Implicit Type Conversions

Francisco Gabriel
Mentor:
Gert-Jan Bottu
Supervisor:
Tom Schrijvers

In this presentation

- What are Implicit Type Conversions (ITC)?
- ITC out there
- What do we want?
- Immediate concerns
- TrIC
- The approach taken
- In-depth ideas

In this presentation

- **What are Implicit Type Conversions (ITC)?**
- ITC out there
- What do we want?
- Immediate concerns
- TrIC
- The approach taken
- In-depth ideas

The idea

Base Types:

In this presentation

- What are Implicit Type Conversions (ITC)?
- **ITC out there**
- What do we want?
- Immediate concerns
- TrIC
- The approach taken
- In-depth ideas

ITC in Scala

```
implicit def conversion (s:String):Int = 3
def function (i:Int):String = "Woohoo!"
println(function("Oops"))
```

RESULT

```
Woohoo!
```

Polymorphic and constrained ITCs

```
implicit def conversion [A] (s:List[A]):A = a.head
```

```
implicit def conv (y:List[Int])(implicit cond:(Int => Char)):Char= cond(y.head)
```

```scala
implicit def conversion1 (s:String):Char = 'c'
implicit def conversion2 (c:Char): Int = 3
def function (i:Int):String = "Woohoo!"
println(function("Oops"))
```

ERROR

```
ScalaFiddle.scala:6: error: type mismatch;
 found   : lang.this.String("Oops")
 required: scala.this.Int
   println(function("Oops"))
                    ^
```

```scala
implicit def conversion (c:Char): Int = 3
def function (c:Char):String =
c match {
  case 3   => "Int 3"
  case 97  => "Int 97"
  case 'a' => "Char 'a'"
  case _   => "Other"
}
println(function('a'))
```

RESULT

```
Int 97
```

In this presentation

- What are Implicit Type Conversions (ITC)?
- ITC out there
- **What do we want?**
- Immediate concerns
- TrIC
- The approach taken
- In-depth ideas

This are the features we want to support

- Transparent extension (including type inference & type classes)
- Polymorphism
- Constrained Conversions
- User-friendliness
- Transitivity
- Local Scoping

Transitivity

$\sim >$

$\sim >$

Local Scoping

```
happiness_calc :: EUR -> City -> Happiness
happiness_calc amount city = case city of
                               Geneva -> ...
                               Zurich -> ...
```

In this presentation

- What are Implicit Type Conversions (ITC)?
- ITC out there
- What do we want?
- **Immediate concerns**
  - **Ambiguity**
- TrIC
- The approach taken
- In-depth ideas

Ambiguity #1

$\rightsquigarrow ct$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

Implicit Environment

Ambiguity #2

$\rightsquigarrow ct$

$\rightsquigarrow ct$?

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

Implicit Environment

Attitude towards ambiguity

If there is no obviously better way to fix the program, don't.

The dominator

$\searrow ct$

$\rightsquigarrow ct$?

$\nearrow ct$

The dominator

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

$\rightsquigarrow ax$

In this presentation

- What are Implicit Type Conversions (ITC)?
- ITC out there
- What do we want?
- Immediate concerns
- **TrIC**
- The approach taken
- In-depth ideas

