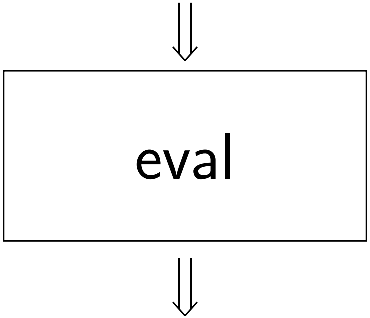
International Conference on Functional Programming 2017
Oxford, UK

Imperative Functional Programs that Explain their Work

Wilmer Ricciotti **Jan Stolarek** Roly Perera James Cheney
University of Edinburgh

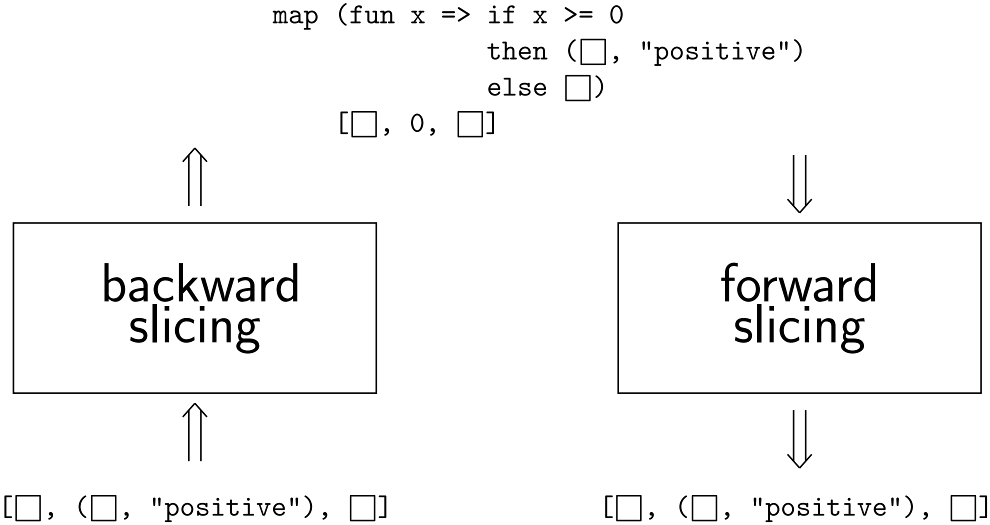
```
map (fun x => if x >= 0
                  then (x, "positive")
                  else (x, "non-positive"))
      [-1, 0, 1]
```




```
[(-1, "non-positive"), (0, "positive"), (1, "positive")]
```

Debugger

Slicing


map (fun x => if x >= 0
then (□, "positive")
else □)
[□, 0, □]
backward slicing
forward slicing
[□, (□, "positive"), □]
[□, (□, "positive"), □]

Imperative Transparent ML

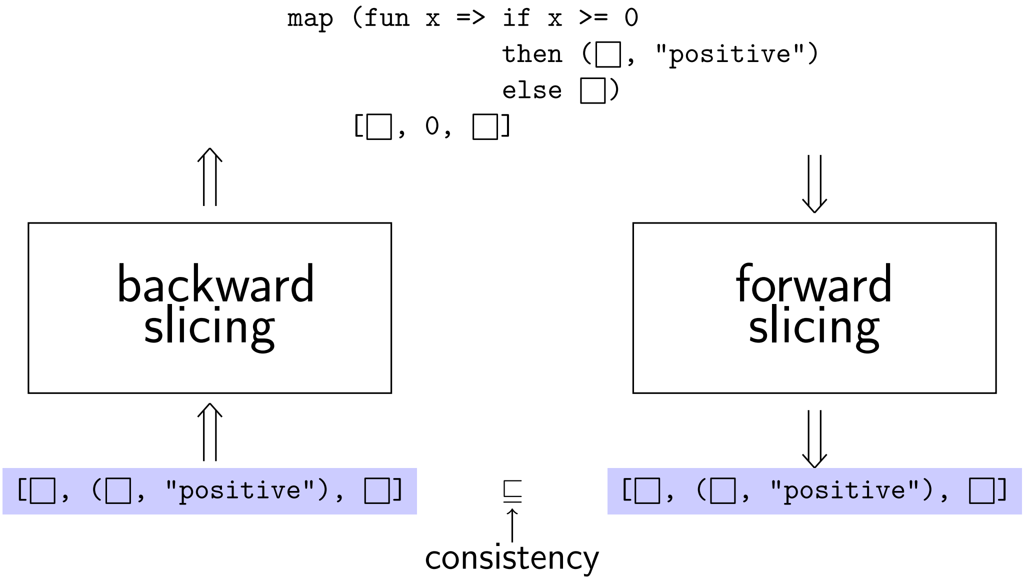
TML is a simple, purely functional, ML-like language:

