# 1 Intro

I need to prove that haskell types and terms that I expose wouldn't break the system. It means two things:

1. Types preserve the same set of invariants
2. Terms have the same interface: any combination of `APPLY` that can be used(ignoring types) to original term must be usable with generated one; and primitives(numbers, strings, ... and their ops) are the same.

# 2 My transformations

## 2.1 Kinds

$$KT[\![Kind]\!] = HaskellKind$$

$$\begin{aligned} KT[\![Set_0]\!] &= * \\ KT[\![Kind_1 \to Kind_2]\!] &= KT[\![Kind_1]\!] \to KT[\![Kind_2]\!] \end{aligned}$$

## 2.2 Type declarations

$DTMA$ gives MAlonzo generated type name.
$DT,\ DTD$ are defined when {`-# EXPORT` $AgdaTypeName\ HaskellTypeName$ `#-`} is specified.

$$\begin{aligned} DTMA[\![AgdaTypeName]\!] &= HaskellTypeName \\ DT[\![AgdaTypeName]\!] &= HaskellTypeName \\ DTD[\![AgdaTypeName]\!] &\doteq HaskellTypeDeclaration \end{aligned}$$

Considering declaration:

**data** $AgdaDataType\ (A_1 : Kind_1) \cdots (A_n : Kind_n) : Kind_{n+1} \to \cdots \to Kind_m \to Set$ **where** $\ldots$

$$\begin{aligned} &DTD[\![AgdaDataType]\!] \doteq \\ &\quad \textbf{newtype}\ DT[\![AgdaDataType]\!]\ (a_0 :: KT[\![Kind_1]\!]) \cdots (a_m :: KT[\![Kind_m]\!]) \\ &\qquad = DT[\![AgdaRecordType]\!]\ (\forall b_0 \cdots b_k.\ DTMA[\![AgdaDataType]\!]\ b_0\ \cdots\ b_k) \end{aligned}$$

$k$ is an arity of type constructor generated by MAlonzo.
It also works for **record**s.

## 2.3 Types

$$\begin{aligned} TT[\![AgdaType]\!](Context) &= HaskellType \\ Context &= \{AgdaTypeVarName \mapsto HaskellTypeVarName\} \end{aligned}$$

$$\begin{aligned} TT[\![A\ args\ldots]\!](\Gamma) &= a\ TT[\![args\ldots]\!](\Gamma), \quad (A \mapsto a) \in \Gamma \\ TT[\![CT\ args\ldots]\!](\Gamma) &= CT\ TT[\![args\ldots]\!](\Gamma), \quad CT \text{ is a } \texttt{COMPILED\_TYPE} \text{ or a primitive postulate} \\ TT[\![ET\ args\ldots]\!](\Gamma) &= DT[\![ET]\!]\ TT[\![args\ldots]\!](\Gamma) \\ TT[\![(A : Kind) \to T]\!](\Gamma) &= \forall (a :: KT[\![Kind]\!]).\ TT[\![T]\!](\Gamma \cup \{A \mapsto a\}) \\ TT[\![(x : T_1) \to T_2]\!](\Gamma) &= TT[\![T_1]\!](\Gamma) \to TT[\![T_2]\!](\Gamma), \quad x \notin freevars(T_2) \\ TT[\![(x : T_1,\ T_2)]\!](\Gamma) &= (TT[\![T_1]\!](\Gamma),\ TT[\![T_2]\!](\Gamma)), \quad x \notin freevars(T_2) \end{aligned}$$

### 2.4 Terms

*Wrap*, *Unwrap* are defined only when $TT[\![AgdaType]\!](\varnothing)$ is defined.

$$
\begin{aligned}
Wrap[\![AgdaType]\!](MAlonzoTerm) &= MyTerm \\
Unwrap[\![AgdaType]\!](MyTerm) &= MAlonzoTerm
\end{aligned}
$$

$$
\begin{aligned}
Wrap[\![A\ args\ldots]\!](term) &= \texttt{unsafeCoerce}\ term \\
Wrap[\![(A : Kind) \to T]\!](term) &= Wrap[\![T]\!](term\ ()) \\
Wrap[\![(x : T_1) \to T_2]\!](term) &= \lambda x.\ Wrap[\![T_2]\!](term\ Unwrap[\![T_1]\!](x)) \\
Wrap[\![(x : T_1,\ T_2)]\!]((term_1,\ term_2)) &= (Wrap[\![T_1]\!](term_1),\ Wrap[\![T_2]\!](term_2))
\end{aligned}
$$

$$
\begin{aligned}
Unwrap[\![A\ args\ldots]\!](term) &= \texttt{unsafeCoerce}\ term \\
Unwrap[\![(A : Kind) \to T]\!](term) &= Unwrap[\![T]\!](\lambda\_.\ term) \\
Unwrap[\![(x : T_1) \to T_2]\!](term) &= \lambda x.\ Unwrap[\![T_2]\!](term\ Wrap[\![T_1]\!](x)) \\
Unwrap[\![(x : T_1,\ T_2)]\!]((term_1,\ term_2)) &= (Unwrap[\![T_1]\!](term_1),\ Unwrap[\![T_2]\!](term_2))
\end{aligned}
$$

### 2.5 Value declarations

$VTMA$ gives MAlonzo generated value name
$VT$, $VTD$ are defined when {`-# EXPORT` $AgdaName\ HaskellName$ `#-`} is specified.

$$
\begin{aligned}
VTMA[\![AgdaName]\!] &= HaskellName \\
VT[\![AgdaName]\!] &= HaskellName \\
VTD[\![AgdaName]\!] &\doteq HaskellDeclaration
\end{aligned}
$$

Considering declaration:

$$
\begin{aligned}
&AgdaName : AgdaType \\
&AgdaName = \ldots
\end{aligned}
$$

$$
\begin{aligned}
&VTD[\![AgdaName]\!] \doteq \\
&\quad VT[\![AgdaName]\!] :: TT[\![AgdaType]\!](\varnothing) \\
&\quad VT[\![AgdaName]\!] = Wrap[\![AgdaType]\!](VTMA[\![AgdaName]\!])
\end{aligned}
$$

It works in the same way for constructors. It also works seamlessly with parametrized modules and, consequently, with record functions.

## 3 Preserving type invariants

Two things to watch for:

- `newtype` wrappers in the first case
- The third case

Every other case is exactly the same.
TODO:

# 4 Preserving term interface

`unsafeCoerce` is legal because it's either:

- The same term(when its type is a type variable)
- A newtype around MAlonzo generated type
- A primitive