Markedly: a cartographic approach for mapping eDSL implementation costs

Karl Cronburg
karl@cs.tufts.edu

Authors: Matthew P. Ahrens, Karl Cronburg, Jeanne-Marie Musca
Siriusly PL Lab @ Tufts University
Preprint: cronburg.com/papers/markedly17.pdf

October 22, 2017

Background

Members of the PL lab at Tufts regularly implement eDSLs in Haskell:

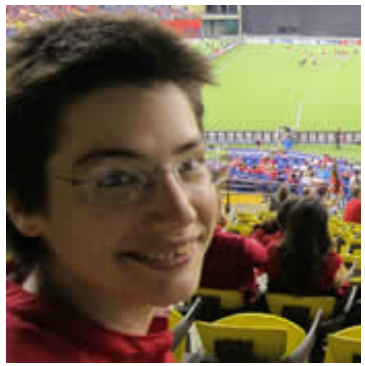
Karl Cronburg

ANTLR-like parsing eDSL

Matt Ahrens

Cyber-physical systems eDSL

Jeanne-Marie Musca



Multi-union data types (extensible records) eDSL

The problem

Embedded domain specific languages (eDSLs) are difficult to design, implement, and maintain to stand the test of time. EDSL creators want to:

- $\mathbb{P}$reserve . . . the correctness of $\boldsymbol{D}$ in $\boldsymbol{I}$
- $\mathbb{E}$xtend . . . $\boldsymbol{D}$ and $\boldsymbol{I}$ to support new features
- $\mathbb{A}$dhere . . . exclusively to $\boldsymbol{D}$ in $\boldsymbol{I}$
- $\mathbb{R}$ecognize . . . spans in $\boldsymbol{I}$ as features in $\boldsymbol{D}$

Design	$\boldsymbol{D}$	set of features, $f \in \boldsymbol{D}$
Implementation	$\boldsymbol{I}$	set of spans, $(s, e) \in \boldsymbol{I}$

Proposed solution

- Provide eDSL creators with a methodology to discuss where costs originate.
- Both the design ***D*** and implementation ***I*** of an eDSL contribute to implementation costs.

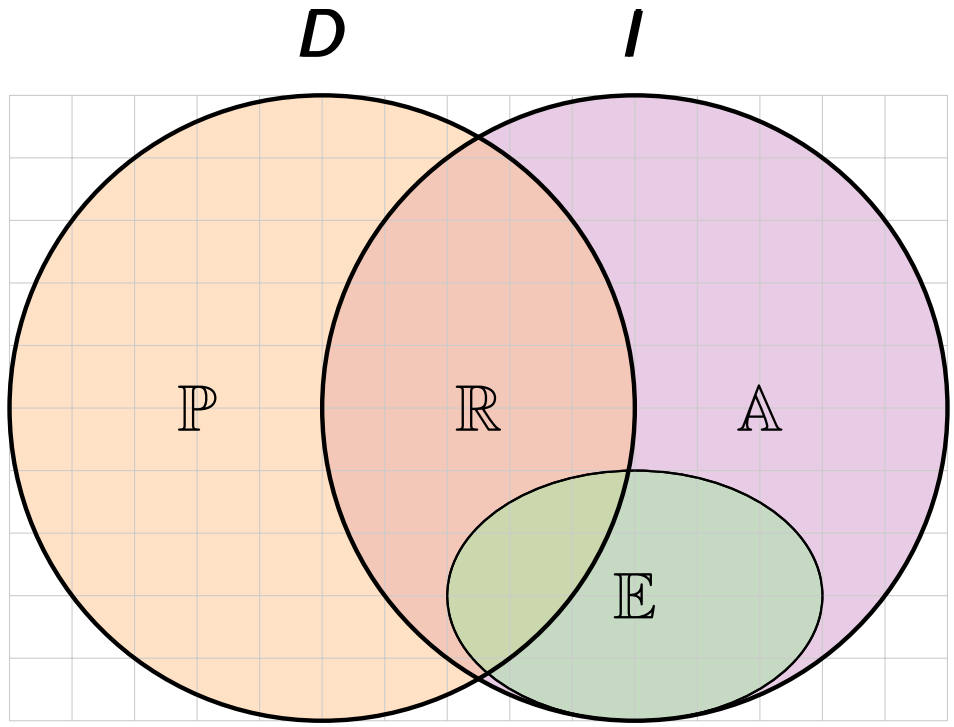


Figure: Sources of $\mathbb{PEAR}$ costs.

