Simple Relational Correctness Proofs for Static Analyses and Program Transformations (Revised, Long Version)

Nick Benton
Microsoft Research
Cambridge UK

February 2005

Technical Report
MSR-TR-2005-26

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

Abstract

We show how some classical static analyses for imperative programs, and the optimizing transformations which they enable, may be expressed and proved correct using elementary logical and denotational techniques. The key ingredients are an interpretation of program properties as relations, rather than predicates, and a realization that although many program analyses are traditionally formulated in very intensional terms, the associated transformations are actually enabled by more liberal extensional properties.

We illustrate our approach with formal systems for analysing and transforming while-programs. The first is a simple type system which tracks constancy and dependency information and can be used to perform dead-code elimination, constant propagation and program slicing as well as capturing a form of secure information flow. The second is a relational version of Hoare logic, which significantly generalizes our first type system and can also justify optimizations including hoisting loop invariants. Finally we show how a simple available expression analysis and redundancy elimination transformation may be justified by translation into relational Hoare logic.

Categories and Subject Descriptors: F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages – *Denotational semantics, Partial evaluation, Program analysis*; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs – *Logics of programs, Invariants*; D.3.4 [Programming Languages]: Processors – *Compilers, Optimization*;

General Terms: Languages, Theory, Verification

Keywords: Program analysis, optimizing compilation, types, denotational semantics, partial equivalence relations, Hoare logic, dependency, information flow, security

Publication History

This is a revised and extended version of a paper of the same name published in POPL 2004:

N. Benton. *Simple Relational Correctness Proofs for Static Analyses and Program Transformations.* In *Proceedings of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, Venice, Italy January 14-16, 2004. Revised version available from http://research.microsoft.com/~nick/publications.htm.

The revised electronic version fixes a couple of small errors in the version in the published proceedings. This technical report additionally includes most of the proofs.

1 Introduction

Although static program analyses are routinely proved correct, the soundness of the optimizing transformations which they enable is much less frequently addressed. Much of the work which *has* been done on formalizing and validating analysis-based transformations comes from the functional programming community (see Section 5 for related work) – the literature on optimizations for imperative languages contains few formal specifications of transformations, let alone proofs of their correctness. One might think that this is because the correctness of most imperative optimizations is entirely trivial, but what literature there is on the subject [22, 18, 19, 20] (not to mention the occasional behaviour of real optimizing compilers) indicates that this is not so.

Why is proving correctness of analysis-based transformations hard? We wish to establish that, given the results of a static analysis, the original program and the transformed program are observationally equivalent. The first problem is that many transformations involve locally replacing some part P of a larger program $C[P]$ with a new version P' which is *not* generally observationally equivalent to P (i.e. $P \not\sim P'$), though they *are* equivalent in that particular context: $C[P] \sim C[P']$. Simply having an analysis which (albeit correctly) deduces that certain predicates $\psi(P)$ hold of program fragments does not straightforwardly allow one to justify such transformations *unless* the predicates $\psi(\cdot)$ also somehow involve sets of contexts $C[\cdot]$, which is often not the case.

The second difficulty in proving correctness of optimizing transformations is that program analyses, especially for imperative languages, are often specified in a very intensional way. For example, "an assignment $[x := a]^l$ *may reach* a certain program point if there is an execution of the program where x was last assigned a value at l when the program point is reached". Notions such as 'program point' and 'where a variable was last assigned' are not present in natural operational or denotational semantics, so the correctness of these analyses is frequently formulated in terms of a new (and essentially bogus) *instrumented semantics* which tracks the extra information. Even where the instrumented semantics is related back to the original one, the relation is usually a rather weak form of adequacy which certainly does not help with establishing equivalences: the instrumented semantics will generally have a weaker equational theory than the original one.

This paper demonstrates that, at least in some simple cases, both of these difficulties can be overcome by use of elementary ideas which are commonplace in the semantics community, but which have not previously been fully exploited in the context of compiler analyses and transformations.

The first difficulty is approached by taking seriously the notion of the semantics of types as (special kinds of) relations, rather than predicates. Typed lambda calculi are routinely presented using judgements of the form

$$\Gamma \vdash M = M' : A$$

which does not assert "under assumptions Γ, M equals M' and they both have type A", but rather "under assumptions Γ, M and M' are equal *at* type A". Such calculi can be modelled by interpreting types as partial equivalence relations over some untyped universe such as D_∞. Many program analyses are presented as non-standard type systems, and partial equivalence relations have been used to give semantics to these non-standard types (equivalently, elements of abstract domains), at least in the cases of binding-time [15] and security analyses [26]. However, even in those cases, the emphasis has been on simple typing judgements rather than equational reasoning. Our approach is to treat all abstract properties as relations, including those which have naive interpretations as predicates (e.g. 'X is 5'), and to present transformations by giving rules for deriving (non-standard) typed equations in context.

The second difficulty, the apparently intensional nature of properties, is often something of a red herring, attributable to a confusion between certain analysis algorithms and the semantics of the information they produce. Of course, analyses relating to properties such as time or space usage can only be justified relative to semantic models which make those aspects of computation explicit. But many transformations performed by optimizing compilers can be justified using more extensional semantics, not only in the weak sense that every input program is provably equivalent to its transformed version, but also in the stronger sense that there is a generic correctness argument for all programs. The true preconditions for applying a transformation tend to be extensional ("this command does not change the value of `X+Y`") even if an analysis algorithm only discovers those properties if a stronger intensional property holds ("this command does not contain any assignments to `X` or `Y`").

As a facetious example of the difference between the intensional and extensional approaches, consider why the following transformation is correct:

```
  X := 7;                 X := 7;
  Y := Y+1;     ==>       Y := Y+1;
  Z := X;                 Z := 7;
```

The extensional answer is "when `X` is evaluated on the last line, its value will always be 7". The intensional answer is something like "the only definition of `X` which reaches its use on line 3 is the one on line 1, and the right hand side of that definition does not contain any variable which is assigned to in lines 1 or 2". This may well be an accurate account of *how* an algorithm works, but it is not a good basis for thinking about *what* it establishes. Things get even worse if we consider a sequence like

```
  X := 7;          X := 7;          X := 7;
  Y := Y+1;   ==>  Y := Y+1;   ==>  Y := Y+1;
  X := 7;          X := 7;
  Z := X;          Z := 7;          Z := 7;
```

After the first transformation, the intensional justification for the change to line 4 refers to the definition of `X` on line 3. But after the second transformation, that definition has gone, which complicates proving the correctness of the combined transformation. Problems of this sort occur both in real compilers (keeping analysis results sound during transformations is notoriously tricky) and in proofs (see for example the discussion of interference between 'forward' analyses and 'backward' transformation patterns in [20]).

Another familiar example of the intensional/extensional distinction arises in optimizing compilation of lazy functional languages [9]. Some analysis algorithms aimed at detecting when CBN can be replaced by CBV look for functions which always evaluate their arguments ('neededness'), which is an intensional property. Their correctness (and that of the associated transformation) can be established in terms of the extensional property of strictness, which is much easier to reason about. Of course, since the extensional property is also weaker (holds of more functions) one is then naturally led to reformulate the analysis to establish the extensional property more directly.

2 The Language of `while`-Programs

The syntax and denotational semantics of the language of `while`-programs are entirely standard (see, e.g. [34]). To fix notation, they are briefly summarized in Figure 1. We sometimes use F_τ as a metavariable ranging over $\tau\ \mathtt{exp}$ where $\tau \in \{\mathtt{int}, \mathtt{bool}\}$.

The denotational semantics is given in the category of ω-complete partial orders (predomains) and continuous functions. We write $\lceil\cdot\rceil : D \to D_\perp$ for the injection of a domain into its lift and $(\cdot)^* : (D \to D'_\perp) \to (D_\perp \to D'_\perp)$ for the associated extension operation. When $R \subseteq D \times D$, $R_\perp \subseteq D_\perp \times D_\perp$ is the relation defined by

$$R_\perp = \{(\lceil x\rceil, \lceil y\rceil) \mid (x, y) \in R\} \cup \{(\perp, \perp)\}$$

If $f : D \to E$, $x \in D$ and $y \in E$ then we define $f[x \mapsto y] : D \to E$ in the usual way:

$$(f[x \mapsto y])(z) = \begin{cases} y & \text{if } z = x \\ f(z) & \text{otherwise} \end{cases}$$

The denotational semantics is fully abstract with respect to the obvious operational semantics and definition of observational equivalence.

2.1 Relations

If X is a set, a binary relation $R \subseteq X \times X$ is a partial equivalence relation (PER) if it is symmetric and transitive. A relation on the carrier of a pointed ω-cpo D is admissible if $(\perp, \perp) \in R$ and for all ascending chains $\langle d_i\rangle$ and $\langle d'_i\rangle$ with $(d_i, d'_i) \in R$, we have $(\sqcup_i d_i, \sqcup_i d'_i) \in R$. If R is a relation on a set X, then $R_\perp$ is an admissible relation on the flat cpo $X_\perp$ and is a PER if R is. The set of PERs on a set is closed under arbitrary intersections and disjoint unions. The set of admissible relations on a pointed cpo is closed under arbitrary intersections and finite unions. If R and S are relation on predomains D and E respectively, we write $R \Rightarrow S$ for the relation on the function space $D \to E$ defined by $(f, g) \in R \Rightarrow S$ iff $\forall (x, y) \in R.(fx, gy) \in S$. This is a PER if R and S are. If E is pointed and S is admissible, then $R \Rightarrow S$ is admissible.

3 Dependency, Dead Code and Constants

In this section we present DDCC, a simple analysis and transformation system for `while`-programs which tracks dependency and constancy information, enabling optimizations such as constant-folding and dead-code elimination. As indicated in the introduction, the system is presented as a non-standard type system for deriving typed equalities between expressions and between commands.

Syntax

$$
\begin{array}{rcl}
X \in \mathbb{V} & = & \{\mathtt{X}, \mathtt{Y}, \ldots\} \\
n \in \mathbb{Z} & & b \in \mathbb{B} = \{\mathit{true}, \mathit{false}\} \\
\mathit{iop} & \in & \{+, \times, -, \ldots\} \ \subseteq \ \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \\
\mathit{bop} & \in & \{<, =, \ldots\} \ \subseteq \ \mathbb{Z} \times \mathbb{Z} \to \mathbb{B} \\
\mathit{lop} & \in & \{\vee, \wedge, \ldots\} \ \subseteq \ \mathbb{B} \times \mathbb{B} \to \mathbb{B} \\
\mathtt{int\ exp} \ni E & := & \mathtt{n} \mid X \mid E \ \mathtt{iop} \ E \\
\mathtt{bool\ exp} \ni B & := & \mathtt{b} \mid E \ \mathtt{bop} \ E \mid \mathtt{not} \ B \mid B \ \mathtt{lop} \ B \\
\mathtt{com} \ni C & := & \mathtt{skip} \mid X \mathtt{:=} E \mid C \mathtt{;} C \mid \mathtt{if} \ B \ \mathtt{then} \ C \ \mathtt{else} \ C \mid \mathtt{while} \ B \ \mathtt{do} \ C
\end{array}
$$

Denotational Semantics

$$S \in \mathbb{S} = \mathbb{V} \to \mathbb{Z}$$

$$\llbracket E \rrbracket \in \mathbb{S} \to \llbracket \mathtt{int} \rrbracket = \mathbb{S} \to \mathbb{Z}$$

$$
\begin{array}{rcl}
\llbracket \mathtt{n} \rrbracket S & = & n \\
\llbracket X \rrbracket S & = & S(X) \\
\llbracket E_1 \ \mathtt{iop} \ E_2 \rrbracket S & = & (\llbracket E_1 \rrbracket S) \ \mathit{iop} \ (\llbracket E_2 \rrbracket S)
\end{array}
$$

$$\llbracket B \rrbracket \in \mathbb{S} \to \llbracket \mathtt{bool} \rrbracket = \mathbb{S} \to \mathbb{B}$$

$$
\begin{array}{rcl}
\llbracket \mathtt{b} \rrbracket S & = & b \\
\llbracket E_1 \ \mathtt{bop} \ E_2 \rrbracket S & = & (\llbracket E_1 \rrbracket S) \ \mathit{bop} \ (\llbracket E_2 \rrbracket S) \\
\llbracket B_1 \ \mathtt{lop} \ B_2 \rrbracket S & = & (\llbracket B_1 \rrbracket S) \ \mathit{lop} \ (\llbracket B_2 \rrbracket S) \\
\llbracket \mathtt{not} B \rrbracket S & = & \neg(\llbracket B \rrbracket S)
\end{array}
$$

$$\llbracket C \rrbracket \in \mathbb{S} \to \mathbb{S}_\perp$$

$$
\begin{array}{rcl}
\llbracket \mathtt{skip} \rrbracket & = & \lambda S. \lceil S \rceil \\
\llbracket X \mathtt{:=} E \rrbracket & = & \lambda S. \lceil S[X \mapsto \llbracket E \rrbracket S] \rceil \\
\llbracket C_1 \mathtt{;} C_2 \rrbracket & = & \llbracket C_2 \rrbracket^* \circ \llbracket C_1 \rrbracket \\
\llbracket \mathtt{if} \ B \ \mathtt{then} \ C_1 \ \mathtt{else} \ C_2 \rrbracket & = & \lambda S. \llbracket B \rrbracket S \implies \llbracket C_1 \rrbracket S \ \mid \ \llbracket C_2 \rrbracket S \\
\llbracket \mathtt{while} \ B \ \mathtt{do} \ C \rrbracket & = & \mathit{fix} \ f. \lambda S. \llbracket B \rrbracket S \implies f^*(\llbracket C \rrbracket S) \mid \lceil S \rceil
\end{array}
$$

Figure 1: Syntax and Semantics of `while` Programs

3.1 DDCC Syntax and Semantics

3.1.1 Formulae

We begin by defining the syntax of some non-standard types for expressions. For $\tau \in \{\texttt{int}, \texttt{bool}\}$, $c \in [\![\tau]\!]$:

$$\phi_\tau \quad := \quad \mathbb{F}_\tau \mid \{c\}_\tau \mid \Delta_\tau \mid \mathbb{T}_\tau$$

Intuitively, $\{c\}_\tau$ is the type of τ-expressions equal to the constant c, Δ_τ is the type of τ-expressions whose value we do not know, whilst $\mathbb{T}_\tau$ is the type of τ-expressions whose value we do not care about. $\mathbb{F}_\tau$ is an empty expression type, which we have included for completeness.[1] Semantically, the denotation of ϕ_τ is a binary relation on $[\![\tau]\!]$:

$$\begin{array}{rcl} [\![\mathbb{F}_\tau]\!] & = & \emptyset \\ [\![\{c\}_\tau]\!] & = & \{(c, c)\} \\ [\![\Delta_\tau]\!] & = & \{(x, x) \mid x \in [\![\tau]\!]\} \\ [\![\mathbb{T}_\tau]\!] & = & [\![\tau]\!] \times [\![\tau]\!] \end{array}$$

Types for states are then finite maps from variables to types for `int exps`, written as lists with the usual conventions. In particular, writing $\Phi, X : \phi_{\texttt{int}}$ implies that X does not occur in Φ.

$$\Phi \quad := \quad - \mid \Phi, X : \phi_{\texttt{int}}$$

State types are interpreted as binary relations on $\mathbb{S}$:

$$\begin{array}{rcl} [\![-]\!] & = & \mathbb{S} \times \mathbb{S} \\ [\![\Phi, X : \phi_{\texttt{int}}]\!] & = & [\![\Phi]\!] \cap \{(S, S') \mid (S(X), S'(X)) \in [\![\phi_{\texttt{int}}]\!]\} \end{array}$$

3.1.2 Entailment

There is a subtyping relation $\leq$ on expression types, which is axiomatised as follows:

$$\mathbb{F}_\tau \leq \phi_\tau \qquad \{c\}_\tau \leq \Delta_\tau$$

$$\phi_\tau \leq \mathbb{T}_\tau \qquad \phi_\tau \leq \phi_\tau$$

$$\frac{\phi_\tau \leq \phi'_\tau \quad \phi'_\tau \leq \phi''_\tau}{\phi_\tau \leq \phi''_\tau}$$

The above induces a depth- and width-subtyping relation on state types:

$$\Phi \leq - \qquad\qquad \Phi, X : \mathbb{F}_{\texttt{int}} \leq \Phi'$$

$$\frac{\Phi \leq \Phi'}{\Phi \leq \Phi', X : \mathbb{T}_{\texttt{int}}} \qquad \frac{\Phi \leq \Phi' \quad \phi_{\texttt{int}} \leq \phi'_{\texttt{int}}}{\Phi, X : \phi_{\texttt{int}} \leq \Phi', X : \phi'_{\texttt{int}}}$$

Because $\Phi, X : \mathbb{T}_{\texttt{int}} \leq \Phi$ and $\Phi \leq \Phi, X : \mathbb{T}_{\texttt{int}}$, absence of a variable from a state type is equivalent to it being present with type $\mathbb{T}_{\texttt{int}}$.

[1] This is really just a matter of taste. $\mathbb{F}_\tau$ does not appear in many interesting derivations.

Lemma 1.

1. *For all ϕ_τ and Φ, $[\![\phi_\tau]\!]$ and $[\![\Phi]\!]$ are partial equivalence relations.*
2. *The $\leq$ relation on state types is reflexive and transitive.*
3. *If $\phi_\tau \leq \phi'_\tau$ then $[\![\phi_\tau]\!] \subseteq [\![\phi'_\tau]\!]$.*
4. *If $\Phi \leq \Phi'$ then $[\![\Phi]\!] \subseteq [\![\Phi']\!]$.*
5. $(S, S') \in [\![\Phi, X : \phi]\!]$ *iff* $\forall m, n.(S[X \mapsto m], S'[X \mapsto n]) \in [\![\Phi]\!]$ *and* $(S(X), S'(X)) \in [\![\phi]\!]$.

Proof.