- sums
- products
- higher-order functions

We created iTML that adds:

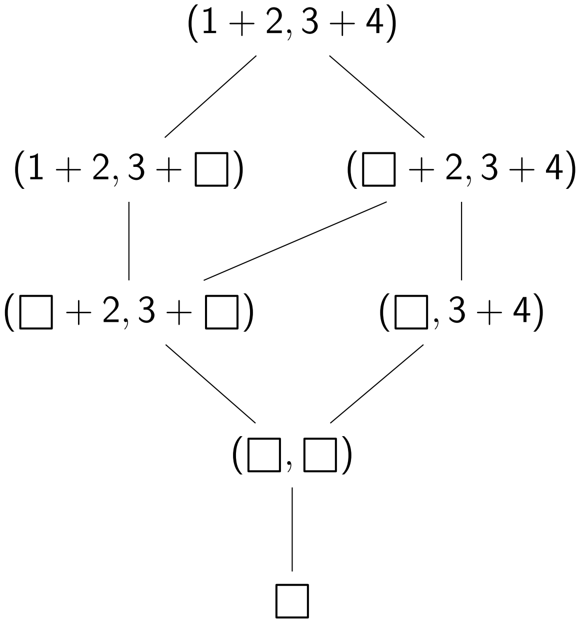
- references
- loops, arrays
- exceptions

```
let a = ref 1 in
let b = ref 2 in
map (fun c -> b := !b - 1 ; 1/!c)
    [a,b]
```

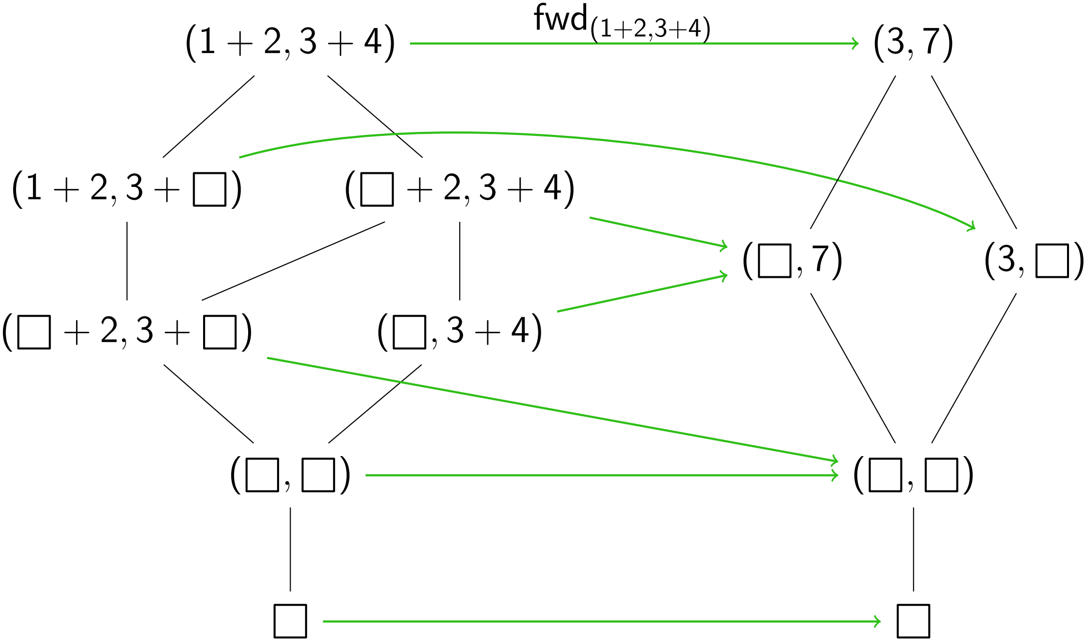

map (fun x => if x >= 0
then (□, "positive")
else □)
[□, 0, □]
backward slicing
forward slicing
[□, (□, "positive"), □]
⊑
[□, (□, "positive"), □]
consistency

