Precise Dataflow Analysis of Event-Driven Applications

Ming-Ho Yee, Ayaz Badouraly, Ondřej Lhoták, Frank Tip, Jan Vitek

January 23, 2020

Event-Driven Programming

```
var fs = require('fs');
var sum;
fs.readdir('.', function f(err, filenames) {
  if (err) throw err;
  sum = 0;
  filenames.forEach(function g(fn) {
    fs.stat('./' + fn, function h(err, stats) {
      if (err) throw err;
      var size = stats.size;
      sum += size;
      console.log(fn + ': ' + size);
      console.log('sum: ' + sum);
    });
  });
});
console.log('done');
```

Event-Driven Programming

```
var fs = require('fs');
var sum;
fs.readdir('.', function f(err, filenames) {
  if (err) throw err;
  sum = 0;
  filenames.forEach(function g(fn) {
    fs.stat('./' + fn, function h(err, stats) {
      if (err) throw err;
      var size = stats.size;
      sum += size;
      console.log(fn + ': ' + size);
      console.log('sum: ' + sum);
    });
  });
});
console.log('done');
```

Event-Driven Programming

```
var fs = require('fs');
var sum;
fs.readdir('.', function f(err, filenames) {
  if (err) throw err;
  sum = 0;
  filenames.forEach(function g(fn) {
    fs.stat('./' + fn, function h(err, stats) {
      if (err) throw err;
      var size = stats.size;
      sum += size;
      console.log(fn + ': ' + size);
      console.log('sum: ' + sum);
    });
  });
});
console.log('done');
```

Event-Driven Programming

```
var fs = require('fs');
var sum;
fs.readdir('.', function f(err, filenames) {
  if (err) throw err;
  sum = 0;
  filenames.forEach(function g(fn) {
    fs.stat('./' + fn, function h(err, stats) {
      if (err) throw err;
      var size = stats.size;
      sum += size;
      console.log(fn + ': ' + size);
      console.log('sum: ' + sum);
    });
  });
});
console.log('done');
```

Event-Driven Programming

```
var fs = require('fs');
var sum;
fs.readdir('.', function f(err, filenames) {
  if (err) throw err;
  sum = 0;
  filenames.forEach(function g(fn) {
    fs.stat('./' + fn, function h(err, stats) {
      if (err) throw err;
      var size = stats.size;
      sum += size;
      console.log(fn + ': ' + size);
      console.log('sum: ' + sum);
    });
  });
});
console.log('done');
```

Modeling Events

$\langle e, f \rangle \in M$ – map of events to functions

$f \in Q$ – queue of functions

Modeling Events

$\langle e, f \rangle \in M$ – map of events to functions

$f \in Q$ – queue of functions

- **Register** function f on event e
  - Add $\langle e, f \rangle$ to M

Modeling Events

$\langle e, f \rangle \in M$ – map of events to functions

$f \in Q$ – queue of functions

- **Register** function f on event e
  - Add $\langle e, f \rangle$ to M
- **Emit** event e
  - Look up $\langle e, f \rangle$ in M, add f to Q

Modeling Events

$\langle e, f \rangle \in M$ – map of events to functions

$f \in Q$ – queue of functions

- **Register** function f on event e
  - Add $\langle e, f \rangle$ to M
- **Emit** event e
  - Look up $\langle e, f \rangle$ in M, add f to Q
- **Invoke** function f
  - When the call stack is empty, remove f from Q and invoke f

IFDS and IDE Frameworks

IFDS – Definition

Interprocedural Finite Distributive Subset

$$P = \langle G^*, D, F, M_F, \sqcap \rangle$$

Distributive: $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$

IFDS – Definition

Interprocedural Finite Distributive Subset

$$P = \langle G^*, D, F, M_F, \sqcap \rangle$$

- $G^* = \langle N^*, E^* \rangle$ is the supergraph

Distributive: $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$

IFDS – Definition

Interprocedural Finite Distributive Subset

$$P = \langle G^*, D, F, M_F, \sqcap \rangle$$

- $G^* = \langle N^*, E^* \rangle$ is the supergraph
- D is a finite set of dataflow facts

Distributive: $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$

IFDS – Definition

Interprocedural Finite Distributive Subset

$$P = \langle G^*, D, F, M_F, \sqcap \rangle$$

- $G^* = \langle N^*, E^* \rangle$ is the supergraph
- D is a finite set of dataflow facts
- $F \subseteq 2^D \rightarrow 2^D$ is a set of distributive dataflow functions

