Type-preserving closure conversion of PCF in Agda (more to come)

Piotr Jander

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2019

Abstract

This is an example of `infthesis` style. The file `skeleton.tex` generates this document and can be used to get a “skeleton” for your thesis. The abstract should summarise your report and fit in the space on the first page. You may, of course, use any other software to write your report, as long as you follow the same style. That means: producing a title page as given here, and including a table of contents and bibliography.

Acknowledgements

Acknowledgements go here.

Contents

Chapter 1

Introduction

The document structure should include:

- The title page in the format used above.
- An optional acknowledgements page.
- The table of contents.
- The report text divided into chapters as appropriate.
- The bibliography.

Commands for generating the title page appear in the skeleton file and are self explanatory. The file also includes commands to choose your report type (project report, thesis or dissertation) and degree. These will be placed in the appropriate place in the title page.

The default behaviour of the documentclass is to produce documents typeset in 12 point. Regardless of the formatting system you use, it is recommended that you submit your thesis printed (or copied) double sided.

The report should be printed single-spaced. It should be 30 to 60 pages long, and preferably no shorter than 20 pages. Appendices are in addition to this and you should place detail here which may be too much or not strictly necessary when reading the relevant section.

1.1 Using Sections

Divide your chapters into sub-parts as appropriate.

1.2 Citations

Note that citations (like [**?**] or [**?**]) can be generated using `BibTeX` or by using the `thebibliography` environment. This makes sure that the table of contents includes an entry for the bibliography. Of course you may use any other method as well.

1.3 Options

There are various documentclass options, see the documentation. Here we are using an option (`bsc` or `minf`) to choose the degree type, plus:

- `frontabs` (recommended) to put the abstract on the front page;
- `twoside` (recommended) to format for two-sided printing, with each chapter starting on a right-hand page;
- `singlespacing` (required) for single-spaced formating; and
- `parskip` (a matter of taste) which alters the paragraph formatting so that paragraphs are separated by a vertical space, and there is no indentation at the start of each paragraph.

Chapter 2

The Real Thing

Chapter 3

Bisimulation

In the previous chapters, we defined the source and target languages of the closure conversion, together with reduction rules for each, and a translation function from source to target.

TODO we or passive voice?

Our implementation of closure conversion is type and scope-preserving by construction. The property of type preservation would be considered a strong indication of correctness in a real-world compiler, but in this theoretical development which deals with a small, toy language, we prove stronger correctness properties which speak about operation correctness.

One such property of operational correctness of a pair of languages is bisimulation. Intuition about bisimulation is captured by a slogan: pairwise similar terms reduce to pairwise similar terms. Before we can formally define bisimulation, we need a definition of similarity between source and target terms of closure conversion.
Definition. Given a source language term M and a target language term M†, the similarity relation M ~M† is defined inductively as follows:

- (Variable) For any given variable (proof of context membership) x, we have S.‘ x ~T.‘ x.
- (Application) If M ~M† and N ~N†, then M S.· N ~M† T.· N†.
- (Abstraction) If N ~T.subst (T.exts E) N†, then S.ƛ N ~T.ƛ N† E.

