

ALEX HOPPEN

AACHEN, 30.7.2020

PROBABILISTIC PROGRAMMING IDE – A FEASIBILITY STUDY

Syntax highlighting



A screenshot of the Xcode IDE showing a Swift file named `WPIInferenceEngine.swift`. The code defines a class `WPIInferenceEngine` with various properties and methods. Syntax highlighting is applied to different parts of the code, such as `public`, `enum`, and `let` keywords, as well as variable names and type annotations. The code includes comments explaining the purpose of certain variables and logic.

```
15
16 public class WPIInferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27     /// Cached inference results. Maps an inference state to the resulting inference state when performing WP-inference to the top of the program.
28     /// The result is `nil` if there is no feasible inference run from the queried state to the top.
29     private var inferenceCache: [WPIInferenceState: WPIInferenceState?] = [:]
30
31     /// Instructions that are likely to be repeatedly visited and for which inference results should be cached.
32     /// At the moment such locations are the condition blocks of loops.
33     private let cachableIntermediateProgramPositions: Set<InstructionPosition>
34
35     public init(program: IRProgram) {
36         self.program = program
37         self.cachableIntermediateProgramPositions = Set(program.loopInducingBlocks.map({ block in
38             let firstNonPhiInstructionInBlock = program.basicBlocks[block].instructions.firstIndex(where: { !$0 is
39                 PhiInstruction } )!
40             return InstructionPosition(basicBlock: block, instructionIndex: firstNonPhiInstructionInBlock)
41         }))
42     }
43
44     /// Perform a single inference step.
45     /// The current instruction of `stateToInfer` has ==not== been inferred yet.
46 }
```

Syntax highlighting



The screenshot shows a portion of a Swift file named `WPIferenceEngine.swift` in an Xcode editor. The code defines a class `WPIferenceEngine` with an enum `ApproximationErrorHandling`. It includes a private let `program` and a private var `inferenceCache` of type `[WPIferenceState: WPIferenceState?]`. A callout bubble is open over the `inferenceCache` declaration, listing actions such as "Jump to Definition", "Show Quick Help", and "Callers...". The code uses standard Swift syntax with annotations explaining the purpose of each part.

```
15
16 public class WPIferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27     /// Cached inference results. Maps an inference state to the resulting inference state when performing WP-inference to the top of the program.
28     /// The result is `nil` if there is no feasible inference run from the queried state to the top.
29     private var inferenceCache: [WPIferenceState: WPIferenceState?] = [:]
30
31     /// Instructions that are
32     /// At the moment such
33     private let cachableInstructions: Set<InstructionPosition>
34
35     public init(program: IRProgram) {
36         self.program = program
37         self.cachableInstructions = Set<InstructionPosition>()
38         let firstNonPhiInstructionIndex = program.loopInducingBlocks.map({ block in
39             basicBlocks[block].instructions.firstIndex(where: { !$0 is
40                 PhiInstruction })
41             ?? 0
42         }).min()!
43
44         return InstructionPosition(basicBlock: nil, instructionIndex: firstNonPhiInstructionIndex)
45     }
46
47     /// Perform a single inference step.
48     /// The current instruction of `stateToInfer` has ==not== been inferred yet.
49 }
```

Static semantic functionality



Syntax highlighting



The screenshot shows the Probabilistic Debugger (ppdb) interface. At the top, there's a navigation bar with icons for file, edit, and search, followed by the path: ProbabilisticDebugger > ProbabilisticDebugger > Sources > WPIInference > WPIInferenceEngine.swift. Below the path is a toolbar with various icons. The main area contains two panes: the left pane shows the Swift code for `WPIInferenceEngine`, and the right pane shows the debugger session. In the debugger session, the command `> step into true` is run, followed by several conditional jumps based on discrete probability distributions. The final command shown is `> display variables`. The bottom status bar indicates "All Output 0".

```
15
16 public class WPIInferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27
28 > step into true
29     int aliceInfections = 1
30     int bobInfected = 0
31     while aliceInfections == 1 {
32     =>     if discrete({0: 0.9, 1: 0.1}) == 1 {
33         bobInfected = 1
34     }
35     if discrete({0: 0.4, 1: 0.6}) == 1 {
36         aliceInfections = 0
37     }
38 }
39 > display variables
40 Currently focused on 16.41% of all initially started runs.
41 Variable values:
42 aliceInfections | 1
43 bobInfected | 0: 81.8%, 1: 19.8%
44
45 >
46 All Output 0
```

Static semantic functionality



Program execution



Syntax highlighting



The screenshot shows a macOS-style application window titled "ppdb" running "ppdb". The main pane displays a Swift file named "WPInferenceEngine.swift" with line numbers 44 through 52. The code implements a function to perform inference steps on a state. A breakpoint is set at line 48, which is highlighted in green. The status bar at the bottom indicates "Thread 1: breakpoint 3.1". Below the code editor, a terminal window shows the execution of the program, starting with "step into true" and then "display variables". The message "Currently focused on 15.89% of all initially started runs." is displayed, followed by "Variable values: (lldb)".

```
44 // The current instruction of 'stateToInfer' has ==not== been inferred yet.
45 // 'previousBlock' is the block that has been inferred before this block. It must be specified when inferring a 'BranchInstruction'. For all other
46 // instructions, it can be omitted.
47 private func performInferenceStep(_ state: WPInferenceState, controlFlowDependencies: [WPTerm: IRVariable]? = nil) ->
48     WPInferenceState? {
49     var state = state
50     let position = state.position
51     let instruction = program.instruction(at: position)!
52     switch instruction {
53     case let instruction as AssignInstruction:
54         state.replace(variable: instruction.assignee, by: WPTerm(instruction.value))
55     }
56     sliceInfections = 0
57 }
58 > step into true
59 int aliceInfections = 1
60 int bobInfected = 0
61 while aliceInfections == 1 {
62-->   if discrete([0: 0.9, 1: 0.1]) == 1 {
63       bobInfected = 1
64   }
65   if discrete([0: 0.4, 1: 0.6]) == 1 {
66       aliceInfections = 0
67   }
68 }
69 > display variables
70 Currently focused on 15.89% of all initially started runs.
71 Variable values:
72 (lldb)
73 All Output 0
```