1. This is immediate for expression types. Then for state types, each set comprehension in the definition defines a PER on the function space $\mathbb{S} \rightarrow [\![\tau]\!]$ and since PERs are closed under intersection we're done.
2. Reflexivity follows by induction from the first $(- \leq -)$ and last rules, and the reflexivity axiom for expression types. Our choice of rules and the presence of $\mathbb{F}$ make showing admissibility of transitivity for state type entailment not entirely trivial:

   First observe that $\phi \leq \mathbb{F}$ implies $\phi = \mathbb{F}$. Then it's a simple induction to show that $\Phi \leq \Phi', X : \mathbb{F}$ implies $\Phi = \Phi'', Y : \mathbb{F}$ for some variable Y. Hence transitivity is immediate if any of the state types involved contain $\mathbb{F}$. WLOG we may therefore just show that $\Phi \leq \Phi'$ and $\Phi' \leq \Phi''$ implies $\Phi \leq \Phi''$ assuming there are no $\mathbb{F}$s in Φ,Φ' or Φ'' (so the second rule is not used in either of the derivations (OK by subformula property)). Then induction on the other three rules shows $\Phi \leq \Phi'$ implies (*) $\forall X \in \mathbb{V}.\Phi(X) \leq \Phi'(X)$ where $\Phi(X)$ is defined to be ϕ if $\Phi = \Phi'', X : \phi$ and $\mathbb{T}$ otherwise. The converse follows by induction on the sum of the length of Φ'. Then since (*) is transitive, we're done. Which is embarassingly involved.
3. Immediate from the definition.
4. Simple induction plus the previous bit.
5. Immediate from definition and assumption in writing $\Phi, X : \phi$ that X does not appear in Φ.

□

3.1.3 Judgements

DDCC has two basic forms of judgement. For expressions, with $F, F' \in \tau$ `exp`, we have judgements of the form

$$\vdash F \sim F' : \Phi \Rightarrow \phi_\tau$$

whilst for commands, $C, C' \in$ `com`, there are judgements of the form

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

We write $\vdash C : \Phi \Rightarrow \Phi'$ as shorthand for $\vdash C \sim C : \Phi \Rightarrow \Phi'$ and similarly for single-subject expression judgements. If we define

$$\begin{array}{rcl}
\llbracket \Phi \Rightarrow \phi_\tau \rrbracket & \subseteq & (\mathbb{S} \to \llbracket \tau \rrbracket) \times (\mathbb{S} \to \llbracket \tau \rrbracket) \\
& \equiv & \{(f, f') \mid \forall (S, S') \in \llbracket \Phi \rrbracket .\ (f S, f' S') \in \llbracket \phi_\tau \rrbracket \}
\end{array}$$

$$\begin{array}{rcl}
\llbracket \Phi \Rightarrow \Phi' \rrbracket & \subseteq & (\mathbb{S} \to \mathbb{S}_\perp) \times (\mathbb{S} \to \mathbb{S}_\perp) \\
& \equiv & \{(f, f') \mid \forall (S, S') \in \llbracket \Phi \rrbracket .\ (f S, f' S') \in \llbracket \Phi' \rrbracket_\perp \}
\end{array}$$

then the intended meanings of the judgements are:

$$\begin{array}{rcl}
\models F_\tau \sim F'_\tau : \Phi \Rightarrow \phi_\tau & \equiv & (\llbracket F_\tau \rrbracket, \llbracket F'_\tau \rrbracket) \in \llbracket \Phi \Rightarrow \phi_\tau \rrbracket \\
\models C \sim C' : \Phi \Rightarrow \Phi' & \equiv & (\llbracket C \rrbracket, \llbracket C' \rrbracket) \in \llbracket \Phi \Rightarrow \Phi' \rrbracket
\end{array}$$

Lemma 2. *$\llbracket \Phi \Rightarrow \phi_\tau \rrbracket$ is a PER and $\llbracket \Phi \Rightarrow \Phi' \rrbracket$ is an admissible PER.*

Proof. Immediate from the basic facts in Section 2.1. □

Some basic rules for deriving DDCC judgements are shown in Figure 2. The rules for expressions refer to abstract versions $\widehat{op}$ of each primitive binary operator op in the language. A typical definition is that for multiplication:

$\widehat{\times}$	$\mathbb{F}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\{n\}_{\mathtt{int}}$	$\Delta_{\mathtt{int}}$	$\mathbb{T}_{\mathtt{int}}$
$\mathbb{F}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$
$\{0\}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$
$\{m\}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\{m \times n\}_{\mathtt{int}}$	$\Delta_{\mathtt{int}}$	$\mathbb{T}_{\mathtt{int}}$
$\Delta_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\Delta_{\mathtt{int}}$	$\Delta_{\mathtt{int}}$	$\mathbb{T}_{\mathtt{int}}$
$\mathbb{T}_{\mathtt{int}}$	$\mathbb{F}_{\mathtt{int}}$	$\{0\}_{\mathtt{int}}$	$\mathbb{T}_{\mathtt{int}}$	$\mathbb{T}_{\mathtt{int}}$	$\mathbb{T}_{\mathtt{int}}$

The general correctness condition for abstract operations is familiar from abstract interpretation:

Definition 1. *We say $\widehat{op}$ soundly abstracts the operation op if*

$$\forall (x, x') \in \llbracket \phi_\tau \rrbracket, (y, y') \in \llbracket \phi'_\tau \rrbracket .\ (x\ \ op\ \ y, x'\ \ op\ \ y') \in \llbracket \phi_\tau\, \widehat{op}\, \phi'_\tau \rrbracket.$$

The most interesting of the rules in Figure 2 are those for conditionals and `while`-loops. Observe that for two conditionals to be related, not only do their true and false branches have to be pairwise related, but they also have to agree on which branch is taken; this is expressed by the use of $\Delta_{\mathtt{bool}}$ in the premises of the rule. Similar considerations apply to the rule for `while`-loops, which ensures that related loops execute in lockstep.

3.2 Equations

Using only the rules in Figure 2, most of the interesting judgements one can prove relate a phrase to itself at some type. In other words, they constitute an analysis system but not yet a program transformation system. However, the advantage of our formulation is that program transformations can now be specified and justified simply by adding new inference rules whose soundness may be straightforwardly and independently checked in terms of the semantics.

Subtyping and Structural

$$\vdash C \sim C' : \Phi, X : \mathbb{F}_{\texttt{int}} \Rightarrow \Phi' \ [\text{D-CT}] \qquad \vdash F_\tau \sim F'_\tau : \Phi \Rightarrow \mathbb{T}_\tau \ [\text{D-ET1}]$$

$$\vdash F_\tau \sim F'_\tau : \Phi, X : \mathbb{F}_{\texttt{int}} \Rightarrow \phi_\tau \ [\text{D-ET2}] \qquad \frac{\vdash F_\tau \sim F'_\tau : \Phi \Rightarrow \phi_\tau}{\vdash F'_\tau \sim F_\tau : \Phi \Rightarrow \phi_\tau} \ [\text{D-ESym}]$$

$$\frac{\vdash F_\tau \sim F'_\tau : \Phi \Rightarrow \phi_\tau \quad \Phi' \leq \Phi \quad \phi_\tau \leq \phi'_\tau}{\vdash F_\tau \sim F'_\tau : \Phi' \Rightarrow \phi'_\tau} \ [\text{D-ESub}]$$

$$\frac{\vdash C \sim C' : \Phi_1 \Rightarrow \Phi_2 \quad \Phi'_1 \leq \Phi_1 \quad \Phi_2 \leq \Phi'_2}{\vdash C \sim C' : \Phi'_1 \Rightarrow \Phi'_2} \ [\text{D-CSub}]$$

$$\frac{\vdash F_\tau \sim F'_\tau : \Phi \Rightarrow \phi_\tau \quad \vdash F'_\tau \sim F''_\tau : \Phi \Rightarrow \phi_\tau}{\vdash F_\tau \sim F''_\tau : \Phi \Rightarrow \phi_\tau} \ [\text{D-ETr}]$$

$$\frac{\vdash C \sim C' : \Phi \Rightarrow \Phi'}{\vdash C' \sim C : \Phi \Rightarrow \Phi'} \ [\text{D-CSym}]$$

$$\frac{\vdash C \sim C' : \Phi \Rightarrow \Phi' \quad \vdash C' \sim C'' : \Phi \Rightarrow \Phi'}{\vdash C \sim C'' : \Phi \Rightarrow \Phi'} \ [\text{D-CTr}]$$

Expressions

$$\vdash X \sim X : \Phi, X : \phi_{\texttt{int}} \Rightarrow \phi_{\texttt{int}} \ [\text{D-V}] \qquad \vdash \texttt{n} \sim \texttt{n} : \Phi \Rightarrow \{n\}_{\texttt{int}} \ [\text{D-N}]$$

$$\vdash \texttt{b} \sim \texttt{b} : \Phi \Rightarrow \{b\}_{\texttt{bool}} \ [\text{D-B}]$$

$$\frac{\vdash F_\tau \sim G_\tau : \Phi \Rightarrow \phi_\tau \quad \vdash F'_\tau \sim G'_\tau : \Phi \Rightarrow \phi'_\tau}{\vdash F_\tau \ \texttt{op} \ F'_\tau \sim G_\tau \ \texttt{op} \ G'_\tau : \Phi \Rightarrow (\phi_\tau \ \widehat{op} \ \phi'_\tau)} \ [\text{D-}op]$$

Commands

$$\vdash \texttt{skip} \sim \texttt{skip} : \Phi \Rightarrow \Phi \ [\text{D-Skip}]$$

$$\frac{\vdash C_1 \sim C'_1 : \Phi \Rightarrow \Phi' \quad \vdash C_2 \sim C'_2 : \Phi' \Rightarrow \Phi''}{\vdash (C_1 ; C_2) \sim (C'_1 ; C'_2) : \Phi \Rightarrow \Phi''} \ [\text{D-Seq}]$$

$$\frac{\vdash E \sim E' : \Phi, X : \phi_{\texttt{int}} \Rightarrow \phi'_{\texttt{int}}}{\vdash X \texttt{:=} E \sim X \texttt{:=} E' : \Phi, X : \phi_{\texttt{int}} \Rightarrow \Phi, X : \phi'_{\texttt{int}}} \ [\text{D-Ass}]$$

$$\frac{\vdash B \sim B' : \Phi \Rightarrow \Delta_{\texttt{bool}} \quad \vdash C \sim C' : \Phi \Rightarrow \Phi}{\vdash (\texttt{while} \ B \ \texttt{do} \ C) \sim (\texttt{while} \ B' \ \texttt{do} \ C') : \Phi \Rightarrow \Phi} \ [\text{D-Whl}]$$

$$\frac{\vdash B \sim B' : \Phi \Rightarrow \Delta_{\texttt{bool}} \quad \vdash C_1 \sim C'_1 : \Phi \Rightarrow \Phi' \quad \vdash C_2 \sim C'_2 : \Phi \Rightarrow \Phi'}{\vdash (\texttt{if} \ B \ \texttt{then} \ C_1 \ \texttt{else} \ C_2) \sim (\texttt{if} \ B' \ \texttt{then} \ C'_1 \ \texttt{else} \ C'_2) : \Phi \Rightarrow \Phi'} \ [\text{D-If}]$$

Figure 2: Core DDCC System

3.2.1 Basic equations

Our first set of transformation rules express universally applicable structural equivalences for `while`-programs, without requiring any of the extra information gathered by the analysis.

Sequential unit laws:

$$\frac{\vdash C : \Phi \Rightarrow \Phi'}{\vdash (\texttt{skip};C) \sim C : \Phi \Rightarrow \Phi'}\ \text{[D-SU1]}$$

$$\frac{\vdash C : \Phi \Rightarrow \Phi'}{\vdash (C;\texttt{skip}) \sim C : \Phi \Rightarrow \Phi'}\ \text{[D-SU2]}$$

Associativity:

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

In practice, one usually identifies programs up to associativity of sequential composition, rather than making explicit use of the rule above.

Commuting conversion for conditional:

$$\frac{\vdash \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 : \Phi \Rightarrow \Phi' \quad \vdash C_3 : \Phi' \Rightarrow \Phi''}{\begin{array}{ll}\vdash & (\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2);C_3 \\ \sim & \texttt{if}\ B\ \texttt{then}\ (C_1;C_3)\ \texttt{else}\ (C_2;C_3) : \Phi \Rightarrow \Phi''\end{array}}\ \text{[D-CC]}$$

Loop unrolling:

$$\frac{\vdash \texttt{while}\ B\ \texttt{do}\ C : \Phi \Rightarrow \Phi' \qquad \text{[D-LU1]}}{\begin{array}{ll}\vdash & \texttt{while}\ B\ \texttt{do}\ C \\ \sim & \texttt{if}\ B\ \texttt{then}\ C;(\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else skip} : \Phi \Rightarrow \Phi'\end{array}}$$

$$\frac{\vdash \texttt{while}\ B\ \texttt{do}\ C : \Phi \Rightarrow \Phi' \qquad \text{[D-LU2]}}{\begin{array}{ll}\vdash & \texttt{while}\ B\ \texttt{do}\ C \\ \sim & \texttt{while}\ B\ \texttt{do}\ (C;\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else skip}) : \Phi \Rightarrow \Phi'\end{array}}$$

Self-assignment elimination:

$$\vdash X\texttt{:=}X \sim \texttt{skip} : \Phi, X : \phi_{\texttt{int}} \Rightarrow \Phi, X : \phi_{\texttt{int}}\ \text{[D-SAs]}$$

In conjunction with the core rules, the rules above can be used to derive many of the basic equalities one might expect.[2] From a pragmatic point of view, however, they are somewhat unwieldy: even very simple proofs get quite large, with many applications of the symmetry and transitivity rules and many repeated sub-derivations. Reformulating the rules as logically equivalent versions which

[2] Though the rules presented are in no sense complete. There are sound rules (arithmetic identities and equivalences for nested conditionals, for example) which are not consequences of the ones we have given.

can be applied in more general contexts helps immensely. For example, a better formulation of one of the `skip` rules is the following:

$$\frac{\vdash C \sim \texttt{skip} : \Phi \Rightarrow \Phi' \quad \vdash C' \sim C'' : \Phi' \Rightarrow \Phi''}{\vdash (C\,;C') \sim C'' : \Phi \Rightarrow \Phi''}\ \text{[D-SU1']}$$

Presenting rules in this style is essentially trying to produce a system with a kind of cut-elimination property, but we leave serious consideration of proof-theoretic matters to future work.

3.2.2 Optimizing Transformations

In this section we consider some more interesting rules, in which equations are predicated on information in the type system.

Dead assignment elimination:

$$\vdash (X \texttt{:=} E) \sim \texttt{skip} : \Phi, X : \phi_{\texttt{int}} \Rightarrow \Phi, X : \mathbb{T}_{\texttt{int}}\ \text{[D-DAs]}$$

Intuitively, the dead assignment rule says that an assignment to a variable is equivalent to `skip` *if* we are in a context in which the value of that variable does not matter.

Equivalent branches for conditional:

$$\frac{\vdash C_1 \sim C_2 : \Phi \Rightarrow \Phi'}{\vdash \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \sim C_1 : \Phi \Rightarrow \Phi'}\ \text{[D-BrE]}$$

An alternative form of this rule, which is a bit prettier, is

$$\frac{\vdash C_1 \sim C : \Phi \Rightarrow \Phi' \quad \vdash C_2 \sim C : \Phi \Rightarrow \Phi'}{\vdash \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \sim C : \Phi \Rightarrow \Phi'}\ \text{[D-BrE']}$$

Constant folding:

$$\frac{\vdash F_\tau : \Phi \Rightarrow \{c\}_\tau}{\vdash F_\tau \sim \texttt{c} : \Phi \Rightarrow \{c\}_\tau}\ \text{[D-CF]}$$

Known branch:

$$\frac{\vdash B : \Phi \Rightarrow \{true\} \quad \vdash C_1 \sim C' : \Phi \Rightarrow \Phi'}{\vdash (\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2) \sim C' : \Phi \Rightarrow \Phi'}\ \text{[D-KBT]}$$

$$\frac{\vdash B : \Phi \Rightarrow \{false\} \quad \vdash C_2 \sim C' : \Phi \Rightarrow \Phi'}{\vdash (\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2) \sim C' : \Phi \Rightarrow \Phi'}\ \text{[D-KBF]}$$

Dead while:

$$\frac{\vdash B : \Phi \Rightarrow \{false\}}{\vdash (\texttt{while}\ B\ \texttt{do}\ C) \sim \texttt{skip} : \Phi \Rightarrow \Phi}\ \text{[D-DWh]}$$

This is actually derivable using loop unrolling [D-LU1] and known branch [D-KBF].

Divergence:

$$\frac{\vdash B : \Phi \Rightarrow \{true\} \quad \vdash C : \Phi \Rightarrow \Phi}{\vdash (\mathtt{while}\ B\ \mathtt{do}\ C) : \Phi \Rightarrow \Phi'}\ \text{[D-Div]}$$

The type Φ' in the conclusion of the rule above is arbitrary because the loop will diverge when executed in any state in the domain of Φ.

The following is an easy induction, relying on Lemmas 1 and 2:

Theorem 1. *Assuming the abstract operations satisfy the correctness condition given in Definition 1, the core DDCC rules of Figure 2 and the additional rules of Section 3.2 are all sound:*

$$\begin{array}{rcl} \vdash F_\tau \sim F'_\tau : \Phi \Rightarrow \phi_\tau & \Longrightarrow & \models F_\tau \sim F'_\tau : \Phi \Rightarrow \phi_\tau \\ \vdash C \sim C' : \Phi \Rightarrow \Phi' & \Longrightarrow & \models C \sim C' : \Phi \Rightarrow \Phi' \end{array}$$

Proof. Induction on derivations, we show that each rule preserves semantic validity.

D-CT $\llbracket \Phi, X : \mathbb{F}_{\mathtt{int}} \rrbracket = \emptyset$ so $\llbracket \Phi, X : \mathbb{F}_{\mathtt{int}} \Rightarrow \Phi' \rrbracket = (\mathbb{S} \to \mathbb{S}_\bot) \times (\mathbb{S} \to \mathbb{S}_\bot)$.

D-ET1 $\llbracket \Phi \Rightarrow \mathbb{T}_\tau \rrbracket = (\mathbb{S} \to \llbracket \tau \rrbracket) \times (\mathbb{S} \to \llbracket \tau \rrbracket)$.

D-ET2 $\llbracket \Phi, X : \mathbb{F}_{\mathtt{int}} \rrbracket = \emptyset$ so $\llbracket \Phi, X : \mathbb{F}_{\mathtt{int}} \Rightarrow \phi_\tau \rrbracket = (\mathbb{S} \to \llbracket \tau \rrbracket) \times (\mathbb{S} \to \llbracket \tau \rrbracket)$.

D-ESym $\llbracket \Phi \to \phi_\tau \rrbracket$ is symmetric as it's a PER, Lemma 2.