Definition. Given two languages S and T and a similarity relation _~_ between them, S and T are in bisimulation if the following holds: Given source language terms M and N, and a target language term M† such that M reduces to N (M —→ N) and M is similar to M† (M M̃†), there exists a target language term N† such that M† reduces to N† (M† —→ N†) and N is similar to N† (N Ñ†),

```
{-# OPTIONS --allow-unsolved-metas #-}
module StateOfTheArt.Bisimulation where

open import Data.List using (List; []; _::_)
```

```
import Relation.Binary.PropositionalEquality as Eq
open Eq using (_≡_; refl; trans; cong; sym; cong₂)
open Eq.≡-Reasoning using (begin_; _≡⟨⟩_; _≡⟨_⟩_; _∎)

open import indexed
open import var hiding (_<$>_ ; get)
open import environment as E hiding (_»_ ; extend)

open import StateOfTheArt.Types
import StateOfTheArt.STLC as S
open S using (_/_)
import StateOfTheArt.Closure as T
import StateOfTheArt.STLC-Thms as ST
import StateOfTheArt.Closure-Thms as TT

convert : ∀ {Γ σ}
  → S.Lam σ Γ
  → T.Lam σ Γ
convert (S.V x) = T.V x
convert (S.A M N) = T.A (convert M) (convert N)
convert (S.L N) = T.L (convert N) T.id-subst

infix 4 _~_

data _~_ : ∀ {Γ σ} → S.Lam σ Γ → T.Lam σ Γ → Set where

  ~V : ∀ {Γ σ} {x : Var σ Γ}
      ----------------
    → S.V x ~ T.V x

  ~L : ∀ {Γ Δ σ τ} {N : S.Lam τ (σ :: Γ)} {N† : T.Lam τ (σ :: Δ)} {E : T.Subst Δ Γ}
    → N ~ T.subst (T.exts E) N†
      ------------------
    → S.L N ~ T.L N† E

  ~A : ∀ {Γ σ τ} {L : S.Lam (σ ⇒ τ) Γ} {L† : T.Lam (σ ⇒ τ) Γ}
          {M : S.Lam σ Γ} {M† : T.Lam σ Γ}
    → L ~ L†
    → M ~ M†
      ----------------------
    → S.A L M ~ T.A L† M†

graph→relation : ∀ {Γ σ} (N : S.Lam σ Γ)
  → N ~ convert N
graph→relation (S.V x) = ~V
graph→relation (S.A f e) = ~A (graph→relation f) (graph→relation e)
```

```
graph→relation (S.L b) = ~L g
  where
  h : T.subst (T.exts T.id-subst) (convert b) ≡ convert b
  h =
    begin
      T.subst (T.exts T.id-subst) (convert b)
    ≡⟨ cong (λ e → T.subst e (convert b)) (sym (env-extensionality TT.exts-id-subst)) ⟩
      T.subst T.id-subst (convert b)
    ≡⟨ TT.subst-id-id (convert b) ⟩
      convert b
    ∎
  g : b ~ T.subst (T.exts T.id-subst) (convert b)
  g rewrite h = graph→relation b

graph←relation : ∀ {Γ σ} {N : S.Lam σ Γ} {N† : T.Lam σ Γ}
  → N ~ N†
  → convert N ≡ N†
graph←relation ~V = refl
graph←relation (~L ~N) = {!!}
graph←relation (~A ~M ~N) = cong₂ T.A (graph←relation ~M) {!graph←relation ~N!}

~val : ∀ {Γ σ} {M : S.Lam σ Γ} {M† : T.Lam σ Γ}
  → M ~ M†
  → S.Value M
    ---------
  → T.Value M†
~val ~V         ()
~val (~L ~N) S.V-L = T.V-L
~val (~A ~M ~N) ()

~rename : ∀ {Γ Δ}