Static semantic functionality



Program execution



Debugger



Syntax highlighting



Static semantic functionality

```
constexpr sampler colorSampler(mip_mirrored::linear,
                                mag_filter::linear,
                                min_filter::linear);

// Return the interpolated color.
float4 color = texture.sample(colorSampler,
    in.texturePosition);

*color = (float4) [0.714, 0.092, 0.043, 1.0]

Values Mask
[0.714, 0.092, 0.043, 1.0]
Min Value [0.0, 0.0, 0.0, 0.0]
Max Value [0.774, 0.774, 0.774, 1.0]

if (color.a == 0) {
    discard_fragment();
}
return color;
```

Program execution



Debugger



Recording-based debugger



Syntax highlighting



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

Static semantic functionality



Program execution



Debugger

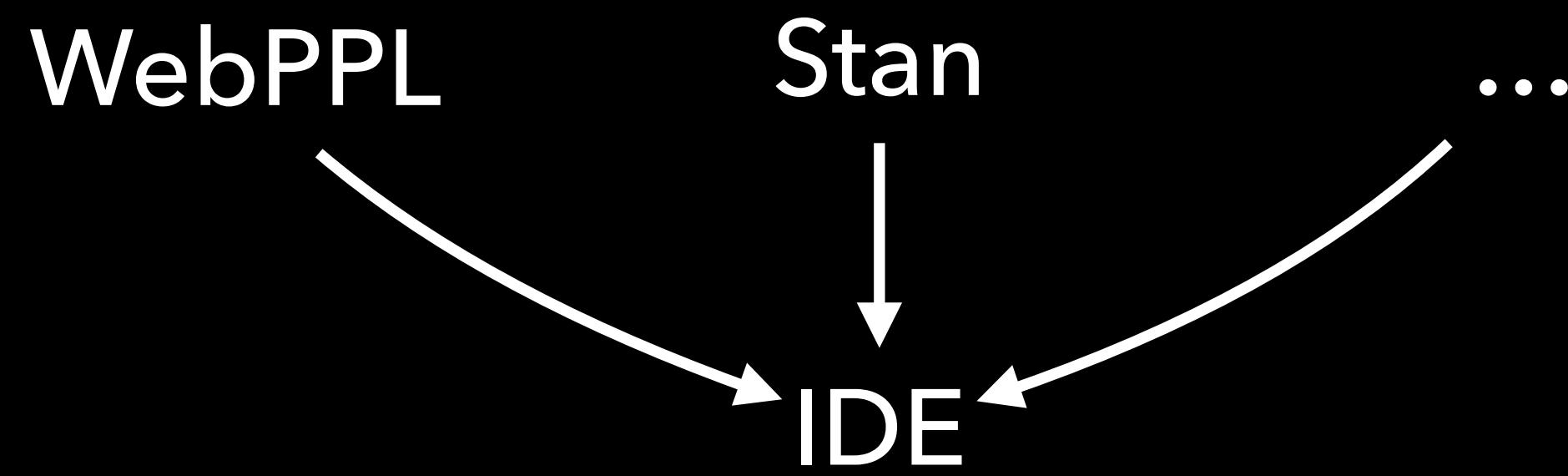


Recording-based debugger



Slicing





- Syntax highlighting
- Static semantic functionality
- Program execution
- Debugger
- Recording-based debugger
- Slicing
- Support for multiple languages

Syntax highlighting



Static semantic functionality



Code

	Samples	Error
int aliceInfections = 1	100.0%	0.0%
int bobInfected = 0	100.0%	0.0%
while aliceInfections == 1 {	100.0%	0.0%
if discrete({0: 0.9, 1: 0.1}) == 1 {	16.0%	0.0%
bobInfected = 1		
if discrete({0: 0.4, 1: 0.6}) == 1 {	16.0%	0.0%
aliceInfections = 0		
}		
}		
> if discrete({0: 0.9, 1: 0.1}) == 1	16.0%	0.0%
> if discrete({0: 0.4, 1: 0.6}) == 1	16.0%	0.0%
end	16.0%	0.0%
> iteration 4	6.4%	0.0%
> iteration 5	2.56%	0.0%
> iteration 6	1.024%	0.0%
> iteration 7	0.41%	0.0%
> iteration 8	0.16%	0.0%
> iteration 9	0.07%	0.0%
> iteration 10	0.03%	0.0%
> iteration 11	0.01%	0.0%
> iteration 12	0.0%	0.0%
> iteration 13	0.0%	0.0%
> Exit states	0.0%	inf%
end	100.0%	=0%

corona.s1

Refine using WP Distribute Error Error: 0% Samples: 16.0%

Variable	Average	Values	Inspect
aliceInfections	1.0	1: 100.0%	⊕
bobInfected	0.19	0: 81.0%, 1: 19.0%	⊕

Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting

A screenshot of an Xcode editor window. The title bar says "ProbabilisticDebugger | Build Succeeded | Today at 19:03". The navigation bar shows "ProbabilisticDebugger > ProbabilisticDebugger > Sources > WPIference > WPIferenceEngine.swift". The main area displays the following Swift code:

```
15
16 public class WPIferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27     /// Cached inference results. Maps an inference state to the resulting inference state when performing WP-inference to the top of the program.
28     /// The result is `nil` if there is no feasible inference run from the queried state to the top.
29     private var inferenceCache: [WPIferenceState: WPIferenceState?] = [:]
30
31     /// Instructions that are likely to be repeatedly visited and for which inference results should be cached.
32     /// At the moment such locations are the condition blocks of loops
33     private let cachableIntermediateProgramPositions: Set<InstructionPosition>
34
35     public init(program: IRProgram) {
36         self.program = program
37         self.cachableIntermediateProgramPositions = Set(program.loopInducingBlocks.map({ block in
38             let firstNonPhiInstructionInBlock = program.basicBlocks[block].instructions.firstIndex(where: { !$0 is
39                 PhiInstruction }) !!
40             return InstructionPosition(basicBlock: block, instructionIndex: firstNonPhiInstructionInBlock)
41         }))
42     }
43
44     /// Perform a single inference step.
45     /// The current instruction of 'stateToInfer' has ==not== been inferred yet.
46 }
```

