
IJ


PS


XCODE


int main(void) {
return 0;
}
static main () {
System.out.print;
}
<?php
?>
print
exit 0;


PC

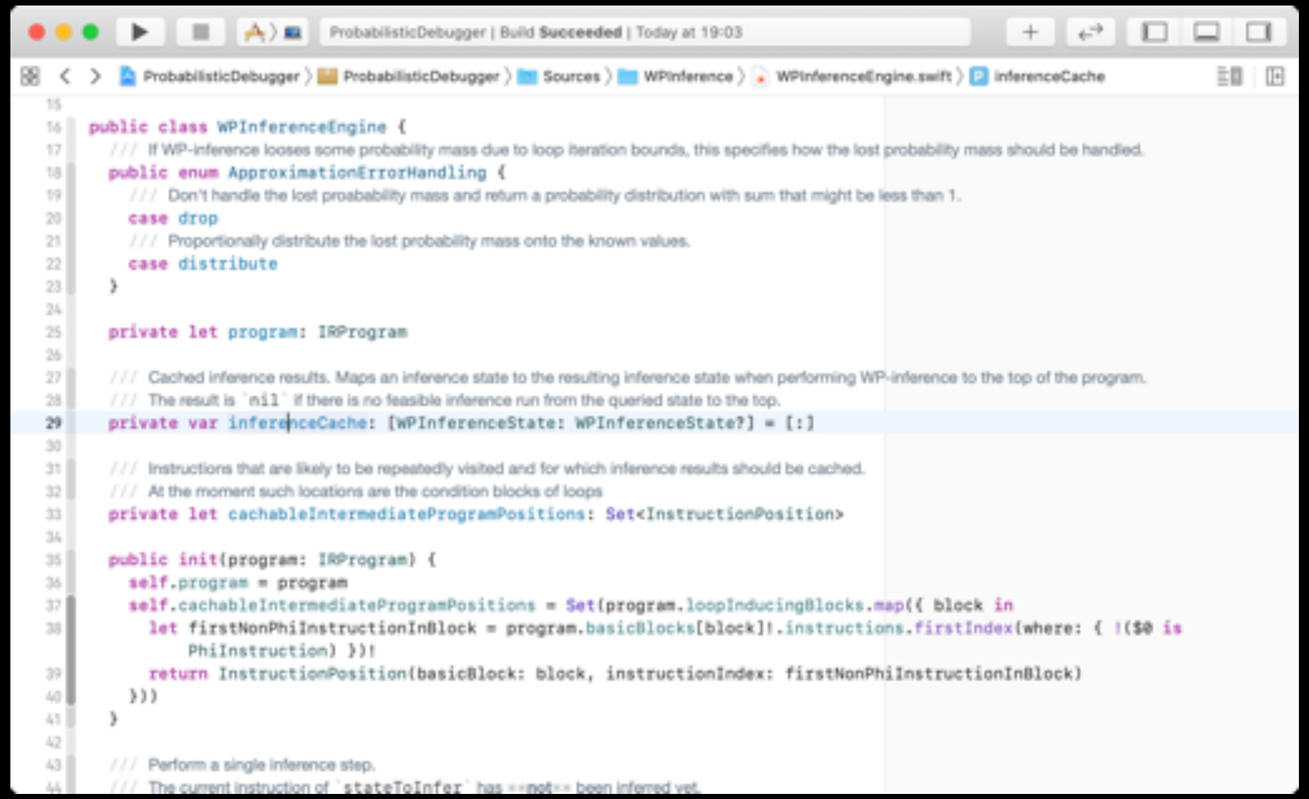

Vim

Alex Hoppen

Aachen, 30.7.2020

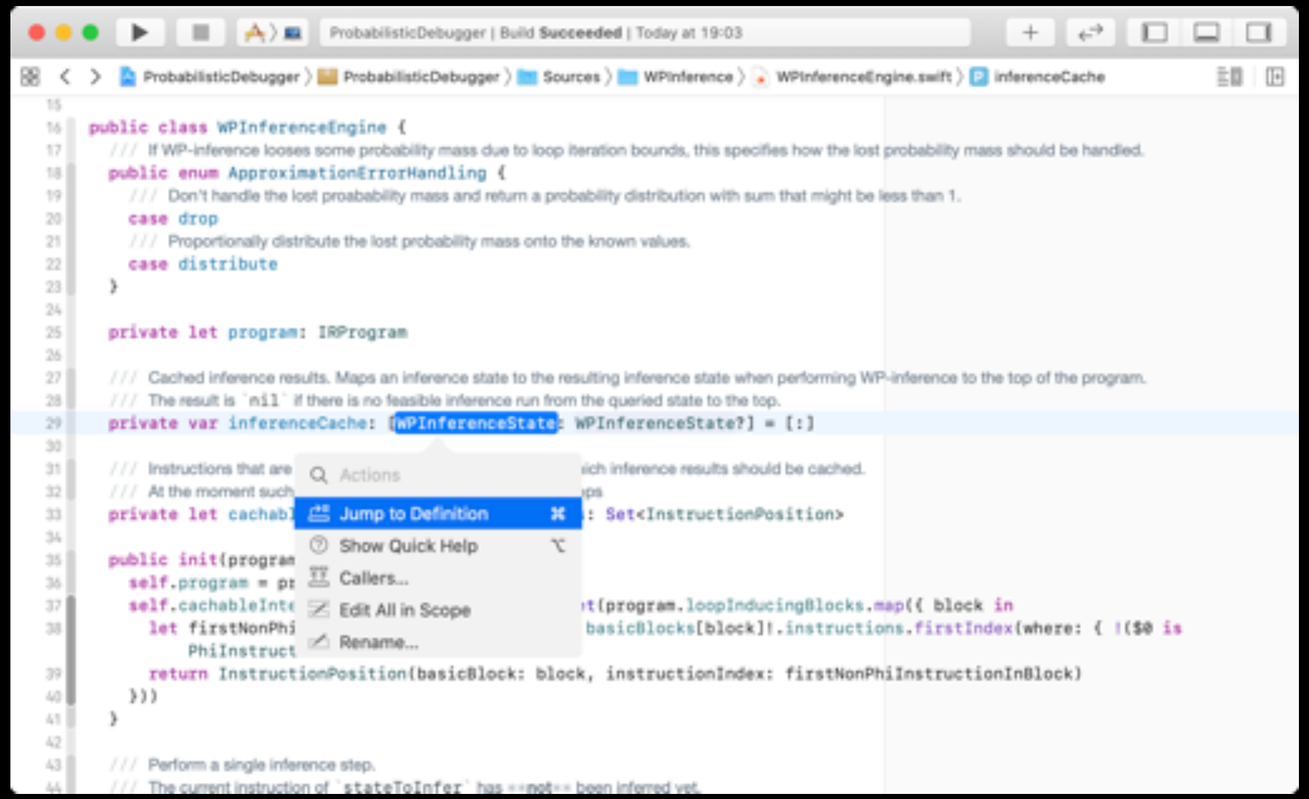
# Probabilistic Programming IDE – A Feasibility Study

