Higher-Order Demand-Driven Symbolic Evaluation

Zachary Palmer Theodore Park Scott F. Smith Shiwei Weng

ICFP 2020, August 24-26, 2020

The Johns Hopkins University and Swarthmore College

Forward vs Demand in Varying Domains

System	Forward	Demand
Logic Programming	Forward-chain (uncommon)	Backward-chain
Tactic-based provers	Forward tactics (uncommon)	Goal-directed tactics
Program Analysis	(most are: kCFA etc)	Reps et al (imperative) DDPA (functional)
Symbolic Execution	(most are)	Snugglebug (imperative) Here: **DDSE** (functional)
Interpreter	(nearly all are)	Here: **DDI** (functional) *no substitition, environment or closures*

Outline

1. The language syntax under study here
2. **DDI**, The novel demand-driven functional interpreter
3. **DDSE**, a demand-driven symbolic evaluator built on DDI
4. Implementation and evaluation of DDSE

Language Features in this Work

In formal theory functions, integers, booleans, conditionals, `input` (for test generation)

Recursion encoded via self-passing

Also in implementation recursive data structures

ANF Used to expose order of operations

e.g. `let x = input in let y = x - 1 in let ret = x * y in ret`

Unique variable names a program point is named by its (unique) defining variable.

The DDI Lookup Function

- Basic idea follows programmer intuition: search upwards in code for variable definitions
- Lookup, $\mathbb{L}([x], @x_{pp}, \underset{\textvisiblespace}{\vdots}) \equiv v$, means x has value v
- $@x_{pp}$ is the program point to begin (reverse) search from
- $\underset{\textvisiblespace}{\vdots}$ is a call stack, of program call points.

```
let y = 1 in
let f = (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

- $\mathbb{L}([\mathrm{y}], @\mathrm{y}, \textvisiblespace) \equiv 1$

The DDI Lookup Function

- Basic idea follows programmer intuition: search upwards in code for variable definitions
- Lookup, $\mathbb{L}([x], @x_{pp}, \vdots) \equiv v$, means x has value v
- $@x_{pp}$ is the program point to begin (reverse) search from
- $\vdots$ is a call stack, of program call points.

```
   let y = 1 in
   let f = (fun x ->
           let fret = x + 1 in fret) in
⇨  let f1 = f y in
   let ret = f f1 in ret
```

- $\mathbb{L}([\mathrm{y}], @\mathrm{y}, \sqcup) \equiv 1$
- $\mathbb{L}([\mathrm{y}], @\mathtt{f1}, \sqcup) \equiv 1$

The DDI Lookup Function

- Basic idea follows programmer intuition: search upwards in code for variable definitions
- Lookup, $\mathbb{L}([x], @x_{pp}, \llcorner\vdots\lrcorner) \equiv v$, means x has value v
- $@x_{pp}$ is the program point to begin (reverse) search from
- $\llcorner\vdots\lrcorner$ is a call stack, of program call points.

```
let y = 1 in
let f = (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

- $\mathbb{L}([\texttt{y}], @\texttt{y}, \llcorner\ \lrcorner) \equiv 1$
- $\mathbb{L}([\texttt{y}], @\texttt{f1}, \llcorner\ \lrcorner) \equiv 1$
- $\mathbb{L}([\texttt{x}], @\texttt{fret}, \llcorner\texttt{f1}\lrcorner) \equiv 1$

Tracing Function Application

Function application requires call-return alignment

```
let y = 1 in
let f =
    (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

1. $\mathbb{L}([\texttt{f1}], @\texttt{f1}, \sqcup)$

Tracing Function Application

Function application requires call-return alignment

```
let y = 1 in
let f =
    (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

1. $\mathbb{L}([\texttt{f1}], @\texttt{f1}, \llcorner\!\_\!\lrcorner)$
2. $\equiv \mathbb{L}([\texttt{fret}], @\texttt{fret}, \llcorner\texttt{f1}\lrcorner)$

Tracing Function Application

Function application requires call-return alignment

```
let y = 1 in
let f =
    (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

1. $\mathbb{L}([\texttt{f1}], @\texttt{f1}, \llcorner\!\lrcorner)$
2. $\equiv \mathbb{L}([\texttt{fret}], @\texttt{fret}, \llcorner\texttt{f1}\lrcorner)$
3. $\equiv \underline{\mathbb{L}([\texttt{x}], @\texttt{fun x}, \llcorner\texttt{f1}\lrcorner)} + 1$

Tracing Function Application

Function application requires call-return alignment

