Utrecht University

Faculty of Science
Department of Information and Computing Sciences
www.cs.uu.nl

Domain-Specific Type Error Diagnosis

Alejandro Serrano

Domain-Specific Languages (DSLs)

Advantages

- Domain experts can use them with little prior knowledge of programming
- Communication between developers, experts and clients becomes more fluid

External DSLs

- New language and compiler is built from scratch
- Complete control, but may reinvent the wheel
- SQL, HTML, and LaTeX are some examples of external DSLs

Internal or Embedded DSLs

- Integrate the language as a
- Less overhead associated to the
- Several DSLs can be combined in the same app
- Very common in the **functional programming** world (Haskell, Scala, F#, ML)

Domain-Specific Type Errors

```
> atop :: Diagram v -> Diagram v -> Diagram v

> atop circle sphere
```

```
-- Puts two diagrams next to each other
-- The vector spaces have to coincide!!
-- Circle is 2-D, sphere is 3-D
```

What we have

```
Couldn't match type 'V2' with type 'V3'
```

What we would prefer to have

```
Vector spaces do not coincide: 2-D vs. 3-D
```

- Embedded DSLs in Haskell use type-level techniques
- Type errors become more and more complicated
- Domain-aware errors reduce the learning steep

Domain-Specific Type Rules

```
atop :: ( d1 ~ Diagram v1 error "First argument is not a diagram, but a $d1"
        , d2 ~ Diagram v2 error "Second argument is not a diagram, but a $d2"
        , v1 ~ v2 error "Vector spaces do not coincide: $v1 vs. $v2")
     => d1 -> d2 -> d1
```

- Integrate error reporting as part of types
- Related work in this direction
  - Heeren and Hage with Helium
  - Wazny with Chameleon

Research Questions

Abstraction

How to prevent duplication describing errors?

```
atop :: Diagram v -> Diagram v -> Diagram v
vtop :: Diagram v -> Diagram v -> Diagram v
```

Those two functions need the exact same errors

- We do not want to write them twice!

Context-awareness

How to specify when a type rule should kick in?

- The same function used in different contexts

```
> atop circle sphere     -- Two arguments!
> map (atop circle) ls   -- Partially applied
```

- The same function used with different types

```
> fmap (+1) [1,2,3]      -- Used with lists
> fmap (+1) stateful     -- Used with state
```

Advanced type systems

Haskell libraries make use of advanced features

- Generalized Algebraic Data Types (GADTs)
- Type families and promoted data kinds
- Higher-rank and impredicative polymorphism

How to make the system independent of the underlying type system of the general language?

Our Solutions

Type-Level Programming of Errors

Use Haskell's powerful type system to abstract

- Type errors are described using constraints
- Type families can generate those constraints from other information, they are "constraint functions"

```
type TopErrors d1 d2 v1 v2
  = ( d1 ~ Diagram v1 error "First arg..."
    , -- rest of errors )

atop :: TopErrors d1 d2 v1 v2
     => d1 -> d2 -> Diagram v1
```

See *Type Error Customization for GHC* (IFL'17)

Conditional type rules

Enlarge the context of type rules

- Complex shape information using regular expr.
- Partial type information

```
rule fmap f xs | type(xs) ~ [e]
  :: ( type(f) ~ a -> b
       error "First arg. must be a fn."
     , type(xs) ~ [a]
       error "Element type must coincide")
  => [b]
```

See *Generic Matching of Tree Regular Expressions over Haskell Data Types* (PADL'16)

and *Type Error Diagnosis for Embedded DSLs by Two-Stage Specialized Type Rules* (ESOP'16)

Extensions to constraint-based inference

Constraint Handling Rules (CHRs) have been used in the past to describe type systems

- Make them aware of type variables
- Integrate with higher-order patterns
- Introduce nested quantification and implication

See *Constraint handling rules with binders, patterns and generic quantification* (ICLP'17)