Minimality:

$$\mathsf{bwd}_e(v') = \bigsqcap\{e' \mid v' \sqsubseteq \mathsf{fwd}_e(e')\}$$

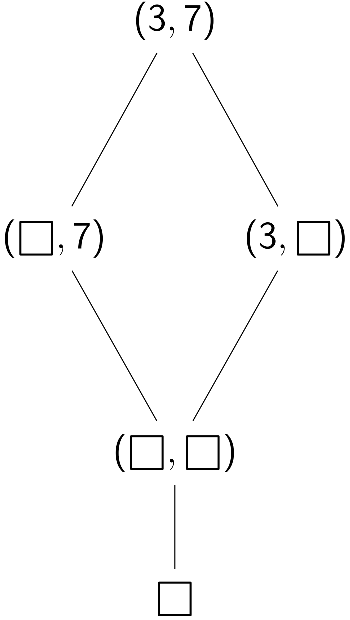


Partial expressions form a *finite lattice*

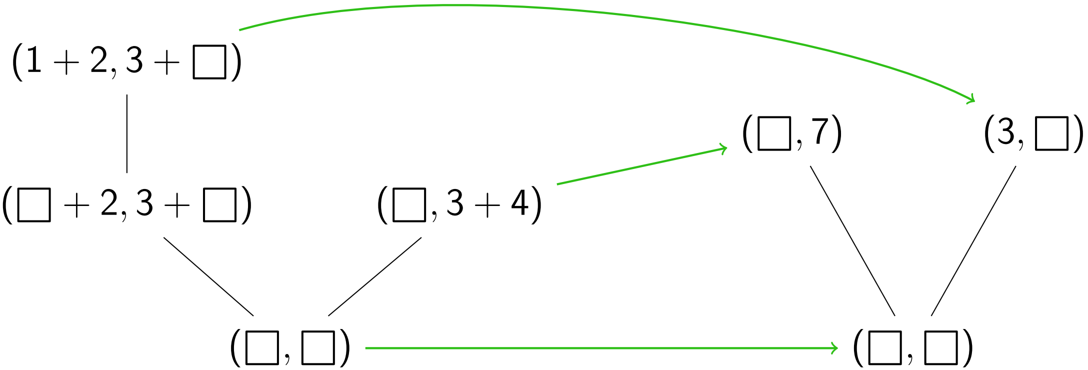


fwd(1+2,3+4)
(1 + 2, 3 + 4)
(3, 7)
(1 + 2, 3 + □)
(□ + 2, 3 + 4)
(□, 7)
(3, □)
(□ + 2, 3 + □)
(□, 3 + 4)
(□, □)
(□, □)
□
□

