Conservativity of Two-Level Type Theory Corresponds to Staged Compilation

András Kovács

Eötvös Loránd University

21 June 2022, TYPES, Nantes
Overview

Two-level TT:

- **Voevodsky**: *A simple type system with two identity types*
- **Anneckov, Capriotti, Kraus, Sattler**: *Two-Level Type Theory and Applications*
- Goal: synthetic homotopy theory
Two-level TT:

- Voevodsky: *A simple type system with two identity types*
- Annekov, Capriotti, Kraus, Sattler: *Two-Level Type Theory and Applications*
- Goal: synthetic homotopy theory

Staged compilation:

- Template Haskell, MetaOCaml
- Goal: code generation (for performance, code reuse)
Two-level TT:

- **Voevodsky**: *A simple type system with two identity types*
- **Annekov, Capriotti, Kraus, Sattler**: *Two-Level Type Theory and Applications*
- **Goal**: synthetic homotopy theory

Staged compilation:

- Template Haskell, MetaOCaml
- **Goal**: code generation (for performance, code reuse)
- **Remark**: staged compilation $\neq$ staged computation
1. Two universes $U_0$, $U_1$, closed under arbitrary type formers.
1. Two universes $U_0$, $U_1$, closed under arbitrary type formers.
2. No elimination allowed from one universe to the other.
Rules

1. Two universes $U_0$, $U_1$, closed under arbitrary type formers.
2. No elimination allowed from one universe to the other.
3. *Lifting:* for $A : U_0$, we have $\uparrow A : U_1$. 

1. Two universes $U_0$, $U_1$, closed under arbitrary type formers.
2. No elimination allowed from one universe to the other.
3. **Lifting**: for $A : U_0$, we have $\uparrow A : U_1$.
4. **Quoting**: for $A : U_0$ and $t : A$, we have $<t> : \uparrow A$. 
1. Two universes $U_0$, $U_1$, closed under arbitrary type formers.
2. No elimination allowed from one universe to the other.
3. *Lifting*: for $A : U_0$, we have $\uparrow A : U_1$.
4. *Quoting*: for $A : U_0$ and $t : A$, we have $\langle t \rangle : \uparrow A$.
5. *Splicing*: for $t : \uparrow A$, we have $\sim t : A$. 


Rules

1. Two universes $U_0$, $U_1$, closed under arbitrary type formers.
2. No elimination allowed from one universe to the other.
3. **Lifting**: for $A : U_0$, we have $⇑A : U_1$.
4. **Quoting**: for $A : U_0$ and $t : A$, we have $⟨t⟩ : ⇑A$.
5. **Splicing**: for $t : ⇑A$, we have $∼t : A$.
6. Quoting and splicing are definitional inverses.
Staging input:

\[
\begin{align*}
\text{two} & : \uparrow\text{Nat}_0 \\
\text{two} & = <\text{suc}_0 (\text{suc}_0 \text{zero}_0)>
\end{align*}
\]

\[
\begin{align*}
f & : \text{Nat}_0 \rightarrow \text{Nat}_0 \\
f & = \lambda x. x + \sim\text{two}
\end{align*}
\]
Staging input:

\[
\begin{align*}
\text{two} &: \uparrow \text{Nat}_0 \\
\text{two} &= \langle \text{suc}_0 (\text{suc}_0 \text{zero}_0) \rangle \\
\text{f} &: \text{Nat}_0 \to \text{Nat}_0 \\
f &= \lambda x. x + \sim \text{two}
\end{align*}
\]

Output:

\[
\begin{align*}
\text{f} &: \text{Nat}_0 \to \text{Nat}_0 \\
f &= \lambda x. x + \text{suc}_0 (\text{suc}_0 \text{zero}_0)
\end{align*}
\]
Input:

\[ id : (A : U_1) \rightarrow A \rightarrow A \]
\[ id = \lambda A \, x. \, x \]

\[ id\text{-}Bool_0 : \text{Bool}_0 \rightarrow \text{Bool}_0 \]
\[ id\text{-}Bool_0 = \lambda x. \sim (id \langle \text{Bool}_0 \rangle \langle x \rangle) \]
Compile-time functions

Input:

\[ \text{id} : (A : U_1) \rightarrow A \rightarrow A \]
\[ \text{id} = \lambda A \ x. \ x \]

\[ \text{idBool}_0 : \text{Bool}_0 \rightarrow \text{Bool}_0 \]
\[ \text{idBool}_0 = \lambda x. ~ (\text{id} \ <\text{Bool}_0> \ <x>) \]

Output:

\[ \text{idBool}_0 : \text{Bool}_0 \rightarrow \text{Bool}_0 \]
\[ \text{idBool}_0 = \lambda x. \ x \]
Inlined map arguments

Input:

\[
\text{inlMap} : \{A, B : U_0\} \rightarrow (\uparrow \sim A \rightarrow \uparrow \sim B) \rightarrow \uparrow (\text{List}_0 \sim A) \rightarrow \uparrow (\text{List}_0 \sim B)
\]

\[
inlMap = \lambda f \ as. \text{foldr}_0 (\lambda a \ bs. \text{cons}_0 \sim (f \langle a \rangle) \ bs) \text{nil}_0 \sim as
\]

\[
f : \text{List}_0 \text{Nat}_0 \rightarrow \text{List}_0 \text{Nat}_0
\]

\[
f = \lambda xs. \sim (\text{inlMap} (\lambda n. \langle \sim n + 2 \rangle) \langle xs \rangle)
\]
Inlined map arguments