Type Level Natural Numbers (TLN)

0 Language creator Mark wants to create an eDSL in Haskell with the ability to annotate Haskell types with natural numbers.

1

Mark first writes down the desired features:

- human-readable natural number weights,
- to attach weights to arbitrary expressions,
- and to sum weights across function application.

The TLN implementation

2 Mark implements a prototype of TLN using GHC's† `DataKinds` and `TypeFamilies` metaprogramming facilities.

```
data Nat = Z | S Nat
type Weighted (w :: Nat) a = a
```

```
type family
  (w1 :: Nat) :+: (w2 :: Nat) :: Nat
```

```
type instance Z :+: m = m
type instance (S n) :+: m = S (n :+: m)
```

```
($$) :: Weighted w1 (a -> b)
     -> Weighted w2 a
     -> Weighted (w1 :+: w2) b
($$) f a = f a
```

† Glasgow Haskell Compiler

The TLN map

3 With Haskell code in hand (previous slide), Mark hand-draws a Markedly map depicting how meta-information† flows through the code.

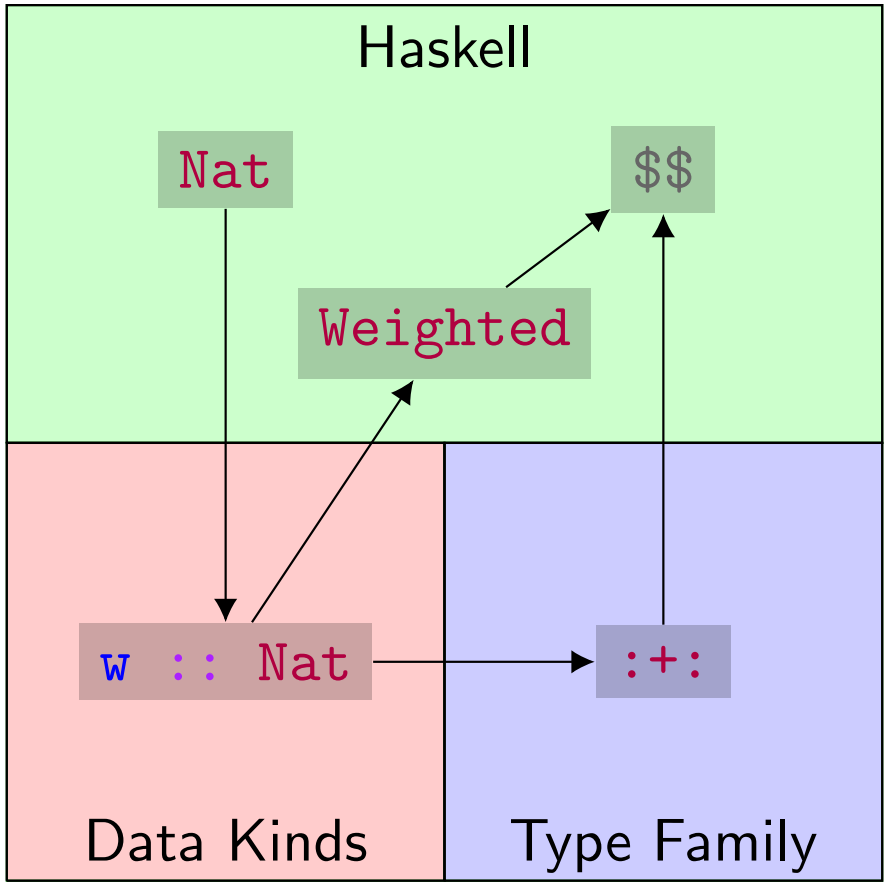


Figure: Haskell map for TLN.