D-ESub If $(S, S') \in \llbracket \Phi' \rrbracket$ and $\Phi' \leq \Phi$ then by Lemma 1, $(S, S') \in \llbracket \Phi \rrbracket$. Thus by induction $(\llbracket F_\tau \rrbracket S, \llbracket F'_\tau \rrbracket S') \in \llbracket \phi_\tau \rrbracket$. Since $\phi_\tau \leq \phi'_\tau$, by Lemma 1 we have $(\llbracket F_\tau \rrbracket S, \llbracket F'_\tau \rrbracket S') \in \llbracket \phi'_\tau \rrbracket$ as required.

D-CSub Observe that $\llbracket \Phi \rrbracket \subseteq \llbracket \Phi' \rrbracket$ implies $\llbracket \Phi \rrbracket_\bot \subseteq \llbracket \Phi' \rrbracket_\bot$, and reason as above.

D-ETr $\llbracket \Phi \to \phi_\tau \rrbracket$ is transitive because it's a PER, Lemma 2.

D-CSym $\llbracket \Phi \to \Phi' \rrbracket$ is symmetric because it's a PER.

D-CTr $\llbracket \Phi \to \Phi' \rrbracket$ is transitive because it's a PER.

D-V $(S, S') \in \llbracket \Phi, X : \phi_{\mathtt{int}} \rrbracket$ implies $(S(X), S'(X)) \in \llbracket \phi_{\mathtt{int}} \rrbracket$. So $(\llbracket X \rrbracket S, \llbracket X \rrbracket S') \in \llbracket \phi_{\mathtt{int}} \rrbracket$ as required.

D-N If $(S, S') \in \llbracket \Phi \rrbracket$, $(\llbracket \mathtt{n} \rrbracket S, \llbracket \mathtt{n} \rrbracket S') = (n, n) \in \llbracket \{n\}_{\mathtt{int}} \rrbracket$ as required.

D-B As above.

D-*op* Assume $(S, S') \in \llbracket \Phi \rrbracket$, then by induction $(\llbracket F \rrbracket S, \llbracket G \rrbracket S') \in \llbracket \phi \rrbracket$ and $(\llbracket F' \rrbracket S, \llbracket G' \rrbracket S') \in \llbracket \phi' \rrbracket$. Now, $(\llbracket F \mathtt{op} F' \rrbracket S, \llbracket G \mathtt{op} G' \rrbracket S') = (\llbracket F \rrbracket S\ op\ \llbracket F' \rrbracket S, \llbracket G \rrbracket S'\ op\ \llbracket G' \rrbracket S') \in \llbracket \phi\, \widehat{op}\, \phi' \rrbracket$ by the definition of soundness for abstract operations.

D-Skip If $(S, S') \in \llbracket \Phi \rrbracket$ then $(\llbracket \mathtt{skip} \rrbracket S, \llbracket \mathtt{skip} \rrbracket S') = (\lceil S \rceil, \lceil S' \rceil) \in \llbracket \Phi \rrbracket_\bot$.

D-Seq Assume $(S, S') \in \llbracket \Phi \rrbracket$. By induction $(\llbracket C_1 \rrbracket S, \llbracket C_2 \rrbracket S') \in \llbracket \Phi' \rrbracket_\bot$ so $(\llbracket C_1 \rrbracket S, \llbracket C_2 \rrbracket S')$ is either equal to $(\bot, \bot)$ or to $(\lceil T \rceil, \lceil T' \rceil)$ with $(T, T') \in \llbracket \Phi' \rrbracket$. In the first case, $(\llbracket C_1 ; C_2 \rrbracket S, \llbracket C'_1 ; C'_2 \rrbracket S') = (\bot, \bot) \in \llbracket \Phi'' \rrbracket_\bot$ as required. In the second case $(\llbracket C_1 ; C_2 \rrbracket S, \llbracket C'_1 ; C'_2 \rrbracket S') = (\llbracket C_2 \rrbracket T, \llbracket C'_2 \rrbracket T') \in \llbracket \Phi'' \rrbracket_\bot$ by induction.

D-Ass If $(S, S') \in \llbracket \Phi, X : \phi \rrbracket$ then by induction $(\llbracket E \rrbracket S, \llbracket E' \rrbracket S') \in \llbracket \phi' \rrbracket$. Thus $(\llbracket X \mathtt{:=} E \rrbracket S, \llbracket X \mathtt{:=} E' \rrbracket S') = (S[X \mapsto \llbracket E \rrbracket S], S'[X \mapsto \llbracket E' \rrbracket S']) \in \llbracket \Phi, X : \phi' \rrbracket$ (last part of Lemma 1).

D-Whl Define $f_n : \mathbb{S} \to \mathbb{S}_\perp$ by $f_0 = \lambda S.\perp$, $f_{n+1} = \lambda S.\llbracket B \rrbracket S \implies f_n^*(\llbracket C \rrbracket S) \mid \lceil S \rceil$ and similarly for f'_n. Clearly $(f_0, f'_0) \in \llbracket \Phi \to \Phi \rrbracket$. Now assume $(f_n, f'_n) \in \llbracket \Phi \to \Phi \rrbracket$ and $(S, S') \in \Phi$. By assumption $(\llbracket B \rrbracket S, \llbracket B' \rrbracket S') \in \llbracket \Delta_{\mathtt{bool}} \rrbracket$, so they're either both true or both false. In the latter case, $(f_{n+1} S, f'_{n+1} S') = (\lceil S \rceil, \lceil S' \rceil) \in \llbracket \Phi \rrbracket_\perp$. In the former case, $(f_{n+1} S, f'_{n+1} S') = (f_n^*(\llbracket C \rrbracket S), {f'_n}^*(\llbracket C' \rrbracket S'))$. By induction either $\llbracket C \rrbracket S = \llbracket C' \rrbracket S' = \perp$, in which case $(f_{n+1} S, f'_{n+1} S') = (\perp, \perp) \in \llbracket \Phi \rrbracket_\perp$, or $(\llbracket C \rrbracket S, \llbracket C' \rrbracket S') = (\lceil T \rceil, \lceil T' \rceil)$, with $(T, T') \in \llbracket \Phi \rrbracket$, in which case $(f_{n+1} S, f'_{n+1} S') = (f_n T, f'_n T') \in \llbracket \Phi \rrbracket_\perp$. Hence by mathematical induction, $(f_n, f'_n) \in \llbracket \Phi \to \Phi \rrbracket$ for all n and by admissibility $(\sqcup_n f_n, \sqcup_n f_n) \in \llbracket \Phi \to \Phi \rrbracket$ as required.

D-If If $(S, S') \in \llbracket \Phi \rrbracket$ then by assumption $(\llbracket B \rrbracket S, \llbracket B' \rrbracket S') \in \llbracket \Delta_{\mathtt{int}} \rrbracket$. Assume that they're both true, then $(\llbracket \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \rrbracket S, \llbracket \mathtt{if}\ B'\ \mathtt{then}\ C'_1\ \mathtt{else}\ C'_2 \rrbracket S') = (\llbracket C_1 \rrbracket S, \llbracket C'_1 \rrbracket S') \in \llbracket \Phi \rrbracket_\perp$ by the assumption about the true branches. Similar reasoning holds for the false branches, so we're done.

Basic equations These are all of the form: if $(\llbracket C \rrbracket, \llbracket C \rrbracket) \in \llbracket \Phi \to \Phi' \rrbracket$ then $(\llbracket C \rrbracket, \llbracket C' \rrbracket) \in \llbracket \Phi \to \Phi' \rrbracket$ where $\llbracket C \rrbracket = \llbracket C' \rrbracket$.

D-DAs Assume $(S, S') \in \llbracket \Phi, X : \phi \rrbracket$ then $(\llbracket X \mathtt{:=} E \rrbracket S, \llbracket \mathtt{skip} \rrbracket S') = (\lceil S[X \mapsto \llbracket E \rrbracket S] \rceil, \lceil S' \rceil)$. By Lemma 1 $(S[X \mapsto \llbracket E \rrbracket S], S') \in \llbracket \Phi \rrbracket$ and since $(\llbracket E \rrbracket S, S'(X)) \in \llbracket \mathbb{T}_{\mathtt{int}} \rrbracket$ we have $(\lceil S[X \mapsto \llbracket E \rrbracket S] \rceil, \lceil S' \rceil) \in \llbracket \Phi, X : \mathbb{T}_{\mathtt{int}} \rrbracket_\perp$ as required.

D-BrE Assume $(S, S') \in \llbracket \Phi \rrbracket$ so we know $(\llbracket C_1 \rrbracket S, \llbracket C_2 \rrbracket S') \in \llbracket \Phi' \rrbracket_\perp$. If $\llbracket B \rrbracket S = \mathit{true}$ we have $(\llbracket \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \rrbracket S, \llbracket C_1 \rrbracket S') = (\llbracket C_1 \rrbracket S, \llbracket C_1 \rrbracket S')$. Now, labouring the relational reasoning a bit (this is essentially just the relexivity on the domain of $\llbracket \Phi \to \Phi' \rrbracket$, which should be proved explicitly in the general bit about relations), $(S' S') \in \llbracket \Phi \rrbracket$ because $\llbracket \Phi \rrbracket$ is a PER. Hence $(\llbracket C_2 \rrbracket S', \llbracket C_1 \rrbracket S') \in \llbracket \Phi' \rrbracket_\perp$ by assumption and symmetry of $\llbracket \Phi' \rrbracket_\perp$ and so $(\llbracket C_1 \rrbracket S, \llbracket C_1 \rrbracket S') \in \llbracket \Phi' \rrbracket_\perp$ by transitivity of $\llbracket \Phi' \rrbracket_\perp$. If, on the other hand, $\llbracket B \rrbracket S = \mathit{false}$ we have $(\llbracket \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \rrbracket S, \llbracket C_1 \rrbracket S') = (\llbracket C_2 \rrbracket S, \llbracket C_1 \rrbracket S')$ and we're done by symmetry of $\llbracket \Phi \to \Phi' \rrbracket$.

D-BrE' This is much easier than the above - better to take this one as primitive in fact, and then just have [D-BrE] as a derived rule in the system (so the semantic reasoning above becomes syntactic reasoning in the proof system). Moreover, this version matches the one for RHL more closely.

D-CF If $(S, S') \in \llbracket \Phi \rrbracket$ then by assumption $(\llbracket F \rrbracket S, \llbracket F \rrbracket S') \in \llbracket \{c\} \rrbracket = \{(c, c)\}$ so $(\llbracket F \rrbracket S, \llbracket \mathtt{c} \rrbracket S') \in \llbracket \{c\} \rrbracket$.

D-KBT If $(S, S') \in \llbracket \Phi \rrbracket$, by the first premiss $\llbracket B \rrbracket S = \llbracket B \rrbracket S' = \mathit{true}$ so

$$(\llbracket \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \rrbracket S, \llbracket C' \rrbracket S') = (\llbracket C_1 \rrbracket S, \llbracket C' \rrbracket S') \in \llbracket \Phi' \rrbracket_\perp$$

by the second premiss.

D-KBF As above.

D-DWh If $(S, S') \in [\![\Phi]\!]$, by assumption $[\![B]\!]S = [\![B]\!]S' = false$ so

$$([\![\texttt{while}\ B\ \texttt{do}\ C]\!]S, [\![\texttt{skip}]\!]S') = (\lceil S \rceil, \lceil S' \rceil) \in [\![\Phi]\!]_\perp.$$

D-Div Define $f_0 = \lambda S.\perp$, $f_{n+1} = \lambda S.[\![B]\!]S \implies f_n^*([\![C]\!]S) \mid \lceil S \rceil$. Then we show that $\forall n.\forall (S, S') \in [\![\Phi]\!].(f_n S, f_n S') = (\perp, \perp)$, from which the result follows. The case $n = 0$ is obvious. Then by the assumption on B we know $(f_{n+1}S, f_{n+1}S') = (f_n^*([\![C]\!]S), (f_n^*([\![C]\!]S')$ and by the assumption on C, either $([\![C]\!]S, [\![C]\!]S') = (\perp, \perp)$ and the result follows by the definition of $(\cdot)^*$, or $([\![C]\!]S, [\![C]\!]S') = (\lceil T \rceil, \lceil T' \rceil)$ with $(T, T') \in [\![\Phi]\!]$ so $(f_{n+1}S, f_{n+1}S') = (f_n T, f_n T')$ and we're done by induction.

□

3.3 Example Transformations

These rules are sufficient to capture some non-trivial transformations, including constant propagation, dead-code elimination and program slicing [33]. Some example derivations are shown in Figure 3. We leave it as an exercise to prove larger examples, such as the slicing transformation:

```
I := 1;                  I := 1;
S := 0;
P := 1;                  P := 1
while I<N do (    ==>    while I<N do (
 S := S+I;
 P := P*I;                P := P*I;
 I := I+1;)               I := I+1;)
```

at type $N : \Delta_{\texttt{int}} \Rightarrow P : \Delta_{\texttt{int}}$. Here we expressed the fact that we were only interested in the final value of `P` simply by transforming it at a result type which only constrains the value of that variable to be preserved – all the others (in particular `S`) are typed at $\mathbb{T}_{\texttt{int}}$ and so are allowed to take any value.

Proof. Let $\Phi_0 = I : \mathbb{T}_{\texttt{int}}, S : \mathbb{T}_{\texttt{int}}, P : \mathbb{T}_{\texttt{int}}, N : \Delta_{\texttt{int}}$ and $\Phi_1 = I : \Delta_{\texttt{int}}, S : \mathbb{T}_{\texttt{int}}, P : \mathbb{T}_{\texttt{int}}, N : \Delta_{\texttt{int}}$. Then

$$\dfrac{\dfrac{\vdash \texttt{1} \sim \texttt{1} : \Phi_0 \Rightarrow \{1\} \quad \{1\} \leq \Delta_{\texttt{int}}}{\vdash \texttt{1} \sim \texttt{1} : \Phi_0 \Rightarrow \Delta_{\texttt{int}}}\text{[D-ESub]}}{\vdash \texttt{I :=1} \sim \texttt{I :=1} : \Phi_0 \Rightarrow \Phi_1}\text{[D-Ass]}$$

Then as

$$\vdash \texttt{S := 0} \sim \texttt{skip} : \Phi_1 \Rightarrow \Phi_1 \text{ [D-DAs]}$$

We can deduce

$$\vdash (\texttt{I := 1;S := 0}) \sim (\texttt{I := 1}) : \Phi_0 \Rightarrow \Phi_1$$

by [D-SU2']. Now let $\Phi_2 = I : \Delta_{\texttt{int}}, S : \mathbb{T}_{\texttt{int}}, P : \Delta_{\texttt{int}}, N : \Delta_{\texttt{int}}$ and similar reasoning yields

$$\vdash (\texttt{I := 1;S := 0;P:=1}) \sim (\texttt{I := 1;P:=1}) : \Phi_0 \Rightarrow \Phi_2$$

Constants, known branches and dead code:

$\mathcal{D}_1$:

$$\dfrac{\dfrac{\dfrac{}{\vdash X:\ \Phi,X:\{3\}\Rightarrow\{3\}}\,[\text{D-V}] \qquad \dfrac{}{\vdash \mathtt{3}:\ \Phi,X:\{3\}\Rightarrow\{3\}}\,[\text{D-N}]}{\vdash X=\mathtt{3}:\ \Phi,X:\{3\}\Rightarrow\{\mathit{true}\}}\,[\text{D-=}] \qquad \dfrac{\dfrac{}{\vdash \mathtt{7}:\ \Phi,X:\{3\}\Rightarrow\{7\}}\,[\text{D-N}]}{\vdash X\mathtt{:=7}:\ \Phi,X:\{3\}\Rightarrow\Phi,X:\{7\}}\,[\text{D-Ass}]}{\vdash(\mathtt{if}\ X=\mathtt{3}\ \mathtt{then}\ X\mathtt{:=7}\ \mathtt{else}\ \mathtt{skip})\sim(X\mathtt{:=7}):\Phi,X:\{3\}\Rightarrow\Phi,X:\{7\}}\,[\text{D-KBT}]$$

$\mathcal{D}_2$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash X:\Phi,X:\{7\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\{7\}}\,[\text{D-V}] \qquad \dfrac{}{\vdash \mathtt{1}:\Phi,X:\{7\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\{1\}}\,[\text{D-N}]}{\vdash X+\mathtt{1}:\Phi,X:\{7\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\{8\}}\,[\text{D-+}]}{\vdash X+\mathtt{1}\sim\mathtt{8}:\Phi,X:\{7\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\{8\}}\,[\text{D-CF}]}{\vdash \begin{array}{l}Z\mathtt{:=}X+\mathtt{1}\\ \sim Z\mathtt{:=8}\end{array} :\Phi,X:\{7\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\{7\},Z:\{8\}}\,[\text{D-Ass}] \qquad \Phi,X:\{7\},Z:\{8\}\ \le\ \Phi,X:\mathbb{T}_{\mathtt{int}},Z:\{8\}}{\vdash \begin{array}{l}Z\mathtt{:=}X+\mathtt{1}\\ \sim Z\mathtt{:=8}\end{array} :\Phi,X:\{7\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\{8\}}\,[\text{D-CSub}]$$

$\mathcal{D}_3$:

$$\dfrac{\dfrac{}{\vdash(X\mathtt{:=7})\sim\mathtt{skip}:\Phi,X:\{3\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\mathbb{T}_{\mathtt{int}}}\,[\text{D-DAss}] \qquad \dfrac{\vdash(\mathtt{8}):\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\{8\}}{\vdash Z\mathtt{:=8}:\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\{8\}}}{\vdash(X\mathtt{:=7};Z\mathtt{:=8})\sim Z\mathtt{:=8}:\Phi,X:\{3\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\{8\}}\,[\text{D-SU1'}]$$

$$\dfrac{\dfrac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\vdash \begin{array}{l}(\mathtt{if}\ X=\mathtt{3}\ \mathtt{then}\ X\mathtt{:=7}\ \mathtt{else}\ \mathtt{skip};Z\mathtt{:=}X+\mathtt{1})\\ \sim(X\mathtt{:=7};Z\mathtt{:=8})\end{array} :\Phi,X:\{3\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\{8\}}\,[\text{D-Seq}] \qquad \mathcal{D}_3}{\vdash \begin{array}{l}(\mathtt{if}\ X=\mathtt{3}\ \mathtt{then}\ X\mathtt{:=7}\ \mathtt{else}\ \mathtt{skip};Z\mathtt{:=}X+\mathtt{1})\\ \sim(Z\mathtt{:=8})\end{array} :\Phi,X:\{3\},Z:\mathbb{T}_{\mathtt{int}}\Rightarrow\Phi,X:\mathbb{T}_{\mathtt{int}},Z:\{8\}}\,[\text{D-CTr}]$$

