3.5 Type conversion: parallel reduction vs. common subexpression reduction

Belo et al. [8] used parallel reduction (defined in Figure 3.8) as the conversion relation between types, while here I've used a structural conversion relation with common subexpression reduction (CSR). In this section, I explain the need for a conversion relation, why I've made the change, and how the two approaches differ.

First, why do we need a type conversion relation at all? Suppose we are trying to prove preservation, in order to find syntactic type soundness. Consider the term $v_1\ e_2$, where v_1 has the type $x{:}T_1 \to T_2$ and e_2 has the type T_1. According to T_App, the type of $v_1\ e_2$ is $T_2[e_2/x]$. What happens when $e_2 \longrightarrow e_2'$? The syntactic type system gives us $v_1\ e_2 : T_2[e_2'/x]$. To finish such a preservation proof, we must know how $T_2[e_2/x]$ and $T_2[e_2'/x]$ relate. Intuitively, they ought to be inhabited by the same values: since our evaluation semantics is deterministic, any value that satisfies the checks in $T_2[e_2/x]$ should also satisfy the checks in $T_2[e_2'/x]$, since the latter type is just a few extra steps along. We must modify the definition of our syntactic type system to make it respect this equivalence in a formal way.

In λ_{H} in Chapter 2, we observe that while $T_2[e_2/x]$ may not reduce to $T_2[e_2'/x]$—types don't reduce at all, in fact—we can relate them as subtypes of each other. For this reason (among others), the λ_{H} system introduces a subtyping relation. But it turns out that subtyping in that language introduces a vicious cycle (see Section 5.2.2), forcing us to adopt a semantic approach to types soundness. I end up showing that that $T_2[e_2/x]$ *parallel reduces* to $T_2[e_2'/x]$. That is, we can take some number of reduction steps in parallel (one for each free occurrence of x) from $T_2[e_2/x]$ to $T_2[e_2'/x]$. I then show that the denotations of such types are equal (Lemma 2.3.17), and then prove *semantic* type soundness for λ_{H} with respect to those denotations. Note that Lemma 2.3.17 depends on a long Coq development showing *coermination*: if $e_1 \Rrightarrow e_2$, then $e_1 \longrightarrow^* v_1$ iff $e_2 \longrightarrow^* v_2$ such that $v_1 \Rrightarrow v_2$ (Lemma `A20` in `thy.v`)—more on this later.

The situation for λ_{H} is somewhat unsatisfying. We set out to prove syntactic type soundness and ended up proving semantic type soundness along the way. While not a serious burden for a language as small as λ_{H}, having to use semantic techniques throughout makes adding some features—polymorphism, state and other effects, concurrency—difficult. For example, a semantic proof of type soundness for F_{H} is very close to a proof of parametricity—must we prove parametricity while proving type soundness?

In originally doing the work in this chapter (Belo et al. [8]), we observed that we could get rid of subtyping and explicitly use the symmetric, transitive closure of parallel reduction as the conversion relation. (Parallel reduction is reflexive by definition.) Between that (and a few other changes), we found subtyping no longer necessary. We still, however, needed cotermination for parallel reduction. We also explicitly depended on *substitutivity*: if $e_1 \Rrightarrow e_2$ and $e_1' \Rrightarrow e_2'$ then $e_1[e_1'/x] \Rrightarrow e_2[e_2'/x]$. It turns