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



```
15
16 public class WPInferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27     /// Cached inference results. Maps an inference state to the resulting inference state when performing WP-inference to the top of the program.
28     /// The result is "nil" if there is no feasible inference run from the queried state to the top.
29     private var inferenceCache: [WPInferenceState: WPInferenceState?] = [:]
30
31     /// Instructions that are likely to be repeatedly visited and for which inference results should be cached.
32     /// At the moment such locations are the condition blocks of loops
33     private let cachableIntermediateProgramPositions: Set<InstructionPosition>
34
35     public init(program: IRProgram) {
36         self.program = program
37         self.cachableIntermediateProgramPositions = Set(program.loopInducingBlocks.map({ block in
38             let firstNonPhiInstructionInBlock = program.basicBlocks[block].instructions.firstIndex(where: { !$0 is
39                 PhiInstruction }) !!
40             return InstructionPosition(basicBlock: block, instructionIndex: firstNonPhiInstructionInBlock)
41         }))
42     }
43
44     /// Perform a single inference step.
45     /// The current instruction of 'stateToInfer' has ==not== been inferred yet.
46 }
```

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting



The screenshot shows a Xcode editor window with a Swift file named `WPInferenceEngine.swift`. The code defines a class `WPInferenceEngine` with various methods and properties. A context menu is open over the variable `inferenceCache`, showing options like "Jump to Definition", "Show Quick Help", and "Callers...". The code uses Swift's modern syntax, including type annotations and optional chaining.

```
15
16 public class WPInferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27     /// Cached inference results. Maps an inference state to the resulting inference state when performing WP-inference to the top of the program.
28     /// The result is "nil" if there is no feasible inference run from the queried state to the top.
29     private var inferenceCache: [WPIInferenceState: WPIInferenceState?] = [:]
30
31     /// Instructions that are
32     /// At the moment such
33     private let cachableInstructions: Set<InstructionPosition>
34
35     public init(program: IRProgram) {
36         self.program = program
37         self.cachableInstructions = Set<InstructionPosition>()
38         let firstNonPhiInstructionIndex = program.loopInducingBlocks.map({ block in
39             basicBlocks[block].instructions.firstIndex(where: { !$0 is
40                 PhiInstruction })
41             .map { InstructionPosition(basicBlock: block, instructionIndex: $0) }
42         }).compactMap { $0 }
43
44         // Perform a single inference step.
45         // The current instruction of 'stateToInfer' has ==not== been inferred yet.
46     }
47 }
```

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



The screenshot shows a Xcode editor window with a Swift file named `WPInferenceEngine.swift`. The code defines a class `WPInferenceEngine` with various methods and properties. A context menu is open over the variable `inferenceCache`, showing options like "Jump to Definition", "Show Quick Help", and "Callers...". The code uses Swift's modern syntax, including type annotations and optional chaining.

```
15
16 public class WPInferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27     /// Cached inference results. Maps an inference state to the resulting inference state when performing WP-inference to the top of the program.
28     /// The result is "nil" if there is no feasible inference run from the queried state to the top.
29     private var inferenceCache: [WPIInferenceState: WPIInferenceState?] = [:]
30
31     /// Instructions that are
32     /// At the moment such
33     private let cachableInstructions: Set<InstructionPosition>
34
35     public init(program: IRProgram) {
36         self.program = program
37         self.cachableInstructions = Set<InstructionPosition>()
38         let firstNonPhiInstructionIndex = program.loopInducingBlocks.map({ block in
39             basicBlocks[block].instructions.firstIndex(where: { !$0 is
40                 PhiInstruction })
41             .map { InstructionPosition(basicBlock: block, instructionIndex: $0) }
42         }).compactMap { $0 }
43
44         // Perform a single inference step.
45         // The current instruction of 'stateToInfer' has ==not== been inferred yet.
46     }
47 }
```

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting



The screenshot shows the Probabilistic Debugger (ppdb) interface running on a Mac. The top navigation bar shows 'ppdb' and 'My Mac'. The title bar says 'Running ppdb : ppdb'. The main window displays a file tree: ProbabilisticDebugger > ProbabilisticDebugger > Sources > WPInference > WPInferenceEngine.swift. The code editor shows the following Swift code:

```
15
16 public class WPInferenceEngine {
17     /// If WP-inference loses some probability mass due to loop iteration bounds, this specifies how the lost probability mass should be handled.
18     public enum ApproximationErrorHandling {
19         /// Don't handle the lost probability mass and return a probability distribution with sum that might be less than 1.
20         case drop
21         /// Proportionally distribute the lost probability mass onto the known values.
22         case distribute
23     }
24
25     private let program: IRProgram
26
27
28 > step into true
29     int aliceInfections = 1
30     int bobInfected = 0
31     while aliceInfections == 1 {
32     =>     if discrete({0: 0.9, 1: 0.1}) == 1 {
33         bobInfected = 1
34     }
35     if discrete({0: 0.4, 1: 0.6}) == 1 {
36         aliceInfections = 0
37     }
38 }
39 > display variables
40 Currently focused on 16.41% of all initially started runs.
41 Variable values:
42 aliceInfections | 1
43 bobInfected | 0: 81.8%, 1: 19.8%
44
45 >
46 All Output 0
```