TrIC - syntax

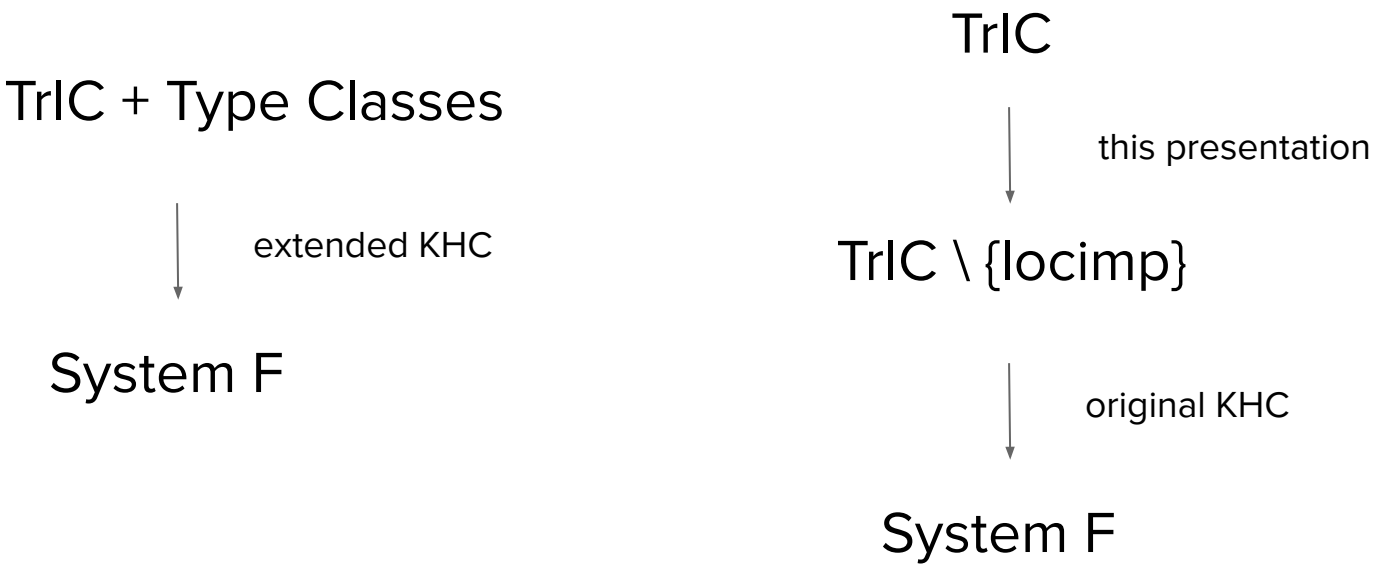
$$e ::=$$

```
data EUR = K_eur Float
data CHF = K_chf Float
data DKK = K_dkk Float
data USD = K_usd Float
data Wallet (a :: *) = K_wallet a a

(locimp \x. case x of K_dkk y -> K_eur (0.13*y) : DKK ~> EUR in
(locimp \x. case x of K_chf y -> K_eur (0.92*y) : CHF ~> EUR in
(locimp \x. case x of K_eur y -> K_usd (1.1*y)  : EUR ~> USD in

(K_wallet (K_chf 5.25)
          (locimp \x.Pi : Int ~> Float in (K_dkk 1)))
)))
```

In this presentation

- What are Implicit Type Conversions (ITC)?
- ITC out there
- What do we want?
- Immediate concerns
- TrIC
- **The approach taken**
- In-depth ideas