Figure 3: Examples of DDCC Transformations

$$\gamma, X : \sigma_{\mathtt{int}} \vdash X : \sigma_{\mathtt{int}} \qquad \gamma \vdash \mathtt{n} : \sigma_{\mathtt{int}} \qquad \gamma \vdash \mathtt{b} : \sigma_{\mathtt{bool}}$$

$$\frac{\gamma \vdash E : \sigma_{\mathtt{int}} \quad \gamma \vdash E' : \sigma_{\mathtt{int}}}{\gamma \vdash E \ \mathtt{iop} E' \ : \sigma_{\mathtt{int}}} \ + \text{similar for } \textit{bop} \text{ and } \textit{lop}$$

$$\frac{\gamma, X : \sigma_{\mathtt{int}} \vdash E : \sigma_{\mathtt{int}}}{\gamma, X : \sigma_{\mathtt{int}} \vdash X \mathtt{:=} E : \sigma_{\mathtt{com}}}$$

$$\frac{\gamma \vdash C : \sigma_{\mathtt{com}} \quad \gamma \vdash C' : \sigma_{\mathtt{com}}}{\gamma \vdash C \mathtt{;} C' : \sigma_{\mathtt{com}}} \qquad \frac{\gamma \vdash B : \sigma_{\mathtt{bool}} \quad \gamma \vdash C : \sigma_{\mathtt{com}} \quad \gamma \vdash C' : \sigma_{\mathtt{com}}}{\gamma \vdash \mathtt{if}\ B\ \mathtt{then}\ C\ \mathtt{else}\ C' : \sigma_{\mathtt{com}}}$$

$$\frac{\gamma \vdash B : L_{\mathtt{bool}} \quad \gamma \vdash C : L_{\mathtt{com}}}{\gamma \vdash \mathtt{while}\ B\ \mathtt{do}\ C : L_{\mathtt{com}}} \qquad \frac{\gamma \vdash F : L_{\tau}}{\gamma \vdash F : H_{\tau}} \qquad \frac{\gamma \vdash C : H_{\mathtt{com}}}{\gamma \vdash C : L_{\mathtt{com}}}$$

Figure 4: Smith/Volpano Type System

Next we show

$$\vdash \mathtt{I} < \mathtt{N} \sim \mathtt{I} < \mathtt{N} : \Phi_2 \Rightarrow \Delta_{\mathtt{bool}}$$

and

$$\vdash (\mathtt{S} := \mathtt{S} + \mathtt{1}; \mathtt{P} := \mathtt{P} * \mathtt{I}; \mathtt{I} := \mathtt{I} + \mathtt{1}) \sim (\mathtt{P} := \mathtt{P} * \mathtt{I}; \mathtt{I} := \mathtt{I} + \mathtt{1}) : \Phi_2 \Rightarrow \Phi_2$$

so that, by [D-Whl]:

$$\vdash \begin{array}{l} (\mathtt{while}\ \mathtt{I} < \mathtt{N}\ \mathtt{do}\ (\mathtt{S} := \mathtt{S} + \mathtt{1}; \mathtt{P} := \mathtt{P} * \mathtt{I}; \mathtt{I} := \mathtt{I} + \mathtt{1})) \sim \\ (\mathtt{while}\ \mathtt{I} < \mathtt{N}\ \mathtt{do}\ (\mathtt{P} := \mathtt{P} * \mathtt{I}; \mathtt{I} := \mathtt{I} + \mathtt{1})) \end{array} : \Phi_2 \Rightarrow \Phi_2$$

Then as $N : \Delta_{\mathtt{int}} \leq \Phi_0$ and $\Phi_2 \leq P : \Delta_{\mathtt{int}}$ we can plug the bits together with [D-Seq] and [D-CSub] and we're done. □

3.4 Secure Information Flow

It is worth observing that the $\mathbb{T}$,Δ fragment of our calculus can be seen as a non-interference type system. Figure 4 presents a version of a type system for secure information flow due to Smith and Volpano [27]. In this system, a *security level*, σ, is either low (L) or high (H). A context γ is then a finite map from variables to security levels:

$$\gamma \quad := \quad - \mid \gamma, X : \sigma_{\mathtt{int}}$$

Given such a context, the type system assigns a security level ($\sigma_{\mathtt{int}}$ or $\sigma_{\mathtt{bool}}$) to each expression and ($\sigma_{\mathtt{com}}$) to each command. The property which the type system ensures is that any typeable command does not allow information to flow (either directly, via assignment, or indirectly, via control flow) from high security variables to low security ones. We define a translation $(\cdot)^*$ from the Smith/Volpano system into DDCC as follows:

Expression types: $L^*_\tau = \Delta_\tau$ and $H^*_\tau = \mathbb{T}_\tau$.

Contexts: $-^* = -$ and $(\gamma, X : \sigma_{\mathtt{int}})^* = \gamma^*, X : \sigma^*_{\mathtt{int}}$.

Judgements:

$$\begin{array}{rcl}(\gamma \vdash F : \sigma_\tau)^* & = & \vdash F \sim F : \gamma^* \Rightarrow \sigma^*_\tau \\ (\gamma \vdash C : L_{\mathtt{com}})^* & = & \vdash C \sim C : \gamma^* \Rightarrow \gamma^* \\ (\gamma \vdash C : H_{\mathtt{com}})^* & = & \vdash C \sim \mathtt{skip} : \gamma^* \Rightarrow \gamma^*\end{array}$$

Theorem 2. *For any judgement J derivable in the Smith/Volpano system, J^* is derivable in DDCC*

Proof. This is a simple induction, relying on the dead assignment axiom in the case of high assignment statements, sequential unit for high sequential compositions and the equivalent branch rule for high conditionals.

Variables $(\gamma, X : \sigma_{\mathtt{int}} \vdash X : \sigma_{\mathtt{int}})^* \quad = \quad \vdash X \sim X : \gamma^*, X : \sigma^*_{\mathtt{int}} \Rightarrow \sigma^*_{\mathtt{int}}$ [D-V].

Constants

$$\frac{\vdash \mathtt{n} \sim \mathtt{n} : \gamma^* \Rightarrow \{n\} \quad \{n\} \leq \sigma^*_{\mathtt{int}}}{(\gamma \vdash \mathtt{n} : \sigma_{\mathtt{int}})^* \; = \vdash \mathtt{n} \sim \mathtt{n} : \gamma^* \Rightarrow \sigma^*_{\mathtt{int}}} \text{[D-ESub]}$$

And similarly for booleans.

Ops By induction have derivations of

$$\begin{array}{rcl}(\gamma \vdash E : \sigma_{\mathtt{int}})^* & = & \vdash E \sim E : \gamma^* \Rightarrow \sigma^*_{\mathtt{int}} \\ (\gamma \vdash E' : \sigma_{\mathtt{int}})^* & = & \vdash E' \sim E' : \gamma^* \Rightarrow \sigma^*_{\mathtt{int}}\end{array}$$

And hence can derive

$$\vdash E \; \mathtt{iop} \; E' \sim E \; \mathtt{iop} \; E' : \gamma^* \Rightarrow \sigma^*_{\mathtt{int}} \, \widehat{iop} \, \sigma^*_{\mathtt{int}}$$

by [D-iop]. Now observe that for any operation $iop : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, there is a sound abstraction $\widehat{iop}$ such that $\Delta_{\mathtt{int}} \, \widehat{iop} \, \Delta_{\mathtt{int}} \leq \Delta_{\mathtt{int}}$ and $\mathbb{T}_{\mathtt{int}} \, \widehat{iop} \, \mathbb{T}_{\mathtt{int}} \leq \mathbb{T}_{\mathtt{int}}$ so for either value of σ, we can get

$$(\gamma \vdash E \; \mathtt{iop} \; E' : \sigma_{\mathtt{int}})^* \; = \vdash E \; \mathtt{iop} \; E' \sim E \; \mathtt{iop} \; E' : \gamma^* \Rightarrow \sigma^*_{\mathtt{int}}$$

by [D-ESub].

Assign (L)

$$\frac{(\gamma, X : L_{\mathtt{int}} \vdash E : L_{\mathtt{int}})^* \; = \vdash E \sim E : \gamma^*, X : \Delta_{\mathtt{int}} \Rightarrow \Delta_{\mathtt{int}}}{(\gamma, X : L_{\mathtt{int}} \vdash X \mathtt{:=} E : L_{\mathtt{com}})^* \; = \vdash X \mathtt{:=} E \sim X \mathtt{:=} E : \gamma^*, X : \Delta_{\mathtt{int}} \Rightarrow \gamma^*, X : \Delta_{\mathtt{int}}} \text{[D-Ass]}$$

Assign (H) We don't even need the inductive hypothesis here:

$$(\gamma, X : H_{\mathtt{int}} \vdash X \mathtt{:=} E : H_{\mathtt{com}})^* \; = \vdash X \mathtt{:=} E \sim \mathtt{skip} : \gamma^*, X : \mathbb{T}_{\mathtt{int}} \Rightarrow \gamma^*, X : \mathbb{T}_{\mathtt{int}} \text{ [D-DAs]}$$

Seq L $$\frac{(\gamma \vdash C : L_{\mathtt{com}})^* = \vdash C \sim C : \gamma^* \Rightarrow \gamma^* \quad (\gamma \vdash C' : L_{\mathtt{com}})^* = \vdash C' \sim C' : \gamma^* \Rightarrow \gamma^*}{(\gamma \vdash C ; C' : L_{\mathtt{com}})^* = \vdash C ; C' \sim C ; C' : \gamma^* \Rightarrow \gamma^*} \text{[D-Seq]}$$

Seq H $$\frac{(\gamma \vdash C : H_{\texttt{com}})^* =\vdash C \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^* \quad (\gamma \vdash C' : H_{\texttt{com}})^* =\vdash C' \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^*}{(\gamma \vdash C;C' : H_{\texttt{com}})^* =\vdash C;C' \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^*} \text{[D-SU1']}$$

Cond L We have by assumption

$$\begin{aligned}(\gamma \vdash B : L_{\texttt{bool}})^* &= \vdash B \sim B : \gamma^* \Rightarrow \Delta_{\texttt{bool}} \\ (\gamma \vdash C : L_{\texttt{com}})^* &= \vdash C \sim C : \gamma^* \Rightarrow \gamma^* \\ (\gamma \vdash C' : L_{\texttt{com}})^* &= \vdash C' \sim C' : \gamma^* \Rightarrow \gamma^*\end{aligned}$$

So applying [D-If] yields

$$\vdash \texttt{if } B \texttt{ then } C \texttt{ else } C' \sim \texttt{if } B \texttt{ then } C \texttt{ else } C' : \gamma^* \Rightarrow \gamma^*$$

which is $(\gamma \vdash \texttt{if } B \texttt{ then } C \texttt{ else } C' : L_{\texttt{com}})^*$ as required.

Cond H We have by assumption

$$\begin{aligned}(\gamma \vdash C : H_{\texttt{com}})^* &= \vdash C \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^* \\ (\gamma \vdash C' : H_{\texttt{com}})^* &= \vdash C' \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^*\end{aligned}$$

The inductive assumption on B is not needed in this case, as we can just apply [D-BrE'] to the above to deduce

$$\vdash \texttt{if } B \texttt{ then } C \texttt{ else } C' \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^*$$

which is $(\gamma \vdash \texttt{if } B \texttt{ then } C \texttt{ else } C' : H_{\texttt{com}})^*$ as required.

While By assumption we have

$$\begin{aligned}(\gamma \vdash B : L_{\texttt{bool}})^* &= \vdash B \sim B : \gamma^* \Rightarrow \Delta_{\texttt{bool}} \\ (\gamma \vdash C : L_{\texttt{com}})^* &= \vdash C \sim C : \gamma^* \Rightarrow \gamma^*\end{aligned}$$

so we can apply [D-Whl] to deduce

$$\vdash \texttt{while } B \texttt{ do } C \sim \texttt{while } B \texttt{ do } C : \gamma^* \Rightarrow \gamma^*$$

which is $(\gamma \vdash \texttt{while } B \texttt{ do } C : L_{\texttt{com}})^*$ as required.

Exp-Sub $$\frac{(\gamma \vdash F : L_\tau)^* \ = \vdash F \sim F : \gamma^* \Rightarrow \Delta_\tau \quad \Delta_\tau \leq \mathbb{T}_\tau}{(\gamma \vdash F : H_\tau)^* \ = \vdash F \sim F : \gamma^* \Rightarrow \mathbb{T}_\tau}$$

Com-Sub By induction have derivation of

$$(\gamma \vdash C : H_{\texttt{com}})^* \ = \vdash C \sim \texttt{skip} : \gamma^* \Rightarrow \gamma^*$$

so we can apply [D-CSym] to get

$$\vdash \texttt{skip} \sim C : \gamma^* \Rightarrow \gamma^*$$

and then [D-CTr] to yield

$$\vdash C \sim C : \gamma^* \Rightarrow \gamma^*$$

which is $(\gamma \vdash C : L_{\texttt{com}})^*$ as required.

□

Definition 2. *In the context of a security type assignment* γ*, a command* C *satisfies* strong sequential noninterference *if* $\models C \sim C : \gamma^* \Rightarrow \gamma^*$.

This version of non-interference is the semantic security property intended by Smith and Volpano, though the actual property established by the soundness proof in [27] is more syntactic and intensional, as it is defined in terms of their particular typing rules. Our notion of intereference is *strong* because it is termination-sensitive: varying the high-security inputs affects neither the low-security outputs *nor* the termination behaviour. In the absence of any termination analysis, this is enforced by the rather brutal approach of making all high-security commands total. The weaker notion of non-interference that is achieved by the earlier system of Volpano, Smith and Irvine [29] does not seem to translate directly into DDCC.

Even without constant tracking, DDCC is marginally more powerful than the Smith/Volpano system. For example, if `H` is a high-security variable, and `L` is low-security then the following are easily shown to satisfy non-interference in DDCC, but would be rejected by the Smith/Volpano system:

1. `if H > 3 then H := L ; L := 1 else L := 1`
2. `L := H ; L := 3`

4 Relational Hoare Logic

There are many common optimizing transformations which are not captured by DDCC. In particular:

- It does not capture any transformations that take advantage of the fact that one knows statically which way a boolean test must have evaluated if one is within a particular branch of a conditional, or either in the body of or have just left a `while`-loop. For example, the judgement

  $$\begin{array}{c}\vdash (\texttt{if}\ X = \texttt{3}\ \texttt{then}\ Y\texttt{:=}X\ \texttt{else}\ Y\texttt{:=3}) \\ \sim (Y\texttt{:=3}) : X : \Delta \Rightarrow Y : \{3\}\end{array}$$

  is semantically valid but not derivable.

- It cannot express the preservation of the values of expressions, except where they are statically known to be a particular constant. These means even trivial code-motion transformations cannot be derived.

We can address these shortcomings by making piecemeal additions to the system, such as quantification over variables ranging over integers or PERs. However, there is a simple and elegant system, which we call Relational Hoare Logic (RHL), into which many of these extensions or alternative type systems can be embedded.

Unlike DDCC, RHL does not look like a conventional type-based analysis system – it has a rather general syntax for relations and is parameterized on some system for deciding the entailment relation between them. The intention is that more specific analyses and transformations can be formulated as subsystems of RHL by restricting the syntax of assertions and providing particular approximations to the entailment relation. Another way in which RHL goes beyond DDCC is that it is not restricted to partial equivalence relations, which deserves some comment.

PERs are certainly privileged: they are the basis of equational reasoning, and we will nearly always be trying to prove that one program phrase is related to another by a PER so that we can perform a rewrite in some context. However, in order to establish that two phrases are related by a PER, we often have to do some local reasoning using more general relations. This is familiar in the semantics of polymorphic type theories: types are interpreted by PERs, and polymorphism by quantification over PERs, but parametricity theorems and equivalence results for implementations of abstract dataypes arise from substituting more general relations. To give some intuition for why this might be so, consider proving the equivalence

```
X := -Y;              X := Y;
Z := Z-X;      ==>    Z := Z+X;
X := -X;
```

at, say, $Y : \Delta_{\mathtt{int}}, Z : \Delta_{\mathtt{int}} \Rightarrow X : \Delta_{\mathtt{int}}, Y : \Delta_{\mathtt{int}}, Z : \Delta_{\mathtt{int}}$. If we try to to establish that the two commands are related by this PER by relating their intermediate states (though this is not the only approach one could take), we will need to use the relation that the value of `X` in one state is the negation of that in the other, which is not a PER.

RHL is an extremely simple variation on traditional Floyd-Hoare logic [13]. Instead of assertions which denote predicates on states and judgements which say that terminating execution of a command in a state satisfying a precondition will yield a state satisfying a postcondition, we directly axiomatise when a *pair* of commands map a given pre-*relation* into a given post-*relation*. Binary relations on states are simply specified by boolean expressions of the language over variables tagged with an indication of which of the two states they refer to. At first sight, this may seem frighteningly simple-minded, but it actually works rather nicely. In this presentation we do not consider quantification over metavariables ("ghost variables") denoting integers: their addition is straighforward, but simple global analyses seem to be expressible without them.

