

$$\begin{aligned}\llbracket C \rrbracket &\in \mathbb{S} \rightarrow \mathbb{S}_\perp \\ \llbracket \text{skip} \rrbracket &= \lambda S. [S] \\ \llbracket X := E \rrbracket &= \lambda S. [S[X \mapsto \llbracket E \rrbracket S]] \\ \llbracket C_1; C_2 \rrbracket &= \llbracket C_2 \rrbracket^* \circ \llbracket C_1 \rrbracket \\ \llbracket \text{if } B \text{ then } C_1 \text{ else } C_2 \rrbracket &= \lambda S. \llbracket B \rrbracket S \Rightarrow \llbracket C_1 \rrbracket \mid \llbracket C_2 \rrbracket \\ \llbracket \text{while } B \text{ do } C \rrbracket &= \text{fix } f. \lambda S. \llbracket B \rrbracket S \Rightarrow f^*(\llbracket C \rrbracket S) \mid [S]\end{aligned}$$

Dependency, Dead Code and Constant (DDCC)

The DDCC type system is used to prove correctness of some program transformations.

- non-standard type system
- it derives typed equality between expressions and commands
- it works on pairs of programs
- has simple types for expressions
- has maps from variables to simple types for states
- can be seen as a non-interference type system
- it captures only decisions based on known variables. So it is not able to capture patterns like in example 2 in table {1}
- does not capture code-motion transformation

Using DDCC we can prove the equivalence of the example 1 in table {1}

A simple type $\phi_\tau := \mathbb{F}_\tau \mid \{c\}_\tau \mid \Delta_\tau \mid \mathbb{T}_\tau$ where $\tau \in \{\text{int}, \text{bool}\}$ and c is a constant.

- \mathbb{F}_τ is an empty type
- $\{c\}_\tau$ is the type of a constant c ($5 \ 5 \in \{5, 5\}_{\text{int}}$)
- Δ_τ is the type of an unknown expression (if we do not know the value of X ($X + X$) $\in \Delta_{\text{int}}$)
- \mathbb{T}_τ is the type of an expression that we do not care any more.

A state type $\Phi := - \mid \Phi, X : \phi_{\text{int}}$ is a map from variable to simple types.

Judgements are of the form

- $\vdash E \sim E' : \Phi \Rightarrow \phi_\tau$ for expressions,
- $\vdash C \sim C' : \Phi \Rightarrow \Phi'$ for commands.

We will use $\vdash C : \Phi \Rightarrow \Phi$ as shorthand for $\vdash C \sim C : \Phi \Rightarrow \Phi$

Some simple DDCC core judgements for are:

- $\vdash n \sim n : \Phi \Rightarrow \{n\}_{int}$
- $\vdash X \sim X : \Phi, X : \phi_{int} \Rightarrow \phi_{int}$
- $\vdash \text{skip} \sim \text{skip} : \Phi \Rightarrow \Phi$
- $$\frac{\vdash C_1 \sim C'_1 : \Phi \Rightarrow \Phi' \quad \vdash C_2 \sim C'_2 : \Phi' \Rightarrow \Phi''}{\vdash (C_1; C_2) \sim (C'_1; C'_2) : \Phi \Rightarrow \Phi''}$$
- $$\frac{\vdash B \sim B' : \Phi \Rightarrow \Delta_{bool} \quad \vdash C \sim C' : \Phi \Rightarrow \Phi}{\vdash (\text{while } B \text{ do } C) \sim (\text{while } B' \text{ do } C') : \Phi \Rightarrow \Phi}$$

These rules are able to prove relations on the same phrase to itself.

Judgements used to prove equivalences of transformed programs.

sequential unit laws

associativity

commuting conversion for conditionals

loop unrolling

self-assignment elimination

dead-assignment elimination

equivalent branches for conditionals

constant folding

known branch for conditional

dead while

divergence for while

$$\frac{\frac{\vdash C \sim C : \Phi \Rightarrow \Phi}{\vdash (\text{skip}; C) \sim C : \Phi \Rightarrow \Phi'}}{\vdash (C_1; C_2); C_3 : \Phi \Rightarrow \Phi'} \\ \vdash ((C_1; C_2); C_3) \sim (C_1; (C_2; C_3)) : \Phi \Rightarrow \Phi'$$

$$\vdash (X := X) \sim \text{skip} : \Phi, X : \phi_{\text{int}} \Rightarrow \Phi, X : \phi_{\text{int}}$$

$$\vdash (X := E) \sim \text{skip} : \Phi, X : \phi_{\text{int}} \Rightarrow \Phi, X : \mathbb{T}_{\text{int}}$$

$$\frac{\vdash C_1 \sim C_2 : \Phi \Rightarrow \Phi'}{\text{if } B \text{ then } C_1 \text{ else } C_2 \sim C_1 : \Phi \Rightarrow \Phi'} \\ \vdash F_{\tau} : \Phi \Rightarrow \{c\}_{\tau}$$

$$\frac{\vdash F_{\tau} \sim c : \Phi \Rightarrow \{c\}_{\tau}}{\vdash B : \Phi \Rightarrow \{\text{true}\} \quad \vdash C_1 \sim C' : \Phi \Rightarrow \Phi'}$$

$$\frac{\text{if } B \text{ then } C_1 \text{ else } C_2 \sim C' : \Phi \Rightarrow \Phi'}{\vdash B : \Phi \Rightarrow \{\text{false}\}} \\ \vdash (\text{while } B \text{ do } C \sim \text{skip} : \Phi \Rightarrow \Phi)$$