Partial values form a *finite lattice*

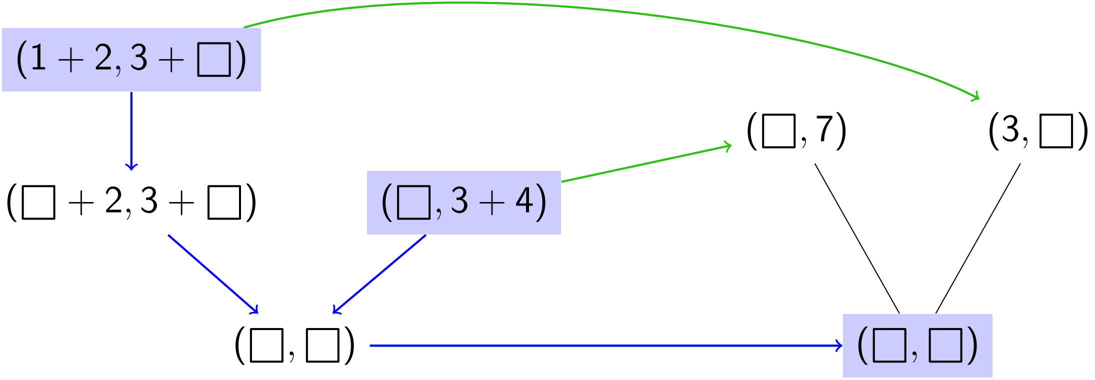


Forward slicing *preserves meets*



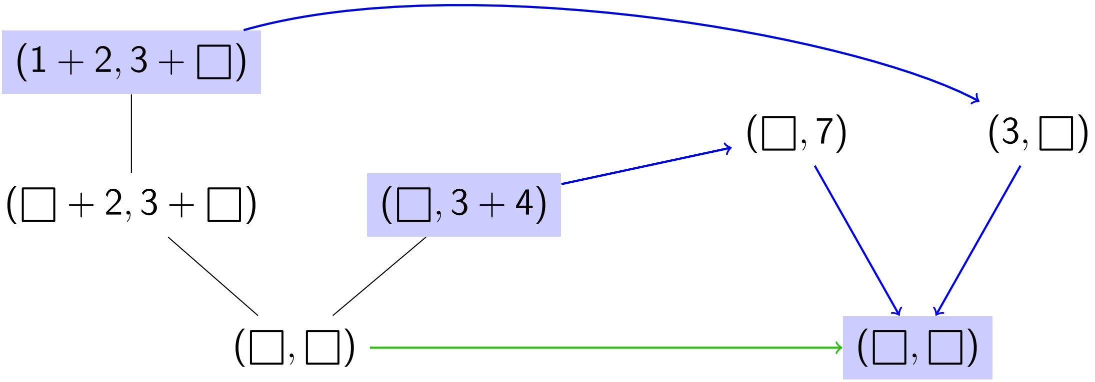
$$\mathit{fwd}(x \sqcap y) \quad = \quad \mathit{fwd}(x) \sqcap \mathit{fwd}(y)$$

Forward slicing *preserves meets*



$$fwd(x \sqcap y) \quad = \quad fwd(x) \sqcap fwd(y)$$

Forward slicing *preserves meets*



$$\mathit{fwd}(x \sqcap y) \quad = \quad \mathit{fwd}(x) \sqcap \mathit{fwd}(y)$$

We know:

- partial expressions and partial values form finite lattices
- forward slicing is meet-preserving
- backward slicing should be consistent and minimal

We know:

- partial expressions and partial values form finite lattices
- forward slicing is meet-preserving
- backward slicing should be consistent and minimal

Corollary (1)

There exists a backward slicing function bwd *such that* bwd $\dashv$ fwd *form a Galois connection.*

$$(P, \sqsubseteq_P), \quad (Q, \sqsubseteq_Q), \quad f : P \to Q, \quad g : Q \to P$$

f and g form a Galois connection (written $f \dashv g$) when

$$f(p) \sqsubseteq_Q q \iff p \sqsubseteq_Q g(q)$$

We know:

- partial expressions and partial values form finite lattices
- forward slicing is meet-preserving
- backward slicing should be consistent and minimal

Corollary (1)

There exists a backward slicing function bwd *such that* bwd $\dashv$ fwd *form a Galois connection.*

$$bwd : values_{\Box} \rightarrow expressions_{\Box}, \quad fwd : expressions_{\Box} \rightarrow values_{\Box}$$

bwd and *fwd* form a Galois connection (written *bwd* $\dashv$ *fwd*) when

$$bwd(\text{slicing criterion}) \sqsubseteq \text{expr}_{\Box} \iff \text{slicing criterion} \sqsubseteq fwd(\text{expr}_{\Box})$$

We know:

- partial expressions and partial values form finite lattices
- forward slicing is meet-preserving
- backward slicing should be consistent and minimal

Corollary (1)

There exists a backward slicing function bwd *such that* bwd $\dashv$ fwd *form a Galois connection.*

Corollary (2)

If bwd $\dashv$ fwd *form a Galois connection then* bwd *is consistent and minimal w.r.t.* fwd.