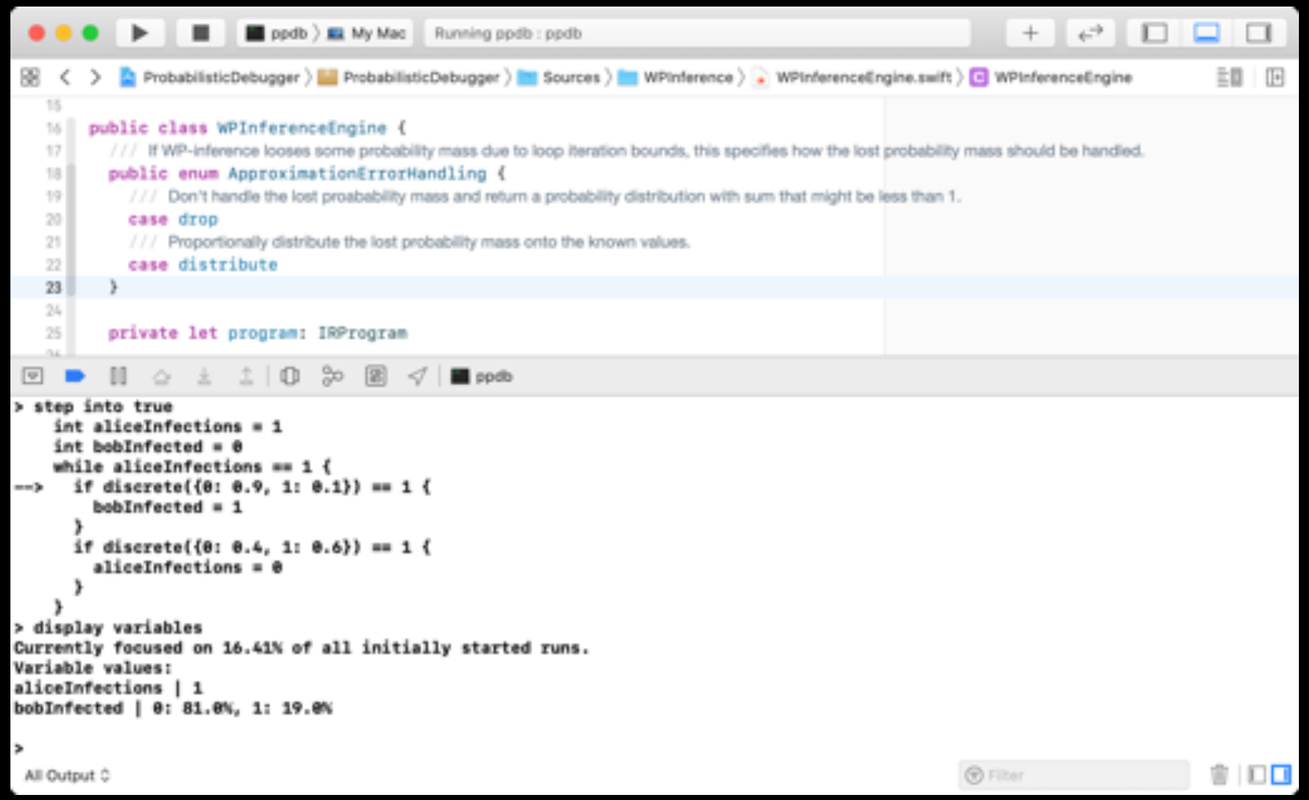
# Syntax highlighting



Syntax highlighting

Static semantic functionality

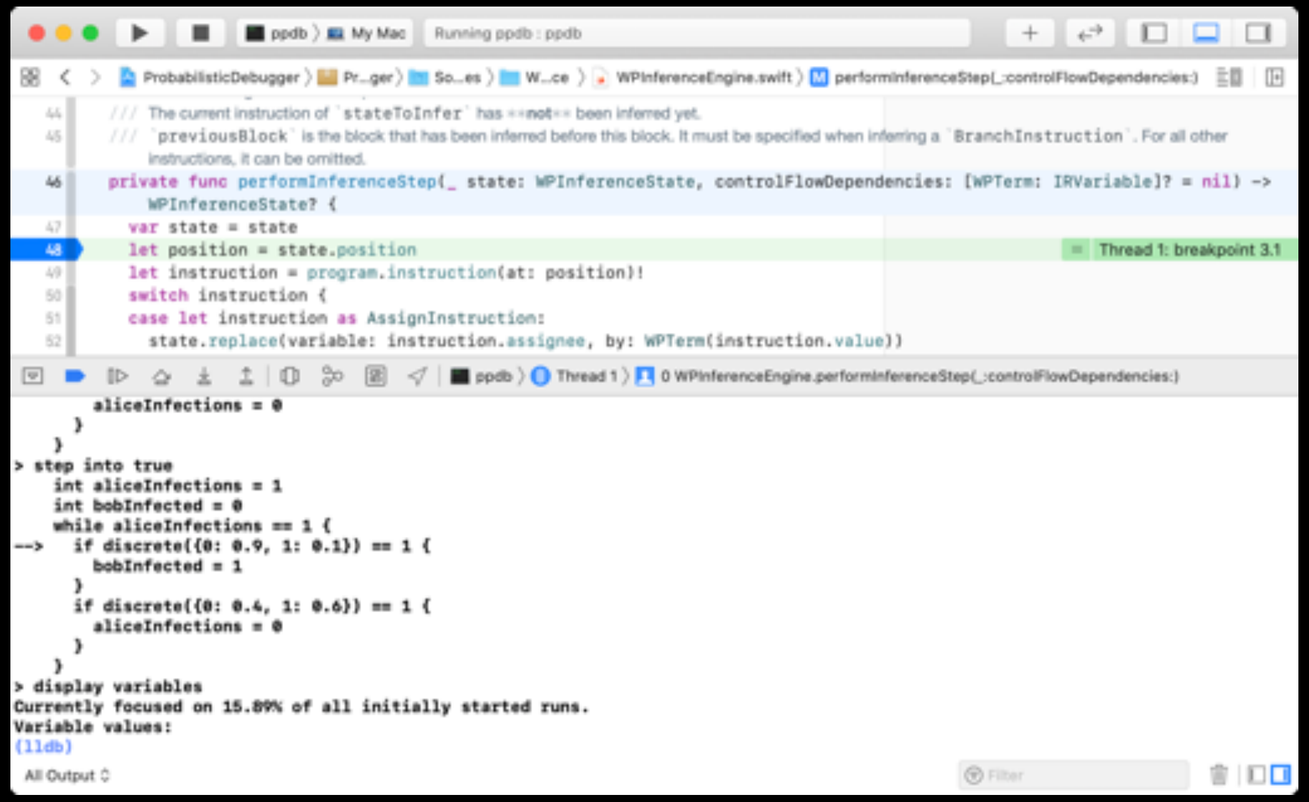





Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

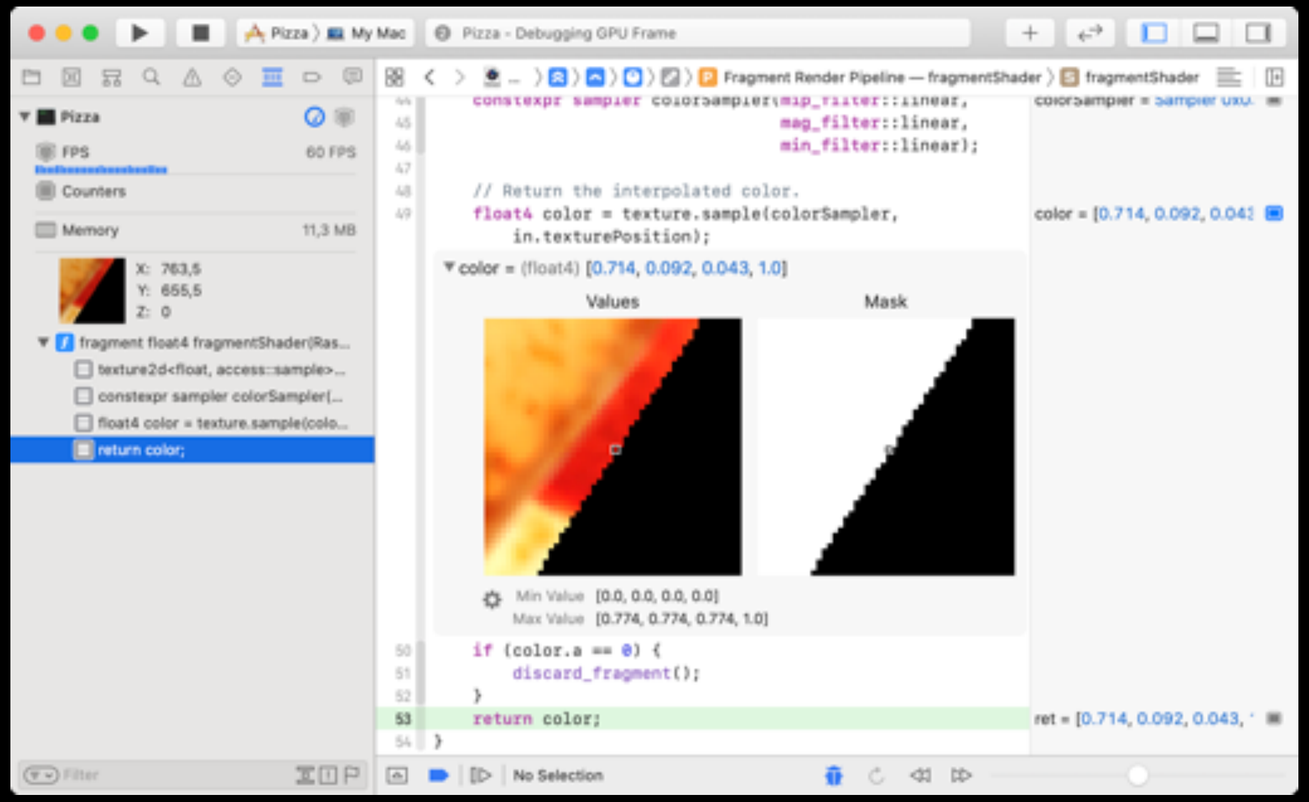



Syntax highlighting

Static semantic functionality

Program execution

Debugger




Syntax highlighting

Static semantic functionality

Program execution

Debugger

Recording-based debugger

```
1 int x = 20
2 int y = 1
3 while x > 0 {
4   x -= 1
5   y = 2 * y
6 }
```

Syntax highlighting

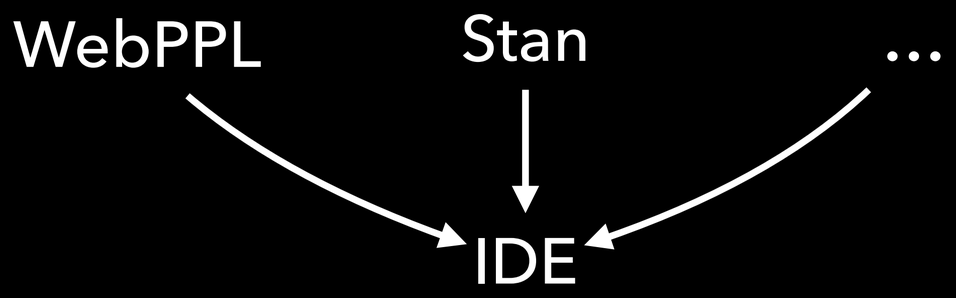
Static semantic functionality

Program execution

Debugger

Recording-based debugger

Slicing




Syntax highlighting

Static semantic functionality

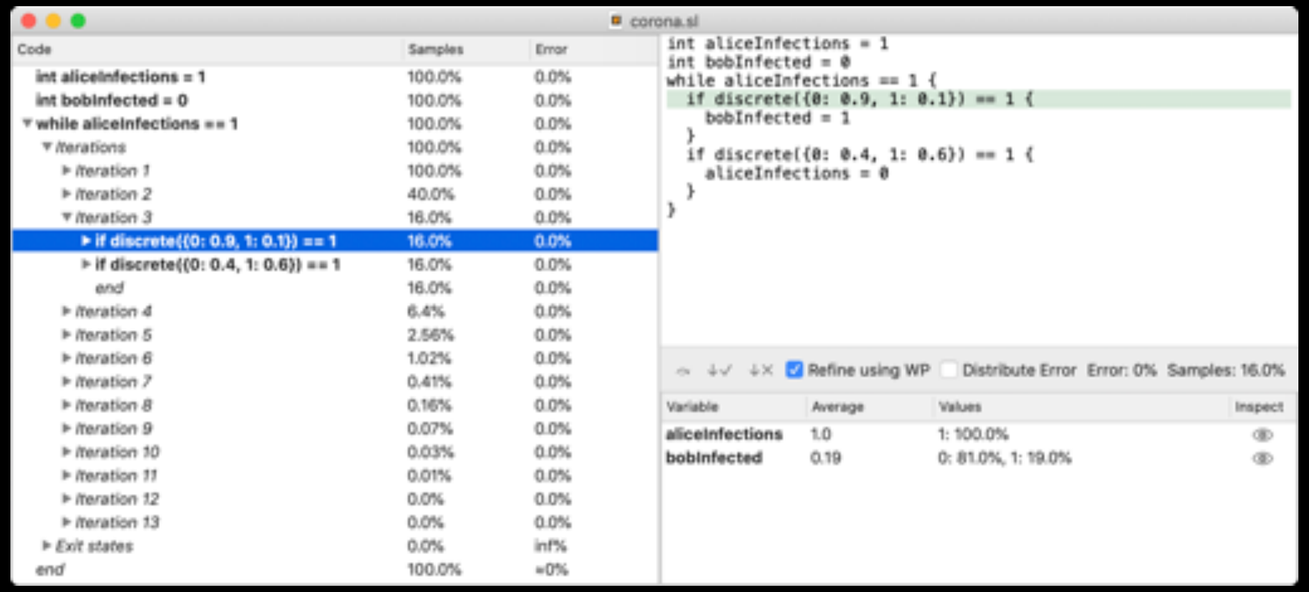
Program execution

Debugger

Recording-based debugger

Slicing

Support for multiple languages




Syntax highlighting

Static semantic functionality

Program execution

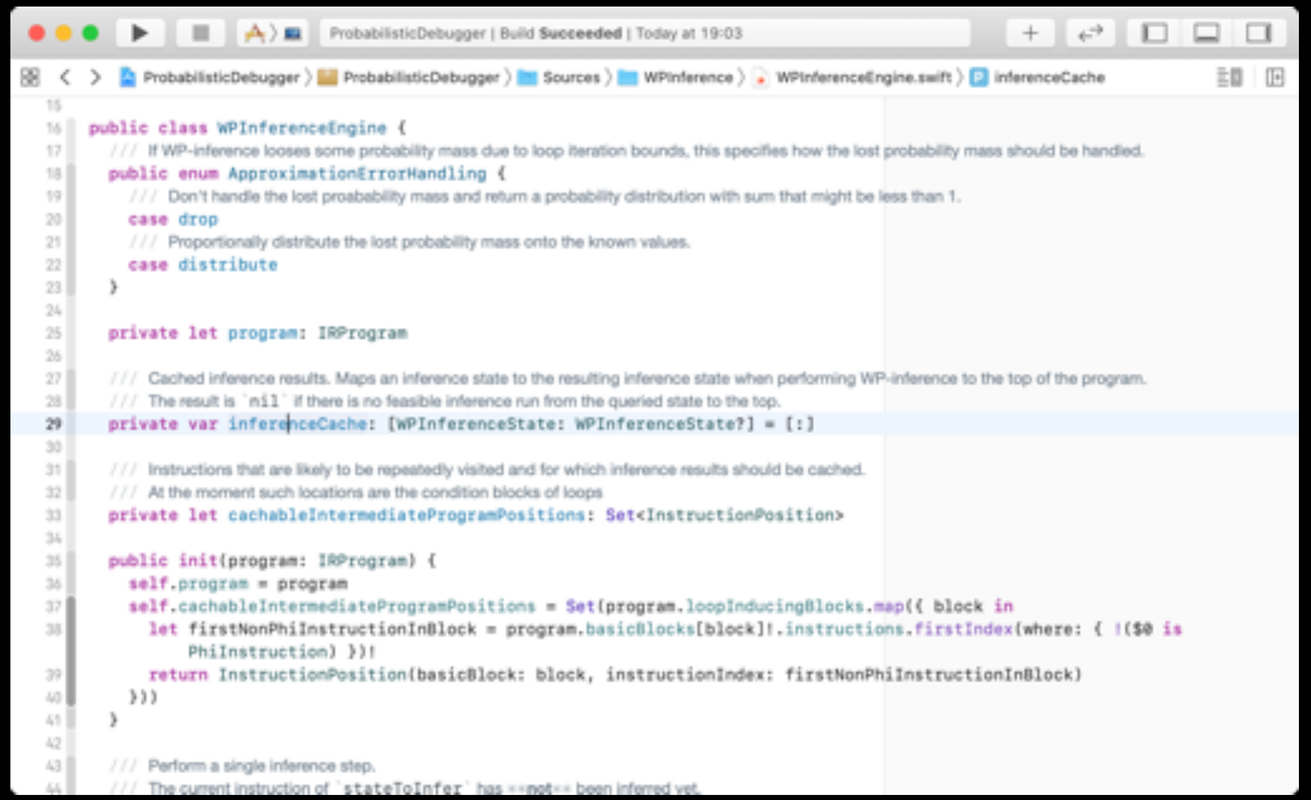
Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation




Syntax highlighting

Static semantic functionality

Program execution

Debugger

Recording-based debugger

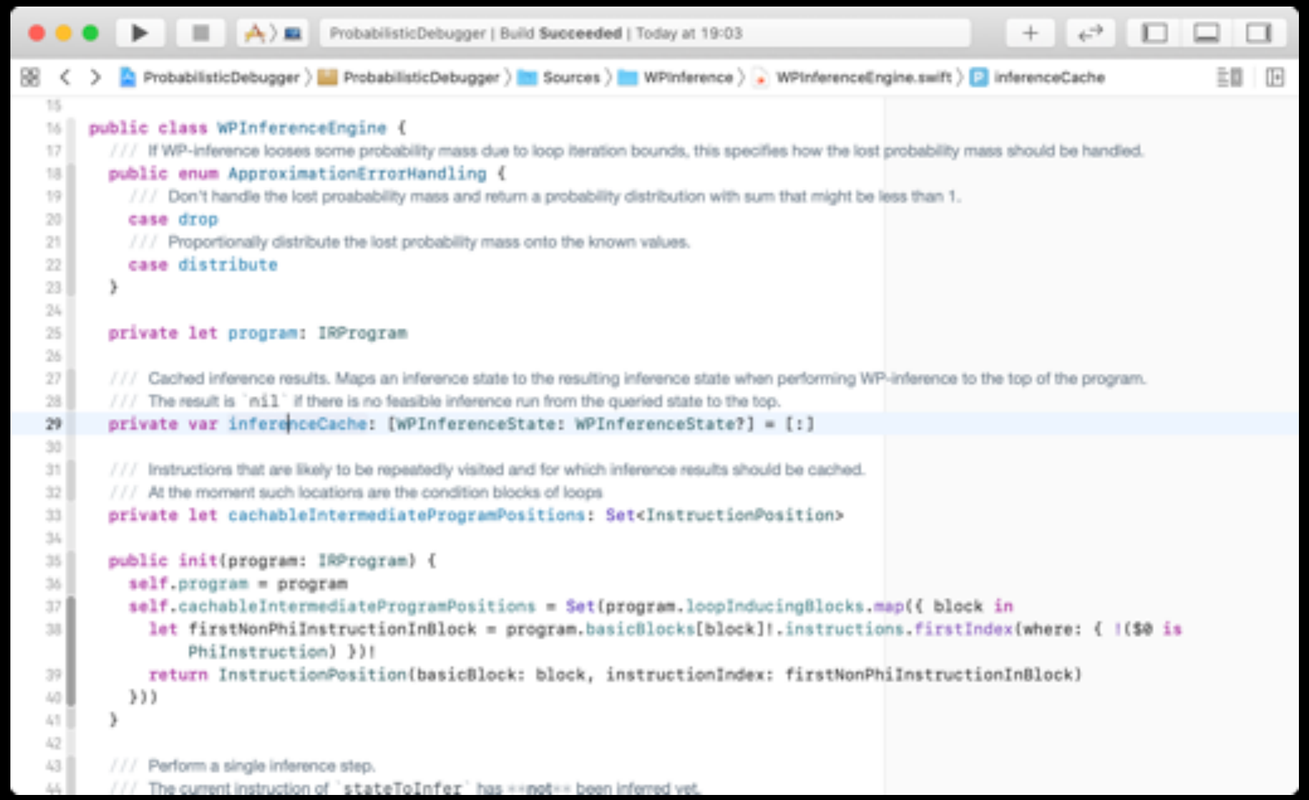
Slicing

Support for multiple languages

Implementation

# Syntax Highlighting

- Additional constructs only extend grammar in a trivial way
  - Probabilistic choice
  - Observe
- Existing techniques also work for probabilistic programming languages
  - Grammar-based syntax highlighting
  - Sublime Text Package for Stan already exists