KHC-dialect = TrIC \ {locimp} + Type Classes



TrIC

Translation

TrIC \ {locimp}

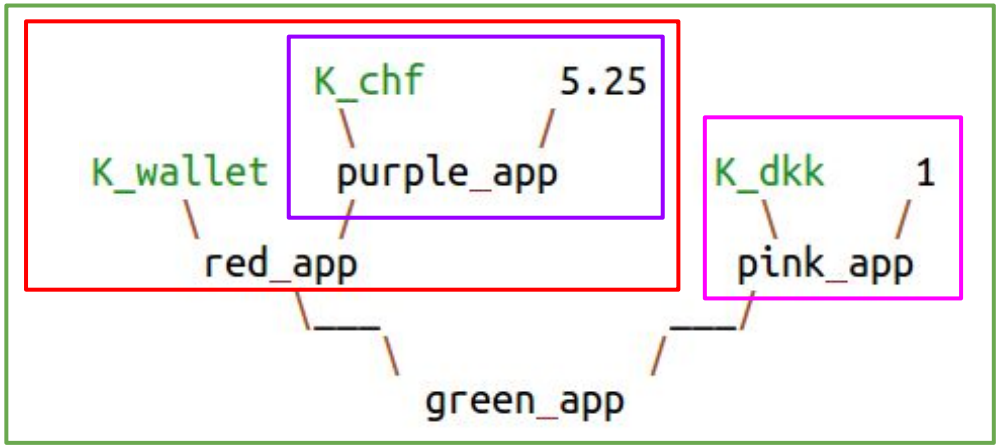
Constraint collection

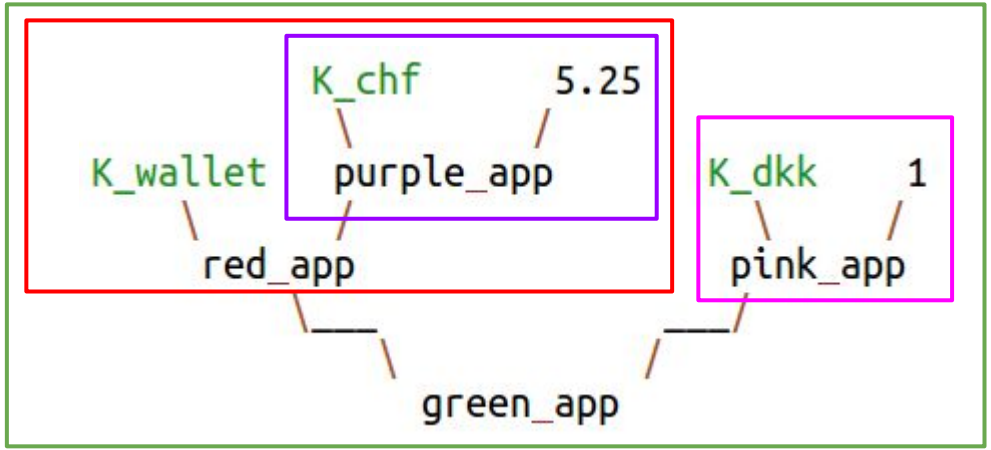
Constraint solving & insertion of converting expressions

$$\frac{tm1 : X \qquad tm2 : Y}{(tm1\ tm2) : idk\ ;\ X \sim (Y \to idk)}$$

$$\frac{tm1 : X \qquad tm2 : Y}{(tm1\ tm2) : idk;\ X \sim (arg \to idk);\ Y \rightsquigarrow arg}$$

```
data EUR = K_eur Float
data CHF = K_chf Float
data DKK = K_dkk Float
data USD = K_usd Float
data Wallet (a :: *) = K_wallet a a

(locimp \x. case x of K_dkk y -> K_eur (0.13*y) : DKK ~> EUR in
(locimp \x. case x of K_chf y -> K_eur (0.92*y) : CHF ~> EUR in
(locimp \x. case x of K_eur y -> K_usd (1.1*y)  : EUR ~> USD in

(K_wallet (K_chf 5.25)
          (locimp \x.Pi : Int ~> Float in (K_dkk 1)))
)))
```





