A typechecker plugin for units of measure:
domain-specific constraint solving in GHC Haskell

Adam Gundry

Workshop on Type Inference and Automated Proving
12th May 2015

EPSRC
Engineering and Physical Sciences
Research Council

Well-Typed
The Haskell Consultants

Goals

- Type inference for units of measure...
  - ...in (GHC-flavoured) Haskell
  - ...without modifying GHC

- Domain-specific constraint solving

Demo

EPSRC
Engineering and Physical Sciences
Research Council

Well-Typed
The Haskell Consultants

Units, kind of

Data kind Unit for units of measure

```
kind Unit   -- well, currently data Unit
```

Base and derived units as type families

```
One  :: Unit
Base :: Symbol → Unit
(⊛)  :: Unit → Unit → Unit
(⊘)  :: Unit → Unit → Unit
```

Type inference modulo

Equational theory: free abelian group

```
Base “m” ⊛ One ~ Base “m”
(One ⊘ Base “s”) ⊛ Base “s” ~ One
u ⊛ v ~ v ⊛ u
```

Non-structural unification

```
 u ⊛ v ~ v ⊛ u      doesn't imply      u ~ v
```

A quantity of units

Phantom newtype for values with units

```
newtype Quantity a (u :: Unit) = MkQ a
```

Unit safe arithmetic operators on quantities

```
(⊛) :: Num a
     ⇒ Quantity a (u ⊘ v)
     → Quantity a v
     → Quantity a u
MkQ x ⊛ MkQ y = MkQ (x * y)
```

Toil and trouble

```
τ ~ Quantity α v
Quantity β One ~ Quantity α (u ⊘ v)

    double :: Num α ⇒ τ → Quantity α u
```

```
x   :: τ
2   :: Quantity β One
(⊛) :: Num α ⇒ Quantity α (u ⊘ v)
     → Quantity α v
     → Quantity α u
```

