Stream Fusion for Isabelle's Code generator

Andreas Lochbihler Alexandra Maximova

Institute of Information Security
ETH Zurich, Switzerland

Interactive Theorem Proving 2015

From formal specifications to implementations

specification

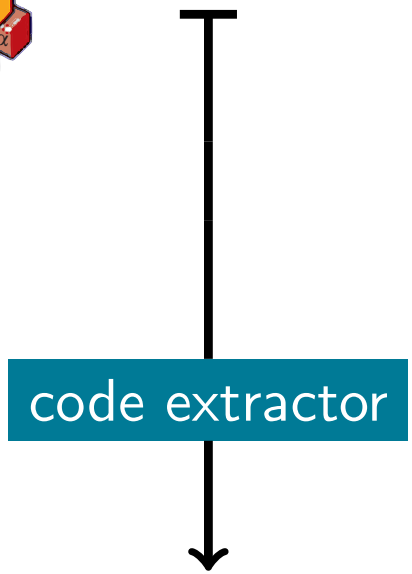
implementation

Standard
ML

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

From formal specifications to implementations

specification



implementation

Standard
ML

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

```
fun sum xs    = ...
fun filter p  = ...
fun prime x   = ...
fun upt i j   = ...
fun sum_primes n =
    sum (filter prime (upt 1 n));
```

From formal specifications to implementations

specification

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

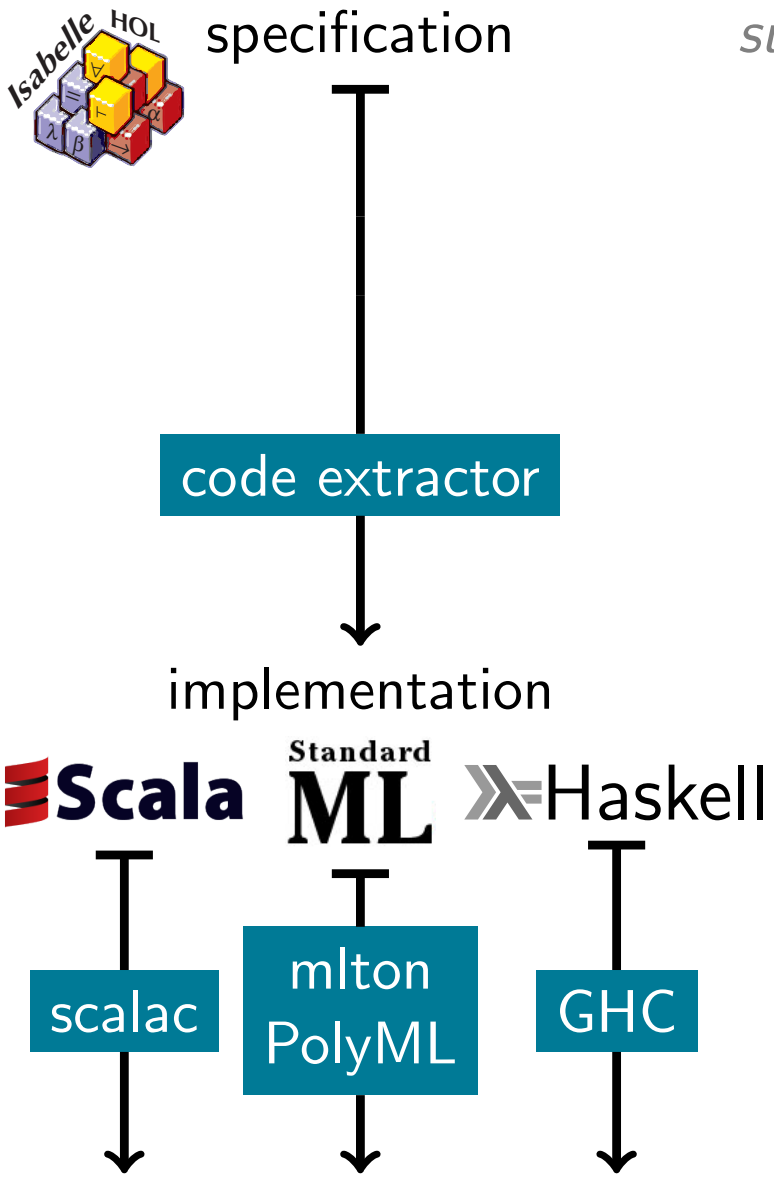
code extractor

implementation

Scala Standard ML Haskell

```
fun sum xs    = ...
fun filter p  = ...
fun prime x   = ...
fun upt i j   = ...
fun sum_primes n =
    sum (filter prime (upt 1 n));
```

From formal specifications to implementations



$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

```
fun sum xs    = ...
fun filter p  = ...
fun prime x   = ...
fun upt i j   = ...
fun sum_primes n =
    sum (filter prime (upt 1 n));
```

From formal specifications to implementations

specification

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

code extractor

implementation

Scala Standard ML Haskell

scalac mlton PolyML GHC

optimise for efficiency

```
fun sum xs    = ...
fun filter p  = ...
fun prime x   = ...
fun upt i j   = ...
fun sum_primes n =
    sum (filter prime (upt 1 n));
```