Input:

\[
\text{inlMap} : \{ A B : \uparrow U_0 \} \to (\uparrow \sim A \to \uparrow \sim B) \to \uparrow (\text{List}_0 \sim A) \to \uparrow (\text{List}_0 \sim B)
\]

\[
\text{inlMap} = \lambda f \ as. \ \langle \text{foldr}_0 (\lambda a \ bs. \ \text{cons}_0 \sim (f \ <a>) \ bs) \ \text{nil}_0 \sim as \rangle
\]

\[
\text{f} : \text{List}_0 \ \text{Nat}_0 \to \text{List}_0 \ \text{Nat}_0
\]

\[
f = \lambda xs. \ \sim (\text{inlMap} (\lambda n. \ \sim n + 2) \ <xs>)
\]

Output:

\[
\text{f} : \text{List}_0 \ \text{Nat}_0 \to \text{List}_0 \ \text{Nat}_0
\]

\[
f = \lambda xs. \ \text{foldr}_0 (\lambda a \ bs. \ \text{cons}_0 (a + 2) \ bs) \ \text{nil}_0 \ xs
\]
Staging Types

Input:

Vec : Nat₁ → ↑U₀ → ↑U₀
Vec zero₁ A = <⊥₀>
Vec (suc₁ n) A = <∼A ×₀ ∼(Vec n A)>

Tuple3 : U₀ → U₀
Tuple3 A = ∼(Vec 3 <A>)
Input:

\[ \text{Vec} : \text{Nat}_1 \rightarrow \Uparrow \text{U}_0 \rightarrow \Uparrow \text{U}_0 \]
\[ \text{Vec zero}_1 \quad A = \langle \top_0 \rangle \]
\[ \text{Vec} (\text{suc}_1 n) \quad A = \langle \sim A \times_0 \sim (\text{Vec} n A) \rangle \]

\[ \text{Tuple3} : \text{U}_0 \rightarrow \text{U}_0 \]
\[ \text{Tuple3} \quad A = \sim (\text{Vec} 3 \langle A \rangle) \]

Output:

\[ \text{Tuple3} : \text{U}_0 \rightarrow \text{U}_0 \]
\[ \text{Tuple3} \quad A = A \times_0 (A \times_0 (A \times_0 \top_0)) \]
map for Vec

Input:

\[
\text{map : } \{ A \, B : \uparrow U_0 \} \rightarrow (n : \text{Nat}_1) \rightarrow (\uparrow \sim A \rightarrow \uparrow \sim B) \\
\rightarrow \uparrow (\text{Vec} \, n \, A) \rightarrow \uparrow (\text{Vec} \, n \, B)
\]

\[
\text{map zero}_1 \quad f \, as = \langle \text{tt}_0 \rangle
\]

\[
\text{map} \, (\text{suc}_1 \, n) \, f \, as = \langle (\sim (f \, \langle \text{fst}_0 \, \sim \, as \rangle), \sim (\text{map} \, n \, f \, \langle \text{snd}_0 \, \sim \, as \rangle)) \rangle
\]

\[
f : \sim (\text{Vec} \, 2 \, \langle \text{Nat}_0 \rangle) \rightarrow \sim (\text{Vec} \, 2 \, \langle \text{Nat}_0 \rangle)
\]

\[
f \, xs = \sim (\text{map} \, 2 \, (\lambda \, x. \langle \sim x + 2 \rangle) \, \langle xs \rangle)
\]
map for Vec

Input:

\[
\text{map} : \{A, B : \uparrow U_0\} \to (n : \text{Nat}_1) \to (\uparrow \sim A \to \uparrow \sim B) \\
\text{map \ zero} \_1 f \ as = <\text{tt}_0> \\
\text{map} (\text{succ}_1 n) f \ as = <(\sim (f \ <\text{fst}_0 \sim as>), \sim (\text{map} n f \ <\text{snd}_0 \sim as>))>
\]

\[
f : \sim(\text{Vec}2 <\text{Nat}_0>) \to \sim(\text{Vec}2 <\text{Nat}_0>) \\
f \ xs = \sim(\text{map} 2 (\lambda x. <\sim x + 2>) \ <xs>)
\]

Output:

\[
f : \text{Nat}_0 \times 0 (\text{Nat}_0 \times 0 \top_0) \to \text{Nat}_0 \times 0 (\text{Nat}_0 \times 0 \top_0) \\
f \ xs = (\text{fst}_0 \ xs + 2, (\text{fst}_0 (\text{snd}_0 \ xs) + 2, \text{tt}_0))
\]
In the demo implementation:

- Bidirectional elaboration
- Coercive subtyping for $\uparrow$ and type formers
- Standard unification techniques

Almost all quotes and splices are inferable in practice.
Staging as Conservativity

The **object theory** is the TT supporting only $U_0$ and its type formers.
The **object theory** is the TT supporting only $U_0$ and its type formers.

The **object-level fragment** of 2LTT contains types in $U_0$, their terms, and only allows contexts with entries in $U_0$. 

Conservativity of 2LTT means

- There's a bijection between object-theoretic types and object-fragment 2LTT types.
- There's also a bijection between object-theoretic terms and object-fragment 2LTT terms.
- (Both up to $\beta\eta$-conversion).

(See proof in the preprint)
The **object theory** is the TT supporting only $U_0$ and its type formers.

The **object-level fragment** of 2LTT contains types in $U_0$, their terms, and only allows contexts with entries in $U_0$.

**Conservativity of 2LTT means**

- There’s a bijection between object-theoretic types and object-fragment 2LTT types.
- There’s also a bijection between object-theoretic terms and object-fragment 2LTT terms.
- (Both up to $\beta\eta$-conversion).

(See proof in the preprint)
ICFP preprint, implementation, tutorial:
github.com/AndrasKovacs/staged

Thanks for your attention!