```
let y = 1 in
let f =
    (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

1. $\mathbb{L}([\texttt{f1}], @\texttt{f1}, \sqcup)$
2. $\equiv \mathbb{L}([\texttt{fret}], @\texttt{fret}, \sqcup\texttt{f1}\sqcup)$
3. $\equiv \underline{\mathbb{L}([\mathrm{x}], @\texttt{fun x}, \sqcup\texttt{f1}\sqcup)} + 1$
4. $\mathbb{L}([\mathrm{x}], @\texttt{fun x}, \sqcup\texttt{f1}\sqcup) \equiv \mathbb{L}([\mathrm{y}], @\texttt{f1}, \sqcup)$

Tracing Function Application

Function application requires call-return alignment

```
let y = 1 in
let f =
    (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in
let ret = f f1 in ret
```

1. $\mathbb{L}([\mathtt{f1}], @\mathtt{f1}, \llcorner\;\;\lrcorner)$
2. $\equiv \mathbb{L}([\mathtt{fret}], @\mathtt{fret}, \llcorner \mathtt{f1} \lrcorner)$
3. $\equiv \underline{\mathbb{L}([\mathtt{x}], @\mathtt{fun\ x}, \llcorner \mathtt{f1} \lrcorner)} + 1$
4. $\mathbb{L}([\mathtt{x}], @\mathtt{fun\ x}, \llcorner \mathtt{f1} \lrcorner) \equiv \mathbb{L}([\mathtt{y}], @\mathtt{f1}, \llcorner\;\;\lrcorner)$
5. $\mathbb{L}([\mathtt{y}], @\mathtt{f1}, \llcorner\;\;\lrcorner) \equiv 1$ so final result is $\mathbb{L}([\mathtt{f1}], @\mathtt{f1}, \llcorner\;\;\lrcorner) \equiv 2$.

Non-Local Variables

```
let g =
  (fun x ->
    let gret = (fun y ->
                 let gyret = x + y in gyret) in gret) in
let g5 = g 5 in
let ret = g5 1 in ret
```

Non-Local Variables

```
    let g =
      (fun x ->
        let gret = (fun y ->
⇨                   let gyret = x + y in gyret) in gret) in
    let g5 = g 5 in
    let ret = g5 1 in ret
```

1. $\ldots \mathbb{L}([\mathrm{x}], \texttt{@gyret}, \underline{\texttt{ret}}) \equiv \mathbb{L}([\mathrm{g5}, \mathrm{x}], \texttt{@g5}, \underline{\quad})$:
    1.1 find definition site for g5;
    1.2 then, resume search for x since that is lexical scope of its def'n.

Non-Local Variables

```
let g =
  (fun x ->
    let gret = (fun y ->
                 let gyret = x + y in gyret) in gret) in
let g5 = g 5 in
let ret = g5 1 in ret
```

1. $\ldots\mathbb{L}([\mathrm{x}], \texttt{@gyret}, \underline{\texttt{ret}}) \equiv \mathbb{L}([\mathrm{g5}, \mathrm{x}], \texttt{@g5}, \underline{\quad})$:
    1.1 find definition site for g5;
    1.2 then, resume search for x since that is lexical scope of its def'n.
2. $\mathbb{L}([\mathrm{g5}, \mathrm{x}], \texttt{@g5}, \underline{\quad}) \equiv \mathbb{L}([\mathrm{gret}, \mathrm{x}], \texttt{@gret}, \underline{\mathrm{g5}})$

Non-Local Variables

```
  let g =
    (fun x ->
      let gret = (fun y ->
                   let gyret = x + y in gyret) in gret) in
  let g5 = g 5 in
  let ret = g5 1 in ret
```

1. $\ldots \mathbb{L}([x], @\texttt{gyret}, \llcorner\texttt{ret}\lrcorner) \equiv \mathbb{L}([g5, x], @g5, \llcorner\;\lrcorner)$:
   1.1 find definition site for g5;
   1.2 then, resume search for x since that is lexical scope of its def'n.
2. $\mathbb{L}([g5, x], @g5, \llcorner\;\lrcorner) \equiv \mathbb{L}([\texttt{gret}, x], @\texttt{gret}, \llcorner g5\lrcorner)$
3. $\mathbb{L}([\texttt{gret}, x], @\texttt{gret}, \llcorner g5\lrcorner) \equiv \mathbb{L}([x], @\texttt{fun x}, \llcorner g5\lrcorner)$