4.1 RHL Syntax and Semantics

4.1.1 Syntax

We define generalized expressions and relational assertions as follows:

$$\begin{array}{rcl} \mathtt{gexp} \ni GE & := & \mathtt{n} \mid X\langle 1\rangle \mid X\langle 2\rangle \mid GE\ \mathtt{iop}\ GE \\ \mathtt{relexp} \ni \Phi & := & \mathtt{b} \mid GE\ \mathtt{bop}\ GE \mid \mathtt{not}\Phi \mid \Phi\ \mathtt{lop}\ \Phi \end{array}$$

We overload the notation $(\cdot)\langle 1\rangle$ and $(\cdot)\langle 2\rangle$ to stand for homomorphic embeddings `int exp` → `gexp` and `bool exp` → `relexp` in the obvious way. The basic judgement form is $\vdash C \sim C' : \Phi \Rightarrow \Phi'$ (though the use of $\sim$ for arbitrary relations is arguably bad).

$$\vdash \texttt{skip} \sim \texttt{skip} : \Phi \Rightarrow \Phi \text{ [R-Skip]}$$

$$\frac{\vdash C \sim C' : \Phi \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \Rightarrow \Phi' \quad \vdash D \sim D' : \Phi \wedge \texttt{not}(B\langle 1\rangle \vee B'\langle 2\rangle) \Rightarrow \Phi'}{\vdash \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ D \sim \texttt{if}\ B'\ \texttt{then}\ C'\ \texttt{else}\ D' : \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle) \Rightarrow \Phi'} \text{ [R-If]}$$

$$\frac{\vdash C \sim C' : \Phi \Rightarrow \Phi' \quad \vdash D \sim D' : \Phi' \Rightarrow \Phi''}{\vdash C\ \texttt{;}\ D \sim C'\ \texttt{;}\ D' : \Phi \Rightarrow \Phi''} \text{ [R-Seq]}$$

$$\vdash X\ \texttt{:=}\ E \sim Y\ \texttt{:=}\ E' : \Phi[E\langle 1\rangle / X\langle 1\rangle, E'\langle 2\rangle / Y\langle 2\rangle] \Rightarrow \Phi \text{ [R-Ass]}$$

$$\frac{\vdash C \sim C' : \Phi \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \Rightarrow \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle)}{\vdash \texttt{while}\ B\ \texttt{do}\ C \sim \texttt{while}\ B'\ \texttt{do}\ C' : \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle) \Rightarrow \Phi \wedge \texttt{not}(B\langle 1\rangle \vee B'\langle 2\rangle)} \text{ [R-Whl]}$$

$$\frac{\vdash C \sim C' : \Phi_1 \Rightarrow \Phi_2 \quad \models \Phi_1' \leq \Phi_1 \quad \models \Phi_2 \leq \Phi_2'}{\vdash C \sim C' : \Phi_1' \Rightarrow \Phi_2'} \text{ [R-Sub]}$$

$$\frac{\vdash C \sim C' : \Phi \Rightarrow \Phi' \quad \models PER(\Phi \Rightarrow \Phi')}{\vdash C' \sim C : \Phi \Rightarrow \Phi'} \text{ [R-Sym]}$$

$$\frac{\vdash C \sim C' : \Phi \Rightarrow \Phi' \quad \vdash C' \sim C'' : \Phi \Rightarrow \Phi' \quad \models PER(\Phi \Rightarrow \Phi')}{\vdash C \sim C'' : \Phi \Rightarrow \Phi'} \text{ [R-Tr]}$$

Figure 5: Core Relational Hoare Logic

4.1.2 Semantics

The semantics of generalized expressions as integer-valued functions of two states, and of relational assertions as relations on states is unsurprising:

$$
\begin{array}{rcl}
[\![GE]\!] & \in & \mathbb{S} \times \mathbb{S} \to \mathbb{Z} \\
[\![\mathtt{n}]\!](S_1, S_2) & = & n \\
[\![X\langle 1\rangle]\!](S_1, S_2) & = & S_1(X) \\
[\![X\langle 2\rangle]\!](S_1, S_2) & = & S_2(X) \\
[\![E \ \mathtt{iop}\ F]\!](S_1, S_2) & = & ([\![E]\!](S_1, S_2)) \ \ iop \ \ ([\![F]\!](S_1, S_2))
\end{array}
$$

$$
\begin{array}{rcl}
[\![\Phi]\!] & \subseteq & \mathbb{S} \times \mathbb{S} \\
 & = & \{(S, S') \mid \chi_\Phi(S, S') = true\} \\
\chi_{\mathtt{true}}(S'S') & = & true \\
\chi_{\mathtt{false}}(S, S') & = & false \\
\chi_{E\ \mathtt{bop}\ F}(S, S') & = & [\![E]\!](S, S') \ \ bop \ \ [\![F]\!](S, S') \\
\chi_{\Phi\ \mathtt{lop}\ \Phi'}(S, S') & = & \chi_\Phi(S, S') \ \ lop \ \ \chi_{\Phi'}(S, S') \\
\chi_{\mathtt{not}\Phi}(S, S') & = & \neg(\chi_\Phi(S, S'))
\end{array}
$$

The intended meaning of judgements is given by

$$
\begin{array}{rl}
 & \models C \sim C' : \Phi \Rightarrow \Phi' \\
\equiv & \forall (S_1, S_2) \in [\![\Phi]\!].\ ([\![C]\!](S_1), [\![C']\!](S_2)) \in [\![\Phi']\!]_\perp
\end{array}
$$

We will also need some auxiliary semantic judgements, whose meanings are as follows:

$$
\begin{array}{rcl}
\models \Phi \leq \Phi' & \equiv & [\![\Phi]\!] \subseteq [\![\Phi']\!] \\
\models PER(\Phi) & \equiv & ([\![\Phi]\!] \circ [\![\Phi]\!] \subseteq [\![\Phi]\!]) \text{ and } ([\![\Phi]\!]^{-1} \subseteq [\![\Phi]\!])
\end{array}
$$

Lemma 3.

1. *For all GE,GE',X,S,S':*

$$
\begin{array}{rl}
 & [\![GE[GE'/X\langle 1\rangle]]\!](S, S') \\
= & [\![GE]\!](S[X \mapsto [\![GE']\!](S, S')], S')
\end{array}
$$

And similarly for $X\langle 2\rangle$ and S'.

2. *For all Φ, GE,X,S,S':*

$$\chi_{\Phi[GE/X\langle 1\rangle]}(S, S') = \chi_\Phi(S[X \mapsto [\![GE]\!](S, S')], S')$$

And similarly for $X\langle 2\rangle$ and S'.

3. *For all Φ and Φ', $[\![\Phi \Rightarrow \Phi']\!]$ is an admissible relation.*

4. *For any $E \in$ **int exp**,S,S', $[\![E\langle 1\rangle]\!](S, S') = [\![E]\!]S$ and similarly for $E\langle 2\rangle$ and S'. For any $B \in$ **bool exp**, $\chi_{B\langle 1\rangle}(S, S') = [\![B]\!]S$ and similarly for $B\langle 2\rangle$ and S’.*

Proof. 1. Induction on GE:

- constants

$$\begin{array}{rcl} LHS & = & \llbracket \mathtt{n}[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket \mathtt{n}\rrbracket(S,S') \\ & = & n \\ RHS & = & \llbracket \mathtt{n}\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S') \\ & = & n \end{array}$$

- variable $X\langle 1\rangle$

$$\begin{array}{rcl} LHS & = & \llbracket X\langle 1\rangle[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket GE'\rrbracket(S,S') \\ RHS & = & \llbracket X\langle 1\rangle\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S') \\ & = & S[X \mapsto \llbracket GE'\rrbracket(S,S')](X) \\ & = & \llbracket GE'\rrbracket(S,S') \end{array}$$

- variable $Y\langle 1\rangle$, $X \neq Y$

$$\begin{array}{rcl} LHS & = & \llbracket Y\langle 1\rangle[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket Y\langle 1\rangle\rrbracket(S,S') \\ & = & S(Y) \\ RHS & = & \llbracket Y\langle 1\rangle\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S') \\ & = & S[X \mapsto \llbracket GE'\rrbracket(S,S')](Y) \\ & = & S(Y) \end{array}$$

- variable $Y\langle 2\rangle$ (any Y)

$$\begin{array}{rcl} LHS & = & \llbracket Y\langle 2\rangle[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket Y\langle 2\rangle\rrbracket(S,S') \\ & = & S'(Y) \\ RHS & = & \llbracket Y\langle 2\rangle\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S') \\ & = & S'(Y) \end{array}$$

- $E\ iop\ F$

$$\begin{array}{rcl} LHS & = & \llbracket (E\mathtt{iop}F)[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket E[GE'/X\langle 1\rangle]\mathtt{iop}F[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket E[GE'/X\langle 1\rangle]\rrbracket(S'S')\ iop\ \llbracket F[GE'/X\langle 1\rangle]\rrbracket(S,S') \\ & = & \llbracket E\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S')\ iop\ \llbracket F\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S') \text{ induction} \\ & = & \llbracket E\mathtt{iop}F\rrbracket(S[X \mapsto \llbracket GE'\rrbracket(S,S')],S') \\ & = & RHS \end{array}$$

2. Induction on Φ:

- true

$$\begin{aligned}
LHS &= \chi_{\texttt{true}[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{\texttt{true}}(S,S') \\
&= true \\
&= \chi_{\texttt{true}}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S') \\
&= RHS
\end{aligned}$$

- false

$$\begin{aligned}
LHS &= \chi_{\texttt{false}[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{\texttt{false}}(S,S') \\
&= false \\
&= \chi_{\texttt{false}}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S') \\
&= RHS
\end{aligned}$$

- $E\texttt{bop}F$

$$\begin{aligned}
LHS &= \chi_{(E\texttt{bop}F)[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{E[GE/X\langle 1\rangle]\texttt{bop}F[GE/X\langle 1\rangle]}(S,S') \\
&= \llbracket E[GE/X\langle 1\rangle]\rrbracket(S,S')\ bop\ \llbracket F[GE/X\langle 1\rangle]\rrbracket(S,S') \\
&= \llbracket E\rrbracket(S[X \mapsto \llbracket GE \rrbracket(S,S')],S')\ bop\ \llbracket F\rrbracket(S[X \mapsto \llbracket GE \rrbracket(S,S')],S')\ \texttt{part1} \\
&= \llbracket E\texttt{bop}F\rrbracket(S[X \mapsto \llbracket GE \rrbracket(S,S')],S') \\
&= RHS
\end{aligned}$$

- $\Phi\texttt{lop}\Phi'$

$$\begin{aligned}
LHS &= \chi_{(\Phi\texttt{lop}\Phi')[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{\Phi[GE/X\langle 1\rangle]\texttt{lop}\Phi'[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{\Phi[GE/X\langle 1\rangle]}(S,S')\ lop\ \chi_{\Phi'[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{\Phi}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S')\ lop\ \chi_{\Phi'}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S')\ \text{induction} \\
&= \chi_{\Phi\texttt{lop}\Phi'}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S') \\
&= RHS
\end{aligned}$$

- $\texttt{not}\Phi$

$$\begin{aligned}
LHS &= \chi_{(\texttt{not}\Phi)[GE/X\langle 1\rangle]}(S,S') \\
&= \chi_{\texttt{not}(\Phi[GE/X\langle 1\rangle])}(S,S') \\
&= \neg(\chi_{\Phi[GE/X\langle 1\rangle]}(S,S')) \\
&= \neg(\chi_{\Phi}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S'))\ \text{induction} \\
&= \chi_{\texttt{not}\Phi}(S[X \mapsto \llbracket GE \rrbracket(S,S')],S') \\
&= RHS
\end{aligned}$$

3. Basic facts about relations on flat domains.

4. Another trivial induction.

□

4.1.3 Inference Rules

The core rules for RHL are shown in Figure 5. Observe that, as was the case in DDCC, the basic rules ensure that the same conditional branches are taken and that loops are executed the same number of times on the two sides. Note also that one could add distinct semantic judgements for symmetry and transitivity, rather than requiring both. The assignment rule is surprisingly liberal, but there is no reason to require the assigned variables to be the same in both commands.

4.2 Equations

As with DDCC, we will specify optimizing transformations by adding extra (sound) rules to the core. But even before we do that, RHL can justify some useful transformations. Here's an example of removing a redundant evaluation:

1. $\vdash \begin{array}{l} Z\texttt{:=}Y\texttt{+1} \\ \sim Z\texttt{:=}X \end{array} : \begin{array}{l} X\langle 1\rangle = X\langle 2\rangle \wedge \\ Y\langle 1\rangle + 1 = X\langle 2\rangle \end{array} \Rightarrow \begin{array}{l} X\langle 1\rangle = X\langle 2\rangle \wedge \\ Z\langle 1\rangle = Z\langle 2\rangle \end{array}$ by [R-Ass]
2. $\vdash \begin{array}{l} X\texttt{:=}Y\texttt{+1} \\ \sim X\texttt{:=}Y\texttt{+1} \end{array} : \begin{array}{l} Y\langle 1\rangle + 1 = Y\langle 2\rangle + 1 \wedge \\ Y\langle 1\rangle + 1 = Y\langle 2\rangle + 1 \end{array} \Rightarrow \begin{array}{l} X\langle 1\rangle = X\langle 2\rangle \wedge \\ Y\langle 1\rangle + 1 = X\langle 2\rangle \end{array}$ by [R-Ass]
3. $\models (Y\langle 1\rangle = Y\langle 2\rangle) \leq \begin{array}{l} Y\langle 1\rangle + 1 = Y\langle 2\rangle + 1 \wedge \\ Y\langle 1\rangle + 1 = Y\langle 2\rangle + 1 \end{array}$ by logic
4. $\vdash \begin{array}{l} X\texttt{:=}Y\texttt{+1} \\ \sim X\texttt{:=}Y\texttt{+1} \end{array} : Y\langle 1\rangle = Y\langle 2\rangle \Rightarrow \begin{array}{l} X\langle 1\rangle = X\langle 2\rangle \wedge \\ Y\langle 1\rangle + 1 = X\langle 2\rangle \end{array}$ by [R-Sub] applied to 2. and 3.
5. $\vdash \begin{array}{l} X\texttt{:=}Y\texttt{+1;}Z\texttt{:=}Y\texttt{+1} \\ \sim X\texttt{:=}Y\texttt{+1;}Z\texttt{:=}X \end{array} : Y\langle 1\rangle = Y\langle 2\rangle \Rightarrow \begin{array}{l} X\langle 1\rangle = X\langle 2\rangle \wedge \\ Z\langle 1\rangle = Z\langle 2\rangle \end{array}$ by [R-Seq] applied to 4. and 1.

4.2.1 Basic Equations

The basic equations we presented in the context of DDCC are still valid for RHL, with the exception of self-assignment elimination, though the contextual versions are now more powerful than the simple ones, so we take [R-SU1'L] and [R-SU2'L] (and their symmetric versions) as basic.

I believe an RHL version of [D-CC] is probably admissible given the other RHL rules presented here, but admissibility is not preserved by adding further rules, so it seems easiest to add something explicit, at least to make sure that the DDCC embedding theorem is valid. There are a number of choices for natural RHL rules which imply [D-CC]. The following is one:

$$\frac{\vdash \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \sim C : \Phi \Rightarrow \Phi'}{\vdash C_1 \sim C : \Phi \wedge B\langle 1\rangle \Rightarrow \Phi'} \text{[R-CBInvTL]}$$

$$\frac{\vdash \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \sim C : \Phi \Rightarrow \Phi'}{\vdash C_2 \sim C : \Phi \wedge \texttt{not} B\langle 1\rangle \Rightarrow \Phi'} \text{[R-CBInvFL]}$$

Plus the obvious ones the other way around. These are the inverted versions of [R-CB] (see later).

We also add the loop unrolling rules [R-LU1L] and [R-LU2L] and their right-handed versions, mainly to ensure that the embedding of DDCC in RHL works. A better treatment of loops in RHL is to have a more powerful inductive rule, but that is not discussed here since this report is intended to match the POPL paper. In passing, it should also be remarked that the messy business of having left and right variants of rules can also be avoided by use of a Φ^{-1} rule, which swaps 1 and 2. (There's also a case to be made for adding syntactic relational composition.)

4.2.2 Optimizing Transformations

Falsity:

$$\vdash C \sim C' : \texttt{false} \Rightarrow \Phi \text{ [R-F]}$$

Dead assignment:

$$\vdash X\texttt{:=}E \sim \texttt{skip} : \Phi[E\langle 1\rangle/X\langle 1\rangle] \Rightarrow \Phi \text{ [R-DAssL]}$$

$$\vdash \texttt{skip} \sim X\texttt{:=}E : \Phi[E\langle 2\rangle/X\langle 2\rangle] \Rightarrow \Phi \text{ [R-DAssR]}$$

These rules subsume our previous dead-assignment and self-assignment rules. With the basic rules for skip, they subsume the [R-Ass] rule too.

Common branch:

$$\frac{\vdash C \sim D : \Phi \wedge B\langle 1\rangle \Rightarrow \Phi' \quad \vdash C' \sim D : \Phi \wedge \texttt{not}B\langle 1\rangle \Rightarrow \Phi'}{\vdash \texttt{if } B \texttt{ then } C \texttt{ else } C' \sim D : \Phi \Rightarrow \Phi'} \text{ [R-CBL]}$$

Plus a version with the conditional on the right. These subsume our earlier equivalent branch rule, and (via the falsity equation) the known-branch rules and the [R-If] rule.

Dead while:

$$\vdash \texttt{while } B \texttt{ do } C \sim \texttt{skip} : \Phi \wedge \texttt{not}B\langle 1\rangle \Rightarrow \Phi \wedge \texttt{not}B\langle 1\rangle \text{ [R-DWhl]}$$

Plus the variant with `skip` on the left.

Soundness is a simple induction, relying on Lemma 3:

Theorem 3. *For all C,C',Φ,Φ', if $\vdash C \sim C' : \Phi \Rightarrow \Phi'$ is derivable using the rules in Figure 5 and Section 4.2 then $\models C \sim C' : \Phi \Rightarrow \Phi'$.*

Proof. Induction on derivations.

R-Skip If $(S, S') \in [\![\Phi]\!]$ then $([\![\texttt{skip}]\!]S, [\![\texttt{skip}]\!]S') = (\lceil S\rceil, \lceil S'\rceil) \in [\![\Phi]\!]_\perp$.

R-If If $(S, S') \in [\![\Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle)]\!]$ then $(S, S') \in [\![\Phi]\!]$ and (by Lemma 3 part 4) $[\![B]\!]S = [\![B']\!]S'$. Assume the common value is *true*, then

$$([\![\texttt{if } B \texttt{ then } C \texttt{ else } D]\!]S, [\![\texttt{if } B' \texttt{ then } C' \texttt{ else } D']\!]S') = ([\![C]\!]S, [\![C']\!]S')$$

and we have $(S, S') \in [\![\Phi \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle)]\!]$ so that by assumption $([\![C]\!]S, [\![C']\!]S') \in [\![\Phi]\!]_\perp$ as required. The $false$ case is similar.

R-Seq This is just the same as the case of [D-Seq].

