A Dependent Language with Type-Safe Extraction

Work in Progress

Greg Brown
greg.brown01@ed.ac.uk

University of Edinburgh

WITS '26

Outline

Dependent
Language
(MLTT)

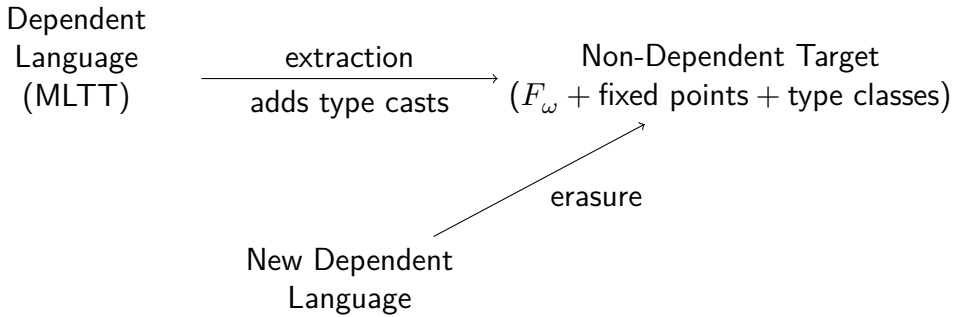
Outline

Dependent Language (MLTT) $\xrightarrow{\text{extraction}}$ Non-Dependent Target (F_ω + fixed points + type classes)

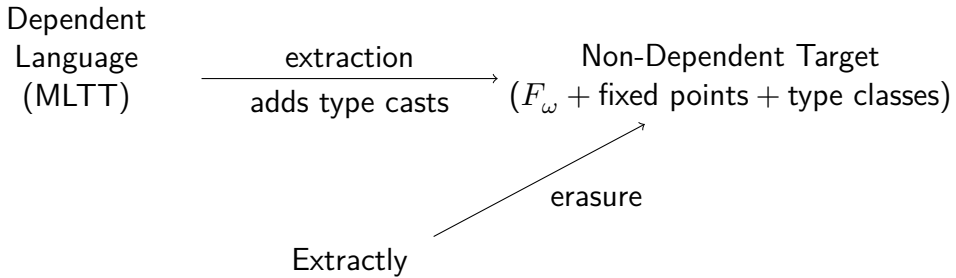
Outline

Dependent Language (MLTT) $\xrightarrow[\text{adds type casts}]{\text{extraction}}$ Non-Dependent Target (F_ω + fixed points + type classes)

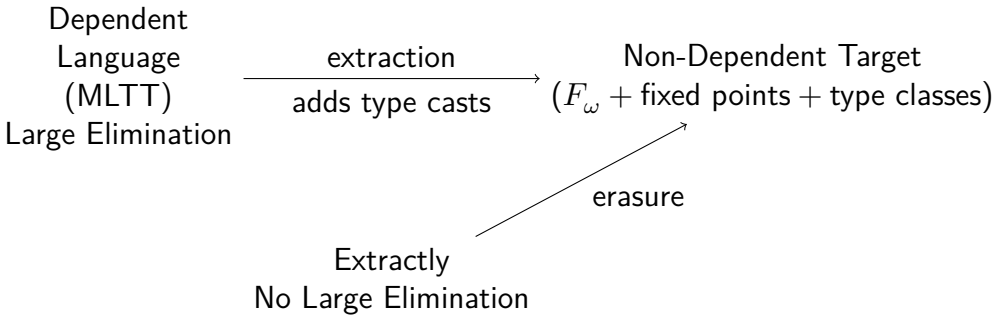
Outline


Dependent
Language
(MLTT)
extraction
adds type casts
Non-Dependent Target
(F_ω + fixed points + type classes)
erasure
New Dependent
Language

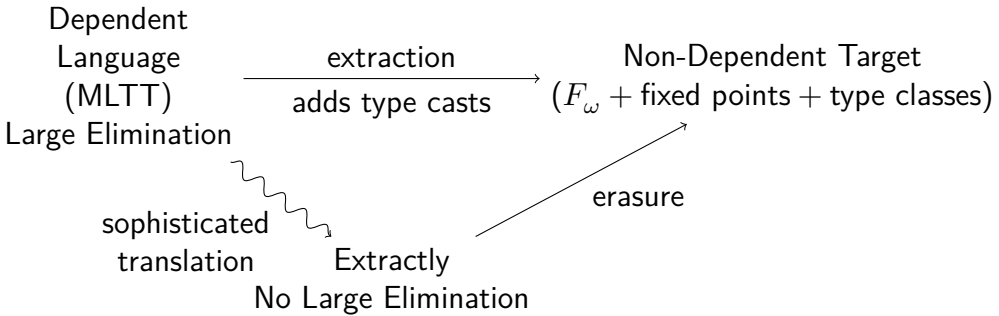
Outline


Dependent
Language
(MLTT)
extraction
adds type casts
Non-Dependent Target
(F_ω + fixed points + type classes)
erasure
Extractly

Outline


Dependent
Language
(MLTT)
Large Elimination
extraction
adds type casts
Non-Dependent Target
(F_ω + fixed points + type classes)
erasure
Extractly
No Large Elimination

Outline


Dependent
Language
(MLTT)
Large Elimination
extraction
adds type casts
Non-Dependent Target
(F_ω + fixed points + type classes)
sophisticated
translation
Extractly
No Large Elimination
erasure

Current Completed Work

Complete

- Type checking

To Do

- Erasure
- Formal definition
- Translation from CIC/MLTT

Try Extractly Yourself

Extractly Features

Inspired by Polarity [Binder et al. 2024]

Data and Codata Types

```
data Nat {Zero : Nat; Succ(pred : Nat) : Nat;}
codata Pair(a b : Set) : Set {first() -> a; second() -> b;}
```

Definable Functions

```
codata Function(a b: Set) : Set {apply(x : a) -> b;}
```

Universe Hierarchy

```
codata Family(a : Set) : Set 1 {at(x : a) -> Set;}
```

Extractly Restrictions

Data Types are Small

```
-- Invalid Extractly
data Anything : Set 1 {
  Something(a : Set, x : a) : Anything;
}
```

No Large Elimination of Data Types

```
-- Invalid Extractly
def vec_prod(a : Set, n : Nat) -> Set {
  match n {Zero() => Unit; Succ(k) => Pair(a, vec_prod(a, k));}
}
```

Names of Types are Independent from Data Values

Conjecture

The "name" of every type does not depend on the values of free variables of data types within its scope.