To increase the capabilities of the analysis this work proposes a system based on Relational Hoare Logics. It is based on

GE	$:=$	$n \mid X\langle 1 \rangle \mid X\langle 2 \rangle \mid GE \text{ iop } GE$:	generalized expressions
Φ	$:=$	$b \mid GE \text{ bop } GE \mid \text{not} \Phi \mid \Phi \text{ lop } \Phi$:	relational assertions

Judgements are of the form:

$$\vdash C \sim C' : \Phi \Rightarrow \Phi'$$

Relational Hoare Logic: semantics

The semantics of GE , and Φ are:

$$\begin{aligned}\llbracket GE \rrbracket &\in \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{Z} \\ \llbracket n \rrbracket(S_1, S_2) &= n \\ \llbracket X \langle 1 \rangle \rrbracket(S_1, S_2) &= S_1(X) \\ \llbracket X \langle 2 \rangle \rrbracket(S_1, S_2) &= S_2(X) \\ \llbracket E \text{ iop } F \rrbracket(S_1, S_2) &= (\llbracket E \rrbracket(S_1, S_2)) \text{ iop } (\llbracket F \rrbracket(S_1, S_2)) \\ \\ \llbracket \Phi \rrbracket &\in \mathbb{S} \times \mathbb{S} \\ &= \{(S, S') \mid \chi_\Phi(S, S') = \text{true}\} \\ \chi_{\text{true}}(S, S') &= \text{true} \\ \chi_{\text{false}}(S, S') &= \text{false} \\ \chi_{E \text{ bop } F}(S, S') &= (\llbracket E \rrbracket(S_1, S_2)) \text{ bop } (\llbracket F \rrbracket(S_1, S_2)) \\ \chi_{\Phi \text{ lop } \Phi'}(S, S') &= (\chi_\Phi(S_1, S_2)) \text{ lop } (\chi_{\Phi'}(S_1, S_2)) \\ \chi_{\text{not } \Phi}(S, S') &= \neg(\chi_\Phi(S_1, S_2))\end{aligned}$$

Some simple RHL core judgements for are:

- $\vdash \text{skip} \sim \text{skip} : \Phi \Rightarrow \Phi$
- $\vdash X := E \sim Y := E' : \Phi[E\langle 1 \rangle / X\langle 1 \rangle, E'\langle 2 \rangle / Y\langle 2 \rangle] \Rightarrow \Phi$
- $$\frac{\vdash C_1 \sim C'_1 : \Phi \Rightarrow \Phi' \quad \vdash C_2 \sim C'_2 : \Phi' \Rightarrow \Phi''}{\vdash (C_1; C_2) \sim (C'_1; C'_2) : \Phi \Rightarrow \Phi''}$$
- $$\frac{\vdash C \sim C' : \Phi \wedge (B\langle 1 \rangle \wedge B'\langle 2 \rangle) \Rightarrow \Phi \wedge (B\langle 1 \rangle = B'\langle 2 \rangle)}{\vdash (\text{while } B \text{ do } C) \sim (\text{while } B' \text{ do } C') : \Phi \wedge (B\langle 1 \rangle = B'\langle 2 \rangle) \Rightarrow \Phi \wedge \text{not}(B\langle 1 \rangle \vee B'\langle 2 \rangle)}$$

Judgements used to prove equivalences of transformed programs.

falsity

$$C \sim C' : \text{false} \Rightarrow \Phi$$

dead-assignment elimination

$$\vdash (X := E) \sim \text{skip} : \Phi[E\langle 1 \rangle / X\langle 1 \rangle] \Rightarrow \Phi$$

common branch

$$\frac{\begin{array}{l} \vdash C \sim D : \Phi \wedge B\langle 1 \rangle \Rightarrow \Phi' \\ \vdash C \sim D : \Phi \wedge \text{not} B\langle 1 \rangle \Rightarrow \Phi' \end{array}}{\text{if } B \text{ then } C \text{ else } C' \sim D : \Phi \Rightarrow \Phi'}$$

dead while

$$\vdash (\text{while } B \text{ do } C \sim \text{skip} : \Phi \wedge \text{not} B\langle 1 \rangle \Rightarrow \Phi \wedge \text{not} B\langle 1 \rangle)$$