From formal specifications to implementations

specification

sum-primes n = *sum* (*filter prime* [1 .. *n*])

preprocessor

perform optimisations before extraction?

code extractor

```
fun sum xs    = ...
fun filter p  = ...
fun prime x   = ...
fun upt i j   = ...
fun sum_primes n =
    sum (filter prime (upt 1 n));
```

implementation

Scala Standard ML Haskell

scalac mlton PolyML GHC

optimise for efficiency

From formal specifications to implementations

specification

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

preprocessor

perform optimisations before extraction?

code extractor

implementation

Scala | Standard ML | Haskell

- ⊕ more structure is available
- ⊕ more flexibility in transformations
- ⊕ exploit proven theorems
- ⊕ kernel checks the transformation

scalac | mlton PolyML | GHC

optimise for efficiency

From formal specifications to implementations

specification

sum-primes n = sum (filter prime [1 .. n])

preprocessor

perform optimisations before extraction?

Focus on enabling transformations

Contributions

1. Formalised stream fusion in HOL
2. Verified a library of list functions
3. Implemented the transformation in Isabelle's code generator

scalac

PolyML

GHC

efficiency

List fusion

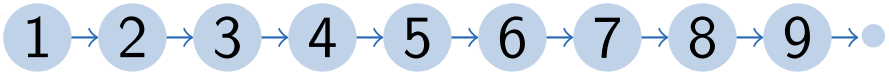
$$\mathit{sum\text{-}primes}\ n = \mathit{sum}\ (\mathit{filter}\ \mathit{prime}\ [1\ ..\ n])$$