Parallel term reduction $\boxed{e_1 \Rrightarrow e_2}$

$$\frac{v_i \Rrightarrow v_i'}{\mathrm{op}\,(v_1,\,\ldots,\,v_n) \Rrightarrow [\![\mathrm{op}]\!]\,(v_1',\,\ldots,\,v_n')}\ \textsc{EP\_ROp} \qquad \frac{e_{12} \Rrightarrow e_{12}' \quad v_2 \Rrightarrow v_2'}{(\lambda x{:}T.\ e_{12})\ v_2 \Rrightarrow e_{12}'[v_2'/x]}\ \textsc{EP\_RBeta}$$

$$\frac{e \Rrightarrow e' \quad T_2 \Rrightarrow T_2'}{(\Lambda\alpha.\ e)\ T_2 \Rrightarrow e'[T_2'/\alpha]}\ \textsc{EP\_RTBeta} \qquad \frac{v \Rrightarrow v'}{\langle T \Rightarrow T\rangle^l\, v \Rrightarrow v'}\ \textsc{EP\_RRefl}$$

$$\frac{T_2 \neq \{x{:}T_1 \mid e\} \quad T_2 \neq \{y{:}\{x{:}T_1 \mid e\} \mid e_2\} \quad T_1 \Rrightarrow T_1' \quad T_2 \Rrightarrow T_2' \quad v \Rrightarrow v'}{\langle \{x{:}T_1 \mid e\} \Rightarrow T_2\rangle^l\, v \Rrightarrow \langle T_1' \Rightarrow T_2'\rangle^l\, v'}\ \textsc{EP\_RForget}$$

$$\frac{T_1 \neq T_2 \quad T_1 \neq \{x{:}T \mid e\} \quad T_1 \Rrightarrow T_1' \quad T_2 \Rrightarrow T_2' \quad e \Rrightarrow e' \quad v \Rrightarrow v'}{\langle T_1 \Rightarrow \{x{:}T_2 \mid e\}\rangle^l\, v \Rrightarrow \langle T_2' \Rightarrow \{x{:}T_2' \mid e'\}\rangle^l\, (\langle T_1' \Rightarrow T_2'\rangle^l\, v')}\ \textsc{EP\_RPreCheck}$$

$$\frac{T \Rrightarrow T' \qquad e \Rrightarrow e' \qquad v \Rrightarrow v'}{\langle T \Rightarrow \{x{:}T \mid e\}\rangle^l\, v \Rrightarrow \langle \{x{:}T' \mid e'\}, e'[v'/x], v'\rangle^l}\ \textsc{EP\_RCheck}$$

$$\frac{v \Rrightarrow v'}{\langle \{x{:}T \mid e_1\}, \mathsf{true}, v\rangle^l \Rrightarrow v'}\ \textsc{EP\_ROK} \qquad \frac{}{\langle \{x{:}T \mid e_1\}, \mathsf{false}, v\rangle^l \Rrightarrow \Uparrow l}\ \textsc{EP\_RFail}$$

$$\frac{\begin{array}{c} x{:}T_{11} \to T_{12} \neq x{:}T_{21} \to T_{22} \\ T_{11} \Rrightarrow T_{11}' \qquad T_{12} \Rrightarrow T_{12}' \qquad T_{21} \Rrightarrow T_{21}' \qquad T_{22} \Rrightarrow T_{22}' \qquad v \Rrightarrow v' \end{array}}{\begin{array}{l} \langle x{:}T_{11} \to T_{12} \Rightarrow x{:}T_{21} \to T_{22}\rangle^l\, v \Rrightarrow \\ \qquad \lambda x{:}T_{21}'.\ (\langle T_{12}'[\langle T_{21}' \Rightarrow T_{11}'\rangle^l\, x/x] \Rightarrow T_{22}'\rangle^l\, (v'\, (\langle T_{21}' \Rightarrow T_{11}'\rangle^l\, x))) \end{array}}\ \textsc{EP\_RFun}$$

$$\frac{\forall\alpha.T_1 \neq \forall\alpha.T_2 \qquad T_1 \Rrightarrow T_1' \qquad T_2 \Rrightarrow T_2' \qquad v \Rrightarrow v'}{\langle \forall\alpha.T_1 \Rightarrow \forall\alpha.T_2\rangle^l\, v \Rrightarrow \Lambda\alpha.\ (\langle T_1' \Rightarrow T_2'\rangle^l\, (v'\,\alpha))}\ \textsc{EP\_RForall}$$

