Relational soundness and optimality proofs for Simple PRE

Ando Saabas Tarmo Uustalu

Theory days, Vanaõue

Background

Sometimes, transformation of program proofs alongside programs is needed (for example in a PCC framework).

- For non-optimizing compilers it is easy: proof compilation is (almost) identity
- Not so when optimizations take place
- Many different optimizations, all have their own particular way of messing up the proof
- There should be a systematic approach to proof transformation

Enter type systems

- Optimizations are based on dataflow analyses
- Dataflow analyses can be described as type systems
- Type systems can have an optimization component

- Type derivation tree matches the Hoare derivation tree
- Idea: types can guide the transformation of the proof tree

Does this approach scale?

This talk

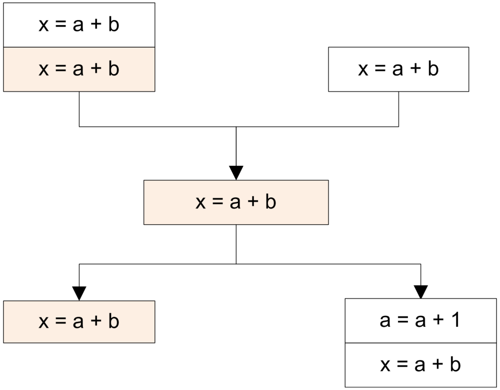
- We look at partial redundancy elimination, a complex optimization which attempts to make as many computations redundant as possible
- As a consequence, it performs common subexpression elimination and code motion (movement of invariant expressions out of loops) at the same time
- It changes the structure of the original program, by inserting new nodes into edges

Makes an interesting case study

Redundant expressions

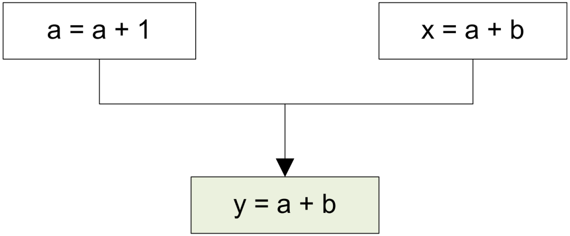
An expression is redundant at program point *p* if on every path to *p*

- It is evaluated before reaching *p*
- None of its variables are redefined before *p*



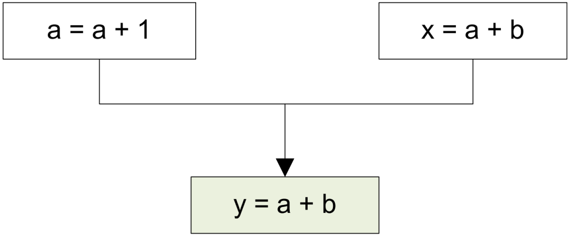
Partially redundant expressions

An expression is partially redundant if it is redundant on some path reaching p.



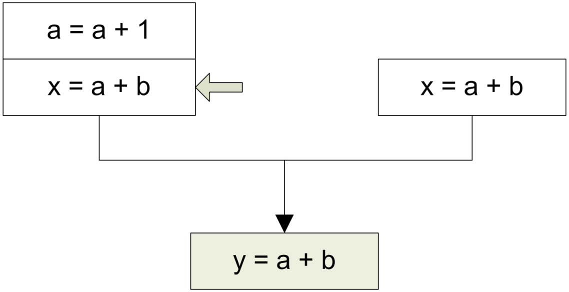
Partially redundant expressions

An expression is partially redundant if it is redundant on some path reaching p. Partial redundancy elimination tries to make it fully redundant



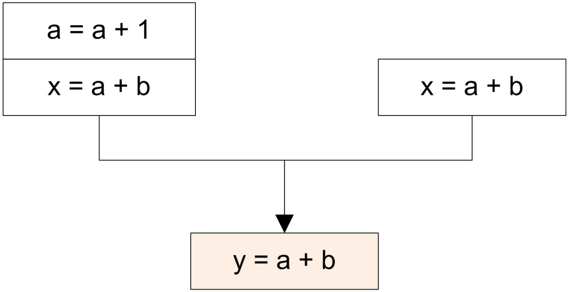
Partially redundant expressions

An expression is partially redundant if it is redundant on some path reaching p. Partial redundancy elimination tries to make it fully redundant