TLN $\mathbb{PEAR}$ costs

4 Finally Mark computes $\mathbb{PEAR}$ costs according to the metrics listed above.

Element	$\mathbb{P}$	$\mathbb{E}$	$\mathbb{A}$	$\mathbb{R}$
`Nat`	1	·	·	·
`Weighted`	·	·	1	·
`w :: Nat`	·	·	·	1
`$$`	·	·	1	·
`Nat` ⟶ `w :: Nat`	·	·	·	1
`w :: Nat` ⟶ `Weighted`	·	·	·	1
`w :: Nat` ⟶ `:+:`	·	·	·	1
$\text{cost}(\boldsymbol{D}, \boldsymbol{I}) = \Sigma \langle \mathbb{P}, \mathbb{E}, \mathbb{A}, \mathbb{R} \rangle =$	1	·	2	4

Figure: Non-zero $\mathbb{PEAR}$ costs of TLN.

PEAR metrics

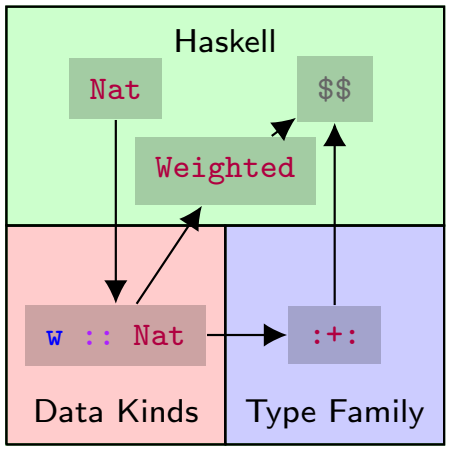
$\mathbb{P}$	How many features in ***D*** are compromised in ***I*** ?
$\mathbb{E}$	How many spans in ***I*** have inaccessible intermediate representations?
$\mathbb{A}$	How many spans in ***I*** are merely supporting annotations for a feature in ***D*** ?
$\mathbb{R}$	Is this source span in ***I*** present in ***D*** ?

PEAR metrics

$$
\begin{array}{c|l}
\mathbb{P} & \exists f \in \boldsymbol{D} \mid \forall (s,e) \in \boldsymbol{I}, \neg \mathrm{impl}\,(f,(s,e)) \\
\mathbb{E} & \exists (s,e) \in \boldsymbol{I} \mid \mathrm{hidden}(s,e) \\
\mathbb{A} & \exists (s,e) \in \boldsymbol{I}, f \in \boldsymbol{D} \mid \mathrm{impl}\,(f,(s,e)) \;\&\&\; \mathrm{annot}(s,e) \\
\mathbb{R} & \exists (s,e) \in \boldsymbol{I} \mid \forall f \in \boldsymbol{D}, \neg \mathrm{impl}(f,(s,e))
\end{array}
$$

Design	$\boldsymbol{D}$	set of features, $f \in \boldsymbol{D}$
Implementation	$\boldsymbol{I}$	set of spans, $(s,e) \in \boldsymbol{I}$
Source code span	(s,e)	start to end (characters)
$\mathrm{impl}(f,(s,e))$		Span (s,e) implements f
$\mathrm{annot}(s,e)$		Span (s,e) is an annotation
$\mathrm{hidden}(s,e)$		(s,e) representation is hidden

Walkthrough of computing $\mathbb{P}$



Element	$\mathbb{P}$	$\mathbb{E}$	$\mathbb{A}$	$\mathbb{R}$
Nat	1	·	·	·
Weighted	·	·	1	·
w :: Nat	·	·	·	1
$$	·	·	1	·
Nat ➤ w :: Nat	·	·	·	1
w :: Nat ➤ Weighted	·	·	·	1
w :: Nat ➤ :+:	·	·	·	1
$\Sigma \langle \mathbb{P}, \mathbb{E}, \mathbb{A}, \mathbb{R} \rangle$	=1	·	2	4

```haskell
data Nat = Z | S Nat
type Weighted (w :: Nat) a = a
```

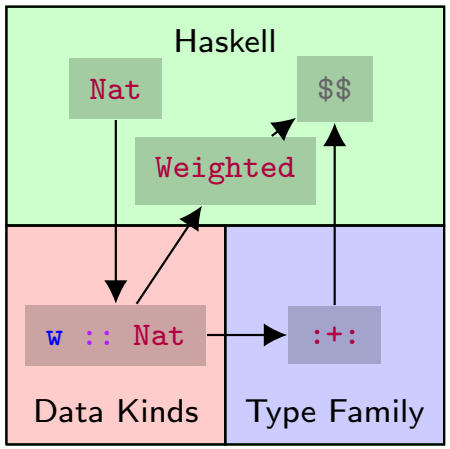
```haskell
type family
  (w1 :: Nat) :+: (w2 :: Nat) :: Nat
```

```haskell
type instance Z :+: m = m
type instance (S n) :+: m = S (n :+: m)
```

```haskell
($$) :: Weighted w1 (a -> b)
     -> Weighted w2 a
     -> Weighted (w1 :+: w2) b
($$) f a = f a
```

$\mathbb{P}$ = How many features in ***D*** are compromised in ***I*** ?

