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*
- *Annekov, Capriotti, Kraus, Sattler: Two-Level Type Theory and Applications*
- Goal: synthetic homotopy theory

Overview

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)

Overview

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*

Rules

1. Two universes U_0, U_1, closed under arbitrary type formers.

Rules

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 $A : \mathsf{U}_0$, we have $\Uparrow A : \mathsf{U}_1$.

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 $A : \mathsf{U}_0$, we have $\Uparrow A : \mathsf{U}_1$.
4. *Quoting:* for $A : \mathsf{U}_0$ and $t : A$, we have $<t> : \Uparrow A$.

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 $A : \mathsf{U}_0$, we have $\Uparrow A : \mathsf{U}_1$.
4. *Quoting:* for $A : \mathsf{U}_0$ and $t : A$, we have $<t> : \Uparrow A$.
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 $A : \mathsf{U}_0$, we have $\Uparrow A : \mathsf{U}_1$.
4. *Quoting:* for $A : \mathsf{U}_0$ and $t : A$, we have $<t> : \Uparrow A$.
5. *Splicing:* for $t : \Uparrow A$, we have $\sim t : A$.
6. Quoting and splicing are definitional inverses.

Inlined definitions

Staging input:

$$\mathsf{two} : {\Uparrow}\mathsf{Nat}_0$$
$$\mathsf{two} = <\mathsf{suc}_0\,(\mathsf{suc}_0\,\mathsf{zero}_0)>$$

$$\mathsf{f} : \mathsf{Nat}_0 \to \mathsf{Nat}_0$$
$$\mathsf{f} = \lambda\, x.\, x + {\sim}\mathsf{two}$$

Inlined definitions

Staging input:

$$\mathsf{two} : \Uparrow \mathsf{Nat}_0$$
$$\mathsf{two} = <\mathsf{suc}_0\,(\mathsf{suc}_0\,\mathsf{zero}_0)>$$

$$\mathsf{f} : \mathsf{Nat}_0 \to \mathsf{Nat}_0$$
$$\mathsf{f} = \lambda\, x.\, x + {\sim}\mathsf{two}$$

Output:

$$\mathsf{f} : \mathsf{Nat}_0 \to \mathsf{Nat}_0$$
$$\mathsf{f} = \lambda\, x.\, x + \mathsf{suc}_0\,(\mathsf{suc}_0\,\mathsf{zero}_0)$$

Compile-time functions

Input:

$$\mathsf{id} : (A : \mathsf{U}_1) \to A \to A$$
$$\mathsf{id} = \lambda\, A\, x.\, x$$

$$\mathsf{idBool}_0 : \mathsf{Bool}_0 \to \mathsf{Bool}_0$$
$$\mathsf{idBool}_0 = \lambda\, x.\, {\sim}(\mathsf{id}\,({\Uparrow}\mathsf{Bool}_0)\,{<}x{>})$$

Compile-time functions

Input:

$$\mathsf{id} : (A : \mathsf{U}_1) \to A \to A$$
$$\mathsf{id} = \lambda\, A\, x.\, x$$

$$\mathsf{idBool}_0 : \mathsf{Bool}_0 \to \mathsf{Bool}_0$$
$$\mathsf{idBool}_0 = \lambda\, x.\, {\sim}(\mathsf{id}\,({\Uparrow}\mathsf{Bool}_0)\, {<}x{>})$$

Output:

$$\mathsf{idBool}_0 : \mathsf{Bool}_0 \to \mathsf{Bool}_0$$
$$\mathsf{idBool}_0 = \lambda\, x.\, x$$

Inlined map arguments

Input:

$$\mathsf{inlMap} : \{A\,B : \Uparrow\mathsf{U}_0\} \to (\Uparrow\sim A \to \Uparrow\sim B) \to \Uparrow(\mathsf{List}_0\sim A) \to \Uparrow(\mathsf{List}_0\sim B)$$
$$\mathsf{inlMap} = \lambda\, f\; as.\; <\mathsf{foldr}_0\,(\lambda\, a\; bs.\; \mathsf{cons}_0 \sim(f\;<a>)\; bs)\; \mathsf{nil}_0 \sim as>$$

$$\mathsf{f} : \mathsf{List}_0\,\mathsf{Nat}_0 \to \mathsf{List}_0\,\mathsf{Nat}_0$$
$$\mathsf{f} = \lambda\, xs.\; \sim(\mathsf{inlMap}\,(\lambda\, n.\, <\sim n + 2>)\, <xs>)$$

Inlined map arguments

Input:

$$\mathsf{inlMap} : \{A\,B : \Uparrow\mathsf{U}_0\} \to (\Uparrow{\sim}A \to \Uparrow{\sim}B) \to \Uparrow(\mathsf{List}_0\,{\sim}A) \to \Uparrow(\mathsf{List}_0\,{\sim}B)$$
$$\mathsf{inlMap} = \lambda\,f\,as.\,{<}\mathsf{foldr}_0\,(\lambda\,a\,bs.\,\mathsf{cons}_0\,{\sim}(f\,{<}a{>})\,bs)\,\mathsf{nil}_0\,{\sim}as{>}$$

$$\mathsf{f} : \mathsf{List}_0\,\mathsf{Nat}_0 \to \mathsf{List}_0\,\mathsf{Nat}_0$$
$$\mathsf{f} = \lambda\,xs.\ {\sim}(\mathsf{inlMap}\,(\lambda\,n.\,{<}{\sim}n + 2{>})\,{<}xs{>})$$