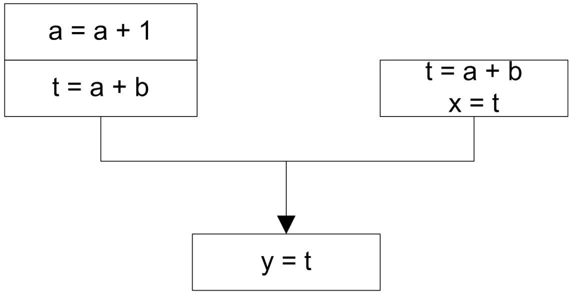
Partially redundant expressions

An expression is partially redundant if it is redundant on some path reaching p. Partial redundancy elimination tries to make it fully redundant



Partially redundant expressions

An expression is partially redundant if it is redundant on some path reaching p. Partial redundancy elimination tries to make it fully redundant



Simple PRE

- In this talk, we look at Simple PRE, which is more conservative version of standard PRE, but easier to perform.
- It requires 2 unidirectional analysis
  - A backward anticipability analysis (an expression is anticipable at a program if it will be evaluated on all paths, before any of its operands is modified)
  - A forward conditional partial availability analysis (an expression is partially available if it is already evaluated and later not modified on some path)

The same approach scales for full PRE (4 dataflow analayses).

Types for Simple PRE

- A type is a pair $(d, e) \in (\wp(\mathbf{AExp}) \times \wp(\mathbf{AExp}))$ satisfying the constraint $e \subseteq d$, where d is an anticipability type, and e is a partial availability type. Subtyping $\leq$ is set inclusion, i.e. $\subseteq$.
- Typing judgements are of the form $s : d', e \longrightarrow d, e'$.

Typing rules for Simple PRE

Typing rules for Simple PRE

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\} \quad \longrightarrow d} \; :=_{1\,\mathrm{pre}}$$

Typing rules for Simple PRE

$$\frac{}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d}\ {:=}_{1\,\mathrm{pre}}$$

Typing rules for Simple PRE

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d} \; {:=}1_{\text{pre}}$$

$$\frac{}{\mathsf{skip} : d, e \longrightarrow d, e} \; \text{skip}_{\text{pre}}$$

Typing rules for Simple PRE

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d} \ {:=_1}_{\mathrm{pre}}$$

$$\frac{}{\mathsf{skip} : d, e \longrightarrow d, e} \ \mathrm{skip}_{\mathrm{pre}}$$

$$\frac{s_0 : d, e \longrightarrow d'', e'' \quad s_1 : d'', e'' \longrightarrow d', e'}{s_0; s_1 : d, e \longrightarrow d', e'} \ \mathrm{comp}_{\mathrm{pre}}$$

Typing rules for Simple PRE

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d} :=_{1\,\mathrm{pre}}$$

$$\frac{}{\mathrm{skip} : d, e \longrightarrow d, e} \mathrm{skip}_{\mathrm{pre}}$$

$$\frac{s_0 : d, e \longrightarrow d'', e'' \quad s_1 : d'', e'' \longrightarrow d', e'}{s_0; s_1 : d, e \longrightarrow d', e'} \mathrm{comp}_{\mathrm{pre}}$$

$$\frac{s_t : d', e \longrightarrow d, e' \quad s_f : d', e \longrightarrow d, e'}{\text{if } b \text{ then } s_t \text{ else } s_f : d', e \longrightarrow d, e'} \mathrm{if}_{\mathrm{pre}}$$

Typing rules for Simple PRE

$$\frac{}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d}\ :=_{1\,\mathrm{pre}}$$

$$\frac{}{\mathsf{skip} : d, e \longrightarrow d, e}\ \mathrm{skip}_{\mathrm{pre}}$$

$$\frac{s_0 : d, e \longrightarrow d'', e'' \quad s_1 : d'', e'' \longrightarrow d', e'}{s_0; s_1 : d, e \longrightarrow d', e'}\ \mathrm{comp}_{\mathrm{pre}}$$

$$\frac{s_t : d', e \longrightarrow d, e' \quad s_f : d', e \longrightarrow d, e'}{\mathsf{if}\ b\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f : d', e \longrightarrow d, e'}\ \mathrm{if}_{\mathrm{pre}}$$

$$\frac{s_t : d, e \longrightarrow d, e}{\mathsf{while}\ b\ \mathsf{do}\ s_t : d, e \longrightarrow d, e}\ \mathrm{while}_{\mathrm{pre}}$$