  → (ρ : Thinning Γ Δ)
    ----------------------------------------------------------
  → ∀ {σ} {M : S.Lam σ Γ} {M† : T.Lam σ Γ} → M ~ M† → S.rename ρ M ~ T.rename ρ M†
~rename ρ ~V = ~V
~rename ρ (~A ~M ~N) = ~A (~rename ρ ~M) (~rename ρ ~N)
~rename ρ (~L {N = N} {N†} {E} ~N) with ~rename (T.ext ρ) ~N
... | ~ρN rewrite TT.lemma-~ren-L ρ E N† = ~L ~ρN

infix 3 _~σ_
record _~σ_ {Γ Δ : Context} (ρ : S.Subst Γ Δ) (ρ† : T.Subst Γ Δ) : Set where
  field ρ~ρ† : ∀ {σ} → (x : Var σ Γ) → lookup ρ x ~ lookup ρ† x
open _~σ_ public

~exts : ∀ {Γ Δ σ}
  → {ρ : S.Subst Γ Δ}
```

```
  → {ρ† : T.Subst Γ Δ}
  → ρ ~σ ρ†
    -----------------------------------------------
  → S.exts {σ = σ} ρ ~σ T.exts ρ†
ρ~ρ† (~exts ~ρ) Z = ~V
ρ~ρ† (~exts {σ = σ} {ρ = ρ} {ρ†} ~ρ) (S x) = ~rename E.extend (ρ~ρ† ~ρ x)

~id-subst : ∀ {Γ} → S.id-subst {Γ} ~σ T.id-subst {Γ}
ρ~ρ† ~id-subst x = ~V

_~•_ : ∀ {Γ Δ σ}
    {ρ : S.Subst Γ Δ} {ρ† : T.Subst Γ Δ}
    {M : S.Lam σ Δ} {M† : T.Lam σ Δ}
  → ρ ~σ ρ†
  → M ~ M†
    -----------------------------------------------
  → ρ • M ~σ ρ† • M†
ρ~ρ† (ρ~σρ† ~• M~M†) Z = M~M†
ρ~ρ† (ρ~σρ† ~• M~M†) (S x) = ρ~ρ† ρ~σρ† x

~subst : ∀ {Γ Δ}
  → {ρ : S.Subst Γ Δ}
  → {ρ† : T.Subst Γ Δ}
  → ρ ~σ ρ†
    -----------------------------------------------------
  → (∀ {τ} {M : S.Lam τ Γ} {M† : T.Lam τ Γ} → M ~ M† → S.subst ρ M ~ T.subst ρ† M†)
~subst ~ρ (~V {x = x}) = ρ~ρ† ~ρ x
~subst {ρ† = ρ†} ~ρ (~L {N = N} {N†} {E} ~N) with ~subst (~exts ~ρ) ~N
... | ~ρN rewrite TT.lemma-~subst-L ρ† E N† = ~L ~ρN
~subst ~ρ (~A ~M ~N) = ~A (~subst ~ρ ~M) (~subst ~ρ ~N)

/V≡E•V† : ∀ {Γ Δ σ τ}
    {N : S.Lam τ (σ :: Γ)} {N† : T.Lam τ (σ :: Δ)} {E : T.Subst Δ Γ}
    {V : S.Lam σ Γ} {V† : T.Lam σ Γ}
  → N ~ T.subst (T.exts E) N†
  → V ~ V†
    -----------------------
  → N / V ~ T.subst (E • V†) N†
/V≡E•V† {N = N} {N†} {E} {VV} {V†} ~N ~VV
  rewrite cong (λ e → (N / VV) ~ e) (sym (TT.subst-E•V N† E V†))
  = ~subst (~id-subst ~• ~VV) ~N

data Leg {Γ σ} (M† : T.Lam σ Γ) (N : S.Lam σ Γ) : Set where

  leg : ∀ {N† : T.Lam σ Γ}
    → N ~ N†
```

```
    → M† T.—→ N†
      ---------
    → Leg M† N

sim : ∀ {Γ σ} {M N : S.Lam σ Γ} {M† : T.Lam σ Γ}
  → M ~ M†
  → M S.—→ N
    ---------
  → Leg M† N
sim ~V ()
sim (~L ~N) ()
sim (~A ~M ~N) (S.ξ-A₁ M—→)
  with sim ~M M—→
... | leg ~M’ M†—→ = leg (~A ~M’ ~N) (T.ξ-A₁ M†—→)
sim (~A ~M ~N) (S.ξ-A₂ VV N—→)
  with sim ~N N—→
... | leg ~N′ N†—→ = leg (~A ~M ~N′) (T.ξ-A₂ (~val ~M VV) N†—→)
sim (~A (~L {N = N} {N†} ~N) ~VV) (S.β-L VV)
  = leg (/V≡E•V† {N = N} {N†} ~N ~VV) (T.β-L (~val ~VV VV))
```

Of course you may want to use several chapters and much more text than here.