Distributive: $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$

IFDS – Definition

Interprocedural Finite Distributive Subset

$$P = \langle G^*, D, F, M_F, \sqcap \rangle$$

- $G^* = \langle N^*, E^* \rangle$ is the supergraph
- D is a finite set of dataflow facts
- $F \subseteq 2^D \to 2^D$ is a set of distributive dataflow functions
- $M_F: E^* \to F$ assigns dataflow functions to supergraph edges

Distributive: $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$

IFDS – Definition

Interprocedural Finite Distributive Subset

$$P = \langle G^*, D, F, M_F, \sqcap \rangle$$

- $G^* = \langle N^*, E^* \rangle$ is the supergraph
- D is a finite set of dataflow facts
- $F \subseteq 2^D \rightarrow 2^D$ is a set of distributive dataflow functions
- $M_F: E^* \rightarrow F$ assigns dataflow functions to supergraph edges
- $\sqcap$ is the meet operator

Distributive: $f(x_1 \sqcap x_2) = f(x_1) \sqcap f(x_2)$

IFDS – Solution

IFDS algorithm computes a meet-over-valid-paths solution:

$$MVP_{IFDS}(P) = \lambda n.\sqcap_{p \in VP(n)} M_F(p)(\emptyset)$$

Valid path: respects call/return of function calls

IFDS – Solution

IFDS algorithm computes a meet-over-valid-paths solution:

$$MVP_{IFDS}(P) = \lambda n. \sqcap_{p \in VP(n)} M_F(p)(\emptyset)$$

Valid path: respects call/return of function calls

IFDS – Solution

IFDS algorithm computes a meet-over-valid-paths solution:

$$MVP_{IFDS}(P) = \lambda n. \sqcap_{p \in VP(n)} M_F(p)(\emptyset)$$

Valid path: respects call/return of function calls

IFDS – Solution

IFDS algorithm computes a meet-over-valid-paths solution:

$$MVP_{IFDS}(P) = \lambda n. \sqcap_{p \in VP(n)} M_F(p)(\emptyset)$$

Valid path: respects call/return of function calls

IFDS – Solution

IFDS algorithm computes a meet-over-valid-paths solution:

$$MVP_{IFDS}(P) = \lambda n. \sqcap_{p \in VP(n)} M_F(p)(\emptyset)$$

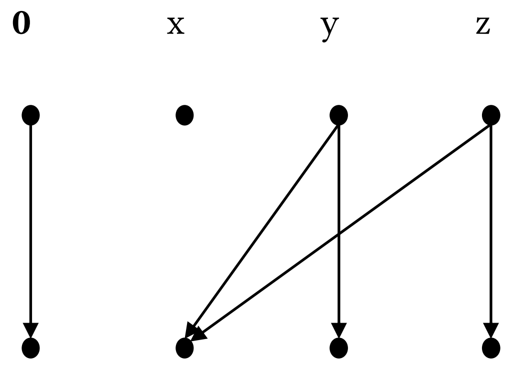
Valid path: respects call/return of function calls

IFDS – Representation Relation

Distributive dataflow function ⇔ representation relation

$$f = \lambda S\,.\,\textbf{if}\ y \in S \lor z \in S\ \textbf{then}\ S \cup \{x\}\ \textbf{else}\ S \setminus \{x\}$$

$R_f =$



IFDS – Exploded Supergraph

Stitch all bipartite graphs to get the exploded supergraph:

$$G_P^\# = \langle N^\#, E^\# \rangle$$

IFDS – Exploded Supergraph

Stitch all bipartite graphs to get the exploded supergraph:

$$G_P^{\#} = \langle N^{\#}, E^{\#} \rangle$$

$P = \langle G^*, D, F, M_F, \sqcap \rangle$ is encoded by $G_P^{\#}$

IFDS – Exploded Supergraph

Stitch all bipartite graphs to get the exploded supergraph:

$$G_P^{\#} = \langle N^{\#}, E^{\#} \rangle$$

$P = \langle G^*, D, F, M_F, \sqcap \rangle$ is encoded by $G_P^{\#}$

$d \in MVP_{IFDS}(P)(n) \Leftrightarrow \langle n, d \rangle$ is reachable from start node