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



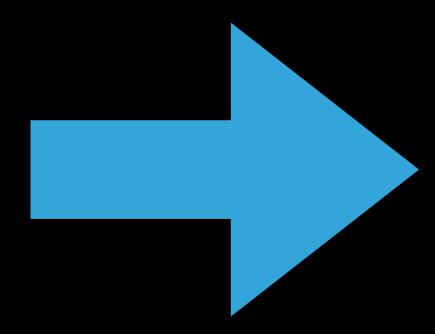

$$\frac{P}{\begin{array}{l} () \\ (P_1, P_2) \\ (\text{int} \mid \text{bool} \mid \varepsilon) \ var \ = \ expr \\ \text{if } expr \ \{ \ P_{\text{if}} \ \} \ \text{else } \{ \ P_{\text{else}} \ \} \\ \text{while } expr \ \{ \ P_{\text{body}} \ \} \\ \text{observe } expr \end{array}}$$
$$\frac{\wp(P, f)}{\begin{array}{l} f \\ \wp(P_1, \wp(P_2, f)) \\ f[v \ := \ expr] \\ [[expr]] \cdot \wp(P_{\text{if}}, f) + [[\neg expr]] \cdot \wp(P_{\text{else}}, f) \\ \text{lfp } X. ([[expr]] \cdot \wp(P_{\text{body}}, X) + [[\neg expr]] \cdot f) \\ [[expr]] \cdot f \end{array}}$$


→ int x = 1

$$\begin{aligned} & \wp(P, [[x = \xi]]) \\ &= [[1 = \xi]] \end{aligned}$$

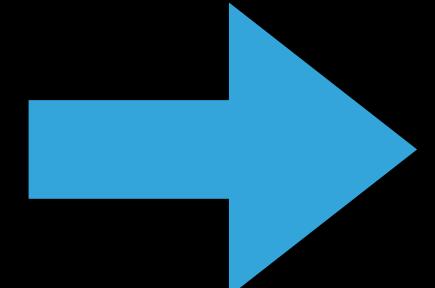



$$\frac{P}{\begin{array}{l} () \\ (P_1, P_2) \\ (\text{int} \mid \text{bool} \mid \varepsilon) \ var \ = \ expr \\ \text{if } expr \ \{ \ P_{\text{if}} \ \} \ \text{else } \{ \ P_{\text{else}} \ \} \\ \text{while } expr \ \{ \ P_{\text{body}} \ \} \\ \text{observe } expr \end{array}}$$
$$\frac{\text{wlp}(P, f)}{\begin{array}{l} f \\ \wp(P_1, \wp(P_2, f)) \\ f[v \ := \ expr] \\ [[expr]] \cdot \wp(P_{\text{if}}, f) + [[\neg expr]] \cdot \wp(P_{\text{else}}, f) \\ \text{gfp } X. ([[expr]] \cdot \text{wlp}(P_{\text{body}}, X) + [[\neg expr]] \cdot f) \\ [[expr]] \cdot f \end{array}}$$



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

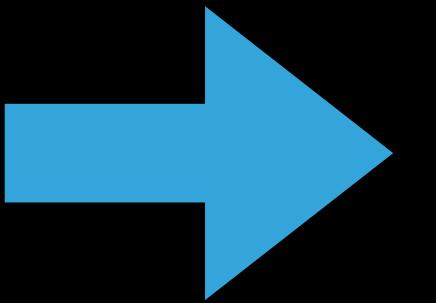


while cond {
 P_{body}
}

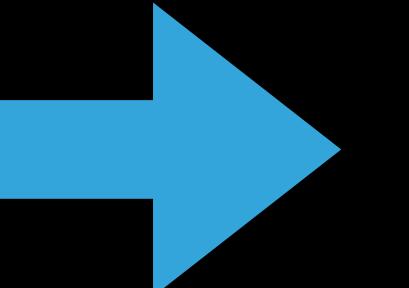


if cond {
 P_{body}
}
if cond {
 P_{body}
}
if cond {
 while true {}
}

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



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



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



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





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

$$\text{wlp}(P, 1)$$

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





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



$$\text{wlp}(P, 1)$$

$$= \text{wp}(P, 1) + (1 - \text{woip}(P, 1))$$

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





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



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



Syntax highlighting



The screenshot shows the Probabilistic Debugger (ppdb) interface. At the top, there's a navigation bar with icons for file, edit, and search, followed by the path: ProbabilisticDebugger > ProbabilisticDebugger > Sources > WPInference > WPInferenceEngine.swift. Below the path is a toolbar with various icons. The main area contains two panes: the left pane shows the source code for `WPInferenceEngine`, and the right pane shows a debugger console. The debugger console has the following output:

```
> step into true
int aliceInfections = 1
int bobInfected = 0
while aliceInfections == 1 {
-->  if discrete({0: 0.9, 1: 0.1}) == 1 {
    bobInfected = 1
}
if discrete({0: 0.4, 1: 0.6}) == 1 {
    aliceInfections = 0
}
}
> display variables
Currently focused on 16.41% of all initially started runs.
Variable values:
aliceInfections | 1
bobInfected | 0: 81.8%, 1: 19.8%
>
```

At the bottom, there's a "Filter" button and a status bar indicating "All Output 0".

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting



The screenshot shows the Probabilistic Debugger (ppdb) interface running on a Mac. The top bar shows the title "ppdb" and "Running ppdb: ppdb". The main window displays a Swift file named "WPInferenceEngine.swift" with syntax highlighting. A breakpoint is set at line 48. The code is as follows:

```
44 // The current instruction of 'stateToInfer' has ==not== been inferred yet.
45 // 'previousBlock' is the block that has been inferred before this block. It must be specified when inferring a 'BranchInstruction'. For all other
46 // instructions, it can be omitted.
47 private func performInferenceStep(_ state: WPInferenceState, controlFlowDependencies: [WPTerm: IRVariable]? = nil) ->
48     WPInferenceState? {
49     var state = state
50     let position = state.position
51     let instruction = program.instruction(at: position)!
52     switch instruction {
53     case let instruction as AssignInstruction:
54         state.replace(variable: instruction.assignee, by: WPTerm(instruction.value))
55     }
56     stateInfections = 0
57 }
58 > step into true
59 int aliceInfections = 1
60 int bobInfected = 0
61 while aliceInfections == 1 {
62-->   if discrete([0: 0.9, 1: 0.1]) == 1 {
63       bobInfected = 1
64   }
65   if discrete([0: 0.4, 1: 0.6]) == 1 {
66       aliceInfections = 0
67   }
68 > display variables
69 Currently focused on 15.89% of all initially started runs.
70 Variable values:
71 (lldb)
72 All Output 0
```