Syntax highlighting

Static semantic functionality

Program execution

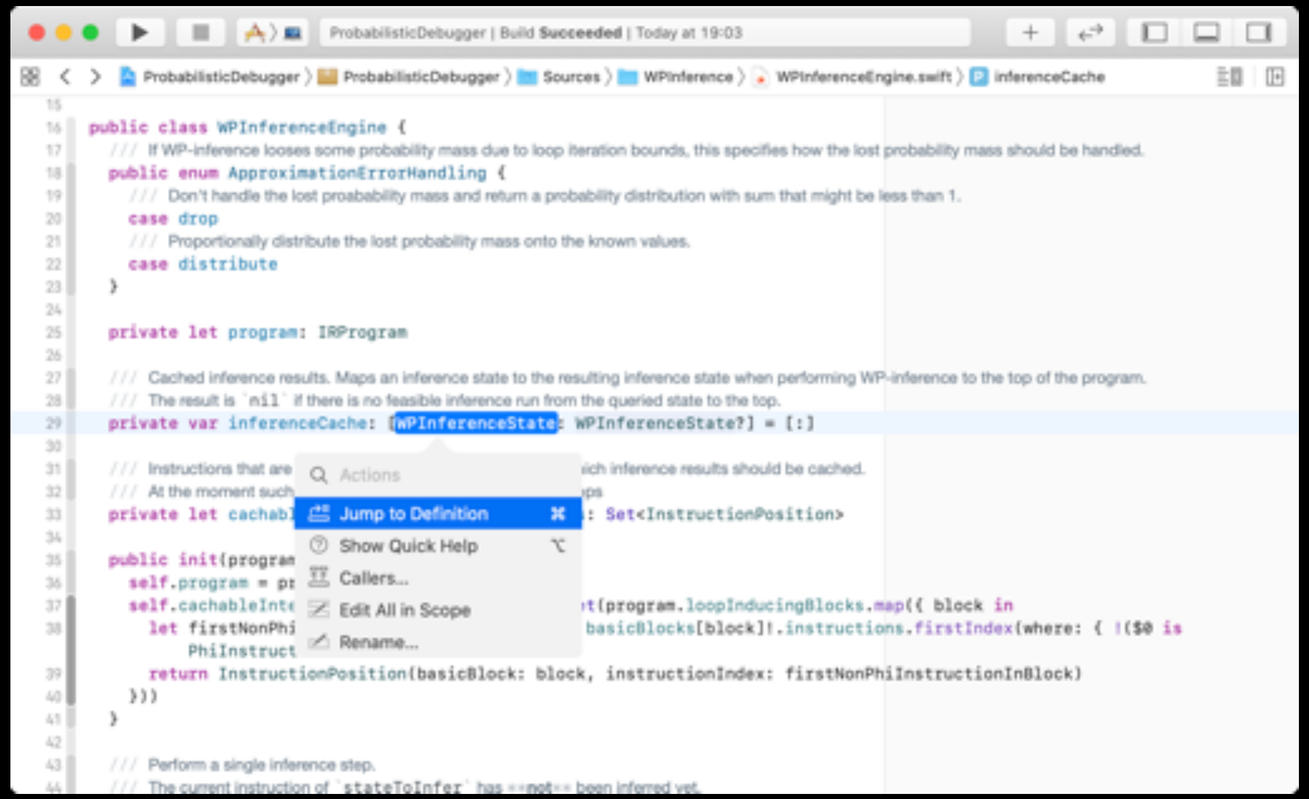
Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation




Syntax highlighting

Static semantic functionality

Program execution

Debugger

Recording-based debugger

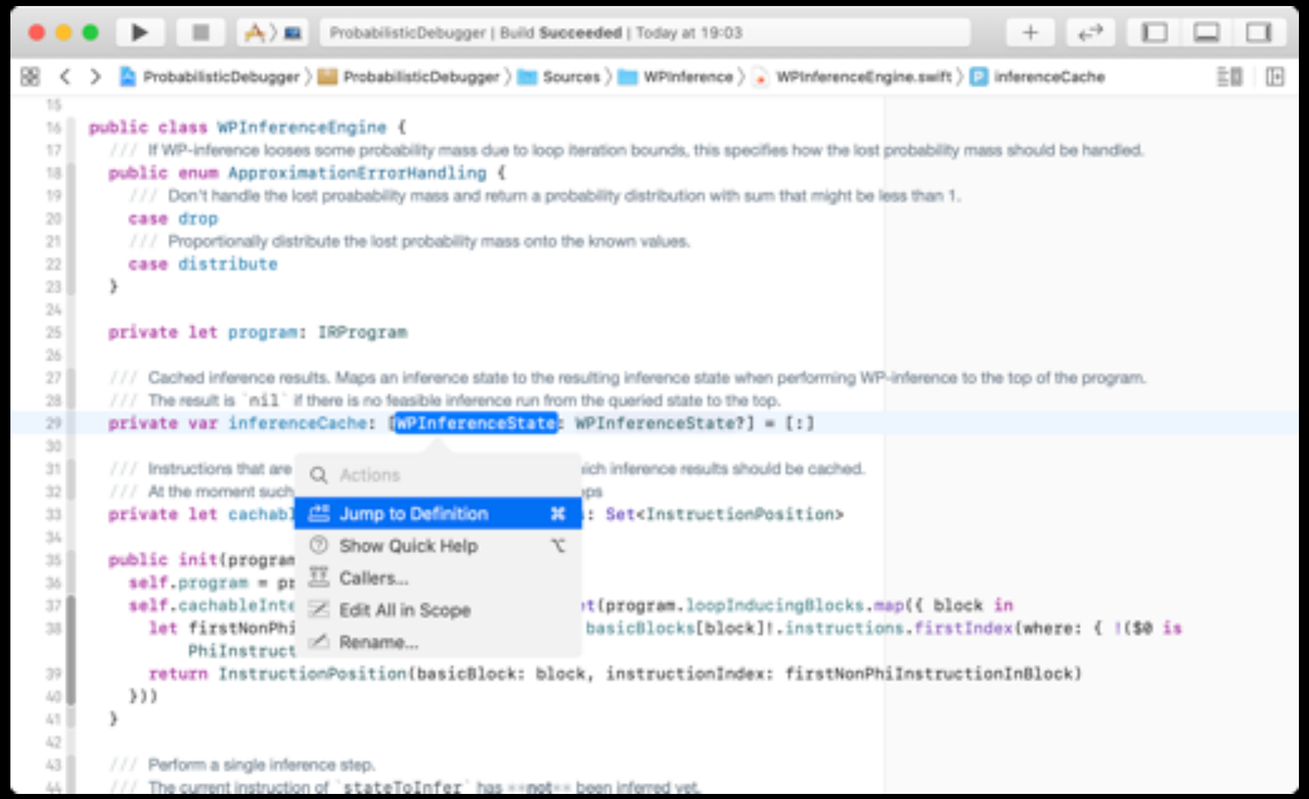
Slicing

Support for multiple languages

Implementation

# Static Semantic Functionality

- Additional constructs don’t add significant static complexity
  - Probabilistic choice
  - Observe
- Traditional approaches can easily be copied




Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

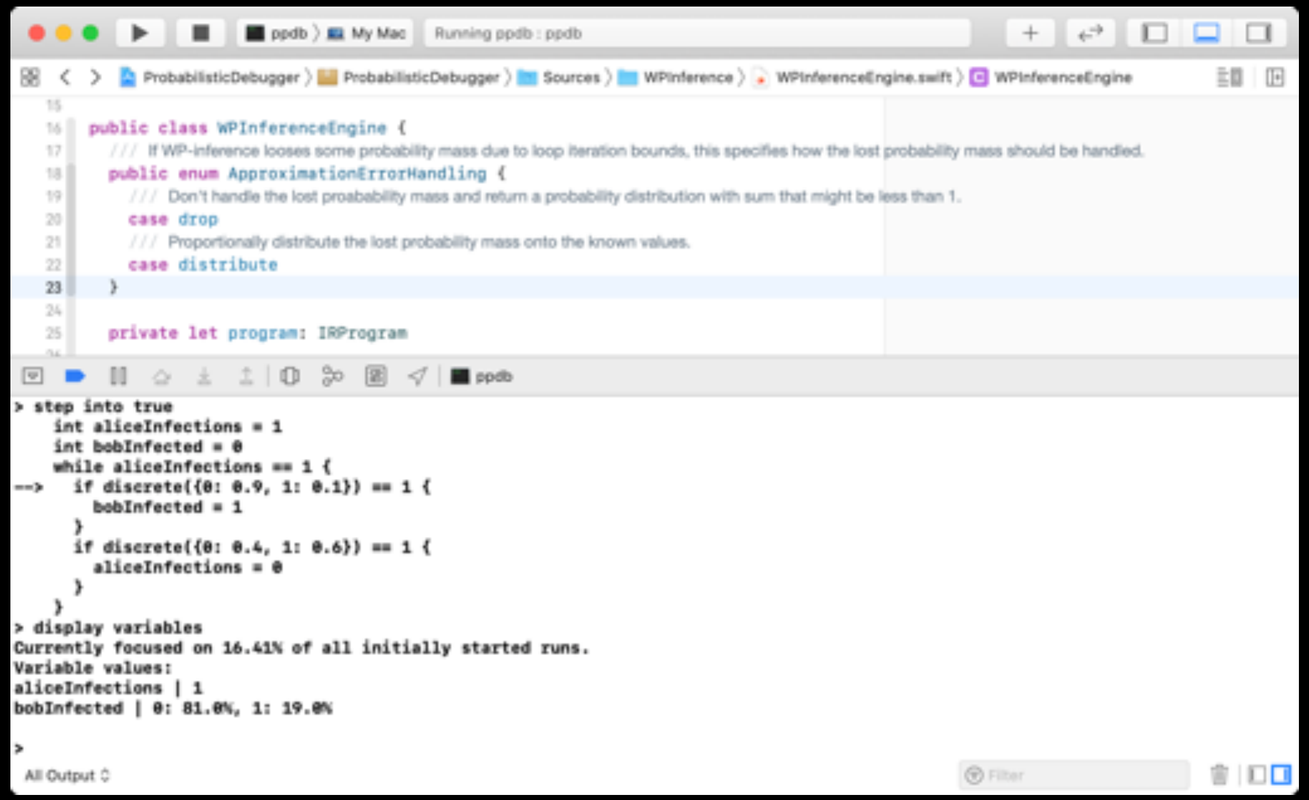
Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✓

Implementation ✓




Syntax highlighting

Static semantic functionality

Program execution

Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation

| $P$ | $\mathsf{wp}(P, f)$ |
|---|---|
| $()$ | $f$ |
| $(P_1, P_2)$ | $\mathsf{wp}(P_1, \mathsf{wp}(P_2, f))$ |
| $(\texttt{int} \mid \texttt{bool} \mid \varepsilon)$ *var* = *expr* | $f[\mathit{var} := \mathit{expr}]$ |
| if *expr* { $P_{\text{if}}$ } else { $P_{\text{else}}$ } | $[\![\mathit{expr}]\!] \cdot \mathsf{wp}(P_{\text{if}}, f) + [\![\neg \mathit{expr}]\!] \cdot \mathsf{wp}(P_{\text{else}}, f)$ |
| while *expr* { $P_{\text{body}}$ } | $\mathsf{lfp}\ X.([\![\mathit{expr}]\!] \cdot \mathsf{wp}(P_{\text{body}}, X) + [\![\neg \mathit{expr}]\!] \cdot f)$ |
| observe *expr* | $[\![\mathit{expr}]\!] \cdot f$ |

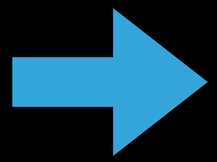
```
int x = 1
```

$$\mathsf{wp}(P, [\![\mathsf{x} = \xi]\!])$$
$$= [\![1 = \xi]\!]$$

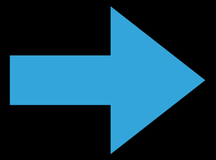
| $P$ | $\mathsf{wlp}(P, f)$ |
|---|---|
| $()$ | $f$ |
| $(P_1, P_2)$ | $\mathsf{wp}(P_1, \mathsf{wp}(P_2, f))$ |
| $(\texttt{int} \mid \texttt{bool} \mid \varepsilon)$ *var* = *expr* | $f[\mathit{var} \coloneqq \mathit{expr}]$ |
| if *expr* { $P_{\text{if}}$ } else { $P_{\text{else}}$ } | $[\![\mathit{expr}]\!] \cdot \mathsf{wp}(P_{\text{if}}, f) + [\![\neg \mathit{expr}]\!] \cdot \mathsf{wp}(P_{\text{else}}, f)$ |
| while *expr* { $P_{\text{body}}$ } | $\mathsf{gfp}\ X.([\![\mathit{expr}]\!] \cdot \mathsf{wlp}(P_{\text{body}}, X) + [\![\neg \mathit{expr}]\!] \cdot f)$ |
| observe *expr* | $[\![\mathit{expr}]\!] \cdot f$ |

$$\frac{\mathsf{wp}(P, f)}{\mathsf{wlp}(P, 1)}$$

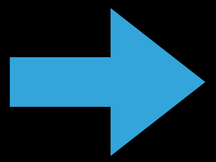
```
while cond {
   P_body
}
```



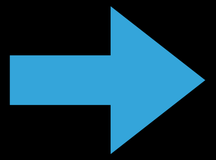
```
if cond {
   P_body
}
if cond {
   P_body
}
if cond {
   while true {}
}
```