Describe higher-rank and impredicative polymorphism using CHR-like formalism

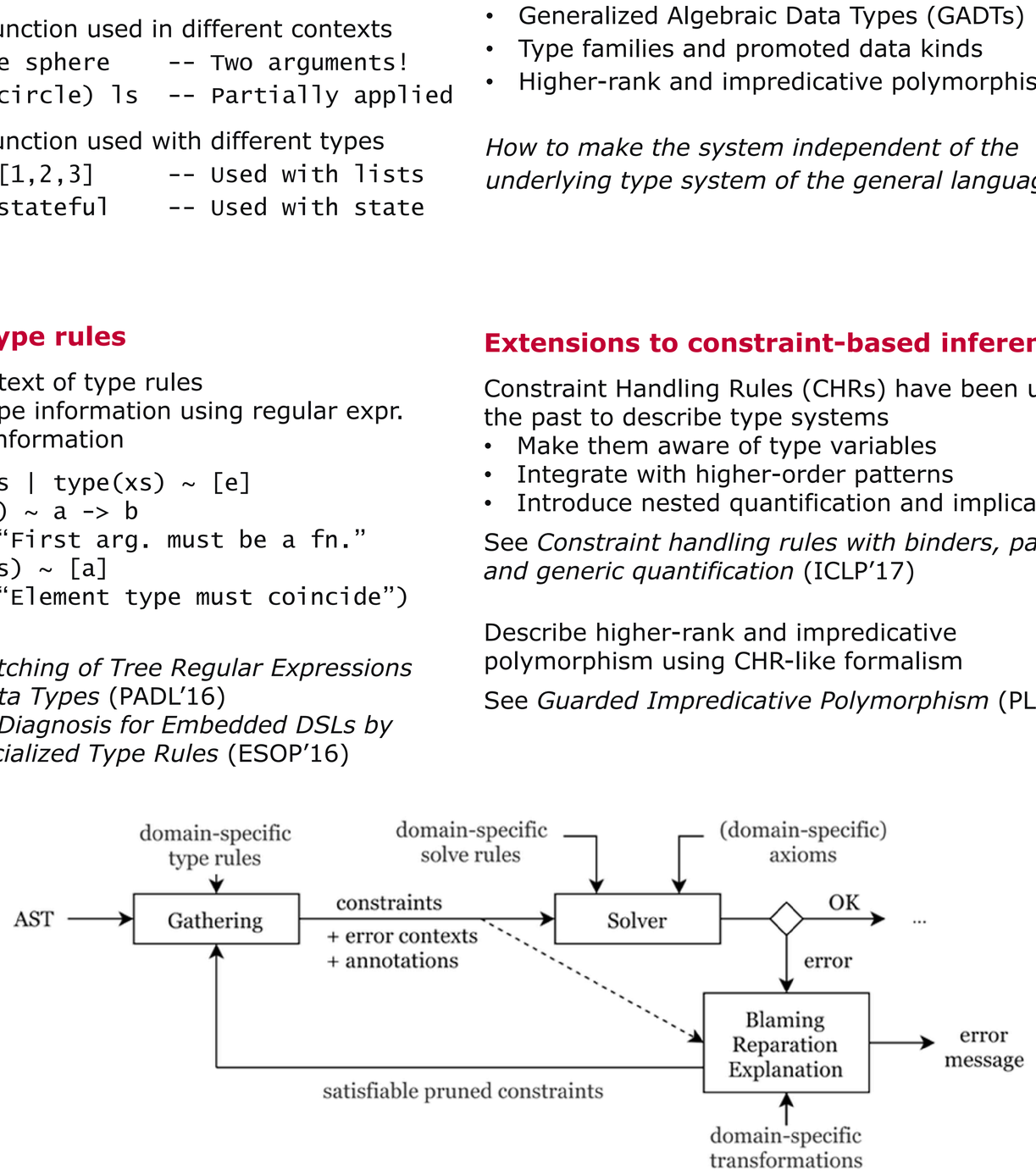
See *Guarded Impredicative Polymorphism* (PLDI'18)

Ensuring Soundness of the System

- Type rules modify the typing rules of the system
- Without any check, programs which were not accepted before could be accepted now!

Core idea: checking the soundness of type rules is similar to the problem of language extensions

See *Lightweight soundness for towers of language extensions* (PEPM'17)



This research was possible thanks to NWO project "Domain-Specific Type Error Diagnosis (DOMSTED)"