Output:

$$\mathsf{f} : \mathsf{List}_0\,\mathsf{Nat}_0 \to \mathsf{List}_0\,\mathsf{Nat}_0$$
$$\mathsf{f} = \lambda\,xs.\,\mathsf{foldr}_0\,(\lambda\,a\,bs.\,\mathsf{cons}_0\,(a + 2)\,bs)\,\mathsf{nil}_0\,xs$$

Staging Types

Input:

$$
\begin{aligned}
&\mathsf{Vec} : \mathsf{Nat}_1 \to {\Uparrow}\mathsf{U}_0 \to {\Uparrow}\mathsf{U}_0 \\
&\mathsf{Vec}\,\mathsf{zero}_1 \quad\;\; A = {<}\top_0{>} \\
&\mathsf{Vec}\,(\mathsf{suc}_1\; n)\, A = {<}{\sim}A \times_0 {\sim}(\mathsf{Vec}\; n\; A){>}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Tuple3} : \mathsf{U}_0 \to \mathsf{U}_0 \\
&\mathsf{Tuple3}\, A = {\sim}(\mathsf{Vec}\; 3\; {<}A{>})
\end{aligned}
$$

Staging Types

Input:

$$
\begin{aligned}
&\mathsf{Vec} : \mathsf{Nat}_1 \to {\Uparrow}\mathsf{U}_0 \to {\Uparrow}\mathsf{U}_0 \\
&\mathsf{Vec}\,\mathsf{zero}_1 \quad\;\; A = <\top_0> \\
&\mathsf{Vec}\,(\mathsf{suc}_1\ n)\,A = <{\sim}A \times_0 {\sim}(\mathsf{Vec}\,n\,A)>
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Tuple3} : \mathsf{U}_0 \to \mathsf{U}_0 \\
&\mathsf{Tuple3}\,A = {\sim}(\mathsf{Vec}\,3\,<A>)
\end{aligned}
$$

Output:

$$
\begin{aligned}
&\mathsf{Tuple3} : \mathsf{U}_0 \to \mathsf{U}_0 \\
&\mathsf{Tuple3}\,A = A \times_0 (A \times_0 (A \times_0 \top_0))
\end{aligned}
$$

map for Vec

Input:

$$\begin{aligned}
&\mathsf{map} : \{A\,B : \Uparrow\mathsf{U}_0\} \to (n : \mathsf{Nat}_1) \to (\Uparrow\sim A \to \Uparrow\sim B) \\
&\qquad\qquad\qquad\qquad\to \Uparrow(\mathsf{Vec}\,n\,A) \to \Uparrow(\mathsf{Vec}\,n\,B) \\
&\mathsf{map}\,\mathsf{zero}_1 \qquad f\ as = <\mathsf{tt}_0> \\
&\mathsf{map}\,(\mathsf{suc}_1\,n)\,f\ as = <(\sim(f\,<\mathsf{fst}_0\sim as>),\ \sim(\mathsf{map}\,n\,f\,<\mathsf{snd}_0\sim as>))>
\end{aligned}$$

$$\begin{aligned}
&\mathsf{f} : \sim(\mathsf{Vec}\,2\,<\mathsf{Nat}_0>) \to \sim(\mathsf{Vec}\,2\,<\mathsf{Nat}_0>) \\
&\mathsf{f}\,xs = \sim(\mathsf{map}\,2\,(\lambda\,x.\,<\sim x + 2>)\,<xs>)
\end{aligned}$$

map for Vec

Input:

$$\begin{aligned}
&\mathsf{map} : \{A\,B : \Uparrow \mathsf{U}_0\} \to (n : \mathsf{Nat}_1) \to (\Uparrow{\sim}A \to \Uparrow{\sim}B) \\
&\qquad\qquad\qquad\quad \to \Uparrow(\mathsf{Vec}\,n\,A) \to \Uparrow(\mathsf{Vec}\,n\,B) \\
&\mathsf{map}\,\mathsf{zero}_1 \quad\; f\;as = <\mathsf{tt}_0> \\
&\mathsf{map}\,(\mathsf{suc}_1\,n)\,f\;as = <(\sim(f\,<\mathsf{fst}_0\,{\sim}as>),\ \sim(\mathsf{map}\,n\,f\,<\mathsf{snd}_0\,{\sim}as>))>
\end{aligned}$$

$$\begin{aligned}
&\mathsf{f} : \sim(\mathsf{Vec}\,2\,<\mathsf{Nat}_0>) \to \sim(\mathsf{Vec}\,2\,<\mathsf{Nat}_0>) \\
&\mathsf{f}\;xs = \sim(\mathsf{map}\,2\,(\lambda\,x.\,<{\sim}x + 2>)\,<xs>)
\end{aligned}$$

Output:

$$\begin{aligned}
&\mathsf{f} : \mathsf{Nat}_0 \times_0 (\mathsf{Nat}_0 \times_0 \top_0) \to \mathsf{Nat}_0 \times_0 (\mathsf{Nat}_0 \times_0 \top_0) \\
&\mathsf{f}\;xs = (\mathsf{fst}_0\,xs + 2,\ (\mathsf{fst}_0\,(\mathsf{snd}_0\,xs) + 2,\ \mathsf{tt}_0))
\end{aligned}$$

Ergonomics

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.

Staging as Conservativity

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.

Staging as Conservativity

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!