$$\frac{}{e \Rrightarrow e}\ \textsc{EP\_Refl} \qquad \frac{T_1 \Rrightarrow T_1' \quad e_{12} \Rrightarrow e_{12}'}{\lambda x{:}T_1.\ e_{12} \Rrightarrow \lambda x{:}T_1'.\ e_{12}'}\ \textsc{EP\_Abs} \qquad \frac{e_1 \Rrightarrow e_1' \quad e_2 \Rrightarrow e_2'}{e_1\ e_2 \Rrightarrow e_1'\ e_2'}\ \textsc{EP\_App}$$

$$\frac{e \Rrightarrow e'}{\Lambda\alpha.\ e \Rrightarrow \Lambda\alpha.\ e'}\ \textsc{EP\_TAbs} \qquad \frac{e_1 \Rrightarrow e_1' \quad T_2 \Rrightarrow T_2'}{e_1\ T_2 \Rrightarrow e_1'\ T_2'}\ \textsc{EP\_TApp}$$

$$\frac{e_i \Rrightarrow e_i'}{\mathrm{op}\,(e_1,\,\ldots,\,e_n) \Rrightarrow \mathrm{op}\,(e_1',\,\ldots,\,e_n')}\ \textsc{EP\_Op} \qquad \frac{T_1 \Rrightarrow T_1' \quad T_2 \Rrightarrow T_2'}{\langle T_1 \Rightarrow T_2\rangle^l \Rrightarrow \langle T_1' \Rightarrow T_2'\rangle^l}\ \textsc{EP\_Cast}$$

$$\frac{T \Rrightarrow T' \quad e \Rrightarrow e'}{\langle T, e, k\rangle^l \Rrightarrow \langle T', e', k\rangle^l}\ \textsc{EP\_Check} \qquad \frac{}{E\,[\Uparrow l] \Rrightarrow \Uparrow l}\ \textsc{EP\_Blame}$$

Parallel type reduction $\boxed{T_1 \Rrightarrow T_2}$

$$\frac{}{T \Rrightarrow T}\ \textsc{EP\_TRefl} \qquad \frac{\sigma_1 \longrightarrow^* \sigma_2 \quad T_1 \Rrightarrow T_2}{\{x{:}T_1 \mid \sigma_1(e)\} \Rrightarrow \{x{:}T_2 \mid \sigma_2(e)\}}\ \textsc{EP\_TRefine}$$

$$\frac{T_1 \Rrightarrow T_1' \quad T_2 \Rrightarrow T_2'}{x{:}T_1 \to T_2 \Rrightarrow x{:}T_1' \to T_2'}\ \textsc{EP\_TFun} \qquad \frac{T \Rrightarrow T'}{\forall\alpha.T \Rrightarrow \forall\alpha.T'}\ \textsc{EP\_TForall}$$

Figure 3.8: Parallel reduction

Conversion $\boxed{\sigma_1 \longrightarrow^* \sigma_2}$ $\qquad$ $\boxed{T_1 \equiv T_2}$

$$\sigma_1 \longrightarrow^* \sigma_2 \iff \begin{array}{l} \mathrm{dom}(\sigma_1) = \mathrm{dom}(\sigma_2) \wedge \\ \forall x \in \mathrm{dom}(\sigma_1).\ \sigma_1(x) \longrightarrow^* \sigma_2(x) \wedge \\ \forall \alpha \in \mathrm{dom}(\sigma_1).\ \sigma_1(\alpha) = \sigma_2(\alpha) \end{array}$$

$$\frac{}{\alpha \equiv \alpha}\ \text{C\_Var} \qquad \frac{}{B \equiv B}\ \text{C\_Base} \qquad \frac{\sigma_1 \longrightarrow^* \sigma_2 \quad T_1 \equiv T_2}{\{x{:}T_1 \mid \sigma_1(e)\} \equiv \{x{:}T_2 \mid \sigma_2(e)\}}\ \text{C\_Refine}$$