Walkthrough of computing $\mathbb{E}$



Element	$\mathbb{P}$	$\mathbb{E}$	$\mathbb{A}$	$\mathbb{R}$
`Nat`	1	·	·	·
`Weighted`	·	·	1	·
`w :: Nat`	·	·	·	1
`$$`	·	·	1	·
`Nat` → `w :: Nat`	·	·	·	1
`w :: Nat` → `Weighted`	·	·	·	1
`w :: Nat` → `:+:`	·	·	·	1
$\Sigma \langle \mathbb{P}, \mathbb{E}, \mathbb{A}, \mathbb{R} \rangle$	=1	·	2	4

```haskell
data Nat = Z | S Nat
type Weighted (w :: Nat) a = a
```

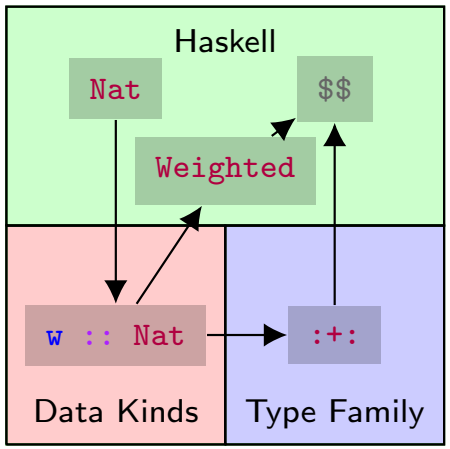
```haskell
type family
  (w1 :: Nat) :+: (w2 :: Nat) :: Nat
```

```haskell
type instance Z :+: m = m
type instance (S n) :+: m = S (n :+: m)
```

```haskell
($$) :: Weighted w1 (a -> b)
     -> Weighted w2 a
     -> Weighted (w1 :+: w2) b
($$) f a = f a
```

$\mathbb{E}$ = How many spans in ***I*** have inaccessible intermediate representations?

Walkthrough of computing $\mathbb{A}$



Element	$\mathbb{P}$	$\mathbb{E}$	$\mathbb{A}$	$\mathbb{R}$
`Nat`	1	·	·	·
`Weighted`	·	·	1	·
`w :: Nat`	·	·	·	1
`$$`	·	·	1	·
`Nat` ▶ `w :: Nat`	·	·	·	1
`w :: Nat` ▶ `Weighted`	·	·	·	1
`w :: Nat` ▶ `:+:`	·	·	·	1
$\Sigma \langle \mathbb{P}, \mathbb{E}, \mathbb{A}, \mathbb{R} \rangle$	=1	·	2	4

```
data Nat = Z | S Nat
type Weighted (w :: Nat) a = a
```

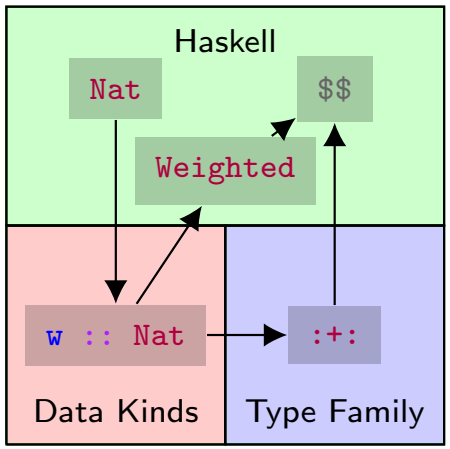
```
type family
  (w1 :: Nat) :+: (w2 :: Nat) :: Nat
```

```
type instance Z :+: m = m
type instance (S n) :+: m = S (n :+: m)
```

```
($$) :: Weighted w1 (a -> b)
     -> Weighted w2 a
     -> Weighted (w1 :+: w2) b
($$) f a = f a
```

$\mathbb{A}$ = How many spans in ***I*** are merely supporting annotations for a feature in ***D*** ?