We know:

- partial expressions and partial values form finite lattices
- forward slicing is meet-preserving
- backward slicing should be consistent and minimal

Corollary (1)

There exists a backward slicing function bwd *such that* bwd $\dashv$ fwd *form a Galois connection.*

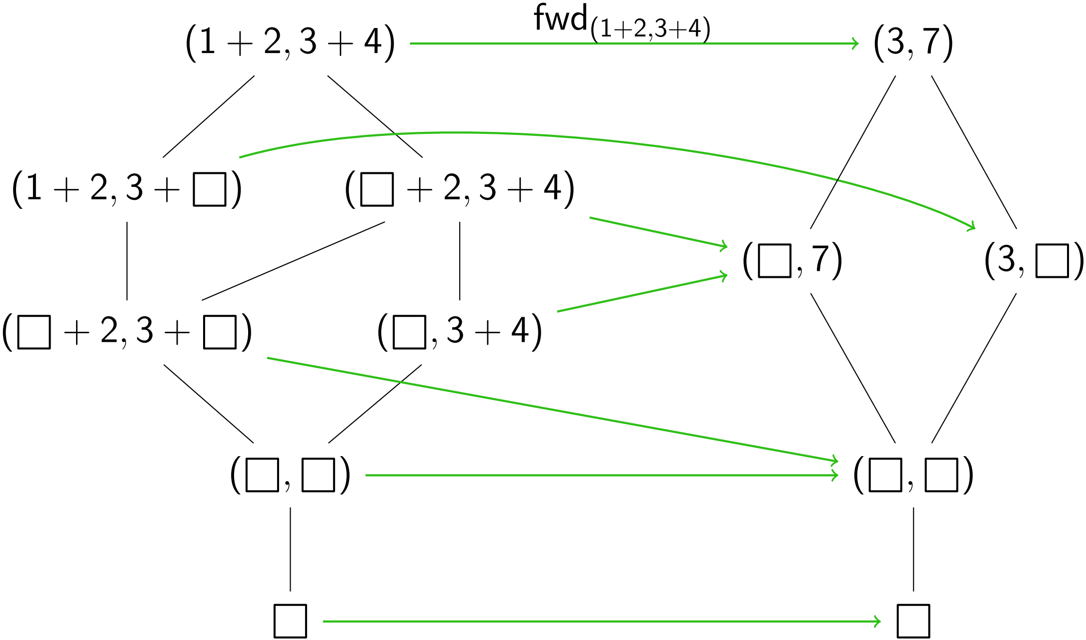
Corollary (2)

If bwd $\dashv$ fwd *form a Galois connection then* bwd *is consistent and minimal w.r.t.* fwd.

Corollary (3)

Choice of fwd *determines* bwd.

See paper for details and proofs.



fwd(1+2,3+4)
(1 + 2, 3 + 4)
(3, 7)
(1 + 2, 3 + □)
(□ + 2, 3 + 4)
(□, 7)
(3, □)
(□ + 2, 3 + □)
(□, 3 + 4)
(□, □)
(□, □)
□
□

$$\mu = [l_1 \mapsto 1, l_2 \mapsto 2]$$

```
l1 := 0; (!l1, !l2)
```

$$\mu = [l_1 \mapsto 1, l_2 \mapsto 2]$$

`l1 := 0; (!l1, !l2)` $\longrightarrow$ `(0, 2)`

$$\mu = [l_1 \mapsto 1, l_2 \mapsto 2]$$

```
□; (!l1, !l2)
```

$$\mu = [l_1 \mapsto 1, l_2 \mapsto 2]$$

$\square$`; (!l1, !l2)` $\longrightarrow$ `(1, 2)`

$$\mu = [l_1 \mapsto \square, l_2 \mapsto \square]$$

$\square$`; (!l1, !l2)` $\longrightarrow$ `(1, 2)`

$$\mu = [l_1 \mapsto \square, l_2 \mapsto \square]$$

$\square$`; (!l1, !l2)` $\longrightarrow$ `(`$\square$`, `$\square$`)`

$$\mu = [l_1 \mapsto \square, l_2 \mapsto 2]$$

$$\square\texttt{; (!l1, !l2)} \longrightarrow \texttt{(}\square\texttt{, 2)}$$

```
raise "foo"; ()
```

raise "foo"; () ⟶ exn "foo"

□; ( ) ⟶ ?

$$\square\ ;\ \texttt{()} \longrightarrow \texttt{exn}\ \square$$

Evaluation

$$T :: \rho, \mu_1, e \Rightarrow \mu_2, R$$

Annotated holes

$$\Box^{\mathscr{k}}_{\mathscr{L}}$$

Backward slicing

$$\mu'_2, R', T \searrow \rho', \mu'_1, e', T'$$

$$\mu'_2 \sqsubseteq \mu_2, R' \sqsubseteq R$$

$$\mu'_1 \sqsubseteq \mu_1, \rho' \sqsubseteq \rho, e' \sqsubseteq e, T' \sqsubseteq T$$