List fusion

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

Evaluate sum-primes 9**:**

alloc

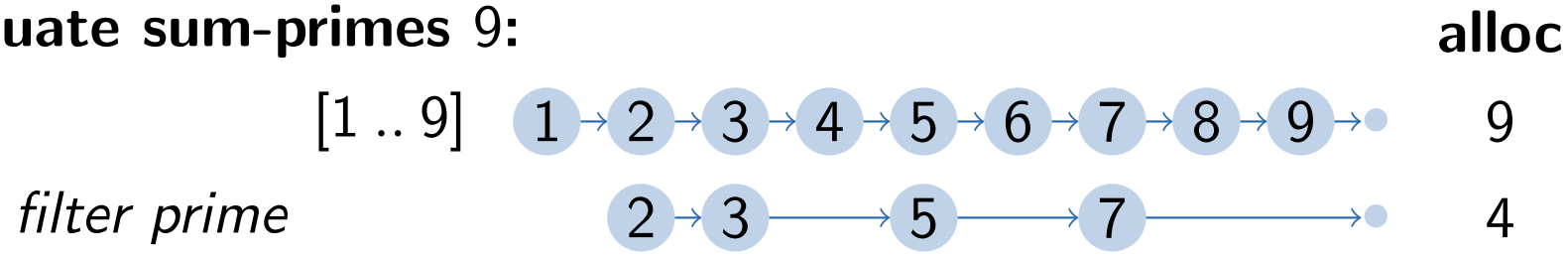
$[1\ ..\ 9]$ 

9

List fusion

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

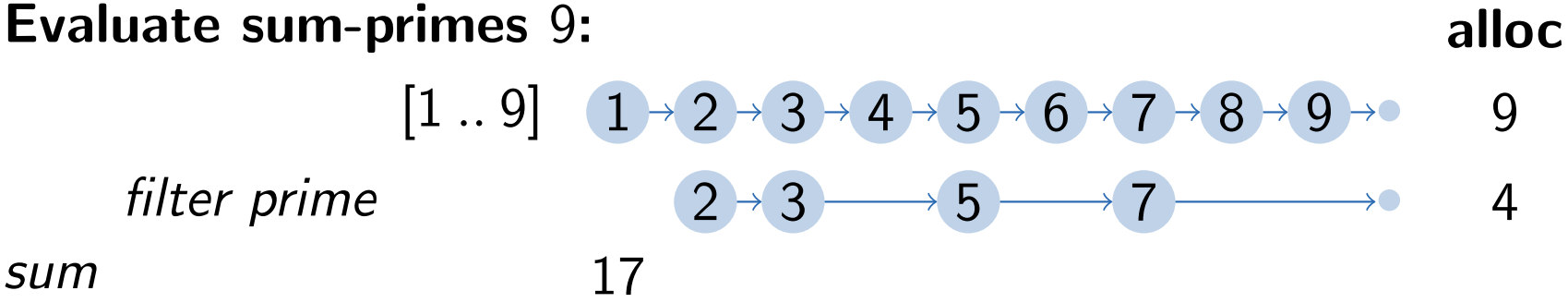
Evaluate sum-primes 9**:**



List fusion

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

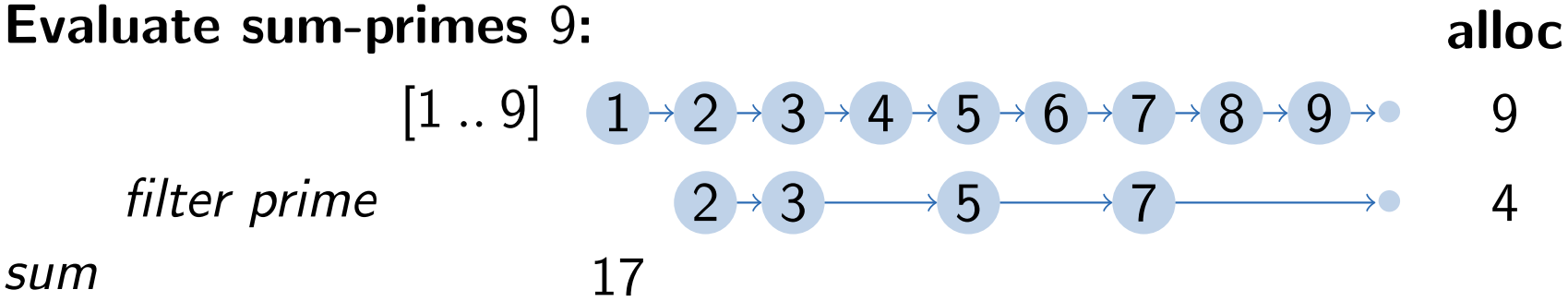
Evaluate sum-primes 9**:**



List fusion

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

Evaluate sum-primes 9**:**

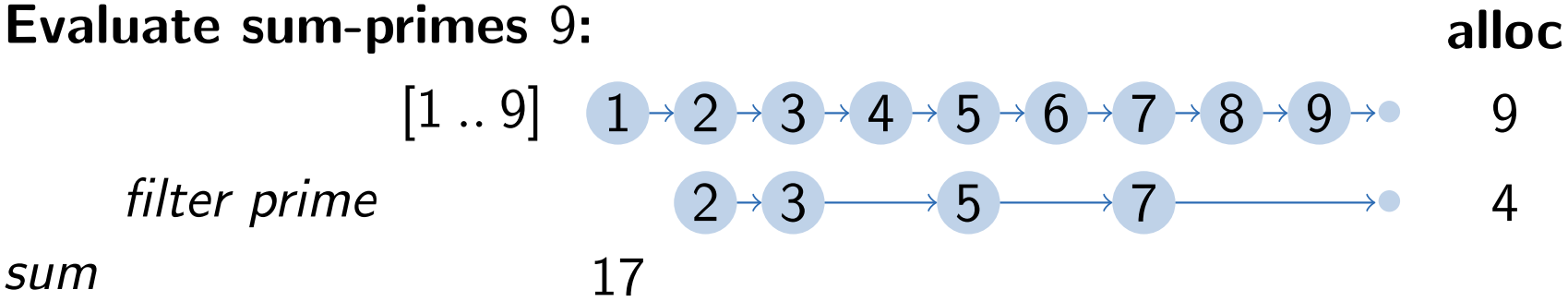


Compilers fail to eliminate intermediate lists
because *filter* and [_ .. _] are recursive.

List fusion

$$sum\text{-}primes\ n = sum\ (filter\ prime\ [1\ ..\ n])$$

Evaluate sum-primes 9**:**



Compilers fail to eliminate intermediate lists
because *filter* and [_ .. _] are recursive.

Express list transformers and producers without recursion
such that they can be inlined into consumers.

Shortcut fusion [Svenningsson 2002]

datatype α *list* $=$
  $[\,]$
 $\mid \alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* $=$
  Done
 $\mid$ *Yield* $\alpha\ \sigma$

type α *stream* $=$
  $\exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

Shortcut fusion [Svenningsson 2002]

datatype α *list* $=$
$\quad [\,]$
$\quad \mid \alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* $=$
$\quad$ *Done*
$\quad \mid$ *Yield* $\alpha\ \sigma$

type α *stream* $=$
$\quad \exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

Stream fusion [Coutts 2010]

datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →

← *unstream*

type α *stream* =
  $\exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

Stream fusion [Coutts 2010]

datatype α *list* $=$
$\quad [\,]$
$\quad \mid \alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* $=$
$\quad$ *Done*
$\quad \mid$ *Yield* $\alpha\ \sigma$
$\quad \mid$ *Skip* σ

stream $\longrightarrow$
unstream $\longrightarrow$

type α *stream* $=$
$\quad \exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* $[1\,..\,n]$) $\longmapsto$ *sumS* (*filterS prime* $[1\,..\,n]_S$)

Stream fusion [Coutts 2010]

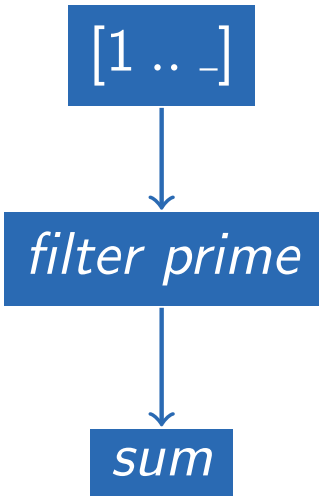
datatype α *list* $=$
$\quad [\,]$
$\quad \mid \alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* $=$
$\quad$ *Done*
$\quad \mid$ *Yield* $\alpha\ \sigma$
$\quad \mid$ *Skip* σ

stream $\longrightarrow$
unstream $\longleftarrow$

type α *stream* $=$
$\quad \exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) $\longmapsto$ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion [Coutts 2010]