R-Ass Assume $(S,S') \in \llbracket \Phi[E\langle 1\rangle/X\langle 1\rangle, E\langle 2\rangle/Y\langle 2\rangle]\rrbracket$. Then

$$\begin{array}{rl} & \chi_\Phi(S[X \mapsto \llbracket E\rrbracket S], S'[Y \mapsto \llbracket E'\rrbracket S']) \\ = & \chi_\Phi(S[X \mapsto \llbracket E\langle 1\rangle\rrbracket(S,S')], S'[Y \mapsto \llbracket E'\langle 2\rangle\rrbracket(S,S')]) \\ = & \chi_{\Phi[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/Y\langle 2\rangle]}(S,S') \\ = & \mathit{true} \end{array}$$

So that $(\llbracket X\texttt{:=}E\rrbracket S, \llbracket Y\texttt{:=}E'\rrbracket S') \in \llbracket\Phi\rrbracket_\bot$ as required.

R-Whl Let $f_0 = \lambda S.\bot$, $f_{n+1} = \lambda S.\llbracket B\rrbracket S \implies f_n^*(\llbracket C\rrbracket S) \mid \lceil S\rceil$ and similarly for f'_n. We prove by induction that for all n, for all $(S,S') \in \llbracket \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle)\rrbracket$, $(f_n S, f'_n S') \in \llbracket \Phi \wedge \neg(B\langle 1\rangle \vee B'\langle 2\rangle)\rrbracket_\bot$. The case $n=0$ is immediate. Then $(f_{n+1}S, f'_{n+1}S') = (\llbracket B\rrbracket S \implies f_n^*(\llbracket C\rrbracket S) \mid \lceil S\rceil, \llbracket B'\rrbracket S' \implies {f'_n}^*(\llbracket C'\rrbracket S') \mid \lceil S'\rceil)$. Since $(S,S') \in \llbracket B\langle 1\rangle = B'\langle 2\rangle\rrbracket$, $\llbracket B\rrbracket S = \llbracket B'\rrbracket S'$. If the common value is false then $(f_{n+1}S, f'_{n+1}S') = (\lceil S\rceil, \lceil S'\rceil) \in \llbracket \Phi \wedge \neg(B\langle 1\rangle \vee B'\langle 2\rangle)\rrbracket_\bot$ as required. Otherwise the common value is true, so $(S,S') \in \llbracket \Phi \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle)\rrbracket$. Thus by assumption $(\llbracket C\rrbracket S, \llbracket C'\rrbracket S') \in \llbracket \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle)\rrbracket_\bot$. Either $(\llbracket C\rrbracket S, \llbracket C'\rrbracket S') = (\bot,\bot)$, in which case $(f_{n+1}S, f'_{n+1}S') = (\bot,\bot)$ and we're done, or $(\llbracket C\rrbracket S, \llbracket C'\rrbracket S') = (\lceil T\rceil, \lceil T'\rceil)$ with $(T,T') \in \llbracket \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle)\rrbracket$. Then by induction $(f_{n+1}S, f'_{n+1}S') = (f_n T, f'_n T') \in \llbracket \Phi \wedge \neg(B\langle 1\rangle \vee B'\langle 2\rangle)\rrbracket_\bot$ as required.

Finally, by admissibility, for all $(S,S') \in \llbracket \Phi \wedge (B\langle 1\rangle = B'\langle 2\rangle)\rrbracket$, $(\sqcup_n f_n S, \sqcup_n f'_n S') \in \llbracket \Phi \wedge \neg(B\langle 1\rangle \vee B'\langle 2\rangle)\rrbracket_\bot$ as required.

R-Sub Immediate.

R-Sym PERs are symmetric.

R-Tr PERs are transitive.

R-F $\forall (S,S') \in \emptyset.(\llbracket C\rrbracket S, \llbracket C'\rrbracket S') \in \llbracket\Phi\rrbracket_\bot$.

R-DAssL Assume $(S,S') \in \llbracket \Phi[E\langle 1\rangle/X\langle 1\rangle]\rrbracket$. Then

$$\begin{array}{rl} & \chi_\Phi(S[X \mapsto \llbracket E\rrbracket S], S') \\ = & \chi_\Phi(S[X \mapsto \llbracket E\langle 1\rangle\rrbracket(S,S')], S') \\ = & \chi_{\Phi[E\langle 1\rangle/X\langle 1\rangle]}(S,S') \\ = & \mathit{true} \end{array}$$

So $(\llbracket X\texttt{:=}E\rrbracket S, \llbracket\texttt{skip}\rrbracket S') \in \llbracket\Phi\rrbracket_\bot$ as required.

R-CBL Assume $(S,S') \in \llbracket\Phi\rrbracket$. If $\llbracket B\rrbracket S = \mathit{true}$ then $(S,S') \in \llbracket \Phi \wedge B\langle 1\rangle\rrbracket$ and

$$\begin{array}{l} (\llbracket \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C'\rrbracket S, \llbracket D\rrbracket S') \\ \quad = \ (\llbracket C\rrbracket S, \llbracket D\rrbracket S') \\ \quad \in \ \llbracket\Phi'\rrbracket_\bot \text{ by assumption} \end{array}$$

Similar reasoning applies in the false case.

R-CBInvTL If $(S,S') \in \llbracket \Phi \wedge B\langle 1\rangle\rrbracket$ then $(S,S') \in \llbracket\Phi\rrbracket$ and $\llbracket B\rrbracket S = \mathit{true}$. Hence $(\llbracket \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\rrbracket S, \llbracket C\rrbracket S') \in \llbracket\Phi'\rrbracket_\bot$ and $\llbracket \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\rrbracket S = \llbracket C_1\rrbracket S$ so that

$$(\llbracket C_1\rrbracket S, \llbracket C\rrbracket S') \in \llbracket\Phi'\rrbracket_\bot$$

as required.

R-DWhlL If $(S, S') \in [\![\Phi \wedge \mathtt{not}B\langle 1\rangle]\!]$ then $[\![B]\!]S = \mathit{false}$ so (strictly speaking by a trivial fixpoint induction)

$$\begin{array}{ll} & ([\![\mathtt{while}\ B\ \mathtt{do}\ C]\!]S, [\![\mathtt{skip}]\!]S') \\ = & (\lceil S\rceil, \lceil S'\rceil) \\ \in & [\![\Phi \wedge \mathtt{not}B\langle 1\rangle]\!]_\perp \end{array}$$

□

4.3 Examples

With these rules, one can prove the correctness of many traditional compiler optimizations, including various forms of code motion and predicated transformation. Producing proofs in RHL is fairly straightforward, so we just give a couple of small examples of the sort of thing one can prove.

Invariant hoisting:

```
while I<N do            X := Y+1;
  X := Y+1;     ==>     while I<N do
  I := I+X;               I := I+X;
```

at type $\Phi \Rightarrow \Phi$ where Φ is $I\langle 1\rangle = I\langle 2\rangle \wedge N\langle 1\rangle = N\langle 2\rangle \wedge Y\langle 1\rangle = Y\langle 2\rangle$. Note that the lifting is only valid because we do not care about the final value of X. The proof makes two uses of the dead-assignment rule, which is a common pattern for performing code-motion in RHL: one effectively adds `skip`s to both sides to make them the same 'shape', shows the equivalence using the congruence rules and then removes the `skip`s.

Proof. Let Φ be as defined above, $B = (I < N)$ and $\Phi' = \Phi \wedge (X\langle 2\rangle = Y\langle 2\rangle + 1) \wedge (B\langle 1\rangle = B\langle 2\rangle)$. Now by [R-Ass]

$$\vdash \mathtt{I\ :=\ I+X} \sim \mathtt{I\ :=\ I+X} : \Phi'[(I\langle 1\rangle{+}X\langle 1\rangle)/I\langle 1\rangle, (I\langle 2\rangle{+}X\langle 2\rangle)/X\langle 2\rangle] \Rightarrow \Phi'$$

and by [R-DAs]

$$\vdash \begin{array}{l} \mathtt{X\ :=\ Y+1} \\ \sim \mathtt{skip} \end{array} : \begin{array}{l} \Phi'[(I\langle 1\rangle + X\langle 1\rangle)/I\langle 1\rangle, (I\langle 2\rangle + X\langle 2\rangle)/X\langle 2\rangle][(Y\langle 1\rangle + 1)/X\langle 1\rangle] \\ \Rightarrow \Phi'[(I\langle 1\rangle + X\langle 1\rangle)/I\langle 1\rangle, (I\langle 2\rangle + X\langle 2\rangle)/X\langle 2\rangle] \end{array}$$

so by [R-SU1'L]

$$\begin{array}{l} \vdash (\mathtt{X\ :=\ Y+1;\ I\ :=\ I+X}) \sim (\mathtt{I\ :=\ I+X}) \\ : \Phi'[(I\langle 1\rangle + X\langle 1\rangle)/I\langle 1\rangle, (I\langle 2\rangle + X\langle 2\rangle)/X\langle 2\rangle][(Y\langle 1\rangle + 1)/X\langle 1\rangle] \Rightarrow \Phi' \end{array}$$

Expanding the substitution on the left gives

$$\begin{array}{rcl} \Phi'' & = & (I\langle 1\rangle + Y\langle 1\rangle + 1) = (I\langle 2\rangle + X\langle 2\rangle) \\ & \wedge & N\langle 1\rangle = N\langle 2\rangle \\ & \wedge & Y\langle 1\rangle = Y\langle 2\rangle \\ & \wedge & X\langle 2\rangle = Y\langle 2\rangle + 1 \\ & \wedge & ((I\langle 1\rangle + (Y\langle 1\rangle + 1)) < N\langle 1\rangle) = (I\langle 2\rangle + X\langle 2\rangle < N\langle 2\rangle) \end{array}$$

Logic and arithmetic then give $\models (\Phi \wedge (X\langle 2\rangle = Y\langle 2\rangle + 1) \wedge B\langle 1\rangle \wedge B\langle 2\rangle) \leq \Phi''$, so by [R-Sub] and [R-Whl]

$$\begin{array}{l}\vdash \texttt{while I<N do (X :=Y+1;I:=I+X)} \sim \texttt{while I<N do I := I+X} \\ \quad : \Phi' \Rightarrow \Phi \wedge (X\langle 2\rangle = Y\langle 2\rangle + 1) \wedge \texttt{not}(B\langle 1\rangle \vee B\langle 2\rangle)\end{array}$$

Now by [R-DAs]

$$\texttt{skip} \sim \texttt{X := Y+1} : \Phi'[(Y\langle 2\rangle + 1)/X\langle 2\rangle] \Rightarrow \Phi'$$

and $\models \Phi \leq \Phi'[(Y\langle 2\rangle + 1)/X\langle 2\rangle]$. So by [R-Sub] and [R-SU1']

$$\begin{array}{l}\vdash \texttt{while I<N do (X :=Y+1;I:=I+X)} \sim \texttt{X := Y+1; while I<N do I := I+X} \\ \quad : \Phi \Rightarrow \Phi \wedge (X\langle 2\rangle = Y\langle 2\rangle + 1) \wedge \texttt{not}(B\langle 1\rangle \vee B\langle 2\rangle)\end{array}$$

and clearly $\models (\Phi \wedge (X\langle 2\rangle = Y\langle 2\rangle + 1) \wedge \texttt{not}(B\langle 1\rangle \vee B\langle 2\rangle)) \leq \Phi$ so we're done by [R-Sub]. □

Sophisticated dead-code:

```
if X>3 then Y := X else Y := 7    ==> skip
```

at type $(X\langle 1\rangle = X\langle 2\rangle \wedge Y\langle 1\rangle > 2 \wedge Y\langle 2\rangle > 2) \Rightarrow (Y\langle 1\rangle > 2 \wedge Y\langle 2\rangle > 2)$. I.e. if all that matters about the value of Y in the rest of the derivation is that it is greater than 2, then the conditional has no effect.

Proof. Write Φ for $(X\langle 1\rangle = X\langle 2\rangle \wedge Y\langle 1\rangle > 2 \wedge Y\langle 2\rangle > 2)$ and Φ' for $(Y\langle 1\rangle > 2 \wedge Y\langle 2\rangle > 2)$. By [R-DAssL]

$$\vdash Y\texttt{:=}X \sim \texttt{skip} : \Phi'[X\langle 1\rangle/Y\langle 1\rangle] \Rightarrow \Phi'$$

and

$$\vdash Y\texttt{:=7} \sim \texttt{skip} : \Phi'[\texttt{7}/Y\langle 1\rangle] \Rightarrow \Phi'$$

It is then trivial to check

$$\begin{array}{l}\models \Phi \wedge (X\langle 1\rangle > 3) \leq \Phi'[X\langle 1\rangle/Y\langle 1\rangle] \quad \text{and} \\ \models \Phi \wedge \texttt{not}(X\langle 1\rangle > 3) \leq \Phi'[\texttt{7}/Y\langle 1\rangle]\end{array}$$

so that by two applications of [R-Sub] and one of [R-CBL] we are done. □

The main weakness of RHL as presented here relates to its treatment of loops. Since we insist that transformed programs have the same termination behaviour as the original, but have no non-trivial termination analysis, this is hardly suprising. I believe it is possible to add sound rules which can justify some cases of loop distribution/fusion, but more ambitious loop optimizations seem to require either a language with restricted iteration constructs or a logic which can reason about termination.

4.4 Embedding Simpler Logics in RHL

RHL is powerful but hardly suitable for direct implementation in a compiler. However, it can provide a useful framework for developing sound type and transformation systems which are more specific. One would start by identifying a restricted sublanguage of relational assertions. For example, several useful analyses can be formulated using only partial equivalence relations generated from axioms such as:

$$\vdash PER(E\langle 1\rangle = E\langle 2\rangle) \qquad \vdash PER(B\langle 1\rangle = B\langle 2\rangle)$$

$$\vdash PER(B\langle 1\rangle \wedge B\langle 2\rangle)$$

plus rules stating that PERs are closed under conjunction, disjoint union and the arrow constructor. Our earlier DDCC system is of this form, with state relations being formed as conjunctions of primitive assertions of the forms $X\langle 1\rangle = X\langle 2\rangle$ and $X\langle 1\rangle = n \wedge X\langle 2\rangle = n$. The rules of DDCC can then be presented as derived rules in RHL.

For $F, F' \in \tau$ `exp` and DDCC expression type ϕ_τ, define the RHL relation $(F \sim F' : \phi_\tau)^*$ as follows:

$$\begin{array}{rcl}
(F \sim F : \mathbb{F})^* & = & \texttt{false} \\
(F \sim F' : \{c\})^* & = & (F\langle 1\rangle = \texttt{c}) \wedge (F'\langle 2\rangle = \texttt{c}) \\
(F \sim F' : \Delta)^* & = & (F\langle 1\rangle = F'\langle 2\rangle) \\
(F \sim F' : \mathbb{T})^* & = & \texttt{true}
\end{array}$$

Then for a DDCC state type Φ, define the RHL relation Φ^* by

$$\begin{array}{rcl}
(-)^* & = & \texttt{true} \\
(\Phi, X : \phi_{\texttt{int}})^* & = & \Phi^* \wedge (X \sim X : \phi_{\texttt{int}})^*
\end{array}$$

Lemma 4. *For all F, F', S, S', Φ, ϕ:*

1. $(S, S') \in [\![(F \sim F' : \phi)^*]\!]^{RHL} \iff ([\![F]\!]S, [\![F']\!]S') \in [\![\phi]\!]^{DDCC}$.

2. $\models PER((F \sim F : \phi)^*)$ *(NB. firstly, it's the same F and, secondly, we don't actually use this in the sequel.)*

3. $[\![\Phi]\!]^{DDCC} = [\![\Phi^*]\!]^{RHL}$

4. $\models PER(\Phi^*)$ *and if* $\vdash \Phi \leq \Phi'$ *then* $\models \Phi^* \leq \Phi'^*$.

Proof. 1. Consider each case for ϕ. $\mathbb{T}$ and $\mathbb{F}$ are obvious. For $\{c\}$ we get (missing out some steps)

$$\begin{array}{rl}
& [\![(F \sim F' : \{c\})^*]\!]^{RHL} \\
= & \{(S, S') \mid [\![F\langle 1\rangle = \texttt{c}]\!](S, S') = \mathit{true} \,\&\, [\![F'\langle 2\rangle = \texttt{c}]\!](S, S') = \mathit{true}\} \\
= & \{(S, S') \mid [\![F]\!]S = c \,\&\, [\![F']\!]S' = c\} \\
= & \{(S, S') \mid ([\![F]\!]S, [\![F']\!]S') \in [\![\{c\}]\!]\}
\end{array}$$

The case $\phi = \Delta$ is similar.

2. For symmetry:

$$\begin{array}{ll} & (S,S') \in [\![(F \sim F : \phi)^*]\!] \\ \iff & ([\![F]\!]S, [\![F]\!]S') \in [\![\phi]\!] \\ \iff & ([\![F]\!]S', [\![F]\!]S) \in [\![\phi]\!] \text{ as } [\![\phi]\!] \text{ is a PER} \\ \iff & (S',S) \in [\![(F \sim F : \phi)^*]\!] \end{array}$$

The reasoning is similar for transitivity.

3. This follows immediately from the first part and induction on the length of Φ:

$$\begin{array}{ll} & (S,S') \in [\![\Phi, X : \phi]\!]^{DDCC} \\ \iff & (S,S') \in [\![\Phi]\!]^{DDCC} \;\&\; (S(X), S'(X)) \in [\![\phi]\!]^{DDCC} \\ \iff & (S,S') \in [\![\Phi^*]\!]^{RHL} \;\&\; ([\![X]\!]S, [\![X]\!]S') \in [\![\phi]\!]^{DDCC} \\ \iff & (S,S') \in [\![\Phi^*]\!]^{RHL} \;\&\; (S,S') \in [\![(X \sim X : \phi)^*]\!]^{RHL} \\ \iff & (S,S') \in [\![\Phi^* \wedge (X \sim X : \phi)^*]\!]^{RHL} \\ \iff & (S,S') \in [\![(\Phi, X : \phi)^*]\!]^{RHL} \end{array}$$

4. Immediate from previous bit, the fact that $[\![\Phi]\!]^{DDCC}$ is a PER, and the soundness of entailment in DDCC (Lemma 1).