Equality constraints

Conversion constraints

Equality constraints

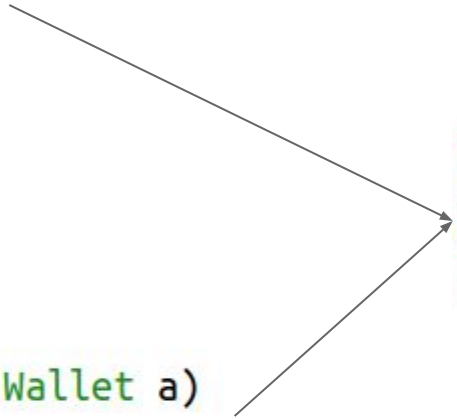
```
(Float -> CHF       ) ~ (ty_conv_from_Float     -> ty_purple)
(a -> a -> Wallet a) ~ (ty_conv_from_ty_purple -> ty_red   )
(Float -> DKK       ) ~ (ty_conv_from_Int       -> ty_pink  )
ty_red                ~ (ty_conv_from_ty_pink   -> ty_green )
```



```
ty_conv_from_Float     |-> Float
ty_purple              |-> CHF
ty_conv_from_ty_purple |-> a
ty_red                 |-> (a -> Wallet a)
ty_conv_from_Int       |-> Float
ty_pink                |-> DKK
ty_conv_from_ty_pink   |-> a
ty_green               |-> Wallet a
```

```
Float     ~> ty_conv_from_Float
ty_puple  ~> ty_conv_from_ty_purple
Int       ~> ty_conv_from_Int
ty_pink   ~> ty_conv_from_ty_pink
```

```
ty_conv_from_Float      |-> Float
ty_purple               |-> CHF
ty_conv_from_ty_purple  |-> a
ty_red                  |-> (a -> Wallet a)
ty_conv_from_Int        |-> Float
ty_pink                 |-> DKK
ty_conv_from_ty_pink    |-> a
ty_green                |-> Wallet a
```



```
Float ~> Float
CHF   ~> a
Int   ~> Float
DKK   ~> a
```

```
Float ~> Float
CHF   ~> a
Int   ~> Float
DKK   ~> a
```

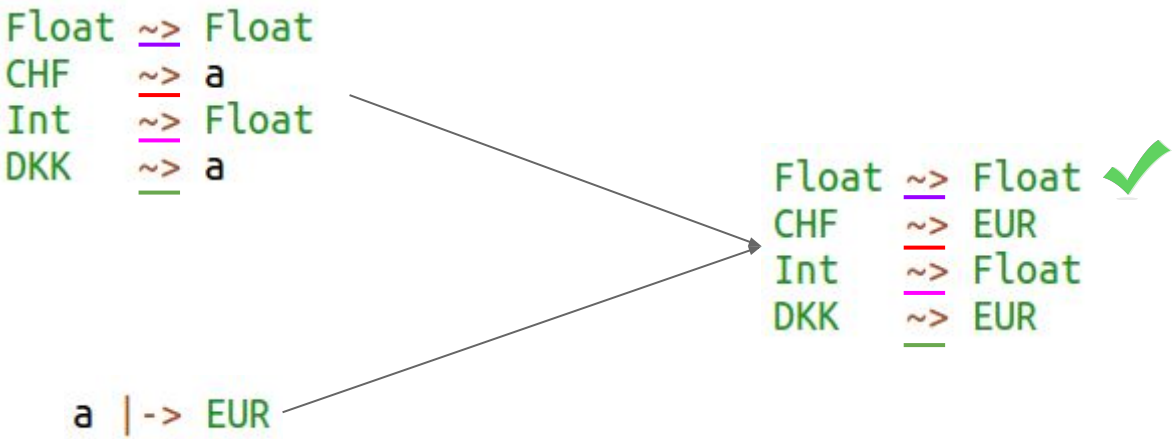
Compute the dominator of the conversion constraints involving each type variable

```
DKK ~> a <~ CHF
```

Implicit environment associated with both ~> and ~> :

```
DKK ~>
          EUR ~> USD
CHF ~>
```

```
a |-> EUR
```


Float ~> Float
CHF ~> a
Int ~> Float
DKK ~> a
a |-> EUR
Float ~> Float
CHF ~> EUR
Int ~> Float
DKK ~> EUR

```
CHF ~> EUR
```

```
DKK ~>
        EUR ~> USD
CHF ~>
```