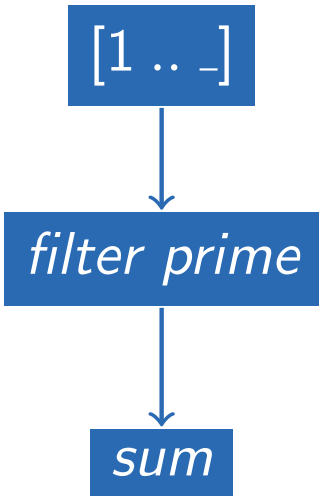
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type α *stream* =
  $\exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion [Coutts 2010]

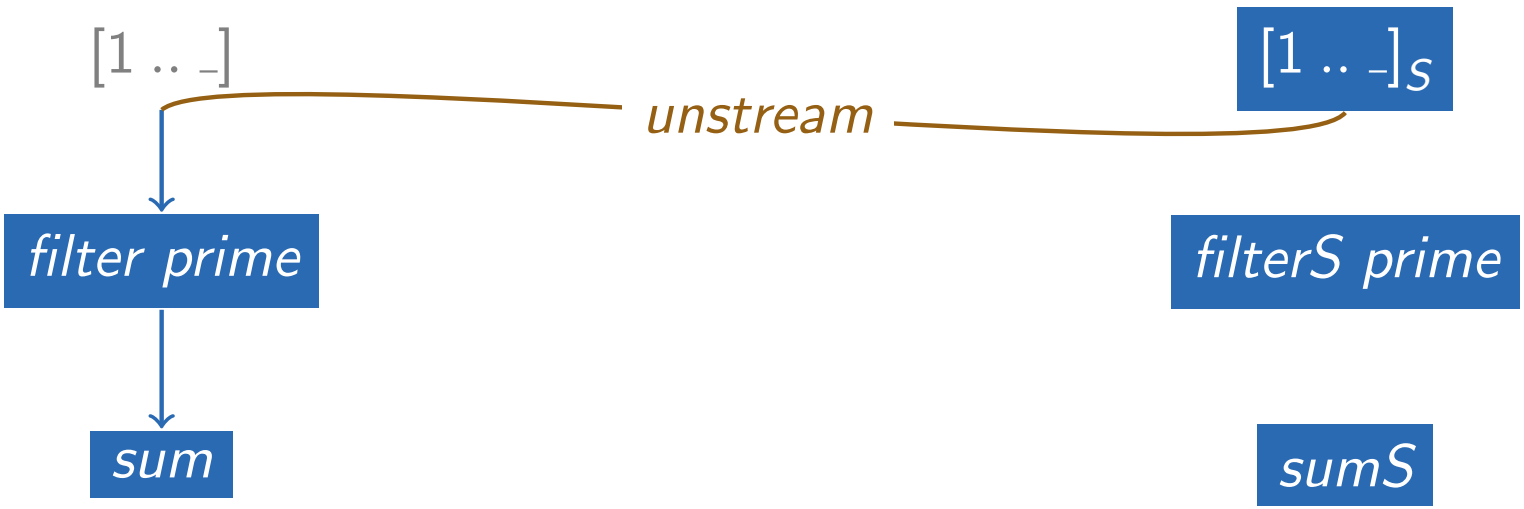
datatype α *list* $=$
$\quad [\,]$
$\quad \mid \alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* $=$
$\quad$ *Done*
$\quad \mid$ *Yield* $\alpha\ \sigma$
$\quad \mid$ *Skip* σ

stream $\longrightarrow$
unstream $\longleftarrow$

type α *stream* $=$
$\quad \exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) $\longmapsto$ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion [Coutts 2010]