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



```
bool alice = true
```

CONTINUESTEP OVERSTEP INTOSTEP OUT

```
bool bob = false
```

```
while alice {
```

```
    if prob(0.1) {
```

```
        bob = true
```

```
}
```

```
    if prob(0.6) {
```

```
        alice = false
```

```
}
```

```
}
```



```
bool alice = true
```

STEP OVER

```
bool bob = false
```

```
while alice {
```

```
    if prob(0.1) {
```

```
        bob = true
```

```
}
```

```
    if prob(0.6) {
```

```
        alice = false
```

```
}
```

```
}
```



```
bool alice = true
```


STEP OVER

```
bool bob = false
```


STEP OVER

```
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



```
bool alice = true
```

STEP OVER

```
bool bob = false
```

STEP OVER

```
while alice {
```

STEP OVER

STEP INTO FALSE

STEP INTO TRUE

```
    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



```
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**STEP OVER****STEP INTO TRUE****STEP INTO TRUE****STEP OVER**

bool alice = true

bool bob = false

while alice

if prob(0.1)

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

 $wph(h^P, f)$ $wph(h^P, 1)$ $woiph(h^P, 1)$

[[bob]]

1

1





STEP OVER

bool alice = true

	wph(h^P, f)	wph($h^P, 1$)	woiph($h^P, 1$)
STEP OVER	0.1	0.1	1
STEP OVER	$0.1 \cdot [\![\text{alice}]\!]$	$0.1 \cdot [\![\text{alice}]\!]$	1
STEP INTO TRUE	$0.1 \cdot [\![\text{alice}]\!]$	$0.1 \cdot [\![\text{alice}]\!]$	1
STEP INTO TRUE	0.1	0.1	1
STEP OVER	1	1	1
	$[\![\text{bob}]\!]$	1	1



STEP INTO TRUE

observe alice

STEP INTO TRUE

observe prob(0.1)

STEP OVER

bob = true



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

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

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



Syntax highlighting



The screenshot shows the Probabilistic Debugger (ppdb) interface running on a Mac. The top bar shows the title "ppdb" and "Running ppdb: ppdb". The main window displays a Swift file named "WPInferenceEngine.swift" with line numbers 44 through 52. Line 48 is highlighted with a blue arrow, indicating it is the current instruction. A green bar at the bottom of the code editor indicates a breakpoint at line 31. The bottom half of the window is a terminal-like interface for the debugger, showing command history and variable values. The command "step into true" has been run, and the debugger is executing a loop. The output includes variables like "aliceInfections" and "bobInfected". The bottom status bar shows "All Output 0".

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting



```
constexpr sampler colorSampler(mip_masher::linear,
                                mag_filter::linear,
                                min_filter::linear);

// Return the interpolated color.
float4 color = texture.sample(colorSampler,
    in.texturePosition);

color = [0.714, 0.092, 0.043, 1.0]

if (color.a == 0) {
    discard_fragment();
}
return color;
```

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting



The screenshot shows the Pizza debugger interface. On the left, there's a sidebar with project navigation, FPS counter (60 FPS), counters, and memory usage (11.3 MB). The main area displays GLSL code for a fragment shader:

```
constexpr sampler colorSampler(mipFilter::linear,
                                mag_filter::linear,
                                min_filter::linear);

// Return the interpolated color.
float4 color = texture.sample(colorSampler,
    in.texturePosition);

color = (float4) [0.714, 0.092, 0.043, 1.0]

fragment float4 fragmentShader(RasterizerOutput in)
{
    // ...
    color.a == 0 {
        discard_fragment();
    }
    return color;
}
```

Below the code, two visualizations are shown: "Values" (a heatmap of pixel colors) and "Mask" (a binary mask indicating where the fragment was discarded).

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
```



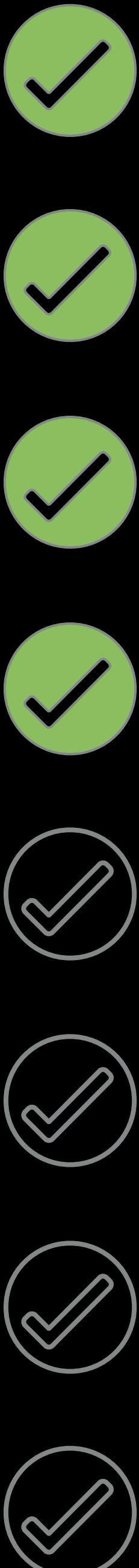
```
bool alice = true  
bool bob = false  
while alice {  
    if prob(0.1) {  
        bob = true  
    }  
    if prob(0.6) {  
        alice = false  
    }  
}
```

```
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



```
constexpr sampler colorSampler(mip_masher::linear,
                                mag_filter::linear,
                                min_filter::linear);

// Return the interpolated color.
float4 color = texture.sample(colorSampler,
    in.texturePosition);

color = [0.714, 0.092, 0.043, 1.0]