Non-Local Variables

```
let g =
  (fun x ->
    let gret = (fun y ->
                 let gyret = x + y in gyret) in gret) in
let g5 = g 5 in
let ret = g5 1 in ret
```

1. $\ldots \mathbb{L}([\mathrm{x}], @\texttt{gyret}, \underline{\texttt{ret}}) \equiv \mathbb{L}([\mathrm{g5}, \mathrm{x}], @\mathrm{g5}, \sqcup)$:
   1.1 find definition site for g5;
   1.2 then, resume search for x since that is lexical scope of its def'n.
2. $\mathbb{L}([\mathrm{g5}, \mathrm{x}], @\mathrm{g5}, \sqcup) \equiv \mathbb{L}([\mathrm{gret}, \mathrm{x}], @\mathrm{gret}, \underline{\mathrm{g5}})$
3. $\mathbb{L}([\mathrm{gret}, \mathrm{x}], @\mathrm{gret}, \underline{\mathrm{g5}}) \equiv \mathbb{L}([\mathrm{x}], @\texttt{fun x}, \underline{\mathrm{g5}})$
4. $\mathbb{L}([\mathrm{x}], @\texttt{fun x}, \underline{\mathrm{g5}}) \equiv 5$

Non-Local Variables

```
let g =
  (fun x ->
    let gret = (fun y ->
                 let gyret = x + y in gyret) in gret) in
let g5 = g 5 in
let ret = g5 1 in ret
```

1. $\ldots \mathbb{L}([\mathtt{x}], @\mathtt{gyret}, \llcorner \mathtt{ret} \lrcorner) \equiv \mathbb{L}([\mathtt{g5}, \mathtt{x}], @\mathtt{g5}, \llcorner \quad \lrcorner)$:
    1.1 find definition site for g5;
    1.2 then, resume search for x since that is lexical scope of its def'n.
2. $\mathbb{L}([\mathtt{g5}, \mathtt{x}], @\mathtt{g5}, \llcorner \quad \lrcorner) \equiv \mathbb{L}([\mathtt{gret}, \mathtt{x}], @\mathtt{gret}, \llcorner \mathtt{g5} \lrcorner)$
3. $\mathbb{L}([\mathtt{gret}, \mathtt{x}], @\mathtt{gret}, \llcorner \mathtt{g5} \lrcorner) \equiv \mathbb{L}([\mathtt{x}], @\mathtt{fun\ x}, \llcorner \mathtt{g5} \lrcorner)$
4. $\mathbb{L}([\mathtt{x}], @\mathtt{fun\ x}, \llcorner \mathtt{g5} \lrcorner) \equiv 5$

General Lookup signature: $\mathbb{L}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \llcorner \vdots \lrcorner) \equiv v$.

Peek at Full Rules for Functional Core

$$\textsc{Value Discovery}\ \frac{\textsc{First}(x, \textsc{Cl}(x), C)}{\mathbb{L}([x], (x = v), C) \equiv v} \qquad \textsc{Value Discard}\ \frac{\mathbb{L}(X, \textsc{Pred}(x), C) \equiv v}{\mathbb{L}([x] \mid\mid X, (x = f), C) \equiv v}$$

$$\textsc{Alias}\ \frac{\mathbb{L}([x'] \mid\mid X, \textsc{Pred}(x), C) \equiv v}{\mathbb{L}([x] \mid\mid X, (x = x'), C) \equiv v}$$

$$\textsc{Function Enter Parameter}\ \frac{c = (x_r = x_f\ x_v) \qquad \mathbb{L}([x_v] \mid\mid X, \textsc{Pred}(c), C) \equiv v \qquad \mathbb{L}([x_f], \textsc{Pred}(c), C) \equiv [\texttt{fun}\ x \texttt{ ->}] \mid\mid e}{\mathbb{L}([x] \mid\mid X, (\texttt{fun}\ x \texttt{ ->}), [c] \mid\mid C) \equiv v}$$