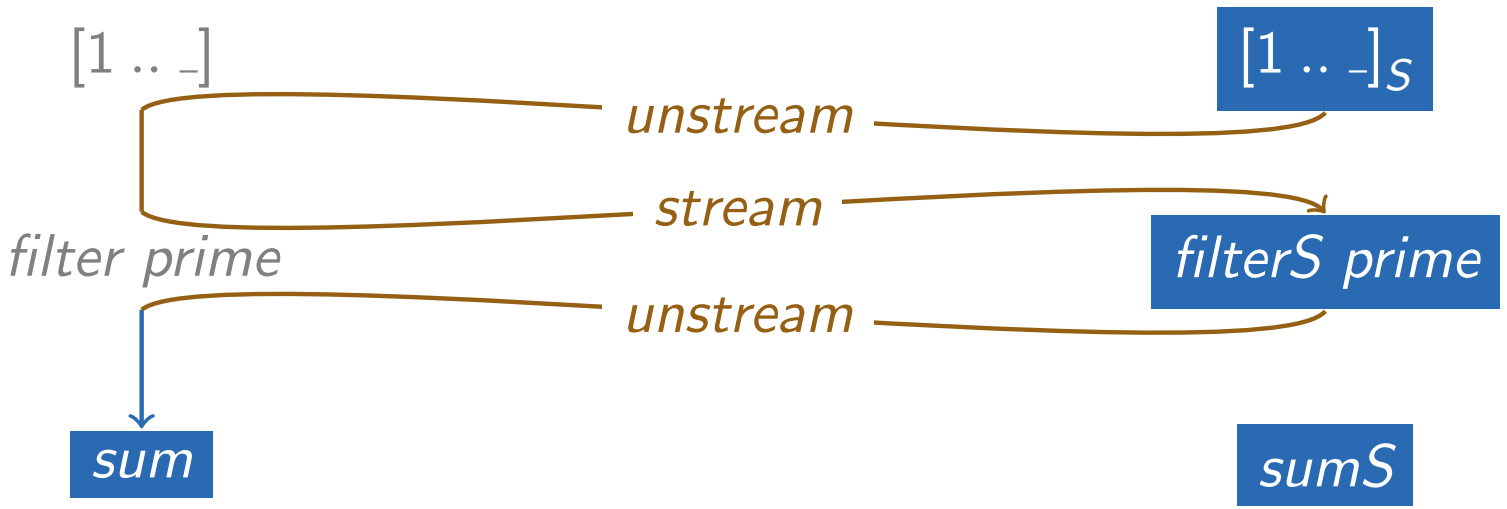
datatype α *list* $=$
$\quad [\,]$
$\quad \mid \alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* $=$
$\quad$ *Done*
$\quad \mid$ *Yield* $\alpha\ \sigma$
$\quad \mid$ *Skip* σ

stream $\longrightarrow$
unstream $\longleftarrow$

type α *stream* $=$
$\quad \exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \textit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) $\longmapsto$ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion [Coutts 2010]

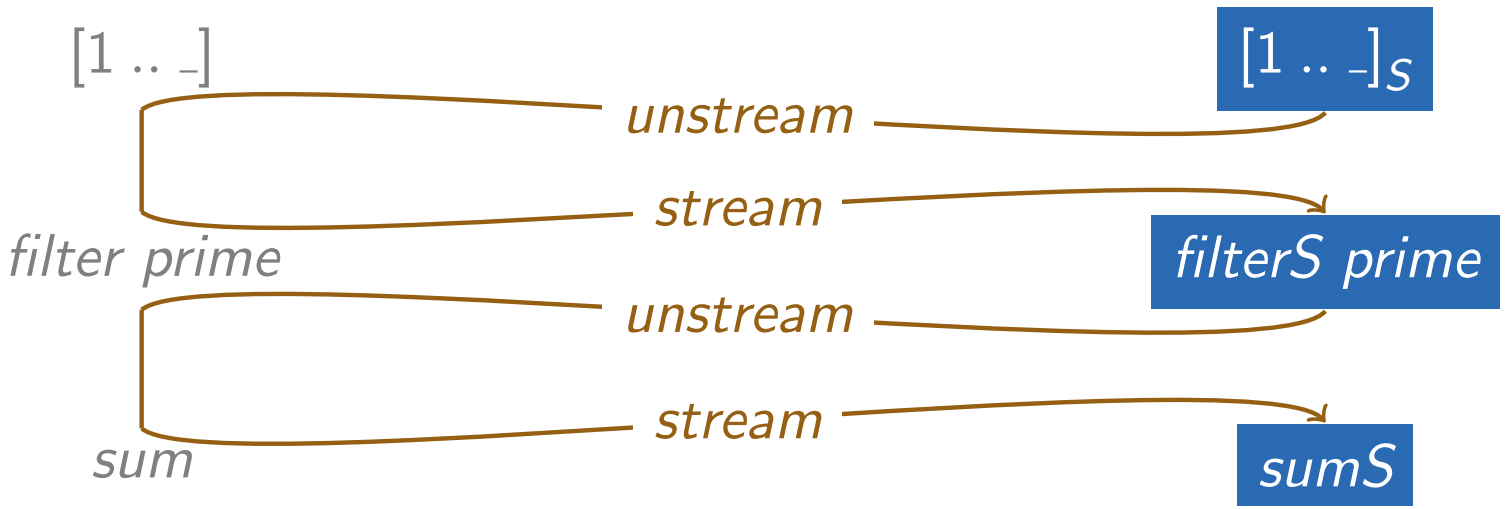
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type α *stream* =
  $\exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion [Coutts 2010]