if (color.a == 0) {
    discard_fragment();
}
return color;
```

Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



Syntax highlighting



Static semantic functionality



Program execution



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

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





EXPECTATION-TRIPLES

- ▶ Expectation-triple $g = (g_f, g_\omega, g_\theta)$
- ▶ $\rho(P, g) = (\text{wp}(P, g_f), \text{wp}(P, g_\omega), \text{woip}(P, g_\theta))$
- ▶ Initial expectation-triple $F = (f, 1, 1)$
- ▶ Fractional value of g : $\bar{g} = \frac{g_f}{1 - g_\theta + g_\omega}$



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



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



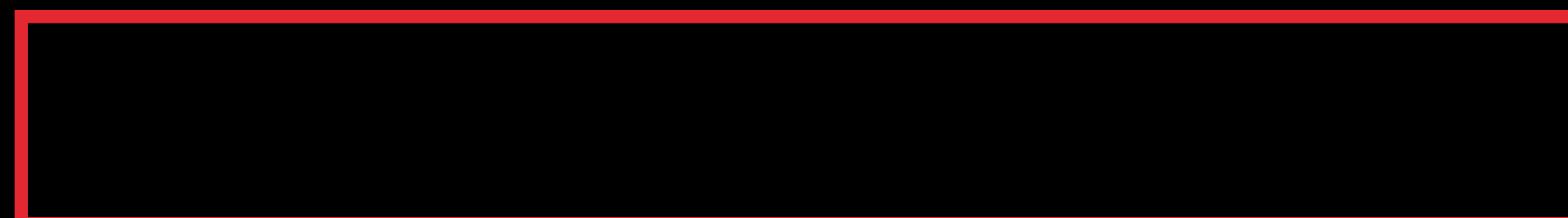
1 int x = 10

2 x -= 1

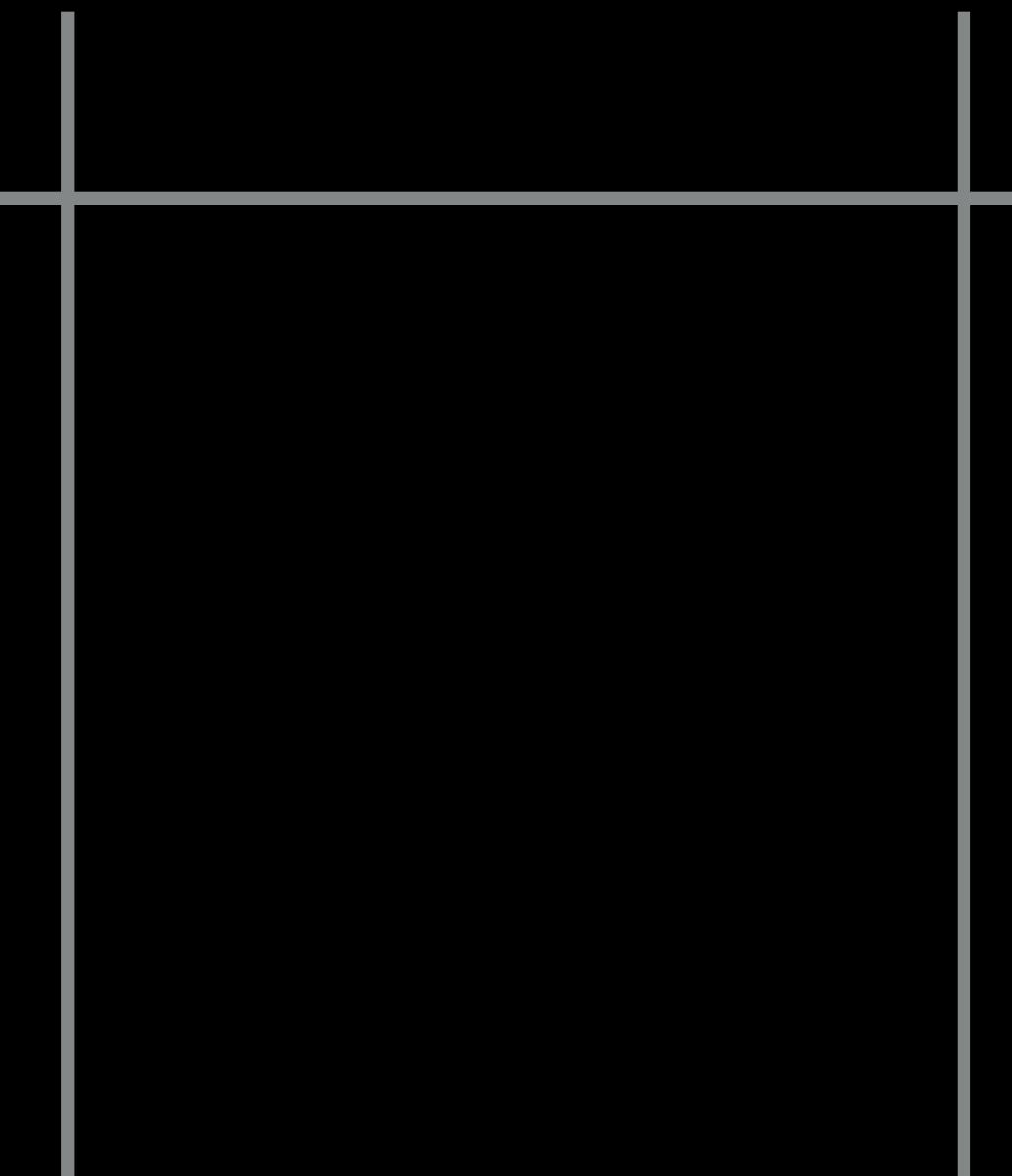
