Step By Step Instructions

This guide explains step-by-step how to compile and evaluate our mechanised proof.

Our proof artifacts consist of the following parts:

- The `cdot/` directory contains sources of the mechanization of the cDOT calculus. The proof is an extension of pDOT soundness proof.
- The `lambda2Gmu/` directory contains sources of the mechanization of the Lambda2Gmu calculus and `lambda2Gmu_annotated/` contains sources of the variant with additional type annotations, as described in the paper.
- The `translation/` directory contains lemmas related to the translation: the typing of the `lib` term and an example showing inversion of tuple equality using our added inversion rules.

Compiling the Proof

Our proof artifacts contain Coq proof scripts of the calculi in our paper. It compiles with Coq 8.13.0 and the TLC library. We assume that you have followed the instructions of our getting-started guide to set up all requirements for compilation.

The `translation` proof depends on both `cdot` and `lambda2Gmu_annotated`, and `lambda2Gmu_annotated` itself depends on `lambda2Gmu`. The following commands will compile the proof in a proper order:

```
make -C cdot/
make -C lambda2Gmu/
make -C lambda2Gmu_annotated/
make -C translation/
```

If each of the `make` command exits without error, all the proof artifacts are compiled successfully.

Paper-to-Artifact Correspondence

Now we explain the correspondence between important definitions and lemmas in the paper and their mechanised versions.

cDOT Calculus

The mechanization of cDOT is in the `cdot/` directory. Based on the soundness proof of pDOT, the soundness proof uses the locally nameless representation with cofinite quantification to represent terms. The Sequences library by Xavier Leroy is also included in our proof to ease the reasoning about the reflexive, transitive closure of binary relations.

Definitions

Definition	Artifact File	Name of Formalization	Proof Notation
Abstract syntax	cdot/Definitions.v	`typ`, `trm`	
Term typing rules	cdot/Definitions.v	`ty_trm`	`G ⊢ t : T`
Definition typing rules	cdot/Definitions.v	`ty_def`, `ty_defs`	`x; bs; G ⊢ d : D`, `x; bs; G ⊢ ds :: T`
Subtyping rules	cdot/Definitions.v	`subtyp`	`G ⊢ T <: U`
Invertible subtyping rules	cdot/SemanticSubtyping.v	`subtyp_s`	
Reduction	cdot/Reduction.v	`red`	`(γ, t) ⟼ (γ', t')`
Lookup	cdot/Lookup.v	`lookup_step`	`γ ⟦ p ↝ t ⟧`, `γ ⟦ p ↝* t ⟧`

Theorems

Theorem	Artifact File	Name
Theorem 4.1 (Type Safety)	cdot/Safety.v	`safety`
Theorem 4.2 (Preservation)	cdot/Safety.v	`preservation`
Theorem 4.3 (Progress)	cdot/Safety.v	`progress`
Lemma 4.1 (Tag Resolution)	cdot/CanonicalForms.v	`tag_resolution`
Lemma 4.2 (Field Inversion)	cdot/GADTRules.v	`invert_subtyp_fld_t`
Lemma 4.3 (Type Member Inversion)	cdot/GADTRules.v	`invert_subtyp_rcd_t`
Lemma 4.4 (<:# to <:##)	cdot/SemanticSubtyping.v	`tight_to_semantic`
Lemma 4.5 (<:## to <:#)	cdot/SemanticSubtyping.v	`semantic_to_tight`
Lemma 4.6 (Field Inversion in <:##)	cdot/GADTRules.v	`invert_subtyp_trm_s`

$\lambda_{2,G\mu}$ Calculus

The `lambda2Gmu/` and `lambda2Gmu_annotated/` directories contain the mechanization of the $\lambda_{2,G\mu}$ calculus we encode into cDOT in the paper. The proof also employs the locally nameless representation with cofinite quantification.

Definition

Definition	Artifact File	Name of Formalization	Proof Notation
Abstract syntax	lambda2Gmu_annotated/Definitions.v	`term`	
Operational semantics	lambda2Gmu_annotated/Definitions.v	`red`	`e1 --> e2`

Theorems

Theorem	Artifact File	Name
Theorem 5.1 (Preservation)	lambda2Gmu_annotated/Preservation.v	`preservation_thm`
Theorem 5.2 (Progress)	lambda2Gmu_annotated/Progress.v	`progress_thm`

Translation

The `translation/` directory contains the formalization of our encoding. It contains the typing of the `lib` term and an example showing inversion of tuple equality using the inversion rules in cDOT.

Definition/Theorem	Artifact File	Name
The `lib` term	translation/Library.v	`libTrm`
Lemma A.2	translation/Library.v	`libTypes`
Tuple inversion example	translation/DestructTupleLemma.v	`destruct_tuple_lemma`