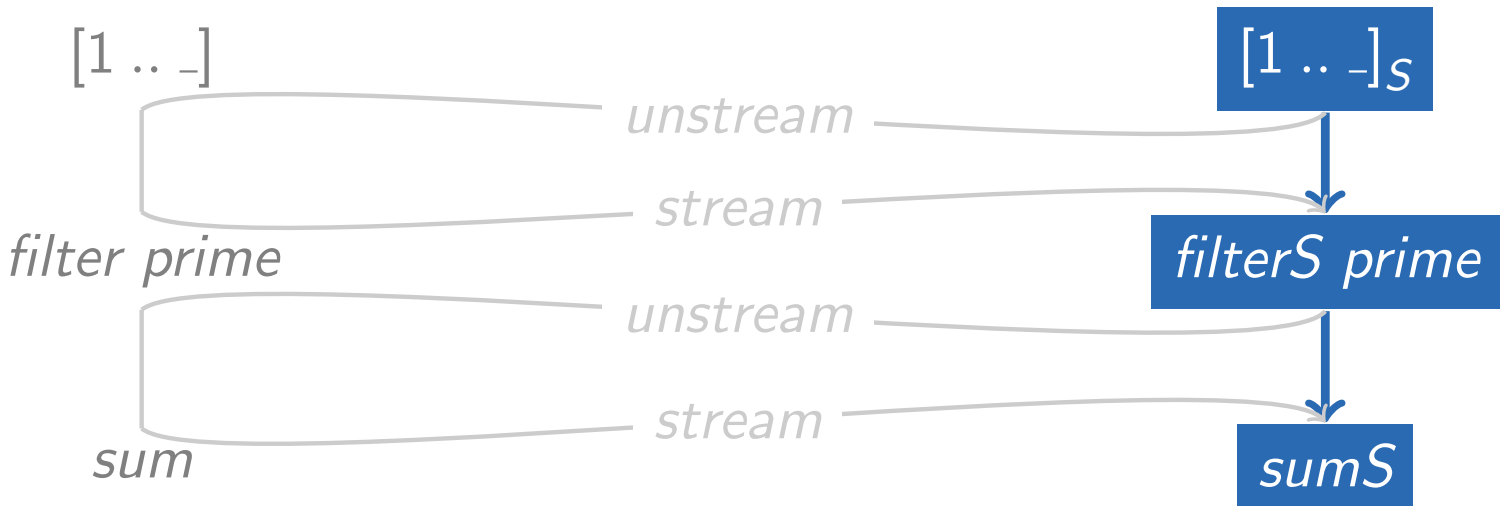
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type α *stream* =
  $\exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion [Coutts 2010]

datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type α *stream* =
  $\exists\sigma.\ (\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)

Stream fusion in HOL [this work]

datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →

← *unstream*

type (α, σ) *stream* =
  ~~$\exists\sigma.$~~ $(\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

Challenges in HOL

1. No existential types
   Fusion equation $\mathit{stream} \circ \mathit{unstream} = \mathit{id}$ does not type-check.

Stream fusion in HOL [this work]

datatype α *list* =
  []
 | $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
 | *Yield* $\alpha\ \sigma$
 | *Skip* σ

stream →
← *unstream*

type (α, σ) *stream* =
  ~~$\exists\sigma.$~~ $(\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

generator

Challenges in HOL

1. No existential types
   Fusion equation *stream* $\circ$ *unstream* $=$ *id* does not type-check.

2. *list* and *stream* are not isomorphic
   inductive lists — use subtype of **terminating** generators
   $\rightsquigarrow$ return *Done* eventually

   coinductive lists — use subtype of **productive** generators
   $\rightsquigarrow$ only finitely many consecutive skips

Stream fusion in HOL [this work]

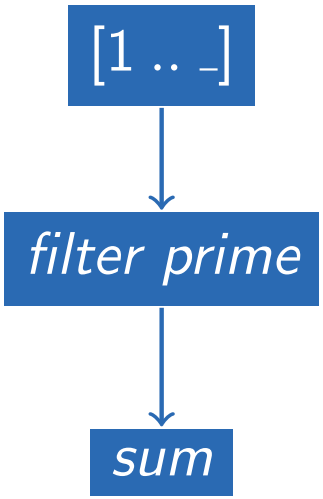
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

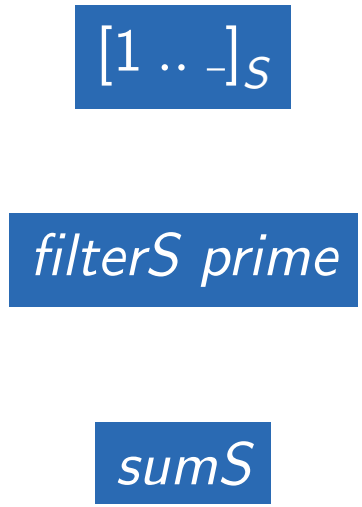
datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type (α, σ) *stream* =
  ~~$\exists\sigma.$~~ $(\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)





Stream fusion in HOL [this work]