3 int y = 5

4 observe prop(0.6)

5 x += 1



$$\frac{[\![x = \xi]\!]}{\varepsilon \rightarrow F}$$



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



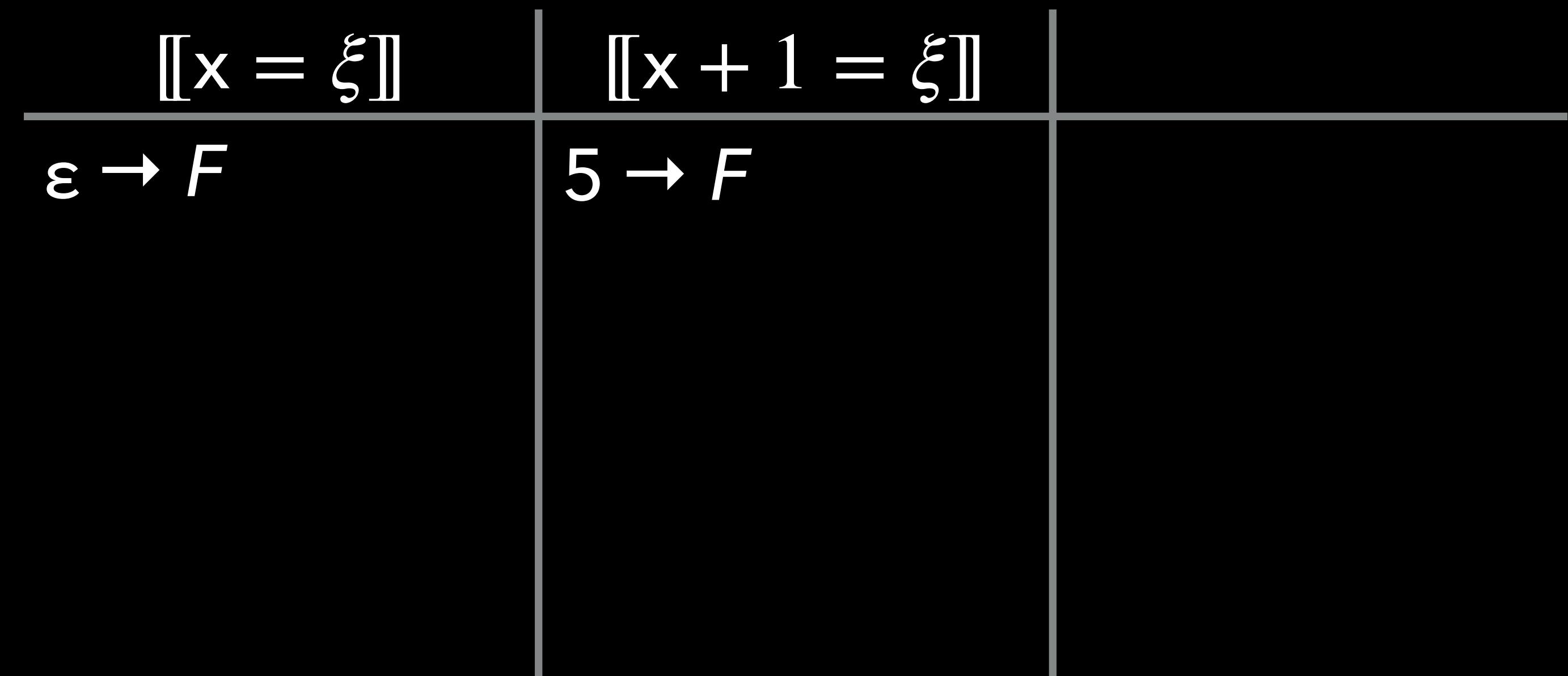
1 int x = 10

2 x -= 1

3 int y = 5

4 observe prop(0.6)

5 x += 1



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



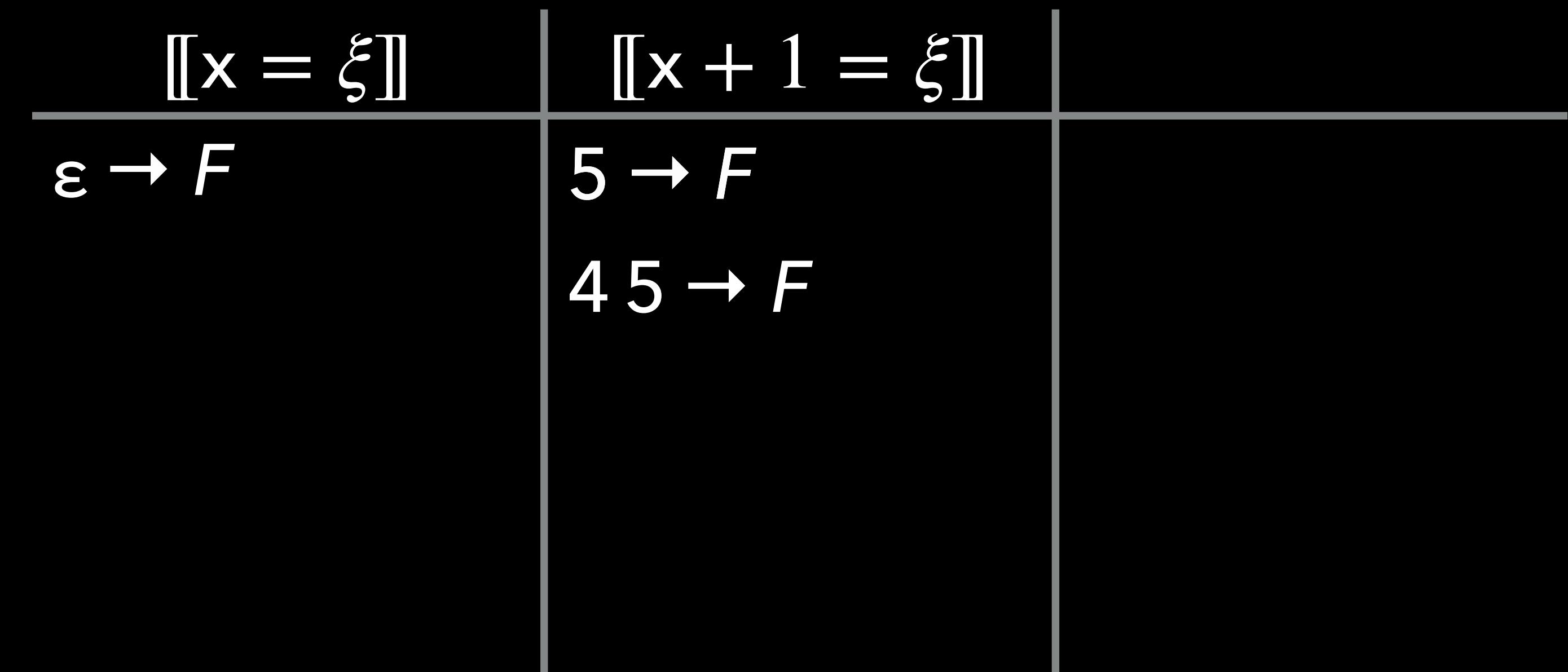
1 int x = 10

2 x -= 1

3 int y = 5

4 observe prop(0.6)

5 x += 1



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



1 int x = 10

2 x -= 1

3 int y = 5

4 observe prop(0.6)

5 x += 1

$\llbracket x = \xi \rrbracket$	$\llbracket x + 1 = \xi \rrbracket$
$\varepsilon \rightarrow F$	$5 \rightarrow F$
	$45 \rightarrow F$
	$35 \rightarrow F$
	$345 \rightarrow F$

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