$$[\![\neg \text{cond}]\!] \cdot f + [\![\text{cond}]\!] \cdot \text{wp}(P, f)$$



$$[\![\neg \text{cond}]\!] \cdot f + [\![\text{cond}]\!] \cdot \text{wp}(P, f)$$



$$[\![\neg \text{cond}]\!] \cdot f$$

$$\lambda^b([\![\neg \text{cond}]\!] \cdot f)$$

$$\lambda(f) = [\![\text{cond}]\!] \cdot \text{wp}(P, f) + [\![\neg \text{cond}]\!] \cdot f$$

$$
\begin{aligned}
&\mathsf{wp}(\text{while expr } \{\, P_{body} \,\}, f) \\
&= \mathsf{lfp}\, X \,.\, \left( [\![\text{expr}]\!] \cdot \mathsf{wp}(P_{body}, X) + [\![\neg \text{expr}]\!] \cdot f \right) \\
&= \mathsf{lfp}\, X \,.\, \varphi(X) \\
&= \lim_{b \to \infty} \varphi^b(0) \\
&\approx \varphi^b(0) \\
&= \lambda^{b-1}([\![\neg \text{cond}]\!] \cdot f)
\end{aligned}
$$

| $P$ | $\mathsf{woip}(P, f)$ |
|---|---|
| $()$ | $f$ |
| $(P_1, P_2)$ | $\mathsf{wp}(P_1, \mathsf{wp}(P_2, f))$ |
| $(\texttt{int} \mid \texttt{bool} \mid \varepsilon)\ \textit{var}\ =\ \textit{expr}$ | $f[\textit{var} := \textit{expr}]$ |
| $\texttt{if}\ \textit{expr}\ \{\ P_{\text{if}}\ \}\ \texttt{else}\ \{\ P_{\text{else}}\ \}$ | $[\![\textit{expr}]\!] \cdot \mathsf{wp}(P_{\text{if}}, f) + [\![\neg \textit{expr}]\!] \cdot \mathsf{wp}(P_{\text{else}}, f)$ |
| $\texttt{while}\ \textit{expr}\ \{\ P_{\text{body}}\ \}$ | $\mathsf{lfp}\ X.([\![\textit{expr}]\!] \cdot \mathsf{wp}(P_{\text{body}}, X) + [\![\neg \textit{expr}]\!] \cdot f)$ |
| $\texttt{observe}\ \textit{expr}$ | $f$ |

$$
\begin{aligned}
&\mathsf{wlp}(P,1) \\
&= \Pr(\text{term.} \wedge \text{obs.}) + \Pr(\neg\text{term.} \wedge \text{obs.}) \\
&= \Pr(\text{term.} \wedge \text{obs.}) + \Pr(\neg\text{term.} \wedge \text{obs.}) + \overbrace{\Pr(\neg\text{term.} \wedge \neg\text{obs.})}^{=0} \\
&= \Pr(\text{term.} \wedge \text{obs.}) + \big(1 - \Pr(\text{term.})\big) \\
&= \mathsf{wp}(P,1) \qquad\qquad + \big(1 - \mathsf{woip}(P,1)\big)
\end{aligned}
$$

| $P$ | $\mathsf{woip}(P, f)$ |
|---|---|
| $()$ | $f$ |
| $(P_1, P_2)$ | $\mathsf{wp}(P_1, \mathsf{wp}(P_2, f))$ |
| $(\texttt{int} \mid \texttt{bool} \mid \varepsilon)\ \textit{var}\ =\ \textit{expr}$ | $f[\textit{var} := \textit{expr}]$ |
| $\texttt{if}\ \textit{expr}\ \{\ P_{\text{if}}\ \}\ \texttt{else}\ \{\ P_{\text{else}}\ \}$ | $[\![\textit{expr}]\!] \cdot \mathsf{wp}(P_{\text{if}}, f) + [\![\neg \textit{expr}]\!] \cdot \mathsf{wp}(P_{\text{else}}, f)$ |
| $\texttt{while}\ \textit{expr}\ \{\ P_{\text{body}}\ \}$ | $\mathsf{lfp}\ X.([\![\textit{expr}]\!] \cdot \mathsf{wp}(P_{\text{body}}, X) + [\![\neg \textit{expr}]\!] \cdot f)$ |
| $\texttt{observe}\ \textit{expr}$ | $f$ |

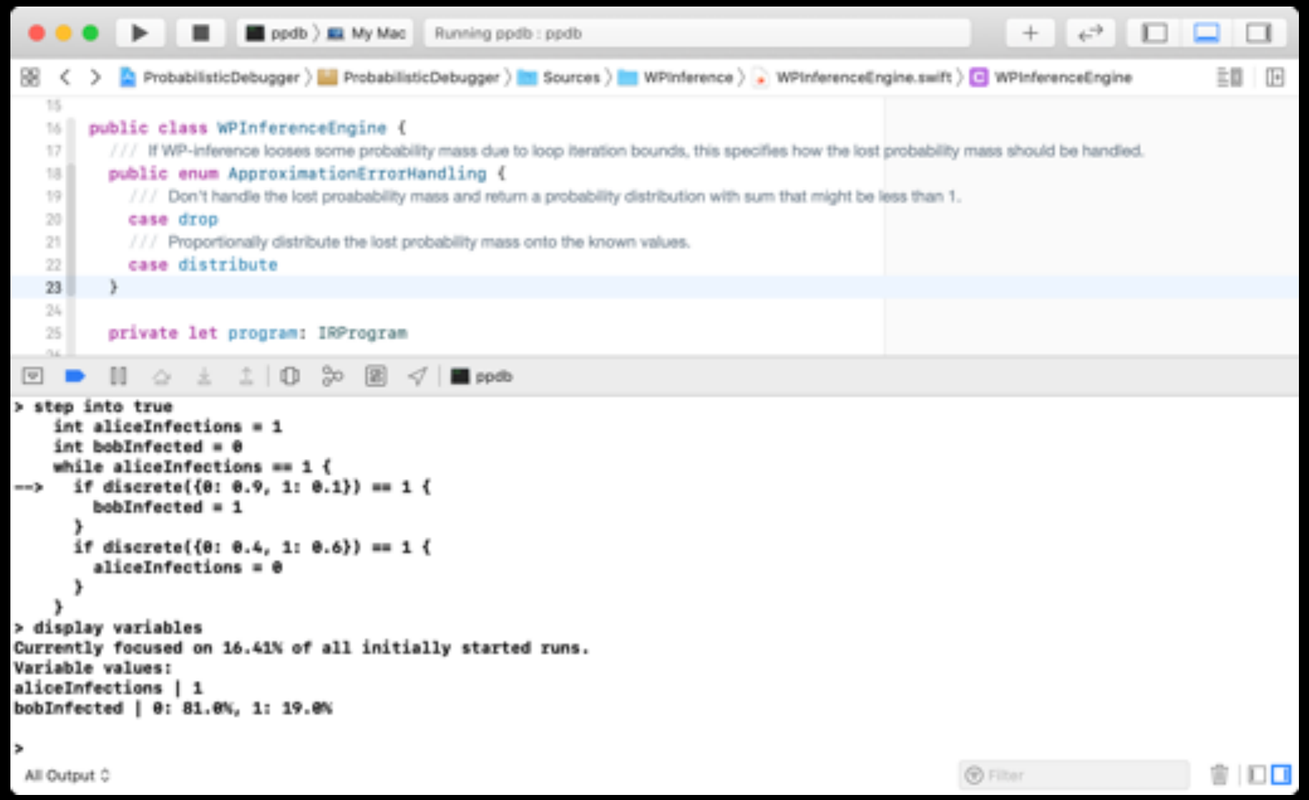
$$\mathsf{wlp}(P,1) = \mathsf{wp}(P,1) + \big(1 - \mathsf{woip}(P,1)\big)$$

$$\frac{\mathsf{wp}(P,f)}{\mathsf{wlp}(P,1)} = \frac{\mathsf{wp}(P,f)}{1 - \mathsf{woip}(P,1) + \mathsf{wp}(P,1)}$$

# How Many Times to Unroll a Loop?

- Execute program using sampling with $n$ samples
- Count how many times a loop is traversed
  - Use maximum as loop iteration bound
- For a.s.t. programs $\frac{1}{n}$ of all runs will not terminate within this bound
  - For 10 000 samples 99.99% of all runs terminate

$$\frac{\mathsf{wp}(P,f)}{\mathsf{wlp}(P,1)} = \frac{\mathsf{wp}_\beta(P,f)}{1-\mathsf{woip}_\beta(P,1)+\mathsf{wp}_\beta(P,1)}\left(\pm\frac{1-\mathsf{woip}_\beta(P,1)}{\mathsf{wp}_\beta(P,1)}\right)$$




Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

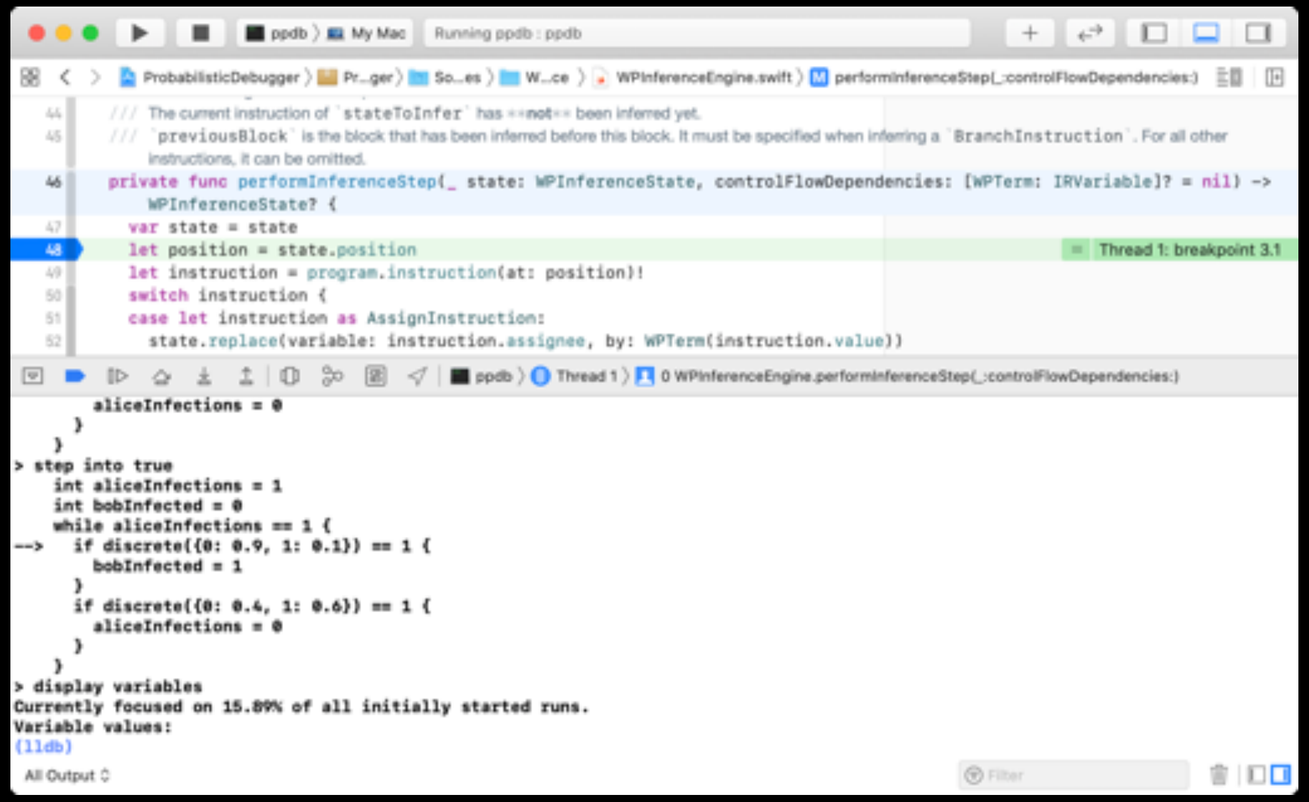
Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✓

Implementation ✓




Syntax highlighting

Static semantic functionality

Program execution

Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Continue | Step Over | Step Into | Step Out

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

Step Over

Step Into False

Step Into True

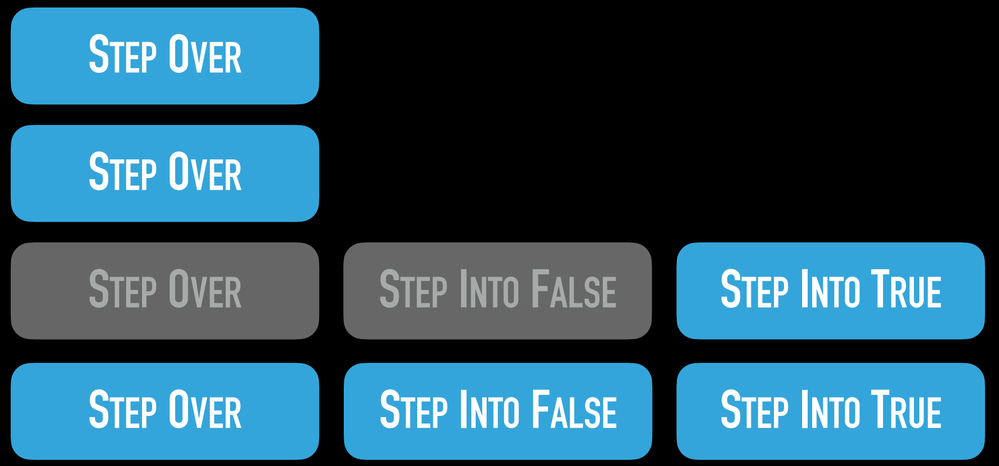
```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

Step Over | Step Into False | Step Into True

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```




```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over
Step Over
Step Over | Step Into False | Step Into True
Step Over | Step Into False | Step Into True

```
     100%
bob:
  true:  10%
  false: 90%
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

Step Over | Step Into False | Step Into True

Step Over | Step Into False | Step Into True

```
     90%
bob:
  true:   0%
  false:100%
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