Typing rules for Simple PRE

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d} :=_{1\,\mathrm{pre}}$$

$$\frac{}{\mathsf{skip} : d, e \longrightarrow d, e} \mathrm{skip_{pre}}$$

$$\frac{s_0 : d, e \longrightarrow d'', e'' \quad s_1 : d'', e'' \longrightarrow d', e'}{s_0; s_1 : d, e \longrightarrow d', e'} \mathrm{comp_{pre}}$$

$$\frac{s_t : d', e \longrightarrow d, e' \quad s_f : d', e \longrightarrow d, e'}{\mathsf{if}\ b\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f : d', e \longrightarrow d, e'} \mathrm{if_{pre}}$$

$$\frac{s_t : d, e \longrightarrow d, e}{\mathsf{while}\ b\ \mathsf{do}\ s_t : d, e \longrightarrow d, e} \mathrm{while_{pre}}$$

$$\frac{d, e \leq d_0, e_0 \quad s : d_0, e_0 \longrightarrow d_0', e_0' \quad d_0', e_0' \leq d', e'}{s : d, e \longrightarrow d', e'} \mathrm{conseq_{pre}}$$

Optimization rules for Simple PRE

$$\frac{}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

$$\frac{}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

$$\frac{a \notin e \quad a \notin d \vee x \in FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

$$\frac{a \notin e \quad a \notin d \vee x \in FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := a}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

$$\frac{a \notin e \quad a \notin d \lor x \in FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := a}$$

...

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

$$\frac{a \notin e \quad a \notin d \vee x \in FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := a}$$

...

$$\frac{d, e \leq d_0, e_0 \quad s : d_0, e_0 \longrightarrow d_0', e_0' \qquad d_0', e_0' \leq d', e'}{s : d, e \longrightarrow d', e'}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := nv(a)}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow nv(a) := a; x := nv(a)}$$

$$\frac{a \notin e \quad a \notin d \vee x \in FV(a)}{x := a : d\backslash\{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\}\backslash\{a' \mid x \in FV(a')\} \cap d \hookrightarrow x := a}$$

...

$$\frac{d, e \leq d_0, e_0 \quad s : d_0, e_0 \longrightarrow d'_0, e'_0 \hookrightarrow s' \quad d'_0, e'_0 \leq d', e'}{s : d, e \longrightarrow d', e'}$$

Optimization rules for Simple PRE

$$\frac{a \in e}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \quad \boxed{\hookrightarrow x := nv(a)}}$$

$$\frac{a \notin e \quad a \in d \quad x \notin FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \quad \boxed{\hookrightarrow nv(a) := a; x := nv(a)}}$$

$$\frac{a \notin e \quad a \notin d \vee x \in FV(a)}{x := a : d \backslash \{a' \mid x \in FV(a')\} \cup \{a\}, e \longrightarrow d, e \cup \{a\} \backslash \{a' \mid x \in FV(a')\} \cap d \quad \boxed{\hookrightarrow x := a}}$$

...

$$\frac{d, e \leq d_0, e_0 \quad s : d_0, e_0 \longrightarrow d_0', e_0' \; \boxed{\hookrightarrow s'} \quad d_0', e_0' \leq d', e'}{s : d, e \longrightarrow d', e' \; \boxed{\hookrightarrow \forall a \in e_0 \backslash e . nv(a) = a; s'; \forall a \in e' \backslash e_0' . nv(a) = a}}$$

Example

Ant Pav

```
if b then
   x := a + b;
else
   a := a + 1;
y := a + b;
```

Example

	Ant	Pav
`if b then`		
`    x := a + b;`		
`else`		
`    a := a + 1;`		
`y := a + b;`	$\{a+b\}$	

Example

	Ant	Pav
`if b then`		
`    x := a + b;`		
`else`		
`    a := a + 1;`	$\emptyset$	
`y := a + b;`	$\{a+b\}$	

Example

	Ant	Pav
`if b then`		
`    x := a + b;`	$\{a+b\}$	
`else`		
`    a := a + 1;`	$\emptyset$	
`y := a + b;`	$\{a+b\}$	

Example

	Ant	Pav
`if b then`	$\emptyset$	
`    x := a + b;`	$\{a + b\}$	
`else`		
`    a := a + 1;`	$\emptyset$	
`y := a + b;`	$\{a + b\}$	

Example

	Ant	Pav
`if b then`	$\emptyset$	$\emptyset$
`    x := a + b;`	$\{a+b\}$	
`else`		
`    a := a + 1;`	$\emptyset$	
`y := a + b;`	$\{a+b\}$	

Example

	Ant	Pav
`if b then`	$\emptyset$	$\emptyset$
`    x := a + b;`	$\{a+b\}$	$\{a+b\}$
`else`		
`    a := a + 1;`	$\emptyset$	
`y := a + b;`	$\{a+b\}$	

Example

	Ant	Pav
`if b then`	$\emptyset$	$\emptyset$
`    x := a + b;`	$\{a+b\}$	$\{a+b\}$
`else`		
`    a := a + 1;`	$\emptyset$	$\emptyset$
`y := a + b;`	$\{a+b\}$	