Implicit Environment

```
data EUR = K_eur Float
data CHF = K_chf Float
data DKK = K_dkk Float
data USD = K_usd Float
data Wallet (a :: *) = K_wallet a a

(locimp \x. case x of K_dkk y -> K_eur (0.13*y) : DKK ~> EUR in
(locimp \x. case x of K_chf y -> K_eur (0.92*y) : CHF ~> EUR in
(locimp \x. case x of K_eur y -> K_usd (1.1*y)  : EUR ~> USD in

(K_wallet (K_chf 5.25)
          (locimp \x.Pi : Int ~> Float in (K_dkk 1)))
)))
```

```
K_wallet (K_chf 5.25)
```



```
K_wallet ((\x. case x of K_chf y -> K_eur (0.92*y)) (K_chf 5.25))
```

```
data EUR = K_eur Float
data CHF = K_chf Float
data DKK = K_dkk Float
data USD = K_usd Float
data Wallet (a :: *) = K_wallet a a

(locimp \x. case x of K_dkk y -> K_eur (0.13*y) : DKK ~> EUR in
(locimp \x. case x of K_chf y -> K_eur (0.92*y) : CHF ~> EUR in
(locimp \x. case x of K_eur y -> K_usd (1.1*y)  : EUR ~> USD in

(K_wallet (K_chf 5.25)
          (locimp \x.Pi : Int ~> Float in (K_dkk 1)))
)))
```



```
data...

(K_wallet ((\x. case x of K_chf y -> K_eur (0.92*y)) (K_chf 5.25))
          ((\x. case x of K_dkk y -> K_eur (0.13*y)) (K_dkk ((\x.Pi) 1))))
```

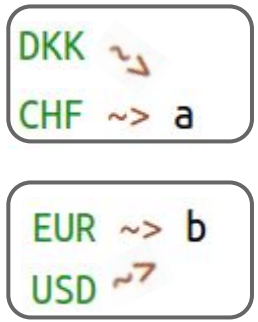
In this presentation

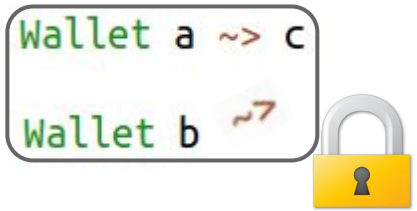
- What are Implicit Type Conversions (ITC)?
- ITC out there
- What do we want?
- Immediate concerns
- TrIC
- The approach taken
- **In-depth ideas**

Randomly choose a variable and compute the dominator?

```
data Wallet (a :: *) = K_wallet a a

(K_wallet
  (K_wallet (K_chf 5.25) (K_dkk 2.5))
  (K_wallet (K_eur 6.5) (K_usd 0.25)))
```

```
CHF ~> a
DKK ~> a
EUR ~> b
USD ~> b
Wallet a ~> c
Wallet b ~> c
```



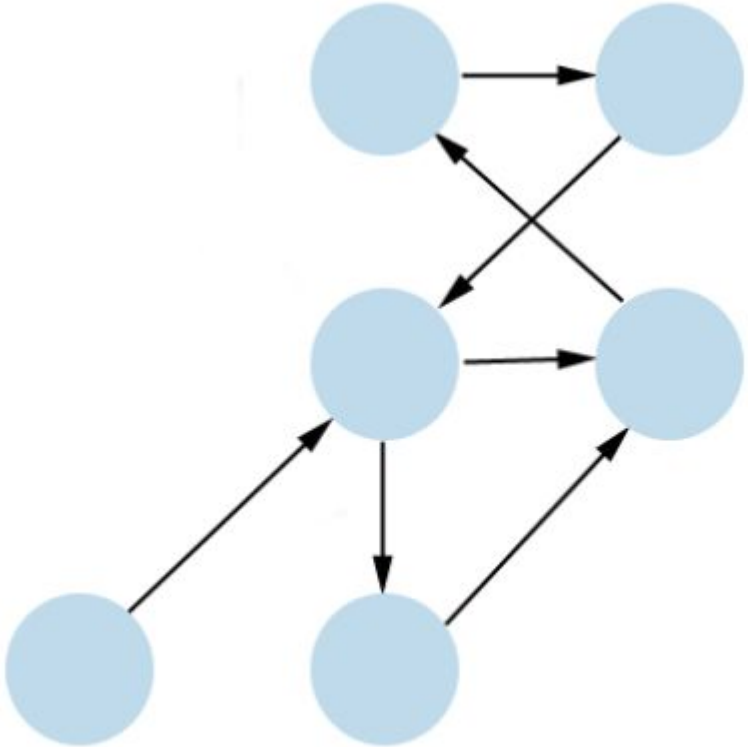


When is an axiom applicable?