IDE – Generalization of IFDS

Interprocedural Distributive Environment

IDE – Generalization of IFDS

Interprocedural Distributive Environment

L is a finite-height lattice used for the analysis

IDE – Generalization of IFDS

Interprocedural Distributive Environment

L is a finite-height lattice used for the analysis

- Environment $D \rightarrow L$
  - Dataflow set D

IDE – Generalization of IFDS

Interprocedural Distributive Environment

L is a finite-height lattice used for the analysis

- Environment $D \to L$
  - Dataflow set D
- Distributive environment transformer $(D \to L) \to (D \to L)$
  - Distributive dataflow function $D \to D$

IDE – Formal Definition

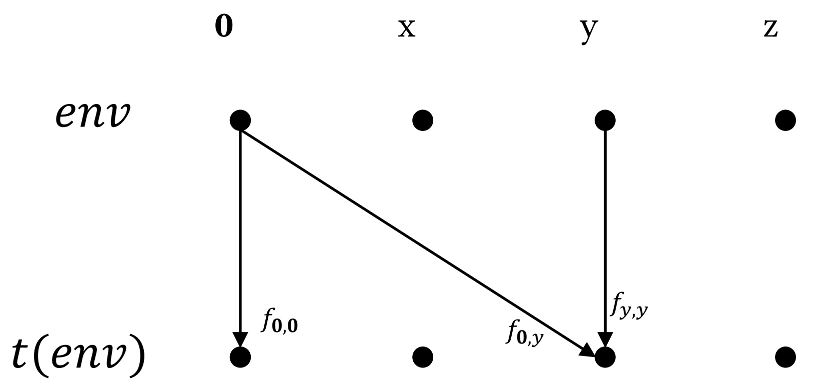
$$P = \langle G^*, D, L, M_{Env} \rangle$$

Meet-over-valid-paths solution:

$$MVP_{IDE}(P) = \lambda n. \sqcap_{p \in VP(n)} M_{Env}(p)(\top_{Env})$$

IDE – Pointwise Representation

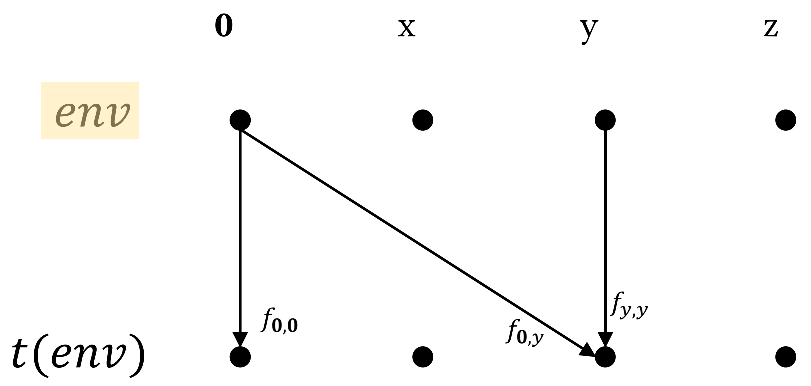
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

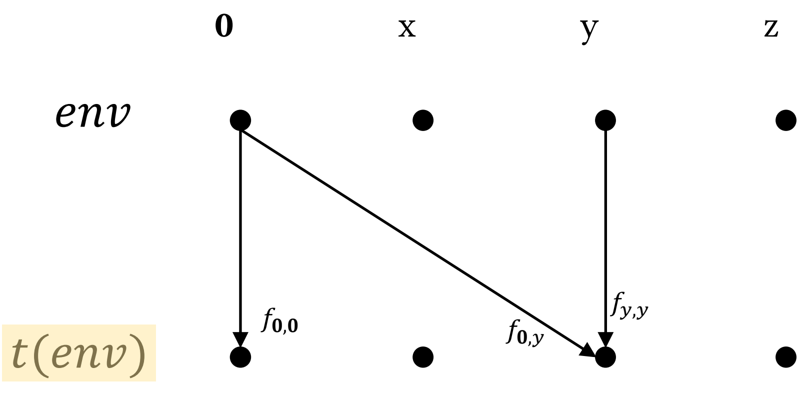
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