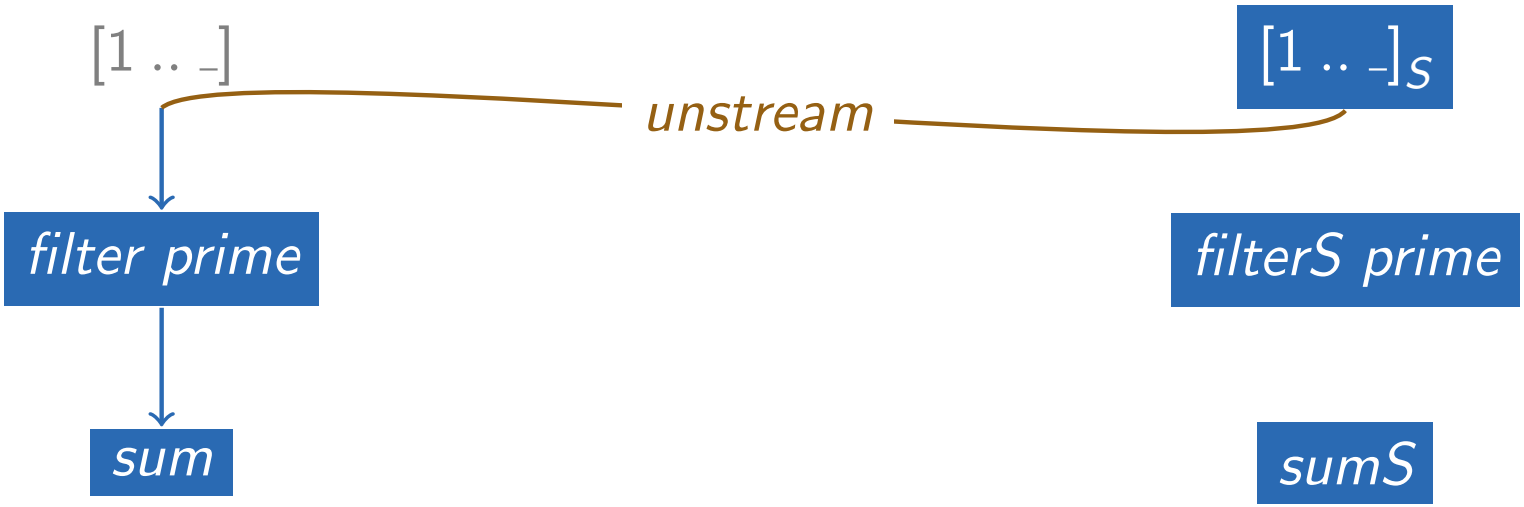
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type (α, σ) *stream* =
  ~~$\exists\sigma.$~~ $(\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion in HOL [this work]

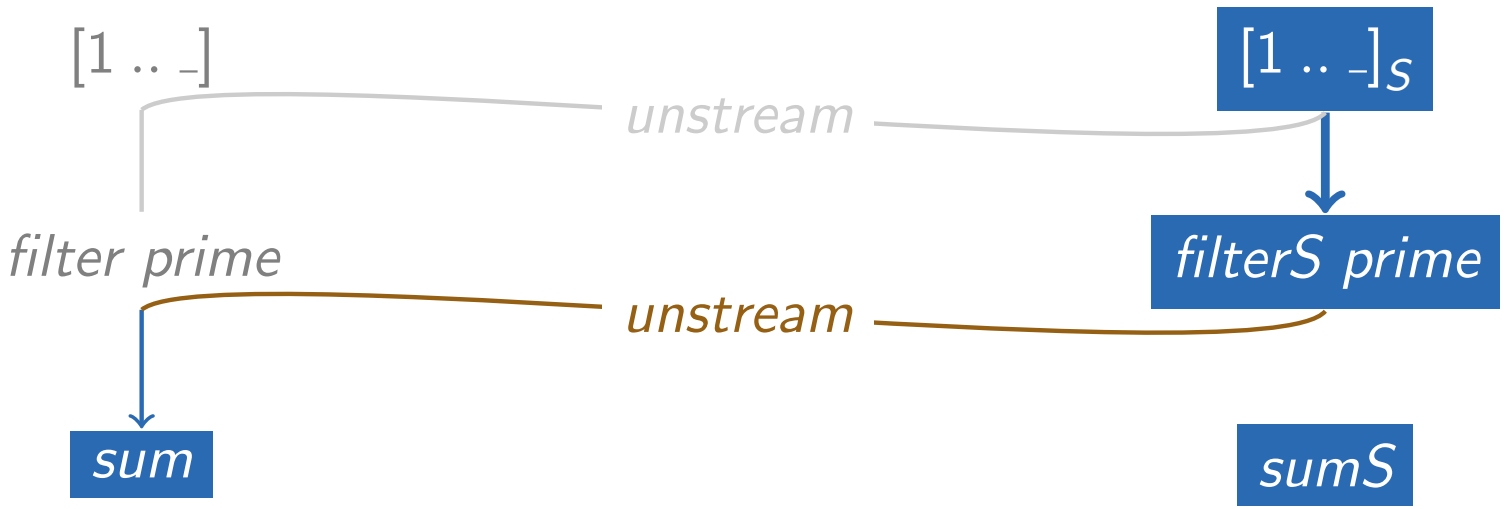
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type (α, σ) *stream* =
  ~~$\exists\sigma.$~~ $(\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Stream fusion in HOL [this work]