Forward slicing

$$\rho', \mu_1', \mathsf{e}', T' \nearrow \mu_2'', R''$$

$$\mu_2' \sqsubseteq \mu_2'' \sqsubseteq \mu_2, R' \sqsubseteq R'' \sqsubseteq R$$

$$\mu = [l_1 \mapsto 1, l_2 \mapsto 2]$$

$$\square^{val}_{\{l_1\}}\texttt{; (!l1, !l2)} \longrightarrow \texttt{(}\square\texttt{, 2)}$$

$$\square^{\mathit{exn}}_{\{\}}\ ;\ \texttt{()} \longrightarrow \texttt{exn}\ \square$$

Summary

We have developed a slicing framework based on Galois connections that handles functional programs with references and exceptions.

More in the paper:

- rules for forward and backward slicing
- proofs
- more examples
- implementation

Implementation available at:

`https://github.com/jstolarek/slicer`

International Conference on Functional Programming 2017

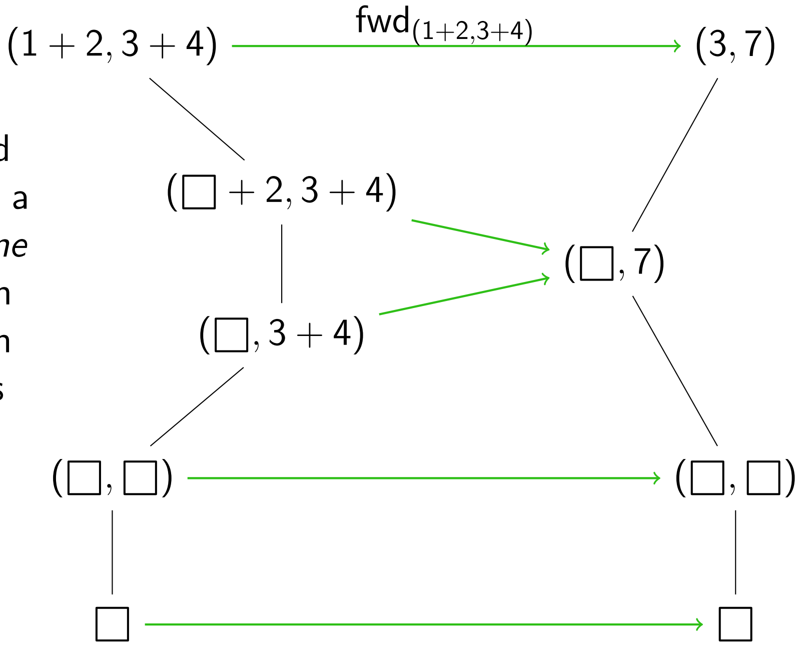
Oxford, UK

Imperative Functional Programs that Explain their Work

Wilmer Ricciotti **Jan Stolarek** Roly Perera James Cheney

University of Edinburgh

Forward slicing is a *monotone* function between lattices



$$\frac{T_1 :: \rho, \mu, \mathsf{e}_1 \Rightarrow \mu', \mathsf{exn}\ v \quad T_2 :: \rho[x \mapsto v], \mu', \mathsf{e}_2 \Rightarrow \mu'', R}{try_F(T_1, x.T_2) :: \rho, \mu, \mathsf{try}\ e_1\ \mathsf{with}\ x \rightarrow e_2 \Rightarrow \mu'', R}$$

$$\frac{\mathscr{L} = \mathsf{writes}(\mathsf{T}) \quad \mu[\ell \mapsto \square \mid \ell \in \mathscr{L}] = \mu}{\mu, \mathscr{k}\square, \mathsf{T} \searrow \square, \mu, \square, \square^{\mathscr{k}}_{\mathscr{L}}}$$

$$\frac{}{\rho, \mu, \mathsf{e}, \square^{\mathscr{k}}_{\mathscr{L}} \nearrow \mu[\ell \mapsto \square \mid \ell \in \mathscr{L}], \mathscr{k}\square}$$