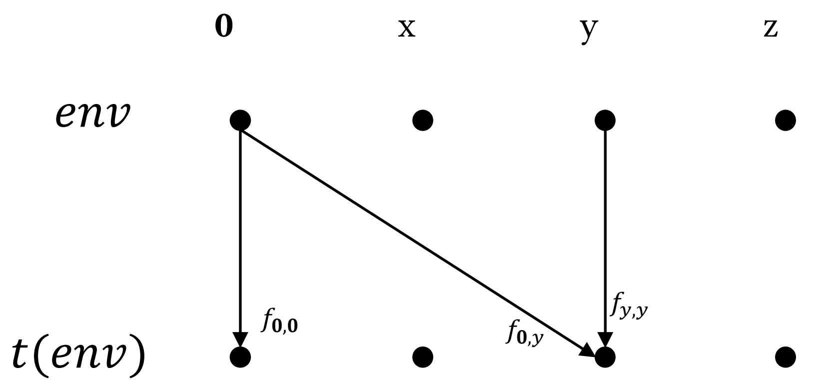
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

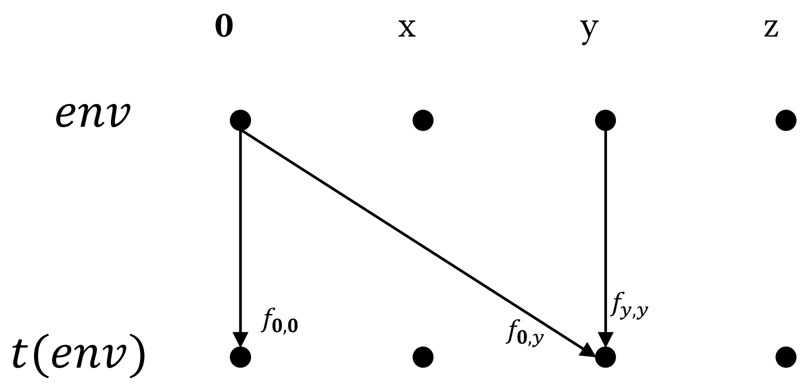
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

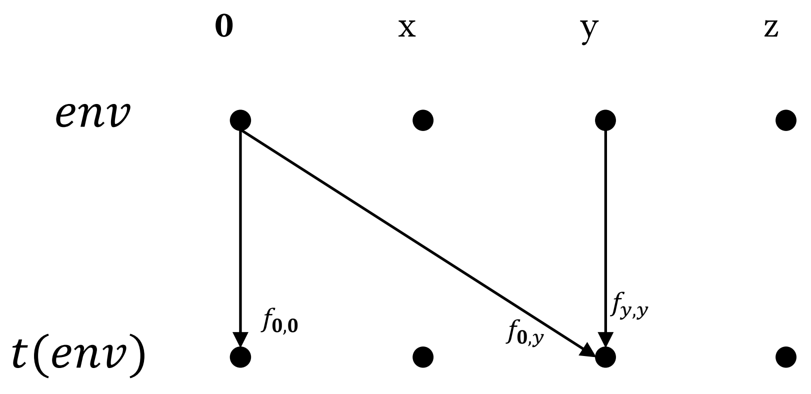
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

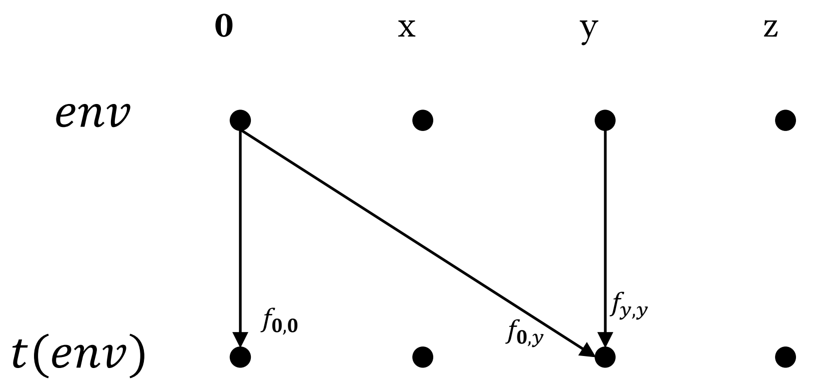
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