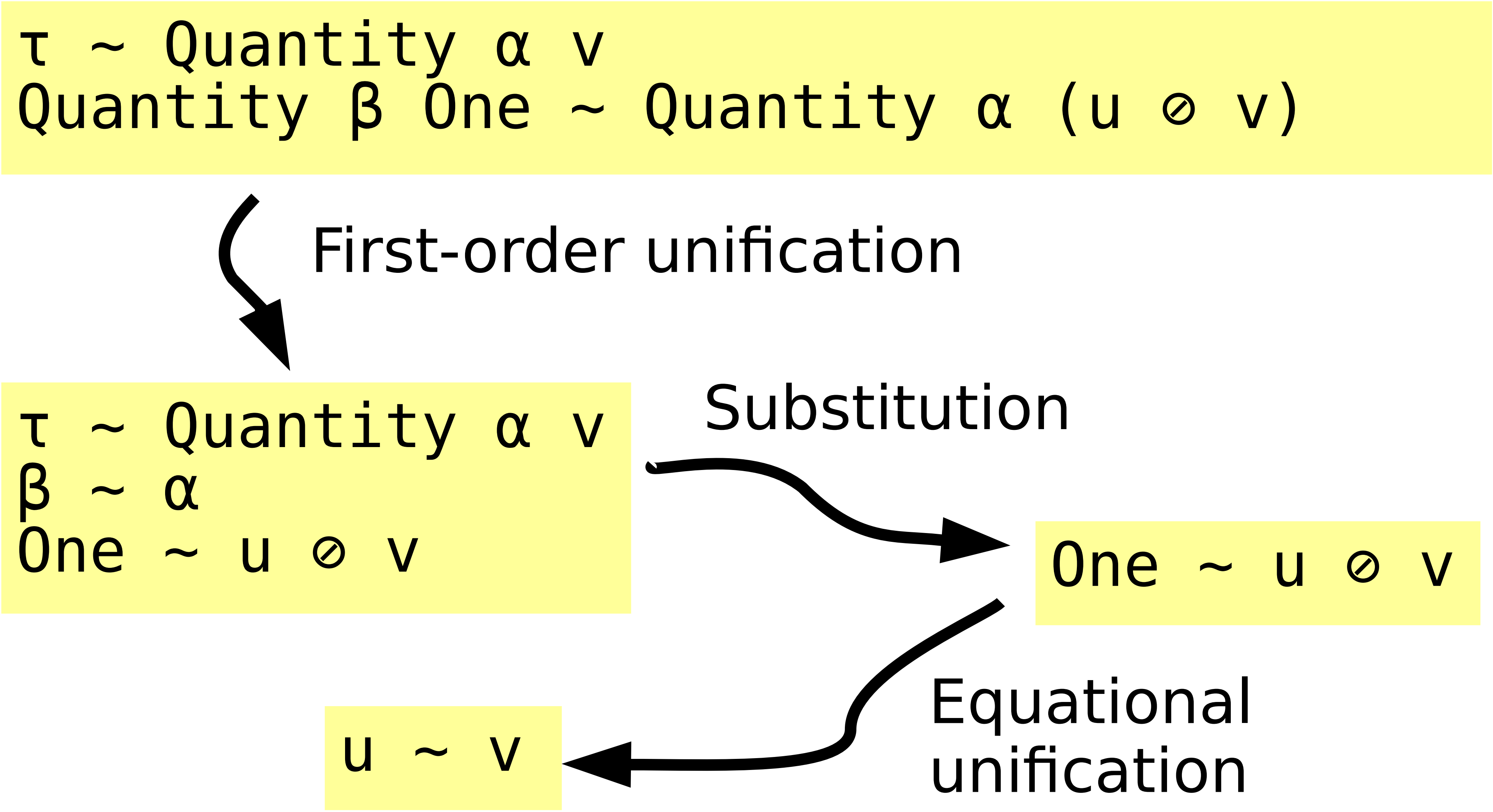
```
double x = 2 ⊛ x
```

EPSRC
Engineering and Physical Sciences
Research Council

http://adam.gundry.co.uk/pub/typechecker-plugins/

Well-Typed
The Haskell Consultants

Toil and trouble



```
τ ~ Quantity α v
Quantity β One ~ Quantity α (u ⊘ v)
```

First-order unification

```
τ ~ Quantity α v
β ~ α
One ~ u ⊘ v
```

Substitution

```
One ~ u ⊘ v
```

Equational unification

```
u ~ v
```

EPSRC
Engineering and Physical Sciences
Research Council

http://adam.gundry.co.uk/pub/typechecker-plugins/

Well-Typed
The Haskell Consultants

Toil and trouble

```
double :: Num α
        ⇒ Quantity α u → Quantity α u
double x = 2 ⊛ x
```

Typechecker plugins

- Extend GHC's constraint solver with user-defined behaviour
- May implement equational unification or other strategies for constraint solving
- Haskell code using the GHC API
- Dynamically loaded during compilation

EPSRC
Engineering and Physical Sciences Research Council

`http://adam.gundry.co.uk/pub/typechecker-plugins/`

Well-Typed
The Haskell Consultants

Defining a typechecker plugin

- Provided with all constraints in scope:
  - Given: facts known to typechecker
  - Wanted: constraints that need to be solved
- May:
  - Identify constraints as inconsistent
  - Solve existing constraints
  - Generate new constraints

Constraint solving with plugins

1. Run main constraint solver
2. Feed leftover constraints into plugin
3. If plugin found an error, stop
4. If plugin solved some constraints, remove them from consideration
5. If plugin yielded new constraints, GOTO 1

How the units plugin works

- Look for equality constraints of kind `Unit`
- Convert both sides to normal form
- Simplify via free abelian group unification
- Might get stuck due to tricky cases
  - Universally quantified variables
  - Type families
  - Local given constraints

Example

- Given: `F (m ⊘ s) ~ ()`
  Wanted: `F α ~ (), α ⊛ s ~ m`
- Plugin adds wanted: `α ~ m ⊘ s`
- GHC solves by substitution, leaving
  `(m ⊘ s) ⊛ s ~ m`
- Plugin runs again to discharge constraint

Tricky case: Type families

- `u ⊛ v ~ F u` where F is user-defined
- Key idea: dynamic unification
- If we get stuck, try to solve other constraints
- Perhaps u will become more defined

Tricky case: Given constraints

- Arise from type signatures, GADT matches
- Given $\mathtt{u}^2 \sim \mathtt{v}^3$, solve $\mathtt{u} \sim \alpha^3$
  where u and v are universally quantified
- Key idea: unify given constraints first
- $\mathtt{u}^2 \sim \mathtt{v}^3$ is equivalent to $\mathtt{u} \sim \mathtt{w}^3,\ \mathtt{v} \sim \mathtt{w}^2$
  for some fresh variable w
- Hence $\alpha \sim \mathtt{w} \sim \mathtt{u} \oslash \mathtt{v}$

Why trust a plugin?

- Plugins can generate evidence for the constraints they solve
- Expressed in System F_C
- Proof-irrelevance means plugins can use arbitrary equality axioms
- Type soundness depends on consistency of those axioms

What plugins can't do

- Add new syntax
  - (but there's always quasiquotes)
- Create entirely new types
- Change the existing typing rules
  - (plugins called only after GHC's main solver)
- Alter presentation of inferred types

The future

- Generate evidence in the plugin, based on the free abelian group axioms
- Consistency in the presence of non-terminating type families
- Termination of the constraint solver?

More plugins

- ghc-typelits-natnormalise (Christian Baaij)
  - sum-of-products normalisation for numeric expressions
  - used for CλaSH hardware description language
  - https://github.com/christiaanb/ghc-typelits-natnormalise

- type-nat-solver (Iavor Diatchki)
  - solves numeric constraints using an SMT solver
  - https://github.com/yav/type-nat-solver

Thank you

EPSRC

Engineering and Physical Sciences Research Council

Well-Typed

The Haskell Consultants