Step Over | Step Into False | Step Into True

Step Over | Step Into False | Step Into True

Step Over

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

Step Over | Step Into False | Step Into True

Step Over | Step Into False | Step Into True

Step Over

```
     10%
bob:
  true: 100%
  false:  0%
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

Step Over

Step Over

Step Into True

Step Into True

Step Over

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

| | |
|---|---|
| Step Over | `bool alice = true` |
| Step Over | `bool bob = false` |
| Step Into True | `while alice` |
| Step Into True | `if prob(0.1)` |
| Step Over | `bob = true` |

Step Over `bool alice = true`

Step Over `bool bob = false`

Step Into True `while alice`

Step Into True `if prob(0.1)`

Step Over `bob = true`

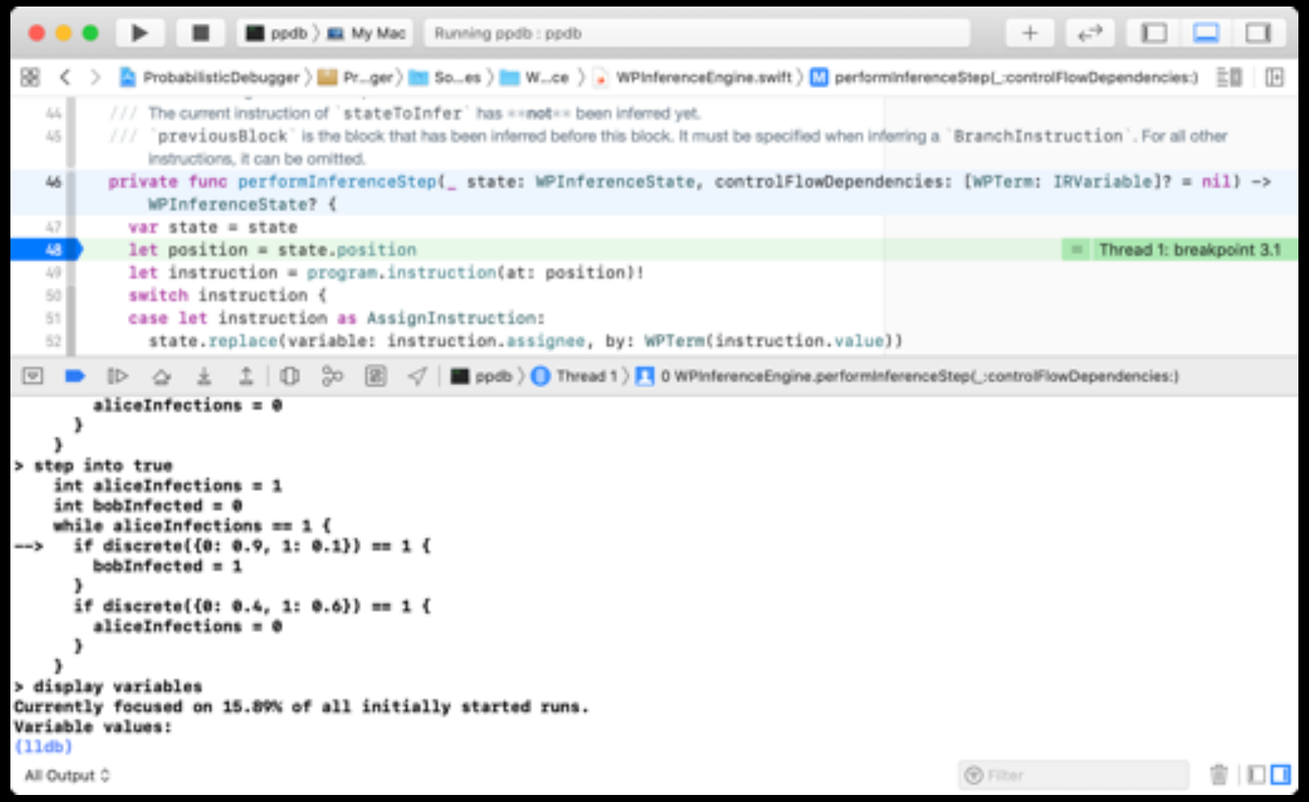
| $\text{wph}(h^P, f)$ | $\text{wph}(h^P, 1)$ | $\text{woiph}(h^P, 1)$ |
|---|---|---|
| $[\![\text{bob}]\!]$ | 1 | 1 |

| | | $\text{wph}(h^P, f)$ | $\text{wph}(h^P, 1)$ | $\text{woiph}(h^P, 1)$ |
|---|---|---|---|---|
| STEP OVER | `bool alice = true` | $0.1$ | $0.1$ | $1$ |
| STEP OVER | `bool bob = false` | $0.1 \cdot [\![\text{alice}]\!]$ | $0.1 \cdot [\![\text{alice}]\!]$ | $1$ |
| STEP INTO TRUE | `observe alice` | $0.1 \cdot [\![\text{alice}]\!]$ | $0.1 \cdot [\![\text{alice}]\!]$ | $1$ |
| STEP INTO TRUE | `observe prob(0.1)` | $0.1$ | $0.1$ | $1$ |
| STEP OVER | `bob = true` | $1$ | $1$ | $1$ |
| | | $[\![\text{bob}]\!]$ | $1$ | $1$ |

$$\frac{\text{wph}_\beta(h^P, f)}{1 - \text{woiph}_\beta(h^P, 1) + \text{wph}_\beta(h^P, 1)} \left( \pm \frac{1 - \text{woiph}_\beta(h^P, 1)}{\text{wph}_\beta(h^P, 1)} \right)$$

$$\frac{0.1}{1 - 1 + 0.1} \left( \pm \frac{1 - 1}{0.1} \right) = 100\,\% \, (\pm 0\,\%)$$

```
      10%
bob:
   true: 100%
   false:  0%
```




Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

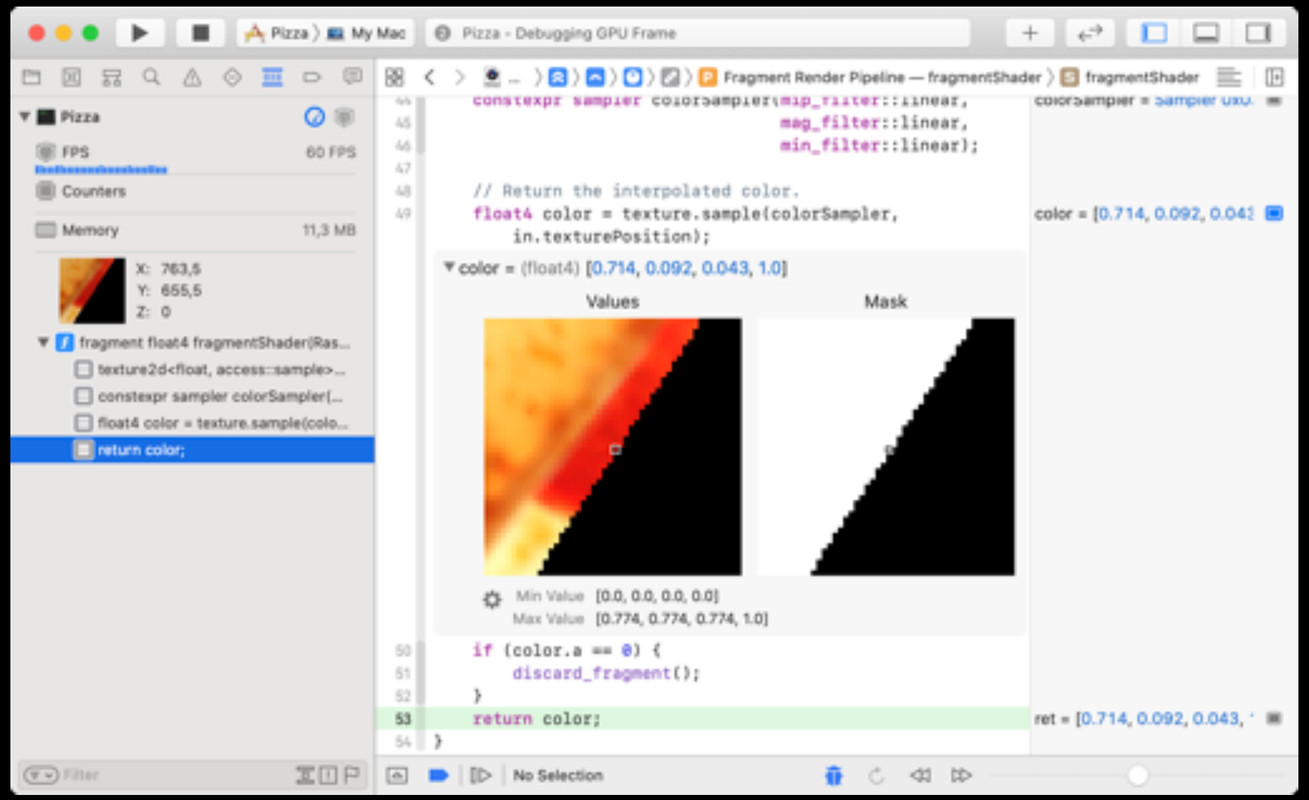
Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✓

Implementation ✓




Syntax highlighting

Static semantic functionality

Program execution

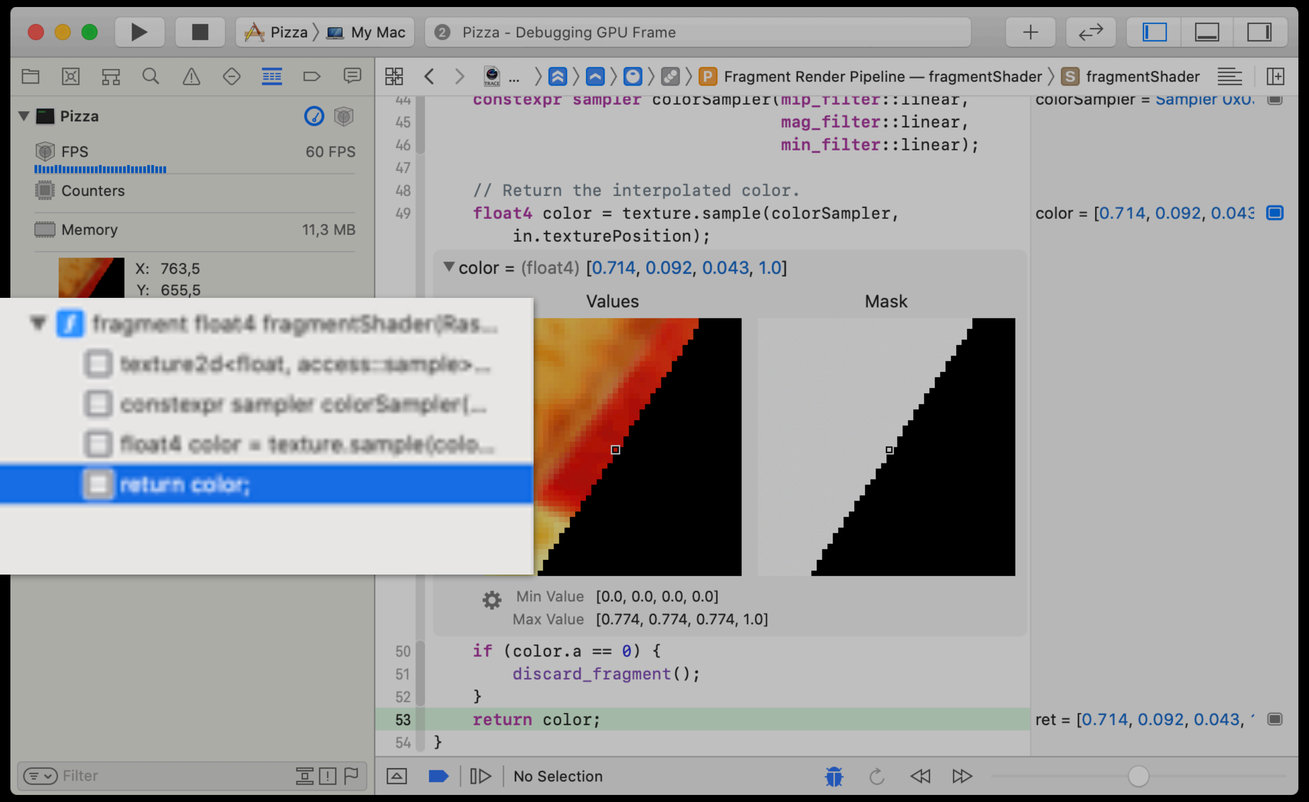
Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation




Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✓

Implementation ✓

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
bool alice = true
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
bool alice = true
bool bob = false
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
bool alice = true
bool bob = false
▼ while alice
▼ Iteration 1
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
  bool alice = true
  bool bob = false
▼ while alice
  ▼ Iteration 1
    ▼ if prob(0.1)
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
  bool alice = true
  bool bob = false
▼ while alice
  ▼ Iteration 1
    ▼ if prob(0.1)
        bob = true
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
  bool alice = true
  bool bob = false
▼ while alice
  ▼ Iteration 1
    ▼ if prob(0.1)
        bob = true
    ▼ if prob(0.6)
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

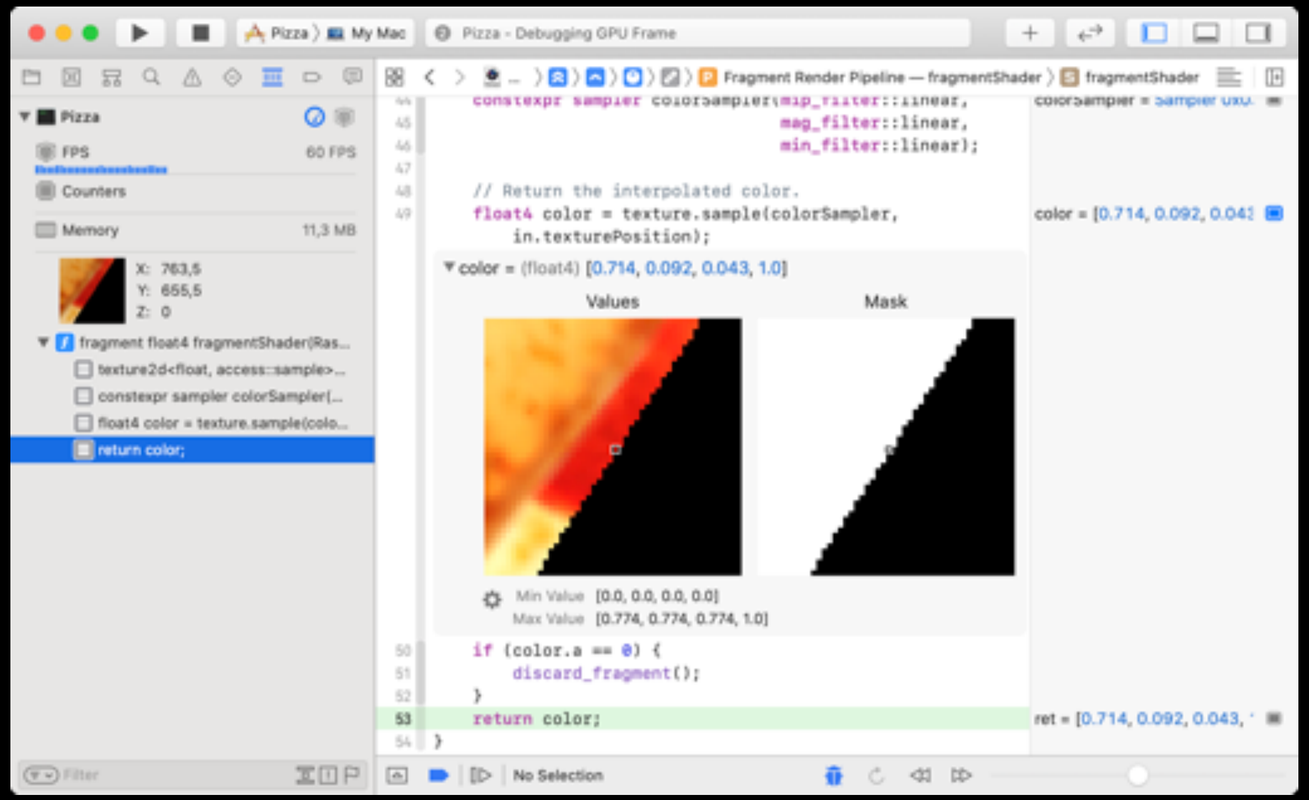
```
  bool alice = true
  bool bob = false
▼ while alice
  ▼ Iteration 1
    ▼ if prob(0.1)
        bob = true
    ▼ if prob(0.6)
        alice = false
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
  bool alice = true
  bool bob = false
▼ while alice
  ▼ Iteration 1
    ▼ if prob(0.1)
        bob = true
    ▼ if prob(0.6)
        alice = false
  ▶ Iteration 2
```

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```