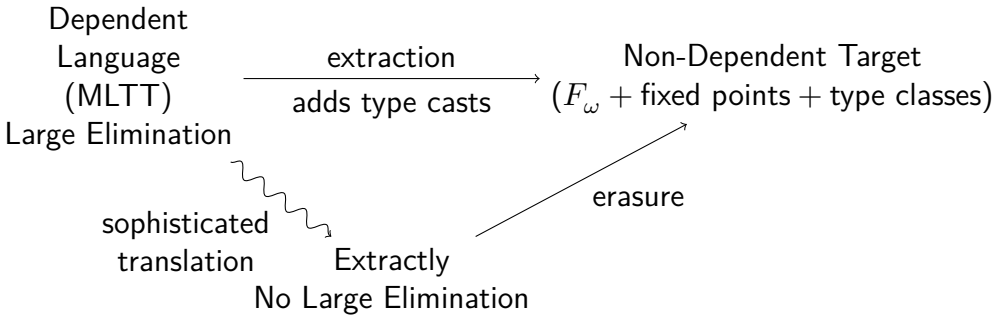
Types Depend on Codata Values

```
codata Family(a : Set) : Set 1 {
  at(x : a) -> Set;
}

codata FamilyFamily(a : Set)
  : Set 1 {
  atFam(f : Family(a)) -> Set;
}

codef AtPoint(a : Set, x : a)
  : FamilyFamily(a) {
  atFam(f) { f.at(x) };
}
```

Outline


Dependent
Language
(MLTT)
Large Elimination
extraction
adds type casts
Non-Dependent Target
(F_ω + fixed points + type classes)
sophisticated
translation
Extractly
No Large Elimination
erasure

Use Indexed Data Types for Large Elimination

Invalid Large Elimination

```
def vec_prod(a : Set, n : Nat)
  -> Set {
  match n {
    Zero() => Unit;
    Succ(k) =>
      Pair(a, vec_prod(a, k));
  }
}
```

Valid Indexed Data Type

```
data Vect(a : Set)
  : Nat -> Set {

  Nil() : Vect(a, Zero);
  Cons(x : a, xs : Vect(a, n))
    : Vect(a, Succ(n));

}
```

Interpretation of STLC in Agda

```
-- Agda
data Ty (X : Set) : Set where
  base  : X           → Ty X
  arrow : Ty X → Ty X → Ty X

Interp : (P : X → Set) → (A : Ty X) → Set
Interp P (base x)     = P x
Interp P (arrow A B) = Interp P A → Interp P B
```

Interpretation of STLC in Extractly

```
codata Family(a : Set) : Set 1 {at(x : a) -> Set;}

data Interp(P : Family(X)) : (A : Ty(X)) {
  Base(prf : P.at(x))
    : Interp(P, Base(x));
  Arrow(preserves : Function(Interp(P, A), Interp(P, B)))
    : Interp(P, Arrow(A, B));
}
```

Limitations

Administrative Singleton Matches

```
def apply(fun : Interp(P, Arrow(A, B)), arg : Interp(P, A))
  -> Interp(P, B) {
  match fun { Arrow(pres) => pres.apply(arg); }
}
```

Type Indices Must Be Small

```
-- Agda
HVec : List Set → Set
HVec []         = Unit
HVec (A :: As) = Pair A (HVec As)
```

Potential Extensions

Atomic Versus Complex Values [Downen 2024]

- Split values into *atomic* and *complex*
- Atomic values are primitives and pointers
- Complex values must be pattern-matched on

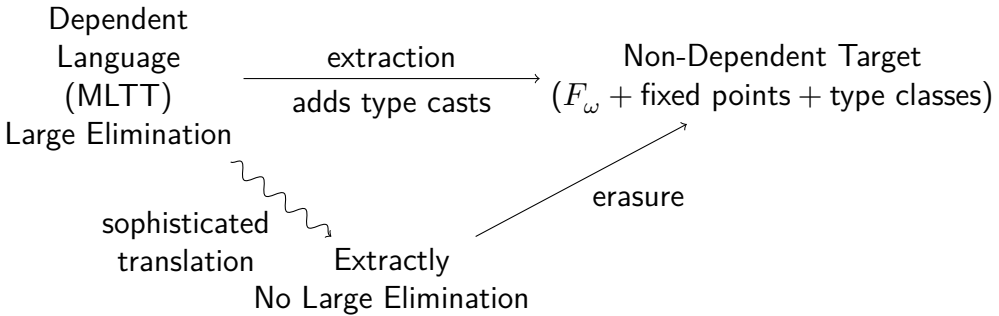
Effect System [Pédrot and Tabareau 2019]

```
def hello_world() -> Unit with (print) {
  x <- print("Hello world");
  return x;
}
```

Consistency

- Polarity has no termination checker

Outline



Looking for Job

Email greg.brown01@ed.ac.uk

Website www.yellowsquid.uk

GitHub yellowsquid