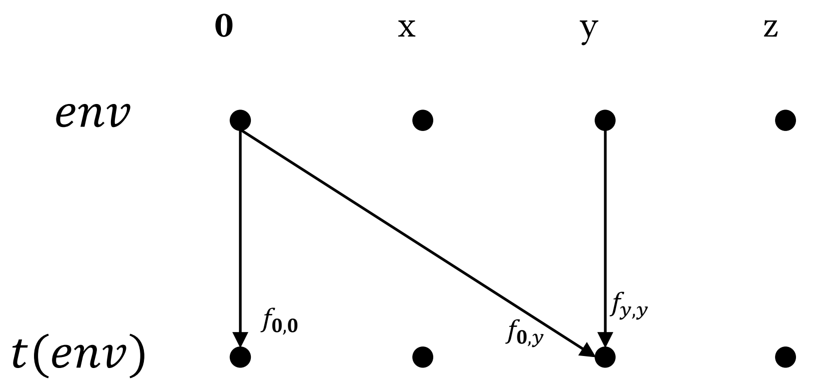
- Edges are labelled with micro-functions in $L \rightarrow L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

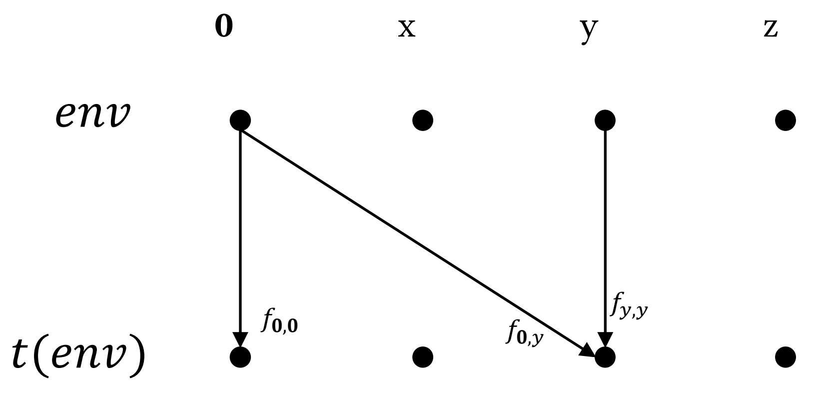
- Edges are labelled with micro-functions in $L \to L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Pointwise Representation

- Edges are labelled with micro-functions in $L \rightarrow L$



$$t(env)(y) = f_{\mathbf{0},y}(\top) \sqcap (\sqcap_{d' \in D} f_{d',y}(env(d')))$$

IDE – Labelled Exploded Supergraph

- Like IFDS exploded supergraph
  - But each edge is labelled with a micro-function

$$\langle G^{\#}, EdgeFn \rangle$$

IDE – Labelled Exploded Supergraph

- Like IFDS exploded supergraph
  - But each edge is labelled with a micro-function

$$\langle G^{\#}, EdgeFn \rangle$$

$P = \langle G^{*}, D, L, M_{Env} \rangle$ is encoded by $\langle G_P^{\#}, EdgeFn_P \rangle$

IFDS to IDE Transformation

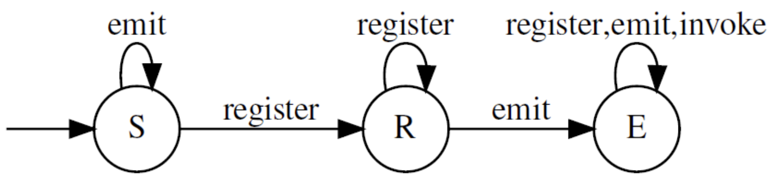
Transformation Overview

Transform IFDS problem instance to IDE problem instance

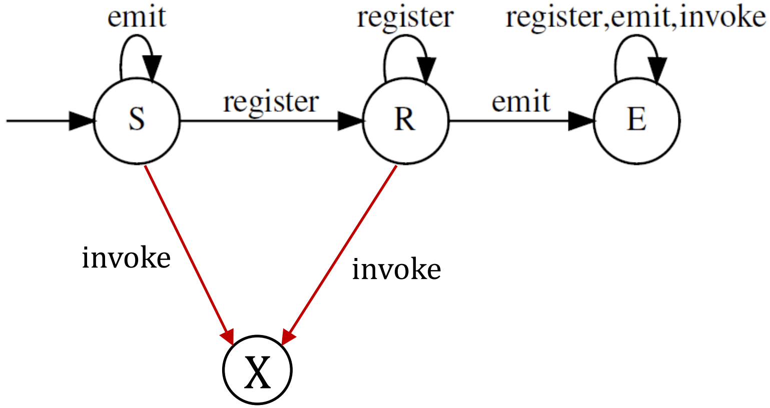
$$T: G^{\#} \rightarrow \langle G^{\#}, EdgeFn \rangle$$