□

Theorem 4. *For all $F, F', \Phi, \Phi', C, C', \phi$:*

1. If $\vdash F \sim F' : \Phi \Rightarrow \phi$ then $\models \Phi^* \leq (F \sim F' : \phi)^*$.

2. If $\vdash C \sim C' : \Phi \Rightarrow \Phi'$ in DDCC then $\vdash C \sim C' : \Phi^* \Rightarrow \Phi'^*$ in RHL.

Proof. 1. This follows from soundness of DDCC expression judgements:

$$\begin{array}{ll} & \vdash E \sim E' : \Phi \Rightarrow \phi \\ \implies & \forall (S,S') \in [\![\Phi]\!]^{DDCC}.([\![E]\!]S, [\![E']\!]S') \in [\![\phi]\!]^{DDCC} \quad \text{(Theorem 1)} \\ \iff & \forall (S,S') \in [\![\Phi]\!]^{DDCC}.(S,S') \in [\![(E \sim E' : \phi)^*]\!]^{RHL} \\ \iff & \forall (S,S') \in [\![\Phi^*]\!]^{RHL}.(S,S') \in [\![(E \sim E' : \phi)^*]\!]^{RHL} \\ \iff & \models \Phi^* \leq (E \sim E' : \phi)^* \end{array}$$

2. Induction on derivations

D-CT $(\Phi, X : \mathbb{F})^* = \Phi^* \wedge (X \sim X : \mathbb{F})^* = \Phi^* \wedge \texttt{false}$ So $\models (\Phi, X : \mathbb{F})^* \leq$ `false` and we're done by [R-F] and [R-Sub].

D-CSub Lemma 4 and [R-Sub].

D-CSym Lemma 4 and [R-Sym].

D-CTr Lemma 4 and [R-Tr].

D-Skip Immediate by [R-Skip]

D-Seq Immediate by [R-Seq]

D-Ass By [R-Ass],

$$\vdash X\texttt{:=}E \sim X\texttt{:=}E' : (\Phi, X : \phi')^*[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle] \Rightarrow (\Phi, X : \phi')^*$$

and examination of the definition of $(\cdot)^*$ shows

$$(\Phi, X : \phi')^*[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle] = \Phi^* \wedge (E \sim E' : \phi')^*$$

By part 1 and the hypothesis we know $\models (\Phi, X : \phi)^* \leq (E \sim E' : \phi')^*$, and clearly $\models (\Phi, X : \phi)^* \leq \Phi^*$ so

$$\models (\Phi, X : \phi)^* \leq \Phi^* \wedge (E \sim E' : \phi')^*$$

so we're done by [R-Sub].

D-Whl By part 1, $\models \Phi^* \leq (B \sim B' : \Delta_{\mathtt{bool}})^*$ which is $\models \Phi^* \leq (B\langle 1\rangle = B'\langle 2\rangle)$, so $\models \Phi^* \leq \Phi^* \wedge (B\langle 1\rangle = B'\langle 2\rangle)$. Clearly $\Phi^* \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \leq \Phi^*$ and, by hypothesis, $\vdash C \sim C' : \Phi^* \Rightarrow \Phi^*$. Thus by [R-Sub]

$$\vdash C \sim C' : \Phi^* \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \Rightarrow \Phi^* \wedge (B\langle 1\rangle = B'\langle 2\rangle)$$

so by [R-Whl]

$$\vdash \mathtt{while}\, B\, \mathtt{do}\, C \sim \mathtt{while}\, B'\, \mathtt{do}\, C' : \Phi^* \wedge (B\langle 1\rangle = B'\langle 2\rangle) \Rightarrow \Phi^* \wedge \mathtt{not}(B\langle 1\rangle \vee B'\langle 2\rangle)$$

And then using $\models \Phi^* \leq \Phi^* \wedge (B\langle 1\rangle = B'\langle 2\rangle)$ again, plus $\models \Phi^* \wedge \mathtt{not}(B\langle 1\rangle \vee B'\langle 2\rangle) \leq \Phi^*$, we can apply [R-Sub] to deduce

$$\vdash \mathtt{while}\ B\ \mathtt{do}\ C \sim \mathtt{while}\ B'\ \mathtt{do}\ C' : \Phi^* \Rightarrow \Phi^*$$

as required.

D-If By hypothesis $\vdash C_i \sim C_i' : \Phi^* \Rightarrow \Phi'^*$ for $i \in \{1, 2\}$. Since $\models \Phi^* \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \leq \Phi^*$ we can use [R-Sub] to deduce

$$\vdash C_1 \sim C_1' : \Phi^* \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \Rightarrow \Phi'^*$$

and similarly

$$\vdash C_2 \sim C_2' : \Phi^* \wedge \mathtt{not}(B\langle 1\rangle \vee B'\langle 2\rangle) \Rightarrow \Phi'^*$$

so that by [R-If]

$$\vdash \mathtt{if}\, B\, \mathtt{then}\, C_1\, \mathtt{else}\, C_2 \sim \mathtt{if}\, B'\, \mathtt{then}\, C_1'\, \mathtt{else}\, C_2' : \Phi^* \wedge (B\langle 1\rangle = B'\langle 2\rangle) \Rightarrow \Phi'^*$$

Then by assumption $\models \Phi^* \leq (B \sim B' : \Delta_{\mathtt{bool}})^*$ so that $\models \Phi^* \leq \Phi^* \wedge (B\langle 1\rangle = B'\langle 2\rangle)$ and we can apply [R-Sub] to deduce

$$\vdash \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \sim \mathtt{if}\ B'\ \mathtt{then}\ C_1'\ \mathtt{else}\ C_2' : \Phi^* \Rightarrow \Phi'^*$$

as required.

D-SU1' Just follows from [R-SU1']. Same goes for variants.

D-DAs By [R-DAssL],

$$\vdash X \mathtt{:=} E \sim \mathtt{skip} : (\Phi, X : \mathbb{T})^*[E\langle 1\rangle / X\langle 1\rangle] \Rightarrow (\Phi, X : \mathbb{T})^*$$

and by the form of DDCC state types, $(\Phi, X : \mathbb{T})^*[E\langle 1\rangle / X\langle 1\rangle] = \Phi^* \wedge \mathtt{true}$. Since $\models \Phi^* \wedge (X \sim X : \phi)^* \leq \Phi^* \wedge \mathtt{true}$ we can apply [R-Sub] to deduce

$$\vdash X \mathtt{:=} E \sim \mathtt{skip} : (\Phi, X : \phi)^* \Rightarrow (\Phi, X : \mathbb{T})^*$$

as required.

D-BrE' Follows from [R-CBL] plus two uses of [R-Sub] since $\models \Phi^* \wedge B\langle 1\rangle \leq \Phi^*$ and $\Phi^* \wedge \mathtt{not} B\langle 1\rangle \leq \Phi^*$.

D-CF

$$\begin{aligned} \models \Phi^* &\leq (F \sim F : \{c\})^* \\ &= (F\langle 1\rangle = \mathtt{c}) \wedge (F\langle 2\rangle = \mathtt{c}) \\ &\leq (F\langle 1\rangle = \mathtt{c}) \wedge (\mathtt{c} = \mathtt{c}) \\ &= (F \sim \mathtt{c} : \{c\})^* \end{aligned}$$

D-KBT By assumption $\models \Phi^* \leq (B\langle 1\rangle = \mathtt{true}) \wedge (B\langle 2\rangle = \mathtt{true})$ and $\vdash C_1 \sim C' : \Phi^* \Rightarrow \Phi'^*$. Hence $\models \Phi^* \wedge B\langle 1\rangle \leq \Phi^*$ and so by [R-Sub], $\vdash C_1 \sim C' : \Phi^* \wedge B\langle 1\rangle \Rightarrow \Phi'^*$. We also have $\vdash C_2 \sim C' : \mathtt{false} \Rightarrow \Phi'^*$ by [R-F], and since $\models \Phi^* \wedge \mathtt{not} B\langle 1\rangle \leq \mathtt{false}$ we can apply [R-Sub] to deduce $\vdash C_2 \sim C' : \Phi^* \wedge \mathtt{not} B\langle 1\rangle \Rightarrow \Phi'^*$. An application of [R-CBL] then gives

$$\vdash \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \sim C' : \Phi^* \Rightarrow \Phi'^*$$

as required.

D-LU1 Immediate from [R-LU1].

D-LU2 Immediate from [R-LU2].

D-CC By assumption

$$\vdash \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \sim \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 : \Phi^* \Rightarrow \Phi'^*$$

and

$$\vdash C_3 \sim C_3 : \Phi'^* \Rightarrow \Phi''^*$$

So by [R-CBInvTL], [R-CBInvFL] and [R-Seq]

$$\vdash C_1\,;C_3 \sim (\mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2)\,;C_3 : \Phi^* \wedge B\langle 1\rangle \Rightarrow \Phi''^*$$

and

$$\vdash C_2\,;C_3 \sim (\mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2)\,;C_3 : \Phi^* \wedge \mathtt{not} B\langle 1\rangle \Rightarrow \Phi''^*$$

so by [R-CBL]

$$\vdash \mathtt{if}\, B\, \mathtt{then}\, C_1\,;C_3\, \mathtt{else}\, C_2\,;C_3 \sim (\mathtt{if}\, B\, \mathtt{then}\, C_1\, \mathtt{else}\, C_2)\,;C_3 : \Phi^* \Rightarrow \Phi''^*$$

as required.

D-DWh By [R-DWhlL], $\vdash \mathtt{while}\, B\, \mathtt{do}\, C \sim \mathtt{skip} : \Phi^* \wedge \mathtt{not} B\langle 1\rangle \Rightarrow \Phi^* \wedge \mathtt{not} B\langle 1\rangle$. Clearly $\models \Phi^* \wedge \mathtt{not} B\langle 1\rangle \leq \Phi^*$ and since by assumption $\models \Phi^* \leq (B\langle 1\rangle = \mathtt{false}) \wedge (B\langle 2\rangle = \mathtt{false})$, $\models \Phi^* \leq \Phi^* \wedge \mathtt{not} B\langle 1\rangle$. Thus by [R-Sub]

$$\vdash \mathtt{while}\ B\ \mathtt{do}\ C \sim \mathtt{skip} : \Phi^* \Rightarrow \Phi^*$$

D-Div By assumption $\models \Phi^* \leq (B\langle 1\rangle = \mathtt{true}) \wedge (B\langle 2\rangle = \mathtt{true})$ and $\vdash C \sim C : \Phi^* \Rightarrow \Phi^*$. So by [R-Sub] $\vdash C \sim C : \Phi^* \wedge (B\langle 1\rangle \wedge B\langle 2\rangle) \Rightarrow \Phi^* \wedge (B\langle 1\rangle = B\langle 2\rangle)$ and thus by [R-Whl]

$$\vdash \mathtt{while}\, B\, \mathtt{do}\, C \sim \mathtt{while}\, B\, \mathtt{do}\, C : \Phi^* \wedge (B\langle 1\rangle = B\langle 2\rangle) \Rightarrow \Phi^* \wedge \mathtt{not}(B\langle 1\rangle \vee B\langle 2\rangle)$$

Then $\models \Phi^* \leq \Phi^* \wedge (B\langle 1\rangle = B\langle 2\rangle)$ and $\models \Phi^* \wedge \texttt{not}(B\langle 1\rangle \vee B\langle 2\rangle) \leq \texttt{false} \leq \Phi'^*$ so by [R-Sub]

$$\vdash \texttt{while}\ B\ \texttt{do}\ C \sim \texttt{while}\ B\ \texttt{do}\ C : \Phi^* \Rightarrow \Phi'^*$$

as required.

□

A natural question is whether the usual Hoare logic can be embedded in RHL. One's first thought might be that a partial correctness judgement $\vdash \{P\}C\{Q\}$ would be equivalent to the 'squared' RHL judgement

$$\vdash C \sim C : P\langle 1\rangle \wedge P\langle 2\rangle \Rightarrow Q\langle 1\rangle \wedge Q\langle 2\rangle$$

but this is not the case because C's termination behaviour might differ on two states satisfying P. Nor can one simply intersect the pre- and post-relations with the identity relation on states, since we do not have syntax for that 'global' identity relation. If we fix C, however, we can conjoin the pre- and post-relations with $X\langle 1\rangle = X\langle 2\rangle$ for every variable X occurring in C and thus effectively recover Hoare logic.[3] Going the other way, one can soundly extend RHL with the squared versions of valid *total* correctness judgements $\vdash [P]C[Q]$.

As a simple, concrete example of the embedding approach, Figure 6 presents (a very naive version of) a type system AERC for available expression analysis and removal of redundant evaluation. State types Θ are finite sets $\{X_i = E_i \mid 1 \leq i \leq n\}$ of equalities between variables and expressions (in which the same variable may occur multiple times on the left) and we write $\Theta \leq \Theta'$ for $\Theta \supseteq \Theta'$. The macros *kill* and *gen* are defined by

$$\begin{array}{lcl} \mathit{kill}(\Theta, X) & = & \{(X_i = E_i) \in \Theta \mid X_i \neq X \wedge X \notin E_i\} \\ \mathit{gen}(X, E) & = & \left\{ \begin{array}{ll} \{X = E\} & \text{if } X \notin E \\ \{\} & \text{otherwise} \end{array} \right. \end{array}$$

The translation of the AERC into RHL is indexed by a finite set V of variables. Define

$$\Theta^*_V \;=\; \bigwedge_{X \in V} (X\langle 1\rangle = X\langle 2\rangle) \wedge \bigwedge_{(X=E)\in\Theta} (X\langle 1\rangle = E\langle 1\rangle)$$

It is easy to see that for any Θ, $\models PER(\Theta^*_V)$ and that $\Theta \leq \Theta'$ implies $\Theta^*_V \leq \Theta'^*_V$. The following asserts the soundness of the translation, and hence of AERC:

Theorem 5. *For any expressions E,F and commands C,D all of whose variables occur in V,*

1. *If* $\vdash E \sim F : \Theta \Rightarrow \tau$ *then* $\models \Theta^*_V \leq (E\langle i\rangle = F\langle j\rangle)$ *for* $i, j \in \{1, 2\}$.
2. *If* $\vdash C \sim D : \Theta \Rightarrow \Theta'$ *in AERC then* $\vdash C \sim D : \Theta^*_V \Rightarrow \Theta'^*_V$ *in RHL.*

Proof. The interesting case is the AERC rule for assignment, which generates the following verification condition in RHL: if $\models \Theta^*_V \leq (E\langle 1\rangle = E'\langle 2\rangle)$ then $\models \Theta^*_V \leq (\mathit{kill}(\Theta, X) \cup \mathit{gen}(X, E) \cup \mathit{gen}(X, E'))^*_V[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$, which is straightforward to check.

[3] The civilised way to do this is to index all our judgements by finite sets of variable names.

$$\vdash X \sim X : \Theta \Rightarrow \texttt{int}\ \text{[A-V]} \qquad \vdash \texttt{n} \sim \texttt{n} : \Theta \Rightarrow \texttt{int}\ \text{[A-N]}$$

$$\vdash \texttt{b} \sim \texttt{b} : \Theta \Rightarrow \texttt{bool}\ \text{[A-B]} \qquad \vdash X \sim E : \Theta \cup \{X = E\} \Rightarrow \texttt{int}\ \text{[A-Red]}$$

$$\vdash \texttt{skip} \sim \texttt{skip} : \Theta \Rightarrow \Theta\ \text{[A-Skp]}$$

$$\frac{\vdash E \sim E' : \Theta \Rightarrow \texttt{int} \quad \vdash F \sim F' : \Theta \Rightarrow \texttt{int}}{\vdash E\ \texttt{iop}\ F \sim E'\ \texttt{iop}\ F' : \Theta \Rightarrow \texttt{int}}\ \text{[A-iop] (+ similar } \mathit{bop} \text{ and } \mathit{lop}\text{)}$$

$$\frac{\vdash C_1 \sim C_1' : \Theta \Rightarrow \Theta' \quad \vdash C_2 \sim C_2' : \Theta' \Rightarrow \Theta''}{\vdash (C_1\texttt{;}C_2) \sim (C_1'\texttt{;}C_2') : \Theta \Rightarrow \Theta''}\ \text{[A-Seq]}$$

$$\frac{\vdash B \sim B' : \Theta \Rightarrow \texttt{bool} \quad \vdash C \sim C' : \Theta \Rightarrow \Theta}{\vdash (\texttt{while}\ B\ \texttt{do}\ C) \sim (\texttt{while}\ B'\ \texttt{do}\ C') : \Theta \Rightarrow \Theta}\ \text{[A-Whl]}$$

$$\frac{\vdash E \sim E' : \Theta \Rightarrow \texttt{int}}{\vdash X\texttt{:=}E \sim X\texttt{:=}E' : \Theta \Rightarrow (\mathit{kill}(\Theta, X) \cup \mathit{gen}(X, E) \cup \mathit{gen}(X, E'))}\ \text{[A-Ass]}$$

$$\frac{\vdash B \sim B' : \Theta \Rightarrow \texttt{bool} \quad \vdash C_1 \sim C_1' : \Theta \Rightarrow \Theta' \quad \vdash C_2 \sim C_2' : \Theta \Rightarrow \Theta'}{\vdash (\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2) \sim (\texttt{if}\ B'\ \texttt{then}\ C_1'\ \texttt{else}\ C_2') : \Theta \Rightarrow \Theta'}\ \text{[A-If]}$$

$$\frac{\vdash C \sim C' : \Theta \Rightarrow \Theta'}{\vdash C' \sim C : \Theta \Rightarrow \Theta'}\ \text{[A-CSym]}$$

$$\frac{\vdash C \sim C' : \Theta_1 \Rightarrow \Theta_2 \quad \Theta_1' \leq \Theta_1 \quad \Theta_2 \leq \Theta_2'}{\vdash C \sim C' : \Theta_1' \Rightarrow \Theta_2'}\ \text{[A-CSub]}$$

$$\frac{\vdash E_\tau \sim E_\tau' : \Theta \Rightarrow \tau \quad \Theta' \leq \Theta}{\vdash E_\tau \sim E_\tau' : \Theta' \Rightarrow \tau}\ \text{[A-ESub]} \qquad \frac{\vdash F_\tau \sim F_\tau' : \Theta \Rightarrow \tau}{\vdash F_\tau' \sim F_\tau : \Theta \Rightarrow \tau}\ \text{[A-ESym]}$$

$$\frac{\vdash C \sim C' : \Theta \Rightarrow \Theta' \quad \vdash C' \sim C'' : \Theta \Rightarrow \Theta'}{\vdash C \sim C'' : \Theta \Rightarrow \Theta'}\ \text{[A-CTr]}$$

$$\frac{\vdash F_\tau \sim F_\tau' : \Theta \Rightarrow \tau \quad \vdash F_\tau' \sim F_\tau'' : \Theta \Rightarrow \tau}{\vdash F_\tau \sim F_\tau'' : \Theta \Rightarrow \tau}\ \text{[A-ETr]}$$

Figure 6: AERC: Available Expressions and Redundant Computation

A-V Clearly $\models \Theta^*_V \leq X\langle i\rangle = X\langle i\rangle$. By assumption $X \in V$ so by the definition of Θ^*_V, $\models \Theta^*_V \leq X\langle 1\rangle = X\langle 2\rangle$ and we're done.