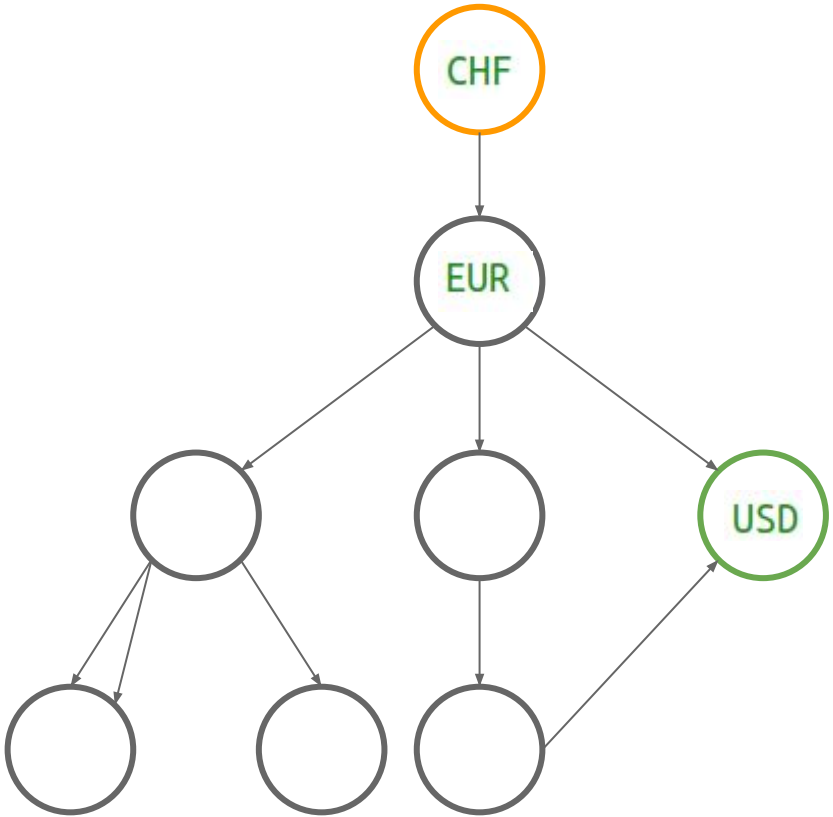
```
                                                  USD ~> EUR  ~> DKK
          Wallet a ~> Wallet EUR                    ⇝   ⇜
  j : a ~> b    => Wallet a ~> Wallet b                 CHF
j : a ~> EUR => Wallet a ~> Wallet EUR
```

Implicit Environment

Paths



Constructing Paths




```
\x.x

\x. exp

eur2usd : EUR ~> USD

\x. (eur2usd exp)
```

Type Classes

```
data DKK = K_dkk Float
data CHF = K_chf Float
data EUR = K_eur Float
data Wallet (a :: *) = K_wallet a a

class Total a :: * where
total :: a -> EUR

instance Total (Wallet EUR) where
total = \x. case x of K_wallet (K_eur x1) (K_eur x2) -> K_eur (x1+x2)

(locimp \x. case x of K_dkk y -> K_eur (0.13*y) : DKK ~> EUR in
total (locimp \x.case x of K_chf y -> K_eur (0.92*y) : CHF ~> EUR in K_wallet (K_dkk 6.3) (K_chf 3.14)))
```


Thanks for listening!