Assign micro-functions to edges of the exploded supergraph

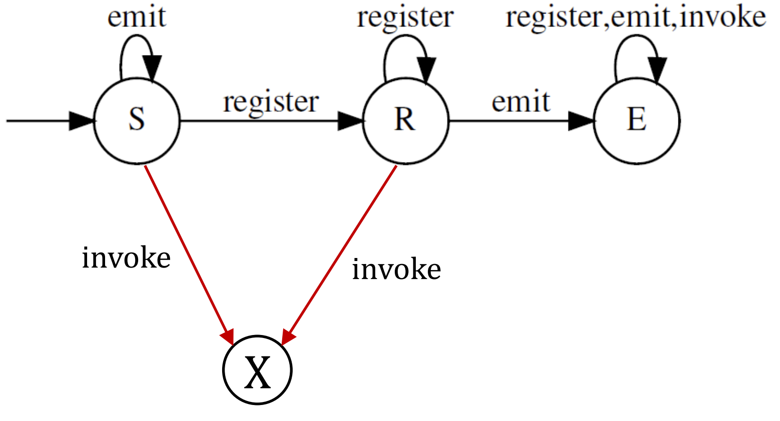
Event Handler State – Model

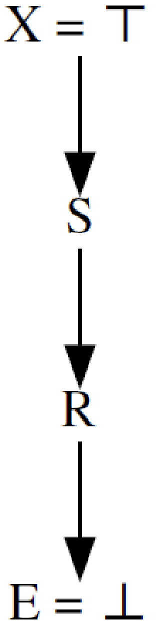


Event Handler State – Model



Event Handler State – Model





Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X
register(S) = R
register(R) = R
register(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X
register(S) = R
register(R) = R
register(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X	emit(X) = X
register(S) = R	emit(S) = S
register(R) = R	emit(R) = E
register(E) = E	emit(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X	emit(X) = X
register(S) = R	emit(S) = S
register(R) = R	emit(R) = E
register(E) = E	emit(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X	emit(X) = X	invoke(X) = X
register(S) = R	emit(S) = S	invoke(S) = X
register(R) = R	emit(R) = E	invoke(R) = X
register(E) = E	emit(E) = E	invoke(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X	emit(X) = X	invoke(X) = X
register(S) = R	emit(S) = S	invoke(S) = X
register(R) = R	emit(R) = E	invoke(R) = X
register(E) = E	emit(E) = E	invoke(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Event Handler State – Micro-functions

- Three basic micro-functions, plus identity
  - Most edges are labelled with the identity micro-function

register(X) = X	emit(X) = X	invoke(X) = X
register(S) = R	emit(S) = S	invoke(S) = X
register(R) = R	emit(R) = E	invoke(R) = X
register(E) = E	emit(E) = E	invoke(E) = E

$$EdgeFn(e) = \begin{cases} register & \text{if edge } e \text{ registers the handler} \\ emit & \text{if edge } e \text{ emits an event for the handler} \\ invoke & \text{if edge } e \text{ invokes the handler from the event loop} \\ id & \text{otherwise} \end{cases}$$

Multiple Event Handlers

Multiple Event Handlers

- Define the IDE lattice $L' = \ H \rightarrow L$
  - H is the set of event handlers in the program
  - L is the event handler state lattice

Multiple Event Handlers

- Define the IDE lattice $L' = H \rightarrow L$
  - H is the set of event handlers in the program
  - L is the event handler state lattice

- IDE computes environments: $D \rightarrow (H \rightarrow L)$
  - For each node n and fact d, we have a map of handlers to states

Multiple Event Handlers

- Define the IDE lattice $L' = H \rightarrow L$
  - H is the set of event handlers in the program
  - L is the event handler state lattice
- IDE computes environments: $D \rightarrow (H \rightarrow L)$
  - For each node n and fact d, we have a map of handlers to states
- Micro-functions: $(H \rightarrow L) \rightarrow (H \rightarrow L)$
  - Alternate representation: $H \rightarrow (L \rightarrow L)$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \rightarrow D$