```
  bool alice = true
  bool bob = false
▼ while alice
  ▼ Iteration 1
    ▼ if prob(0.1)
        bob = true
    ▼ if prob(0.6)
        alice = false
  ▶ Iteration 2
  end
```




Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✓

Implementation ✓

```
1 int x = 20
2 int y = 1
3 while x > 0 {
4   x -= 1
5   y = 2 * y
6 }
```

Syntax highlighting

Static semantic functionality

Program execution

Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation

# Slicing

- Find smaller program that has the same semantics w.r.t. some variable
- For probabilistic programs
  - Same semantics w.r.t. postexpectation $f \Leftrightarrow$ weakest preexpectation of $f$

    matches: $\dfrac{\text{wp}(P, f)}{1 - \text{woip}(P, 1) + \text{wp}(P, 1)}$
- Basic idea: Perform WP-inference and keep track of preexpectations produced by slices

# Expectation-Triples

- Expectation-triple $g = (g_f, g_\omega, g_\theta)$
- $\rho(P, g) = \big(\mathsf{wp}(P, g_f), \mathsf{wp}(P, g_\omega), \mathsf{woip}(P, g_\theta)\big)$
- Initial expectation-triple $F = (f,1,1)$
- Fractional value of $g$: $\overline{g} = \dfrac{g_f}{1 - g_\theta + g_\omega}$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

$$\overline{\rho(\varepsilon, F)} = \frac{[\![\mathsf{x} = \xi]\!]}{1 - 1 + 1} = [\![\mathsf{x} = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![\mathsf{x} = \xi]\!]$ | | |
|---|---|---|
| $\varepsilon \to F$ | | |

$$\overline{\rho(\varepsilon, F)} = \frac{[\![\mathsf{x} = \xi]\!]}{1 - 1 + 1} = [\![\mathsf{x} = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $5 \to F$ |

$$\overline{\rho(P_5, F)} = \frac{[\![x + 1 = \xi]\!]}{1 - 1 + 1} = [\![x + 1 = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $5 \to F$ |
| | $4\,5 \to F$ |

$$\overline{\rho(P_{4-5}, F)} = \frac{0.6 \cdot [\![x + 1 = \xi]\!]}{1 - 1 + 0.6} = [\![x + 1 = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $5 \to F$ |
| | $4\,5 \to F$ |
| | $3\,5 \to F$ |
| | $3\,4\,5 \to F$ |

$$\overline{\rho(P_{3-5}, F)} = \frac{0.6 \cdot [\![x + 1 = \xi]\!]}{1 - 1 + 0.6} = [\![x + 1 = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![\mathtt{x} = \xi]\!]$ | $[\![\mathtt{x} + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $5 \to F$ |
| $2\,5 \to F$ | $4\,5 \to F$ |
| $2\,4\,5 \to F$ | $3\,5 \to F$ |
| $2\,3\,5 \to F$ | $3\,4\,5 \to F$ |
| $2\,3\,4\,5 \to F$ | |

$$\overline{\rho(P_{2-5}, F)} = \frac{0.6 \cdot [\![\mathtt{x} = \xi]\!]}{1 - 1 + 0.6} = [\![x = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![10 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F$ | $5 \to F$ | $1 \to F$ |
| $2\,5 \to F$ | $4\,5 \to F$ | $1\,2\,5 \to F$ |
| $2\,4\,5 \to F$ | $3\,5 \to F$ | $1\,2\,4\,5 \to F$ |
| $2\,3\,5 \to F$ | $3\,4\,5 \to F$ | $1\,2\,3\,5 \to F$ |
| $2\,3\,4\,5 \to F$ | | $1\,2\,3\,4\,5 \to F$ |

$$\overline{\rho(P_{1-5}, F)} = \frac{0.6 \cdot [\![10 = \xi]\!]}{1 - 1 + 0.6} = [\![10 = \xi]\!]$$

```
1 int x = 10
2 x -= 1
3 int y = 5
4 observe prop(0.6)
5 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![10 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F$ | $5 \to F$ | $1 \to F$ |
| $2\,5 \to F$ | $4\,5 \to F$ | $1\,2\,5 \to F$ |
| $2\,4\,5 \to F$ | $3\,5 \to F$ | $1\,2\,4\,5 \to F$ |
| $2\,3\,5 \to F$ | $3\,4\,5 \to F$ | $1\,2\,3\,5 \to F$ |
| $2\,3\,4\,5 \to F$ | | $1\,2\,3\,4\,5 \to F$ |

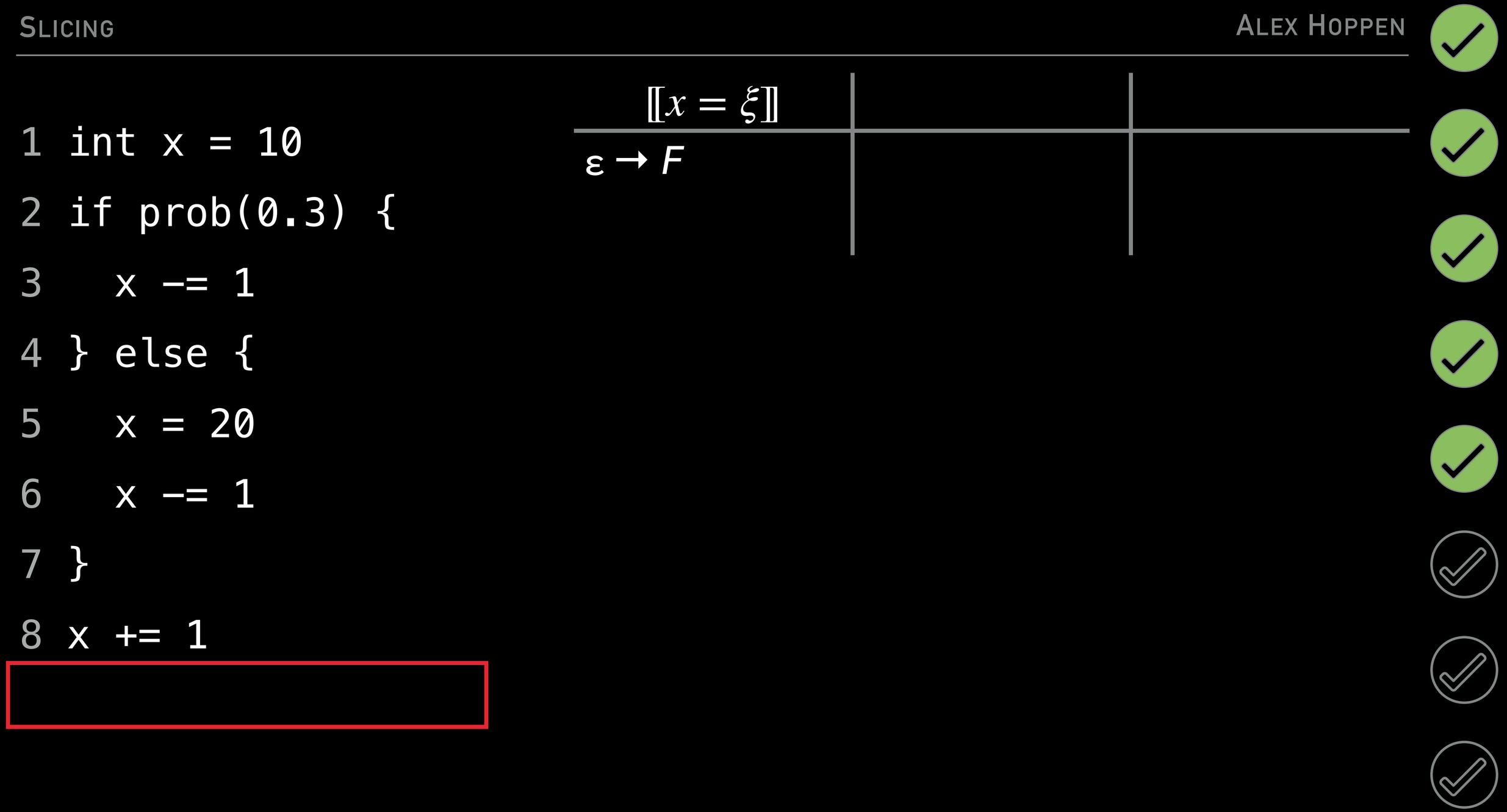
$$\overline{\rho(P_{1-5}, F)} = \frac{0.6 \cdot [\![10 = \xi]\!]}{1 - 1 + 0.6} = [\![10 = \xi]\!]$$

```
1 int x = 10
2 if prob(0.3) {
3   x -= 1
4 } else {
5   x = 20
6   x -= 1
7 }
8 x += 1
```

```
1 int x = 10
2 if prob(0.3) {
3   x -= 1
4 } else {
5   x = 20
6   x -= 1
7 }
8 x += 1
```

```
1 int x = 10
2 if prob(0.3) {
3   x -= 1
4 } else {
5   x = 20
6   x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | | |
|---|---|---|
| $\varepsilon \to F$ | | |



```
1 int x = 10
2 if prob(0.3) {
3   x -= 1
4 } else {
5   x = 20
6   x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $8 \to F$ |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $8 \to F$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F_x$ | $\varepsilon \to F_{x+1}$ |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F$ | $8 \to F$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ |
|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F_x$ | $\varepsilon \to F_{x+1}$ | |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F_x$<br>$3 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F_x$<br>$3 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | |

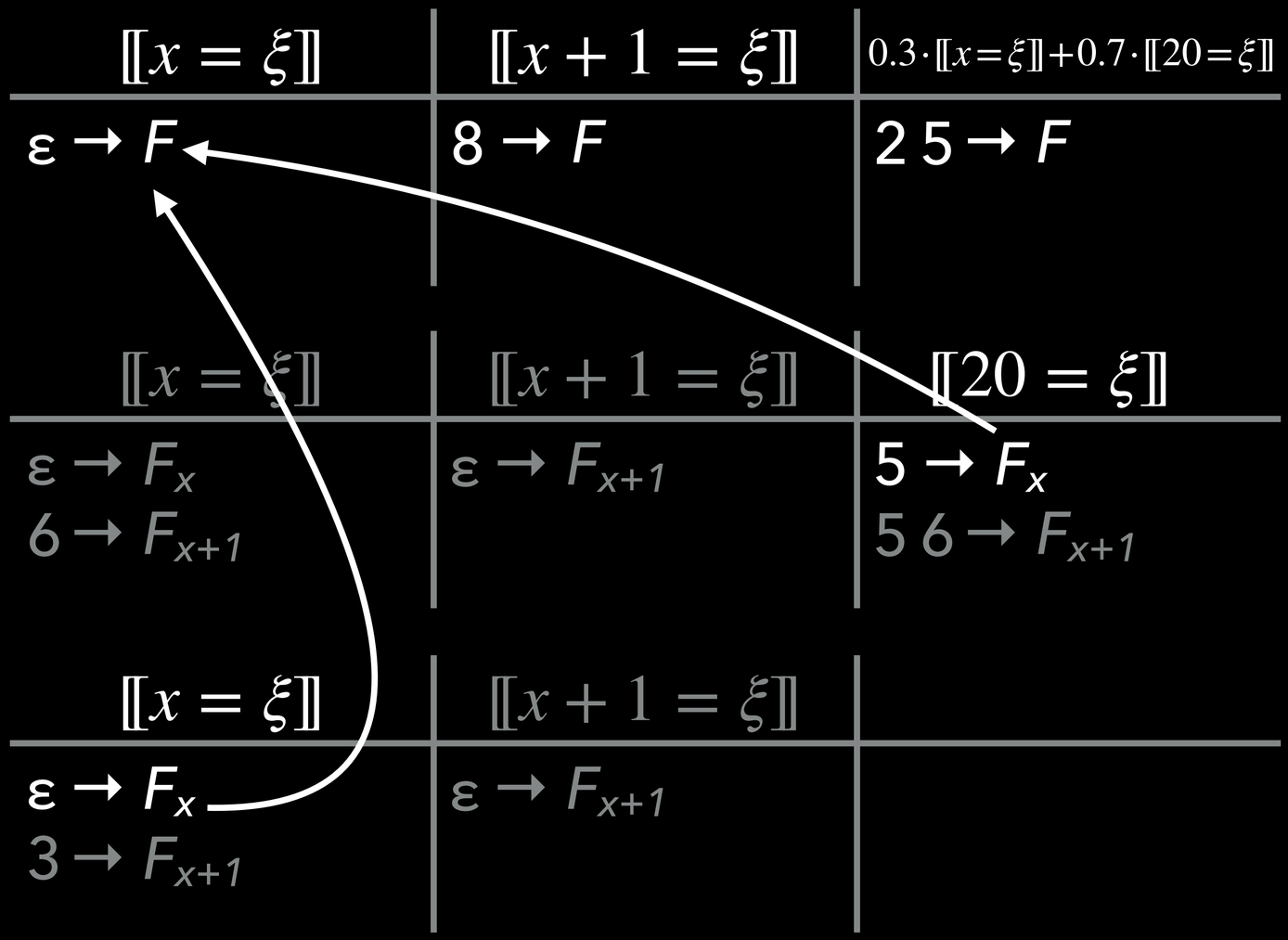
```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $0.3 \cdot [\![x = \xi]\!] + 0.7 \cdot [\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F_x$<br>$3 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```