$$\frac{T_1 \equiv T_1' \quad T_2 \equiv T_2'}{x{:}T_1 \to T_2 \equiv x{:}T_1' \to T_2'}\ \text{C\_Fun} \qquad \frac{T \equiv T'}{\forall \alpha.\, T \equiv \forall \alpha.\, T'}\ \text{C\_Forall}$$

$$\frac{T_2 \equiv T_1}{T_1 \equiv T_2}\ \text{C\_Sym} \qquad \frac{T_1 \equiv T_2 \quad T_2 \equiv T_3}{T_1 \equiv T_3}\ \text{C\_Trans}$$

Figure 3.9: Type conversion via common subexpression reduction

out that the proof of cotermination for λ_{H} also needs substitutivity, Lemma `A3` in `thy.v`, but we needed it for the substitution in types discussed above. Unfortunately, parallel reduction in F_{H} is *not* substitutive [60]. There are two counterexamples, both in Figure 3.10.

Why doesn't substitutivity hold in F_{H}, when it did (so easily) in λ_{H}? There are two reasons. First, the cast semantics of F_{H} is much more complicated than that of λ_{H} (six rules, as opposed to two). The F_{H} rules depend on upon certain (syntactic) equalities between types—both counterexamples in Figure 3.10 take advantage these equalities to break substitutivity. Second, λ_{H} treats $\langle x{:}T_{11} \to T_{12} \Rightarrow x{:}T_{21} \to T_{22} \rangle^l\, v$ as a value, while it is a redex in F_{H}. This syntactic coincidence makes an exact cotermination lemma possible. But as the second counterexample shows, in F_{H} it's possible to have a substitution introduce a function proxy in e_2 but not e_1. While I conjecture that e_1 and e_2 are *contextually* equivalent, they won't yield values that parallel reduce to each other. The rules that are the source of the problem for substitutivity of parallel reduction are the EP_R... rules, where a reduction in the outer term happens at the same time as parallel reductions deep inside the term. Note that both counterexamples make use of such rules.

The semantics of F_{H} as published in ESOP 2011 are wrong. To fix them, I introduced a simpler conversion relation, defined in Figure 3.9. Instead of allowing full parallel reduction, I restrict convertible types to the CSR $\sigma_1 \longrightarrow^* \sigma_2$, i.e., substitutions over the same set of term and type variables where (a) every term binding in σ_1 reduces to its corresponding binding in σ_2, and (b) the type bindings are identical. My conversion relation is essentially the symmetric, transitive closure[5] of parallel reduction—*without* these reducing rules.

Phrasing the conversion relation in terms of CSR gives us substitutivity nearly automatically, but cotermination remains an issue. In Conjecture 3.2.1, I suggest that

[5]I prove that my relation is reflexive in Lemma 3.2.3.

Counterexample 1

Let T be a type with a free variable x.

$$\begin{array}{rcl} e_1 & = & \langle T \Rightarrow \{y{:}T[5/x] \mid \mathsf{true}\}\rangle^l\, 0 \\ e_2 & = & \langle T[5/x] \Rightarrow \{y{:}T[5/x] \mid \mathsf{true}\}\rangle^l\, (\langle T \Rightarrow T[5/x]\rangle^l\, 0) \\ e_1' = e_2' & = & 5 \end{array}$$

Observe that $e_1' \Rrightarrow e_2'$ (by EP_Refl) and $e_1 \Rrightarrow e_2$ (by EP_RPreCheck) but $e_1[5/x] = \langle T[5/x] \Rightarrow \{y{:}T[5/x] \mid \mathsf{true}\}\rangle^l\, 0 \Rrightarrow \langle \{y{:}T[5/x] \mid \mathsf{true}\}, \mathsf{true}, 0\rangle^l$ by EP_RCheck, not $e_2[5/x]$.