IDE result: $N^* \rightarrow \left(D \rightarrow (H \rightarrow L)\right)$

Theoretical Properties

Theoretical Properties

Soundness

Result computed along concrete execution path

➔ Result computed by our technique

Theoretical Properties

Soundness

Result computed along concrete execution path

➔ Result computed by our technique

Precision

Result from our technique $\subseteq$ Result computed by IFDS

Theoretical Properties

Soundness

Result computed along concrete execution path

➔ Result computed by our technique

Precision

Result from our technique $\subseteq$ Result computed by IFDS

Formal statements and proofs in the paper

Conclusion

- Problem: static analysis of event-driven programs does not respect event handler ordering
- Our approach: transform an existing IFDS problem to an IDE problem
  - IDE problem maintains information about event handler state
- Transformation is sound and precise
  - Formal statements and proofs in paper

Extra Slides

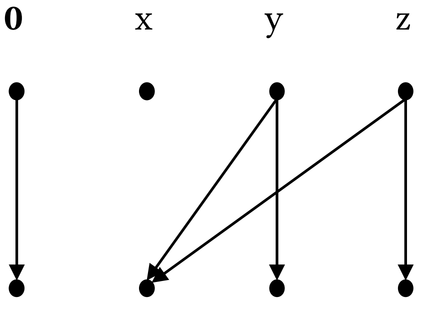
IFDS – Representation Relation

Distributive dataflow function ⇔ representation relation

$$R_g = \{\langle \mathbf{0}, \mathbf{0} \rangle\} \cup \{\langle \mathbf{0}, d \rangle \mid d \in g(\emptyset)\} \cup \{\langle d_1, d_2 \rangle \mid d_2 \in g(\{d_1\}) \wedge d_2 \notin g(\emptyset)\}$$

$f = \lambda S$. **if** $y \in S \vee z \in S$
then $S \cup \{x\}$
else $S \setminus \{x\}$

$$R_f = \{\langle \mathbf{0}, \mathbf{0} \rangle, \langle y, x \rangle, \langle y, y \rangle, \langle z, x \rangle, \langle z, z \rangle\}$$



IDE – Lattices

If L is a lattice with top element $\top$, the pair $L \times L$ is a lattice:

- Top: $\langle \top, \top \rangle$
- Meet: $\langle x_1, y_1 \rangle \sqcap \langle x_2, y_2 \rangle = \langle x_1 \sqcap x_2, y_1 \sqcap y_2 \rangle$

The map $D \rightarrow \ L$ is also a lattice:

- Top: $T_{Env} = \lambda d. \top$
- Meet: $m_1 \sqcap m_2 = \lambda d. \left(env_1(d) \sqcap env_2(d) \right)$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \left(D \to (H \to L)\right)$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n. \{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X\}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n. \{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X\}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n. \{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X\}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

“Untransform” function U applied to IDE result R:

$$U(R) = \lambda n.\{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X\}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n. \{ d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X \}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n. \{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X\}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

“Untransform” function U applied to IDE result R:

$$U(R) = \lambda n. \{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X \}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \big(D \to (H \to L)\big)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n.\{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X\}$$

Transforming IDE Results

For a result, if *any* handler is in state X, discard that result

IFDS result: $N^* \to D$

IDE result: $N^* \to \left(D \to (H \to L)\right)$

"Untransform" function U applied to IDE result R:

$$U(R) = \lambda n. \{d \mid \forall h \in H \,.\, R(n)(d)(h) \neq X \}$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \dots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \dots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \dots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \dots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Soundness

Let P be an IFDS problem, $p = [start, \ldots, n]$ be a concrete execution path, and $d \in D$ a dataflow fact. Then:

$$d \in M_F(p)(\emptyset) \Longrightarrow d \in U(MVP_{IDE}(T(P)))(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}\big(T(P)\big)\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}(T(P))\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}(T(P))\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}(T(P))\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}\left(T(P)\right)\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}(T(P))\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}(T(P))\right)(n) \subseteq MVP_{IFDS}(P)(n)$$

Theoretical Results – Precision

Let P be an IFDS problem and $n \in N^*$ be any node in the supergraph. Then:

$$U\left(MVP_{IDE}\big(T(P)\big)\right)(n) \subseteq MVP_{IFDS}(P)(n)$$