$$\textsc{Function Enter Non-Local}\ \frac{\begin{array}{c} x'' \neq x \qquad c = (x_r = x_f\ x_v) \\ \mathbb{L}([x_f, x] \mid\mid X, \textsc{Pred}(c), C) \equiv v \qquad \mathbb{L}([x_f], \textsc{Pred}(c), C) \equiv [\texttt{fun}\ x'' \texttt{ ->}] \mid\mid e \end{array}}{\mathbb{L}([x] \mid\mid X, (\texttt{fun}\ x'' \texttt{ ->}), [c] \mid\mid C) \equiv v}$$

$$\textsc{Function Exit}\ \frac{\begin{array}{c} \mathbb{L}([x'] \mid\mid X, (x' = b), [\textsc{Cl}(x)] \mid\mid C) \equiv v \\ \textsc{RetCl}(e) = (x' = b) \qquad \mathbb{L}([x_f], \textsc{Pred}(c), C) \equiv [\texttt{fun}\ x'' \texttt{ ->}] \mid\mid e \end{array}}{\mathbb{L}([x] \mid\mid X, (x = x_f\ x_v), C) \equiv v}$$

$$\textsc{Skip}\ \frac{x'' \neq x \qquad \mathbb{L}([x] \mid\mid X, \textsc{Pred}(x''), C) \equiv v \qquad \exists v_0 .\ \mathbb{L}([x''], \textsc{Cl}(x''), C) \equiv v_0}{\mathbb{L}([x] \mid\mid X, (x'' = b), C) \equiv v}$$

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{\mathrm{S}}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \overset{\vdots}{\sqcup}) \equiv \overset{\vdots}{\sqcup} x$ over Φ

- Lookup returns a variable activation now: a pair $\overset{\vdots}{\sqcup} x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
     (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{\mathrm{S}}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \overset{\vdots}{\sqcup}) \equiv \overset{\vdots}{\sqcup} x$ over Φ

- Lookup returns a variable activation now: a pair $\overset{\vdots}{\sqcup} x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
     (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

1. $\mathbb{L}^{\mathrm{S}}([\mathtt{f1}], @\mathtt{f1}, \sqcup)$

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{\mathrm{S}}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \underset{\vdots}{\sqcup}) \equiv \underset{\vdots}{\sqcup}x$ over Φ

- Lookup returns a variable activation now: a pair $\underset{\vdots}{\sqcup}x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
     (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

1. $\mathbb{L}^{\mathrm{S}}([\texttt{f1}], @\texttt{f1}, \sqcup) \equiv \mathbb{L}^{\mathrm{S}}([\texttt{fret}], @\texttt{fret}, \underset{\texttt{f1}}{\sqcup})$

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{S}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \underbrace{\;\vdots\;}) \equiv \underbrace{\;\vdots\;}x$ over Φ

- Lookup returns a variable activation now: a pair $\underbrace{\;\vdots\;}x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
     (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

1. $\mathbb{L}^{S}([\mathtt{f1}], @\mathtt{f1}, \underbrace{\;\;\;}) \equiv \mathbb{L}^{S}([\mathtt{fret}], @\mathtt{fret}, \underbrace{\mathtt{f1}})$
2. $\equiv \underbrace{\mathtt{f1}}\mathtt{fret};\ (\underbrace{\mathtt{f1}}\mathtt{fret} = \underline{\mathbb{L}([\mathtt{x}], @\mathtt{fun\ x}, \underbrace{\mathtt{f1}})} + 1) \in \Phi$

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{S}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \underline{\;\vdots\;}) \equiv \overset{\vdots}{\sqcup} x$ over Φ

- Lookup returns a variable activation now: a pair $\overset{\vdots}{\sqcup} x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
     (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

1. $\mathbb{L}^{S}([\texttt{f1}], @\texttt{f1}, \sqcup) \equiv \mathbb{L}^{S}([\texttt{fret}], @\texttt{fret}, \underline{\texttt{f1}})$
2. $\equiv \overset{\texttt{f1}}{\sqcup}\texttt{fret}$; $(\overset{\texttt{f1}}{\sqcup}\texttt{fret} = \underline{\mathbb{L}([\texttt{x}], @\texttt{fun x}, \underline{\texttt{f1}})} + 1) \in \Phi$
3. $\mathbb{L}([\texttt{x}], @\texttt{fun x}, \underline{\texttt{f1}}) \equiv \mathbb{L}([\texttt{y}], \texttt{f}, \sqcup)$