Counterexample 2

Let T_2 be a type with a free variable x.

$$\begin{array}{rcl} e_1 & = & \langle T_1 \to T_2 \Rightarrow T_1 \to T_2[5/x]\rangle^l\, v \\ e_2 & = & \lambda y{:}T_1.\ \langle T_2 \Rightarrow T_2[5/x]\rangle^l\, (v\, (\langle T_1 \Rightarrow T_1\rangle^l\, y)) \\ e_1' = e_2' & = & 5 \end{array}$$

Observe that $e_1' \Rrightarrow e_2'$ (by EP_Refl) and $e_1 \Rrightarrow e_2$ (by EP_RFun). We have $e_1[5/x] = \langle T_1 \to T_2[5/x] \Rightarrow T_1 \to T_2[5/x]\rangle^l\, v \Rrightarrow v[5/x]$ by EP_RRefl, not $e_2[5/x]$.

Figure 3.10: Counterexamples to substitutivity of parallel reduction in $\mathrm{F_H}$

terms related by CSR coterminate at true; this is enough to prove type soundness and parametricity of $\mathrm{F_H}$.

It is unclear if cotermination of parallel reduction holds in $\mathrm{F_H}$ despite the absence of substitutivity, i.e., whether if $e_1 \Rrightarrow e_2$ then $e_1 \longrightarrow^* v_1$ iff $e_2 \longrightarrow v_2$ such that $v_1 \Rrightarrow v_2$. But it turns out that we can get by with a simpler property: cotermination at true, rather than at arbitrary values. This property is a corollary of cotermination, since $\mathsf{true} \Rrightarrow \mathsf{true}$, but it is less likely to be interfered with by function proxies which may be introduced as in the second counterexample. The intuition that leads me to believe that cotermination at true holds for CSR is that in a well typed program, any extra checks or function proxies introduced due to differing substitutions must eventually disappear if the type of the final expression is Bool.

I believe that weak bisimulation is a promising proof technique: it's syntactic enough to avoid issues of circularity with typing, but semantic enough to relate terms that are related by $\longrightarrow^*$ reductions. I think *weak* bisimulations in particular are appropriate, because of the need for $\longrightarrow^*$ reductions on both sides of the relation. The system as defined may make such a proof slightly difficult. Recalling the first counterexample to substitutivity, we will need to have $e_1[e_1'/x]$ and $e_2[e_2'/x]$ in the relation, but how can the relation "remember" that any checks that occur on one side but not the other inevitably succeed? Introducing explicit tagging, as I do in Chapter 4, is an attractive approach to solving this technical problem. In an explicitly tagged manifest contract system, the only values inhabiting refinement types are tagged as such, e.g., $(v, \{x{:}T \mid e\})$; the operational semantics then manages tags on

values, tagging in E_CheckOK and untagging in E_Forget. Explicit tagging has several advantages: it clarifies the staging of the operational semantics; it eliminates the need for a T_Forget rule; it gives value inversion directly (Lemma 3.2.11). Finally, any proof of cotermination at true (Conjecture 3.2.1) must be careful to *not* rely on type soundness, preservation, or substitution properties. Those theorems in F_H rely on Conjecture 3.2.1, so we can't use them in its proof.

Finally: what kind of calculus *wouldn't* have cotermination at true? In a non-deterministic language, CSR may make one choice with σ_1 and another with σ_2. Fortunately, F_H is deterministic. In a deterministic language, cotermination at true may not hold for CSR if the evaluation relation abuses equalities that are violated by reduction. F_H's semantics *does* use equalities that are violated by term reduction; I believe that "abuse" means using an equality on part of a term to determine which step to take, but then ignoring that part of the term later in evaluation. Since F_H *doesn't* do that, I am confident enough to conjecture that cotermination at true holds.