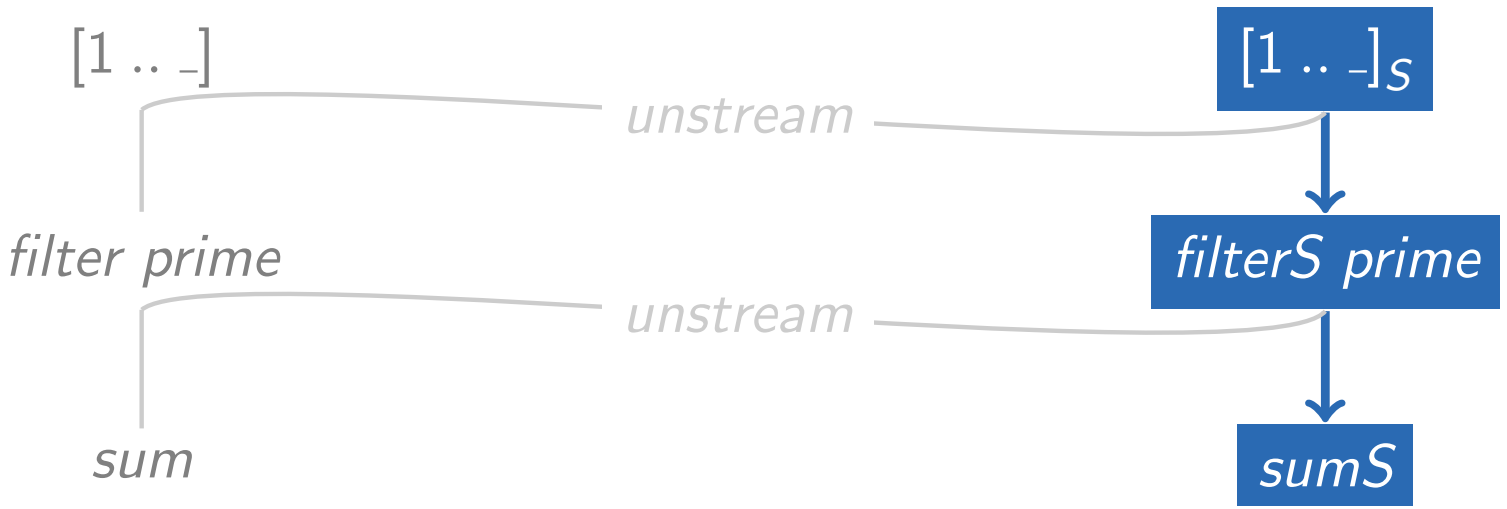
datatype α *list* =
  []
| $\alpha \cdot \alpha$ *list*

datatype $(\alpha,\ \sigma)$ *step* =
  Done
| *Yield* $\alpha\ \sigma$
| *Skip* σ

stream →
← *unstream*

type (α, σ) *stream* =
  ~~$\exists\sigma.$~~ $(\sigma \Rightarrow (\alpha,\ \sigma)\ \mathit{step}) \times \sigma$

sum (*filter prime* [1 .. *n*]) ⟼ *sumS* (*filterS prime* $[1\ ..\ n]_S$)



Evaluation

Implemented fusion for Isabelle's library of lists in Isabelle's code generator

Evaluation

Evaluation

Implemented fusion for Isabelle's library of lists in Isabelle's code generator

Evaluation

on micro-benchmarks

compiler	speed up
mlton 20100608	12 x
GHC 7.8.4 -O3 + SAT	1.4–1.7 x
PolyML 5.5.2	1.3–1.4 x

Evaluation

Implemented fusion for Isabelle's library of lists in Isabelle's code generator

Evaluation

on micro-benchmarks

compiler	speed up
mlton 20100608	12 x
GHC 7.8.4 -O3 + SAT	1.4–1.7 x
PolyML 5.5.2	1.3–1.4 x

on real-world application CeTA

- ⛔ fusion triggers only once

  no measurable impact

Evaluation

Implemented fusion for Isabelle's library of lists in Isabelle's code generator

Evaluation

on micro-benchmarks

compiler	speed up
mlton 20100608	12 x
GHC 7.8.4 -O3 + SAT	1.4–1.7 x
PolyML 5.5.2	1.3–1.4 x

on real-world application CeTA

- ⊖ fusion triggers only once

  no measurable impact

Conclusions

- ⊕ Transformations before code extraction have big potential,
- ⊖ but more engineering in the code generator is needed

Evaluation

Implemented fusion for Isabelle's library of lists in Isabelle's code generator

Evaluation

on micro-benchmarks

compiler	speed up
mlton 20100608	12 x
GHC 7.8.4 -O3 + SAT	1.4–1.7 x
PolyML 5.5.2	1.3–1.4 x

on real-world application CeTA

- ⊖ fusion triggers only once

  no measurable impact

Conclusions

- ⊕ Transformations before code extraction have big potential,
- ⊖ but more engineering in the code generator is needed

Available in the Archive of Formal Proofs

`http://afp.sf.net/entries/Stream_Fusion_Code.shtml`