A-N Since $\mathtt{n}\langle i\rangle = n$ this is trivial.

A-B as above

A-Red By assumption on V, $\models (\Theta \cup \{X = E\})^*_V \leq Y\langle 1\rangle = Y\langle 2\rangle$ for all $Y \in \mathit{vars}(E) \cup \{X\}$. Also $\models (\Theta \cup \{X = E\})^*_V \leq X\langle 1\rangle = E\langle 1\rangle$ by definition of $(\cdot)^*_V$. Hence by elementary properties of equality, $\models (\Theta \cup \{X = E\})^*_V \leq X\langle i\rangle = E\langle j\rangle$ for any $i, j \in \{1, 2\}$.

A-iop By induction $\models \Theta^*_V \leq (E\langle i\rangle = E'\langle j\rangle)$ and $\models \Theta^*_V \leq (F\langle i\rangle = F'\langle j\rangle)$, and since $(E\ \mathtt{iop}\ F)\langle i\rangle = E\langle i\rangle\ \mathtt{iop}\ F\langle i\rangle$ and $(E'\ \mathtt{iop}\ F')\langle j\rangle = E'\langle j\rangle\ \mathtt{iop}\ F'\langle j\rangle$ it's clear that

$$\models \Theta^*_V \leq (E\ \mathtt{iop}\ F)\langle i\rangle = (E'\ \mathtt{iop}\ F')\langle j\rangle$$

as required.

A-ESub By induction $\models \Theta^*_V \leq (E\langle i\rangle = E'\langle j\rangle)$ and $\models \Theta'^*_V \leq \Theta^*_V$ so $\models \Theta'^*_V \leq (E\langle i\rangle = E'\langle j\rangle)$.

A-ETr For any i, j, $\models \Theta^*_V \leq (F\langle i\rangle = F'\langle 1\rangle)$ and $\models \Theta^*_V \leq (F'\langle 1\rangle = F''\langle j\rangle)$ so by transitivity of equality $\models \Theta^*_V \leq (F\langle i\rangle = F''\langle j\rangle)$.

A-ESym By induction $\models \Theta^*_V \leq F\langle i\rangle = F'\langle j\rangle$ for all i, j, so $\models \Theta^*_V \leq F'\langle i\rangle = F\langle j\rangle$ for all i, j.

A-Skip By [R-Skip], $\vdash \mathtt{skip} \sim \mathtt{skip} : \Theta^*_V \Rightarrow \Theta^*_V$ in RHL.

A-Seq Immediate by [R-Seq].

A-Whl By induction $\models \Theta^*_V \leq (B\langle 1\rangle = B'\langle 2\rangle)$ and $\vdash C \sim C' : \Theta^*_V \Rightarrow \Theta^*_V$. So $\models \Theta^*_V \leq \Theta^*_V \wedge (B\langle 1\rangle = B'\langle 2\rangle)$ and $\models \Theta^*_V \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \leq \Theta^*_V$ and by [R-Sub] and [R-Whl]

$$\vdash \mathtt{while}\ B\ \mathtt{do}\ C \sim \mathtt{while}\ B'\ \mathtt{do}\ C' : \Theta^*_V \wedge (B\langle 1\rangle = B'\langle 2\rangle) \Rightarrow \Theta^*_V \wedge \mathtt{not}(B\langle 1\rangle \vee B'\langle 2\rangle)$$

Then as $\models \Theta^*_V \leq \Theta^*_V \wedge (B\langle 1\rangle = B'\langle 2\rangle)$ and $\models \Theta^*_V \wedge \mathtt{not}(B\langle 1\rangle \vee B'\langle 2\rangle) \leq \Theta^*_V$ we can apply [R-Sub] again to deduce

$$\vdash \mathtt{while}\ B\ \mathtt{do}\ C \sim \mathtt{while}\ B'\ \mathtt{do}\ C' : \Theta^*_V \Rightarrow \Theta^*_V$$

as required.

A-Ass Let $\Theta' = (\mathit{kill}(\Theta, X) \cup \mathit{gen}(X, E) \cup \mathit{gen}(X, E'))$, then by[R-Ass]

$$\vdash X\mathtt{:=}E \sim X\mathtt{:=}E' : \Theta'^*_V[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle] \Rightarrow \Theta'^*_V$$

So by [R-Sub], we're done if we can show

$$\models \Theta^*_V \leq \Theta'^*_V[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$$

given that we know by induction that $\models \Theta^*_V \leq (E\langle i\rangle = E'\langle j\rangle)$ for each i, j. Expanding out the definition of $\Theta'^*_V[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$ we find it comprises the following four conjuncts:

1. $\bigwedge_{Y\in V}(Y\langle 1\rangle = Y\langle 2\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$.
2. $\bigwedge_{(Y=F)\in \mathit{kill}(\Theta,X)}(Y\langle 1\rangle = F\langle 1\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$.
3. $(X\langle 1\rangle = E\langle 1\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$ provided $X \notin E$.
4. $(X\langle 1\rangle = E'\langle 1\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$ provided $X \notin E'$.

so we have to show that Θ^*_V entails each of the conjuncts.

1. The first conjunct is logically equivalent to

$$\left(\bigwedge_{Y\in V\setminus\{X\}}(Y\langle 1\rangle = Y\langle 2\rangle)\ \wedge\ (X\langle 1\rangle = X\langle 2\rangle)\right)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]$$

   which is

$$\bigwedge_{Y\in V\setminus\{X\}}(Y\langle 1\rangle = Y\langle 2\rangle)\ \wedge\ (E\langle 1\rangle = E'\langle 2\rangle)$$

   It is clear that Θ^*_V entails the first part of this. That it also entails the second part is an instance of our inductive hypothesis.

2. $(Y = F) \in \mathit{kill}(\Theta, X)$ implies $Y \neq X$ and $X \notin F$. Hence

$$(Y\langle 1\rangle = F\langle 1\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]\ =\ (Y\langle 1\rangle = F\langle 1\rangle)$$

   and so, as $(Y = F) \in \Theta$, which implies $\models \Theta^*_V \leq (Y\langle 1\rangle = F\langle 1\rangle)$, we are done.

3. If $X \notin E$ then

$$(X\langle 1\rangle = E\langle 1\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]\ =\ (E\langle 1\rangle = E\langle 1\rangle)$$

   which is trivially entailed by Θ^*_V.

4. If $X \notin E'$ then

$$(X\langle 1\rangle = E'\langle 1\rangle)[E\langle 1\rangle/X\langle 1\rangle, E'\langle 2\rangle/X\langle 2\rangle]\ =\ (E\langle 1\rangle = E'\langle 1\rangle)$$

   and that Θ^*_V entails the RHS of the above was an induction hypothesis.

A-If By induction $\vdash C_1 \sim C'_1 : \Theta^*_V \Rightarrow \Theta'^*_V$ so by [R-Sub] $\vdash C_1 \sim C'_1 : \Theta^*_V \wedge (B\langle 1\rangle \wedge B'\langle 2\rangle) \Rightarrow \Theta'^*_V$, and similarly for C_2 and C'_2. Thus by [R-If]

$$\vdash \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \sim \texttt{if}\ B'\ \texttt{then}\ C'_1\ \texttt{else}\ C'_2 : \Theta^*_V \wedge (B\langle 1\rangle = B'\langle 2\rangle) \Rightarrow \Theta'^*_V$$

Then we also have by assumption that $\models \Theta^*_V \leq (B\langle 1\rangle = B'\langle 2\rangle)$ so we can apply [R-Sub] again to get

$$\vdash \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \sim \texttt{if}\ B'\ \texttt{then}\ C'_1\ \texttt{else}\ C'_2 : \Theta^*_V \Rightarrow \Theta'^*_V$$

as required.

A-CSub Immediate by [R-Sub].

A-CSym Immediate from [R-Sym] and the fact that $\models PER(\Theta^*_V \Rightarrow \Theta'^*_V)$.

A-CTr Immediate from [R-Tr] as above.

□

5 Related Work

As we said in the introduction, there has been a good deal of work on proving the correctness of optimizing transformations for functional languages, especially from the group at Northeastern [30, 28, 32, 31] but also by Amtoft on strictness analysis [5], Damiani and Giannini on dead-variables [10, 11], Kobayashi on dead-variables [17] and Benton and Kennedy on effects [8]. Damiani and Giannini explicitly use PERs in giving the semantics of their analysis system but give a more algorithmic account its use in transformation. Benton and Kennedy present optimizing transformations as equations in context, but derive those (rather clumsily) from a predicate-based semantics for the analysis.

Recently Lacey et al. [19] described how some of the classical [16, 12, 4] transformations considered here (dead code elimination, constant folding and a simple code-motion transformation) can be formulated as conditional rewrite rules on control flow graphs. The rewrites are predicated on temporal logic formulae expressing (intensionally) the contexts in which the rewrites may be applied. The authors then use a small-step operational semantics to verify that under these conditions, their transformations preserve the observable behaviour of programs. Lacey et al. express strongly the view that more traditional semantic techniques, in particular denotational ones, are either unable to express the properties which justify optimizing transformations or can only do so at the cost of complex proofs and 'mathematically sophisticated' techniques. I believe the present paper provides some evidence to the contrary.

Lerner et al. [20] have built an implementation of a domain specific language for specifying and justifying rewrites on a simple imperative language which interfaces to a theorem prover for checking the supplied justification. This system also uses a (rather restricted) language of temporal logic formulae for specifying optimizations over flow graphs.

Kozen and Patron [18] describe an algebraic approach to proving some traditional optimizations correct. There is no mention of relations in their work, and they abstract rather severely from the actual language (there are no assignments, just unspecified atomic programs including one which makes a variable 'undefined'), but the connections between their work and ours seem worth further study.

The work that is most closely related to that presented here has been done in the contexts of *credible compilation* [24, 25] and *translation validation* [21, 36]. These both take the view that formal verification of complete optimizing compilers is impractical, but that one might realistically produce a correctness proof relating the input and output of particular compilations. Translation validation tries to do this without modifying the compiler, using an independent tool that tries to infer that the output is a correct translation of the input. Credible compilation envisages an instrumented compiler producing a putative proof that the transformations it performed in each particular case were safe; these proofs can then be examined by a comparatively simple proof-checker. The basic technical ideas used in credible and validated compilation are very close indeed to the ones presented here (developed quite independently). The main difference is that we use the language of types, denotational semantics and PERs instead of that of control-flow graphs, operational semantics and simulation relations. Inspired by Rinard's work, Yang [35] has recently used a version of relational Hoare logic in reasoning about the correctness of the

Schorr-Waite graph marking algorithm.

The idea of directly axiomatising a logic of PERs [3] and more general relations was inspired by the work of Abadi et al on a formal logic for parametric polymorphism [2].

We have already mentioned some of the large amount of recent work using PERs (and domain-theoretic projections) to give semantics to analyses for non-interference, slicing, secure information flow, binding time analysis. An elegant general calculus, DCC, for such dependency-based analyses has been defined by Abadi et al. [1]. DCC seems comparable to a higher-order version of our DDCC, though it is not explicitly presented as an equational calculus and is more directly in the style of type systems for secure information flow.

The work of Hughes on type specialization [14] seems to have interesting connections with (a higher-order version of) the work presented here. Hughes has formulated a type-based analysis which essentially uses a form of singleton type, and proved the correctness of an associated transformation system which changes types. Singleton types and their PER semantics have also been studied in some depth by Aspinall [6].

6 Conclusions and Further Work

We have shown how some very elementary techniques can be used to prove the combined correctness of analyses and transformations for simple imperative programs. From a purely semantic perspective, there is nothing very surprising here. But that is as it should be: we have finally shown that something that appears simple actually *is* simple.

One obvious shortcoming of the present work is that it says nothing about concrete inference or transformation algorithms. Although there are benefits in factoring a correctness proof into the soundness of a declarative set of inference rules and the correctness of an inference algorithm, one does ultimately have to provide both parts. Although there seems no reason why the approach taken here should not carry over to the control-flow graphs more commonly used in optimizing compilers for imperative languages, proving directly that analyses as they are actually implemented in real compilers imply our extensional properties seems likely to be somewhat messy. Combining our extensional relational approach to correctness with a more algorithmic, but still declarative, framework for specifying transformations (such as that of Lacey et al.) seems a more reasonable next step.

Relational Hoare logic is a promising formalism which certainly merits further study. A limitation of the system presented here is that it cannot justify any transformations which remove loops, except in the special case that they can be completely unrolled at compile-time. This naturally suggests investigating a total-correctness variant of the logic, but one might also consider a version which allows termination-improving transformations. A further possibility is to axiomatise the version of relational lifting which maps Φ to $\{(d,d') \in \mathbb{S}_\perp \times \mathbb{S}_\perp \mid (d = \lceil s \rceil \wedge d' = \lceil s' \rceil) \implies (s,s') \in \Phi\}$. This is inappropriate for optimizations, but seems useful in reasoning about termination-insensitive information flow [7].

There are many natural ways to develop the ideas here, both in terms of the language features and analyses addressed (higher-types, higher-typed store, dynamic allocation) and in terms of the features of the logics (e.g. quantification, conjunctive and disjunctive types). Doing some of these would require working with relations on recursively-defined domains, for which we expect to use the techniques described by Pitts [23]. If the technique extends to imperative programs with higher-typed store, a promising idea is to look at optimizations on low level code that are justified by relational invariants passed down from a high-level compiler.

Acknowledgements. Thanks to Tony Hoare, Andrew Kennedy, David Naumann, Matthew Parkinson, Claudio Russo and the referees for useful discussions, feedback and pointers to related work.

References

[1] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Conference Record of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 147–160. ACM Press, January 1999.

[2] M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 121, 1993.

[3] M. Abadi and G. D. Plotkin. A PER model of polymorphism and recursive types. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 355–365. IEEE Computer Society Press, June 1990.

[4] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison Wesley, 1986.

[5] T. Amtoft. Minimal thunkification. In P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, editors, *Proceedings of the 3rd International Workshop on Static Analysis, Padova, Italy*, volume 724 of *Lecture Notes in Computer Science*, pages 218–229. Springer-Verlag, September 1993.

[6] D. Aspinall. Subtyping with singleton types. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th International Workshop (CSL'94)*, number 933 in Lecture Notes in Computer Science. Springer-Verlag, 1995.

[7] A. Banerjee and D. Naumann. Secure information flow and pointer confinement in a Java-like language. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 253–267. IEEE Computer Society Press, June 2002.

[8] N. Benton and A. Kennedy. Monads, effects and transformations. In *Third International Workshop on Higher Order Operational Techniques in Semantics (HOOTS), Paris*, volume 26 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 1999.

[9] P. N. Benton. *Strictness Analysis of Lazy Functional Programs*. PhD thesis, Computer Laboratory, University of Cambridge, December 1992.

[10] F. Damiani. Useless-code Detection and Elimination for PCF with Algebraic Datatypes. In *4th International Conference on Typed Lambda Calculi and Applications (TLCA)*, volume 1581 of *Lecture Notes in Computer Science*, pages 83–97. Springer-Verlag, 1999.

[11] F. Damiani and P. Giannini. Automatic useless-code detection and elimination for hot functional programs. *Journal of Functional Programming*, pages 509–559, 2000.

[12] M S. Hecht. *Flow Analysis of Computer Programs*. Elsevier North-Holland, 1977.

[13] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–585, October 1969.

[14] J. Hughes. Type specialization for the lambda calculus. In *Proceedings of the Dagstuhl Seminar on Partial Evaluation*, 1996.

[15] S. Hunt and D. Sands. Binding time analysis: A new PERspective. In *Proceedings ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, June 1991.

[16] G. A. Kildall. A unified approach to global program optimization. In *Proceedings of the 1st ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 194–206. ACM Press, 1973.

[17] N. Kobayashi. Type-based useless variable elimination. In *Proceeedings of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, 2000.

[18] D. Kozen and M. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *Proceedings of the 1st International Conference in Computational Logic*, volume 1861 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2000.

[19] D. Lacey, N. D. Jones, E. Van Wyk, and C. C. Frederiksen. Proving correctness of compiler optimizations by temporal logic. In *Proceedings of the 29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages, Portland*, January 2002.

[20] S. Lerner, T. Millstein, and C. Chambers. Automatically proving the correctness of compiler optimizations. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[21] G. C. Necula. Translation validation for an optimizing compiler. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 83–95, 2000.

[22] F. Nielson. Program transformations in a denotational setting. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):359–379, July 1985.

[23] A.M. Pitts. Relational properties of domains. *Information and Computation*, 127, 1996.

[24] M. Rinard. Credible compilation. Technical Report MIT-LCS-TR-776, Massachusets Institute of Technology, March 1999.

[25] M. Rinard and D. Marinov. Credible compilation with pointers. In *Proceedings of the FLoC Workshop on Run-Time Result Verification*, July 1999.

[26] A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, March 2001.

[27] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of the 25th ACM Symposium on Principles of Porgramming Languages (POPL)*, January 1998.

[28] P. A. Steckler and M. Wand. Lightweight closure conversion. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, pages 48–86, January 1997. Original version appeared in Proceedings 21st ACM Symposium on Principles of Programming Languages, 1994.

[29] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:167–187, December 1996.

[30] M. Wand. Specifying the correctness of binding-time analysis. In *Conference Record of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 137–143. ACM, January 1993.

[31] M. Wand and W. D. Clinger. Set constraints for destructive array update optimization. *Journal of Functional Programming*, 11(3):319–346, May 2001. Preliminary version appeared in International Conference on Computer Languages, 1998.

[32] M. Wand and I. Siveroni. Constraint systems for useless variable elimination. In *Proceedings 26th ACM Symposium on Principles of Programming Languages*, pages 291–302, 1999.

[33] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.

[34] G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press, 1993.

[35] H. Yang. Verification of the Schorr-Waite graph marking algorithm by refinement. Slides from talk at Dagstuhl Seminar 03101, March 2003.

[36] L. Zuck, A. Pnueli, Y. Fang, and B. Goldberg. VOC: A methodology for the translation validation for optimizing compilers. *Journal of Universal Computer Science*, 9(3):223–247, 2003.