Walkthrough of computing $\mathbb{R}$



Element	$\mathbb{P}$	$\mathbb{E}$	$\mathbb{A}$	$\mathbb{R}$
Nat	1	·	·	·
Weighted	·	·	1	·
w :: Nat	·	·	·	1
$$	·	·	1	·
Nat ▶ w :: Nat	·	·	·	1
w :: Nat ▶ Weighted	·	·	·	1
w :: Nat ▶ :+:	·	·	·	1
$\Sigma\ \langle \mathbb{P}, \mathbb{E}, \mathbb{A}, \mathbb{R} \rangle$	=1	·	2	4

```
data Nat = Z | S Nat
type Weighted (w :: Nat) a = a
```

```
type family
  (w1 :: Nat) :+: (w2 :: Nat) :: Nat
```

```
type instance Z :+: m = m
type instance (S n) :+: m = S (n :+: m)
```

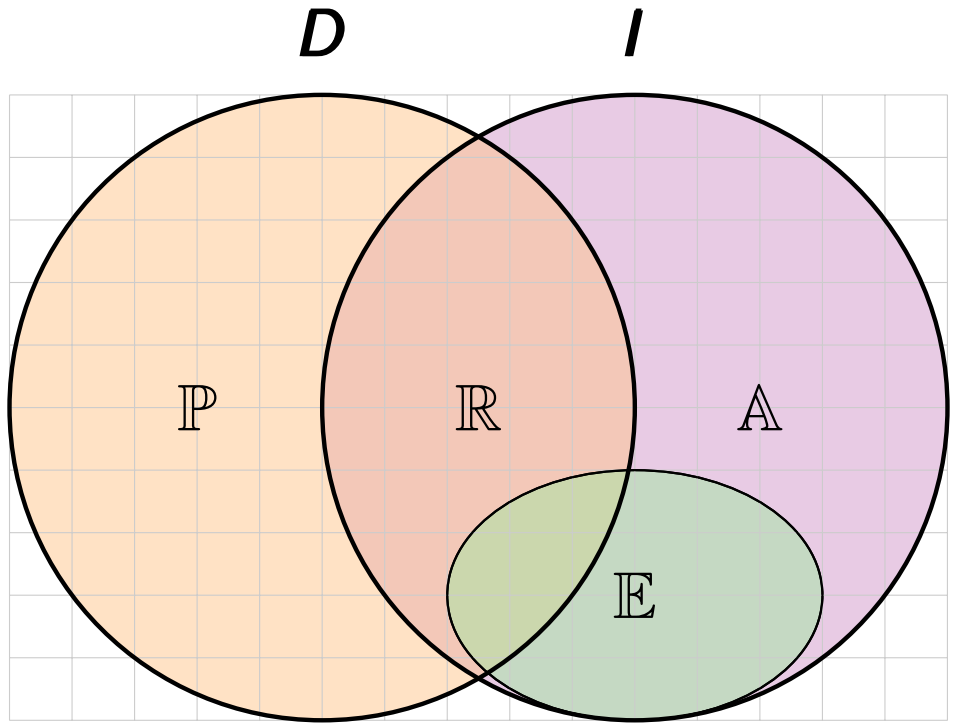
```
($$) :: Weighted w1 (a -> b)
     -> Weighted w2 a
     -> Weighted (w1 :+: w2) b
($$) f a = f a
```

$\mathbb{R}$ = Is this source span in ***I*** present in ***D*** ?

Future Work

- Markedly user study.
- Source span highlighting IDE support.
- Git integration.
- Extensible (multi-union) data types.

Conclusion



Web: `http://siriusly.cs.tufts.edu/`

Email: `karl@cs.tufts.edu`

Preprint: `cronburg.com/papers/markedly17.pdf`

Markedly is sponsored by the Defense Advanced Research Projects Agency (contract: FA8750-15-2-0033). Markedly does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

User Study

Programmers with functional programming experience will be tasked with implementing eDSLs over the course of a few months:

- Half of the programmers will be given a number of example Markedly maps.
- This half will be asked to recompute $\mathbb{PEAR}$ costs after substantial design and implementation progress.

Hypothesis: Markedly-enabled programmers will implement more design-preserving, extensible, adherent, and recognizable eDSLs.