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{S}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \overset{\vdots}{\sqcup}) \equiv \overset{\vdots}{\sqcup}x$ over Φ

- Lookup returns a variable activation now: a pair $\overset{\vdots}{\sqcup}x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
    (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

1. $\mathbb{L}^{S}([\texttt{f1}], @\texttt{f1}, \sqcup) \equiv \mathbb{L}^{S}([\texttt{fret}], @\texttt{fret}, \overset{\texttt{f1}}{\sqcup})$
2. $\equiv \overset{\texttt{f1}}{\sqcup}\texttt{fret};\ (\overset{\texttt{f1}}{\sqcup}\texttt{fret} = \underline{\mathbb{L}([\texttt{x}], @\texttt{fun x}, \overset{\texttt{f1}}{\sqcup})} + 1) \in \Phi$
3. $\mathbb{L}([\texttt{x}], @\texttt{fun x}, \overset{\texttt{f1}}{\sqcup}) \equiv \mathbb{L}([\texttt{y}], \texttt{f}, \sqcup)$
4. $\equiv \mathbb{L}([\texttt{y}], @\texttt{f}, \sqcup) \equiv \mathbb{L}([\texttt{y}], @\texttt{y}, \sqcup) \equiv \sqcup\texttt{y}$

From Demand Interpreter to Demand Symbolic Evaluation

Symbolic lookup: $\mathbb{L}^{\mathrm{S}}([x_{f_1}, \ldots, x_{f_n}, x], @x_{pp}, \overset{\vdots}{\sqcup}) \equiv \overset{\vdots}{\sqcup}x$ over Φ

- Lookup returns a variable activation now: a pair $\overset{\vdots}{\sqcup}x$
- Φ equationally constrains variables, must be satisfiable

```
let y = input in
let f =
     (fun x ->
        let fret = x + 1 in fret) in
let f1 = f y in ret
let ret = f f1 in ret
```

1. $\mathbb{L}^{\mathrm{S}}([\texttt{f1}], @\texttt{f1}, \sqcup) \equiv \mathbb{L}^{\mathrm{S}}([\texttt{fret}], @\texttt{fret}, \overset{\texttt{f1}}{\sqcup})$
2. $\equiv \overset{\texttt{f1}}{\sqcup}\texttt{fret}$; $(\overset{\texttt{f1}}{\sqcup}\texttt{fret} = \underline{\mathbb{L}([\texttt{x}], @\texttt{fun x}, \overset{\texttt{f1}}{\sqcup})} + 1) \in \Phi$
3. $\mathbb{L}([\texttt{x}], @\texttt{fun x}, \overset{\texttt{f1}}{\sqcup}) \equiv \mathbb{L}([\texttt{y}], \texttt{f}, \sqcup)$
4. $\equiv \mathbb{L}([\texttt{y}], @\texttt{f}, \sqcup) \equiv \mathbb{L}([\texttt{y}], @\texttt{y}, \sqcup) \equiv \sqcup\texttt{y}$
5. Final $\Phi = \{\overset{\texttt{f1}}{\sqcup}\texttt{fret} = \sqcup\texttt{y} + 1\}$; satisfiable.

Formal Development

- **Theorem:** Demand operational semantics
  $$\equiv$$
  Forward operational semantics
- **Theorem:** DDSE is sound and complete with respect to operational semantics
- Several subtle issues had to be skipped in talk:
  1. Call stack must be inferred when lookup initiated in middle of program
  2. Input order of demand-driven lookup is not forward order; sorting step needed

DDSE Implementation

- Artifact is a test generator: given program and target line, search for inputs which reach the target line of code
- Initial proof-of-concept implementation in OCaml
- Need to dovetail on different search paths
  $\Rightarrow$ coroutine/nondeterminism monad used
- Successfully solves all benchmarks from Cruanes [CADE '17], a higher-order forward symbolic evaluator implementation; see paper for details

Comparison with Select Related Work

- Snugglebug, PLDI '09: *Imperative* demand symbolic execution, no correctness
- Cruanes, Satisfiability Modulo Bounded Checking, CADE '17: Functional *forward* symbolic execution, no correctness proof, no input, no unbounded recursion
- Rosette, PLDI '14: a *forward* symbolic execution framework implementation; bounded datatypes only
- This work: **functional**, **demand**, **arbitrary** datatypes and recursion, **proven** sound and complete.