Example

	Ant	Pav
`if b then`	$\emptyset$	$\emptyset$
`    x := a + b;`	$\{a+b\}$	$\{a+b\}$
`else`		
`    a := a + 1;`	$\emptyset$	$\emptyset$
`y := a + b;`	$\{a+b\}$	$\{a+b\}$

Example

	Ant	Pav
`if b then`	$\emptyset$	$\emptyset$
`    x := a + b;`	$\{a+b\}$	$\{a+b\}$
`else`		
`    a := a + 1;`	$\emptyset$	$\emptyset$
`y := a + b;`	$\{a+b\}$	$\{a+b\}$

$\longrightarrow$

```
if b then
    t := a + b;
    x := t;
else
    a := a + 1;
    t := a + b;
y := t;
```

Soundness of Simple PRE

The typesystematic formulation of PRE leads to a simple correctness proof

- At corresponding program points, the states of the two programs agree to each other on all normal program variables
- Values of the expressions in the type and the values of the corresponding auxiliary variables also coincide

Soundness of Simple PRE

Let $\sigma \sim_e \sigma'$ denote that two states σ and σ' agree on $e \subseteq$ **AExp** in the following sense:
$(\forall x \in \mathbf{Var}.\sigma(x) = \sigma'(x)) \wedge (\forall a \in e.[\![a]\!]\sigma = \sigma'(nv(a)))$.

Theorem (Correctness of simple PRE)

If $s : d', e \longrightarrow d, e' \hookrightarrow s_*$ and $\sigma \sim_e \sigma_*$, then
— $\sigma \succ s \rightarrow \sigma'$ implies the existence of σ'_* such that $\sigma' \sim_{e'} \sigma'_*$ and $\sigma_* \succ s_* \rightarrow \sigma'_*$,
— $\sigma_* \succ s_* \rightarrow \sigma'_*$ implies the existence of σ' such that $\sigma' \sim_{e'} \sigma'_*$ and $\sigma \succ s \rightarrow \sigma'$.

Proof transformation

From soundness we get proof transformation. Let $P|_e$ to be $P \wedge \bigwedge[a = nv(a) | a \in e]$.

Theorem

If $s : d', e \longrightarrow d, e' \hookrightarrow s'$, *then* $\{P\}\, s\, \{Q\}$ *implies* $\{P|_e\}\, s'\, \{Q|_{e'}\}$,

The constructive proof for this gives us an algorithm which transforms the derivation of $\{P\}\, s\, \{Q\}$ into the derivation of $\{P|_e\}\, s'\, \{Q|_{e'}\}$

Improvement property

- From the definition of the optimization, it is not obvious that code motion does not introduce unneeded evaluations.
- This property can be shown via simple instrumented semantics, which counts the number of evaluations of every expression.
- The semantic judgement is of the form $(\sigma, r) \succ s \rightarrow (\sigma', r')$, where σ and σ' are usual states, and r and r' show how many times a particular expression has been evaluated.

Improvement property

Let $(\sigma, r) \sim_e (\sigma', r')$ denote that two states (σ, r) and (σ', r') agree on $e \subseteq \mathbf{AExp}$ in the following sense:
$(\forall x \in \mathbf{Var}.\sigma(x) = \sigma'(x)) \wedge (\forall a \in e.[\![a]\!]\sigma = \sigma'(nv(a)))$.
Furthermore $\forall a \notin e.r'(a) \leq r(a)$ and $\forall a \in e.r'(a) \leq r'(a) + 1$.

Theorem (Improvement property of simple PRE)

If $s : d', e \longrightarrow d, e' \hookrightarrow s_*$ and $\sigma \sim_e \sigma_*$, then
— $(\sigma, r) \succ s \rightarrow (\sigma', r')$ implies the existence of (σ'_*, r'_*) such that $(\sigma', r') \sim_{e'} (\sigma'_*, r'_*)$ and $(\sigma_*, r_*) \succ s_* \rightarrow (\sigma'_*, r'_*)$,
— $(\sigma_*, r_*) \succ s_* \rightarrow (\sigma'_*, r'_*)$ implies the existence of (σ', r') such that $(\sigma', r') \sim_{e'} (\sigma'_*, r'_*)$ and $(\sigma, r) \succ s \rightarrow (\sigma', r')$.

Conclusions

- The type-systematic approach to dataflow analyses scales up to complicated analyses which change code structure.
- Using this approach, it is possible to prove properties beyond soundness, like optimality results.
- Soundness of the optimization makes it possible to transform a program's proof along the program guided by its analysis type derivation.