```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $0.3 \cdot [\![x = \xi]\!] + 0.7 \cdot [\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | $25 \to F$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$56 \to F_{x+1}$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F_x$<br>$3 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | |

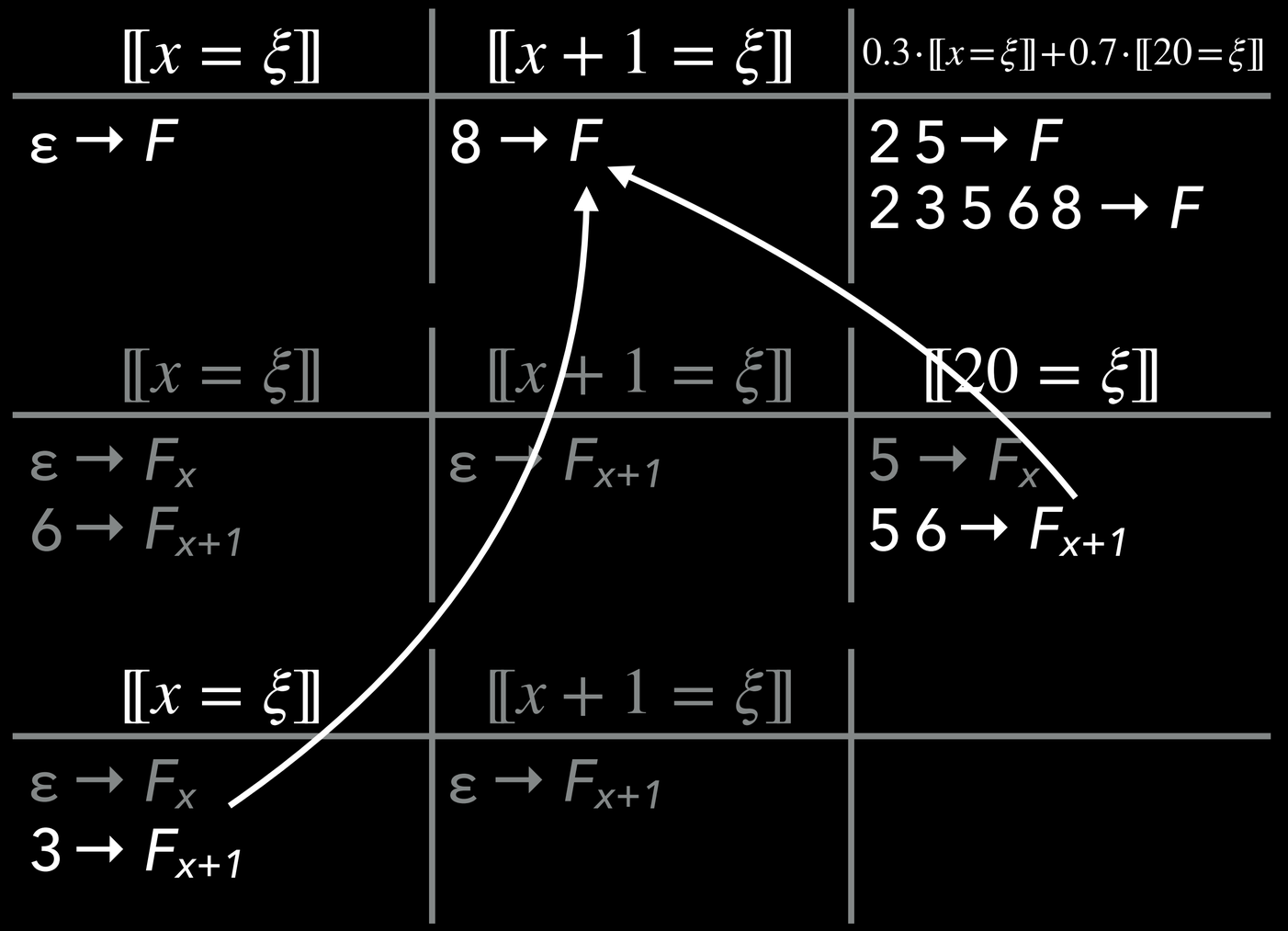
```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $0.3 \cdot [\![x = \xi]\!] + 0.7 \cdot [\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | $2\,5 \to F$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $[\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F_x$<br>$6 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | $5 \to F_x$<br>$5\,6 \to F_{x+1}$ |

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | |
|---|---|---|
| $\varepsilon \to F_x$<br>$3 \to F_{x+1}$ | $\varepsilon \to F_{x+1}$ | |

```
1 int x = 10
2 if prob(0.3) {
3    x -= 1
4 } else {
5    x = 20
6    x -= 1
7 }
8 x += 1
```

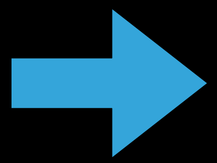


```
1 int x = 10
2 if prob(0.3) {
3   x -= 1
4 } else {
5   x = 20
6   x -= 1
7 }
8 x += 1
```

| $[\![x = \xi]\!]$ | $[\![x + 1 = \xi]\!]$ | $0.3 \cdot [\![x = \xi]\!] + 0.7 \cdot [\![20 = \xi]\!]$ |
|---|---|---|
| $\varepsilon \to F$ | $8 \to F$ | $25 \to F$<br>$23568 \to F$ |

```
1 int x = discrete({1: 0.5, 2: 0.5})
2 if x == 1 {
3   observe prob(0.1)
4 }
```

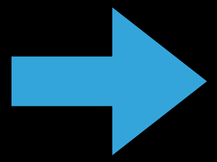
```
1 int x = discrete({1: 0.5, 2: 0.5})
2 if x == 1 {
3   observe prob(0.1)
4 }
```



Slice w.r.t. expectation-triple instead of the fractional value if wp($P$,1) and woip($P$,1) of bodies doesn't match

```
1 int x = 10
2 if prob(0.3) {
3   x += 1
4 } else {
5   x += 1
6 }
```

```
1 int x = 10
2 if prob(0.3) {
3   x += 1
4 } else {
5   x += 1
6 }
```



If expectation-triples of both bodies match, the condition itself can be removed

```
1 while x > 0 {
2   x = x - y
3   y = y + 1
4   z = z + 1
5 }
```

```
1 int x = 20
2 int y = 1
3 while x > 0 {
4   x -= 1
5   y = 2 * y
6 }
```

Syntax highlighting

Static semantic functionality

Program execution

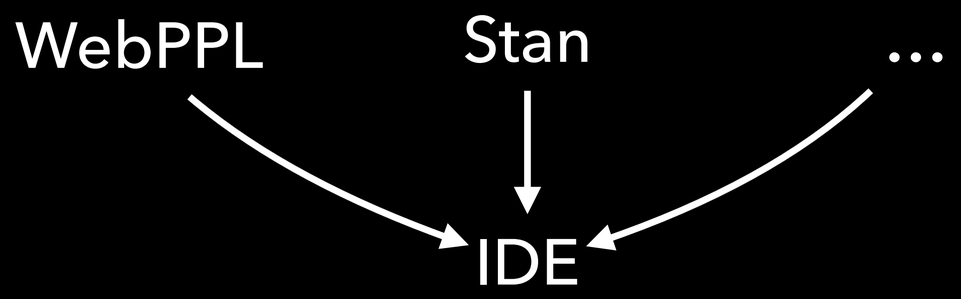
Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation




Syntax highlighting

Static semantic functionality

Program execution

Debugger

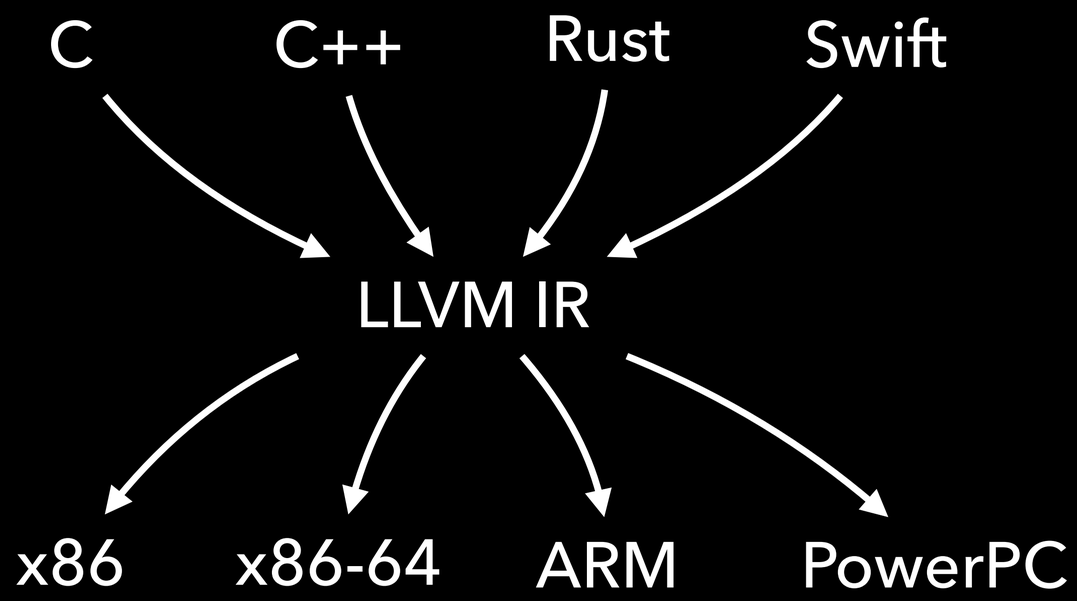
Recording-based debugger

Slicing

Support for multiple languages

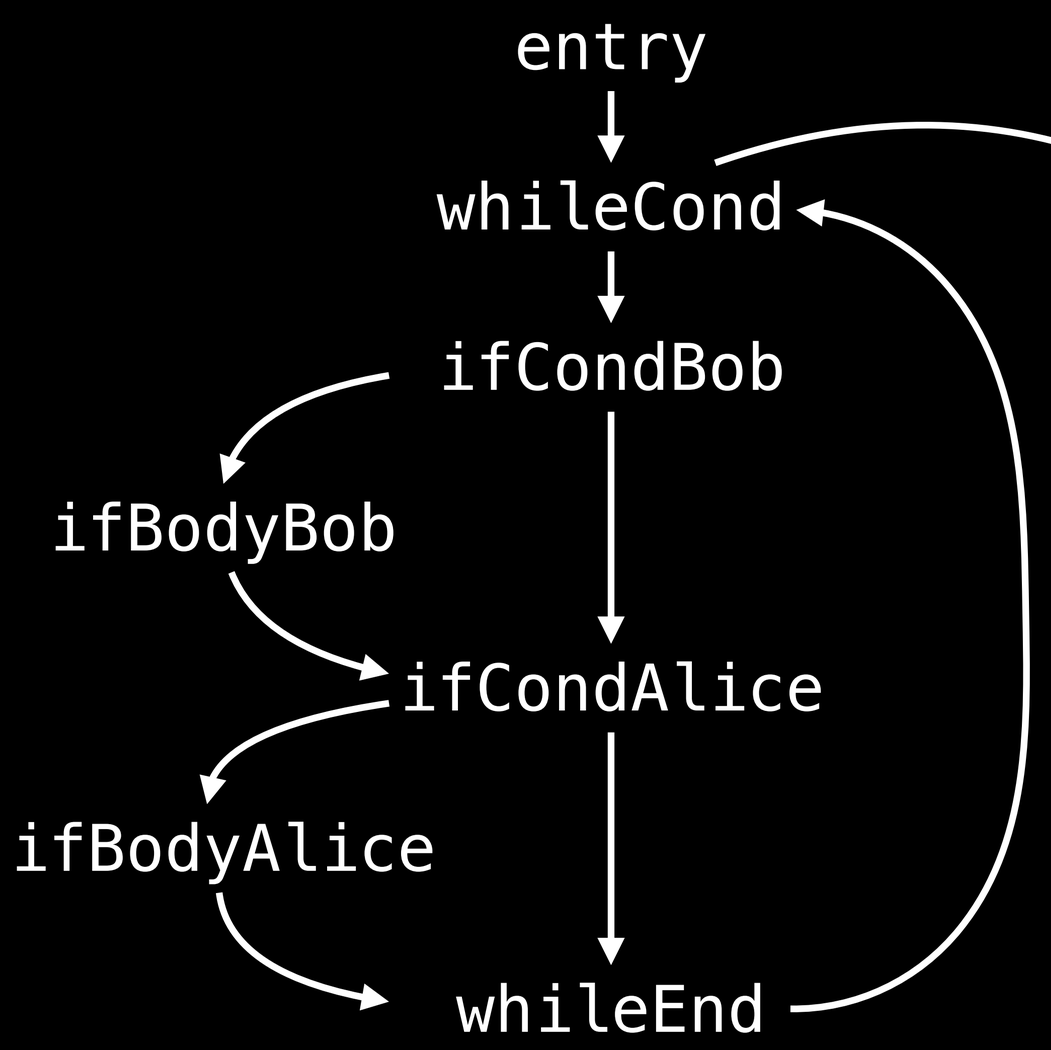
Implementation

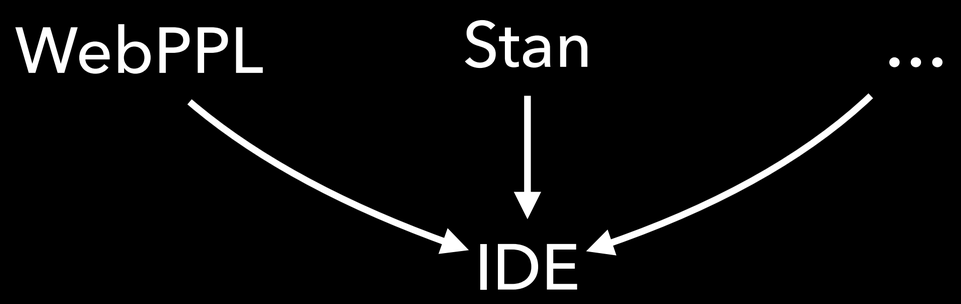
| | | |
|---|---|---|
| | Probability | |
| greta[46] | TensorFlow | R |
| pomegranate[47] | Python | Python |
| Lea[48] | Python | Python |
| WebPPL[49] | JavaScript | JavaScript |
| Picture[4] | Julia | Julia |
| Turing.jl[50] | Julia | Julia |
| Gen[51] | Julia | Julia |
| Low-level First-order PPL[52] | Python, Clojure, Pytorch | Various: Python, Clojure |
| Trolll[53] | | Moscow ML |
| Edward[54] | TensorFlow | Python |
| TensorFlow Probability[55] | TensorFlow | Python |
| Edward2[56] | TensorFlow Probability | Python |
| Pyro[57] | PyTorch | Python |
| Saul[58] | Scala | Scala |
| RankPL[59] | | Java |
| Birch[60] | | C++ |
| PSI[61] | | D |




Source: Wikipedia

```
bool alice = true
bool bob = false
while alice {
  if prob(0.1) {
    bob = true
  }
  if prob(0.6) {
    alice = false
  }
}
```






Syntax highlighting

Static semantic functionality

Program execution

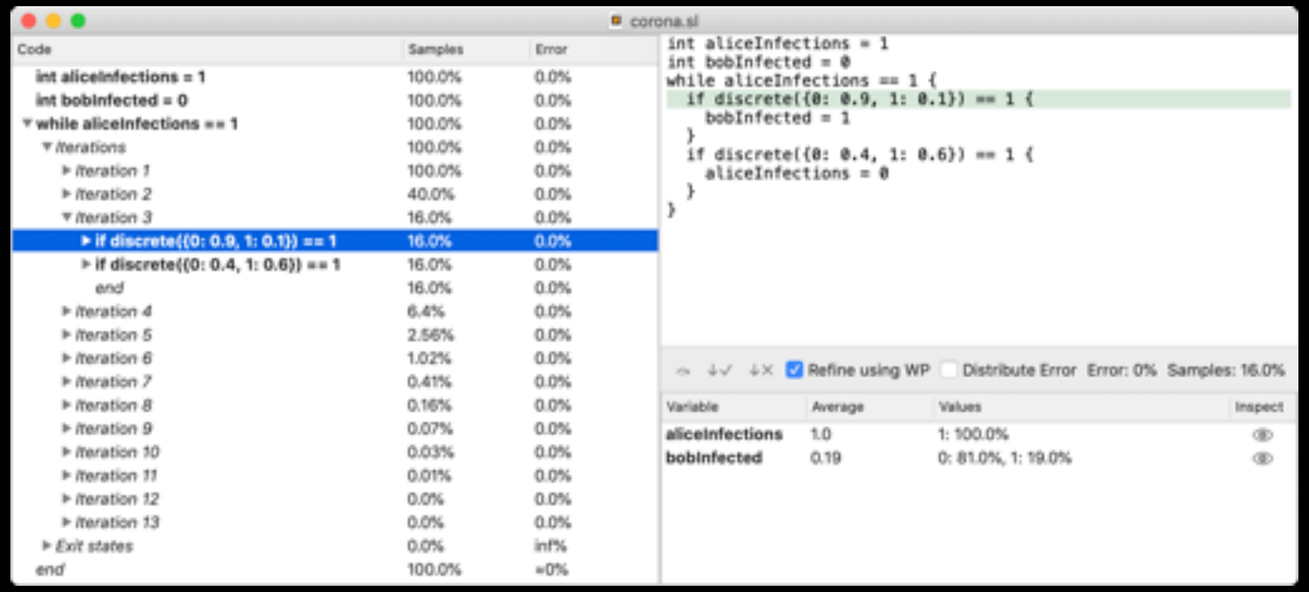
Debugger

Recording-based debugger

Slicing

Support for multiple languages

Implementation




Syntax highlighting

Static semantic functionality

Program execution

Debugger

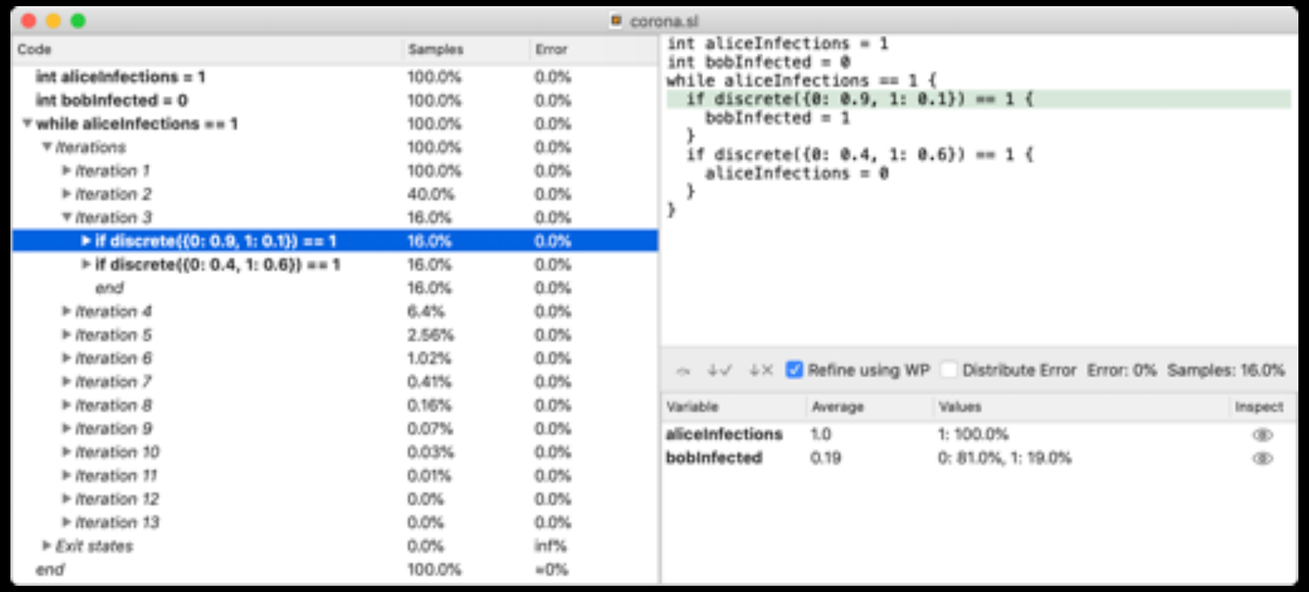
Recording-based debugger

Slicing

Support for multiple languages

Implementation

# Demo




Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✗

Implementation ✓

- Possible using existing techniques

Syntax highlighting 

Static semantic functionality 

Program execution 

Debugger 

Recording-based debugger 

Slicing 

Support for multiple languages 

Implementation 

- Possible using existing techniques

Syntax highlighting 

Static semantic functionality 

Program execution 

Debugger 

Recording-based debugger 

Slicing 

Support for multiple languages 

Implementation 

- Weakest preexpectation
- Loop iteration bounds
  - Determined using sampling-based execution
- Approximation error $\left( \pm \frac{1 - \mathsf{woip}_{\beta}(P,1)}{\mathsf{wp}_{\beta}(P,1)} \right)$

Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✗

Implementation ✓

- Debugger commands
  - Step Over
  - Step Into True
  - Step Into False
- Weakest preexpectations of execution history

Syntax highlighting ✓

Static semantic functionality ✓

Program execution ✓

Debugger ✓

Recording-based debugger ✓

Slicing ✓

Support for multiple languages ✗

Implementation ✓

- Description of execution states using execution history
- Generation of execution outline from source code structure

Syntax highlighting 

Static semantic functionality 

Program execution 

Debugger 

Recording-based debugger 

Slicing 

Support for multiple languages 

Implementation 

- Semantic slicing using weakest preexpectations
- Slicing correct w.r.t. loop iteration bounds

Syntax highlighting 

Static semantic functionality 

Program execution 

Debugger 

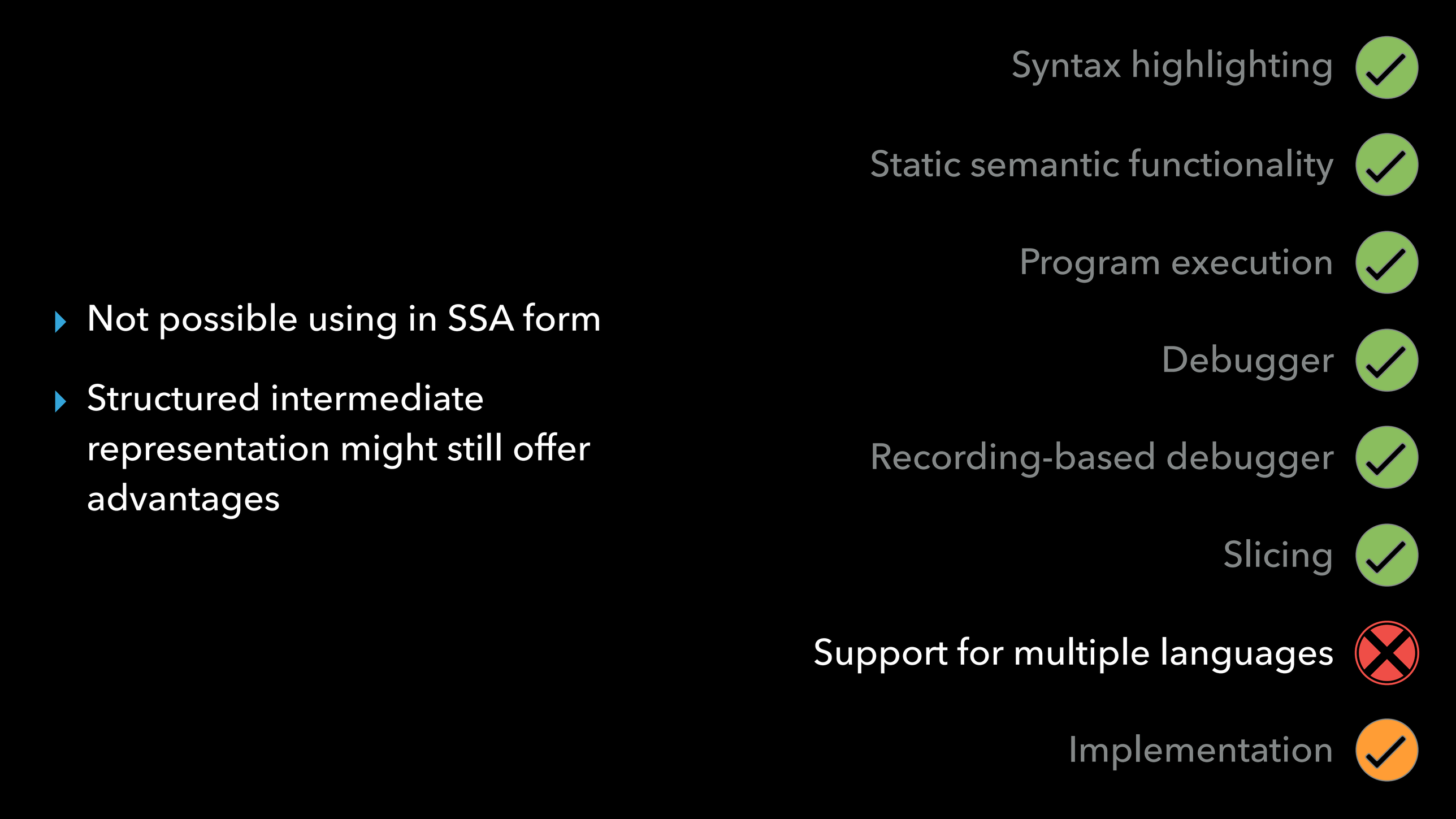
Recording-based debugger 

Slicing 

Support for multiple languages 

Implementation 

- Not possible using in SSA form
- Structured intermediate representation might still offer advantages



- Available as Mac-App and Unix Command Line tool

Syntax highlighting 

Static semantic functionality 

Program execution 

Debugger 

Recording-based debugger 

Slicing 

Support for multiple languages 

Implementation 

## Probabilistic Programs Are Inherently Different

# It is Possible