```
1 int x = 10
```

```
2 x -= 1
```

```
3 int y = 5
```

```
4 observe prop(0.6)
```

```
5 x += 1
```

$\llbracket x = \xi \rrbracket$	$\llbracket x + 1 = \xi \rrbracket$
$\varepsilon \rightarrow F$	$5 \rightarrow F$
$25 \rightarrow F$	$45 \rightarrow F$
$245 \rightarrow F$	$35 \rightarrow F$
$235 \rightarrow F$	$345 \rightarrow F$
$2345 \rightarrow F$	



$$\overline{\rho(P_{2-5}, F)} = \frac{0.6 \cdot \llbracket x = \xi \rrbracket}{1 - 1 + 0.6} = \llbracket x = \xi \rrbracket$$





1 int x = 10

2 x -= 1

3 int y = 5

4 observe prop(0.6)

5 x += 1

	$\llbracket x = \xi \rrbracket$	$\llbracket x + 1 = \xi \rrbracket$	$\llbracket 10 = \xi \rrbracket$
	$\varepsilon \rightarrow F$	$5 \rightarrow F$	$1 \rightarrow F$
2	$25 \rightarrow F$	$45 \rightarrow F$	$125 \rightarrow F$
3	$245 \rightarrow F$	$35 \rightarrow F$	$1245 \rightarrow F$
4	$235 \rightarrow F$	$345 \rightarrow F$	$1235 \rightarrow F$
5	$2345 \rightarrow F$		$12345 \rightarrow F$



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





1 int x = 10

2 x -= 1

3 int y = 5

4 observe prop(0.6)

5 x += 1

	$\llbracket x = \xi \rrbracket$	$\llbracket x + 1 = \xi \rrbracket$	$\llbracket 10 = \xi \rrbracket$
	$\varepsilon \rightarrow F$	$5 \rightarrow F$	$1 \rightarrow F$
2	$25 \rightarrow F$	$45 \rightarrow F$	$125 \rightarrow F$
3	$245 \rightarrow F$	$35 \rightarrow F$	$1245 \rightarrow F$
4	$235 \rightarrow F$	$345 \rightarrow F$	$1235 \rightarrow F$
5	$2345 \rightarrow F$		$12345 \rightarrow F$



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



```
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
```

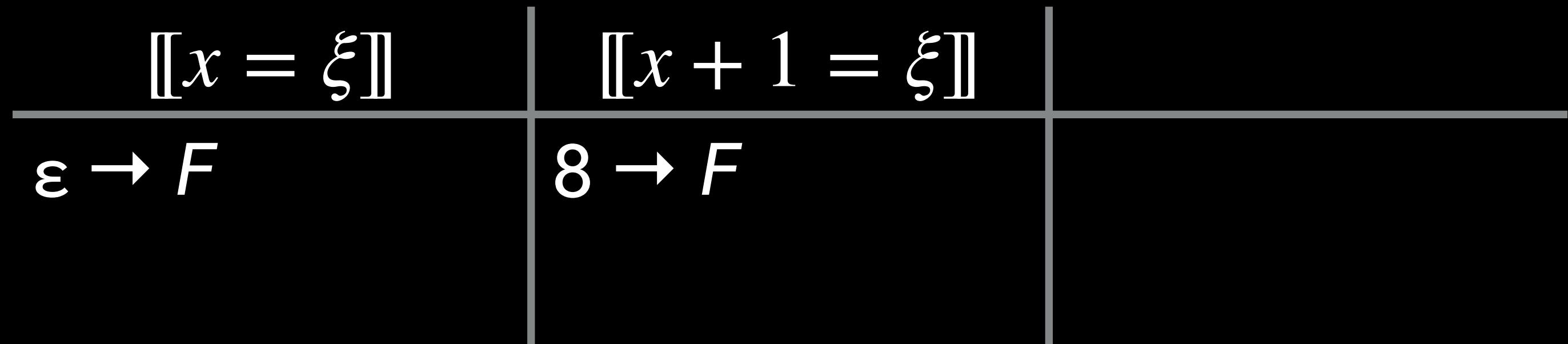


```
1 int x = 10
2 if prob(0.3) {
3     x -= 1
4 } else {
5     x = 20
6     x -= 1
7 }
8 x += 1
```

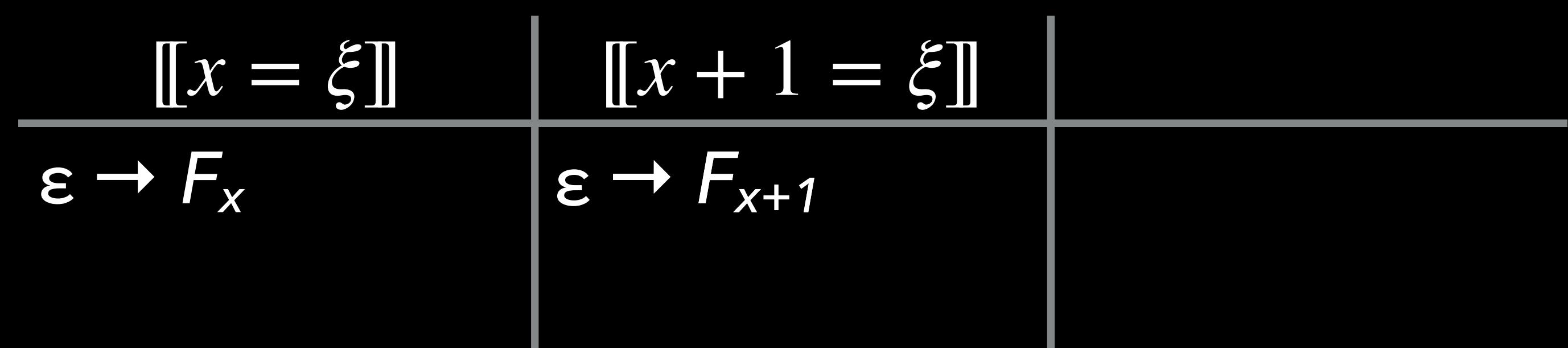
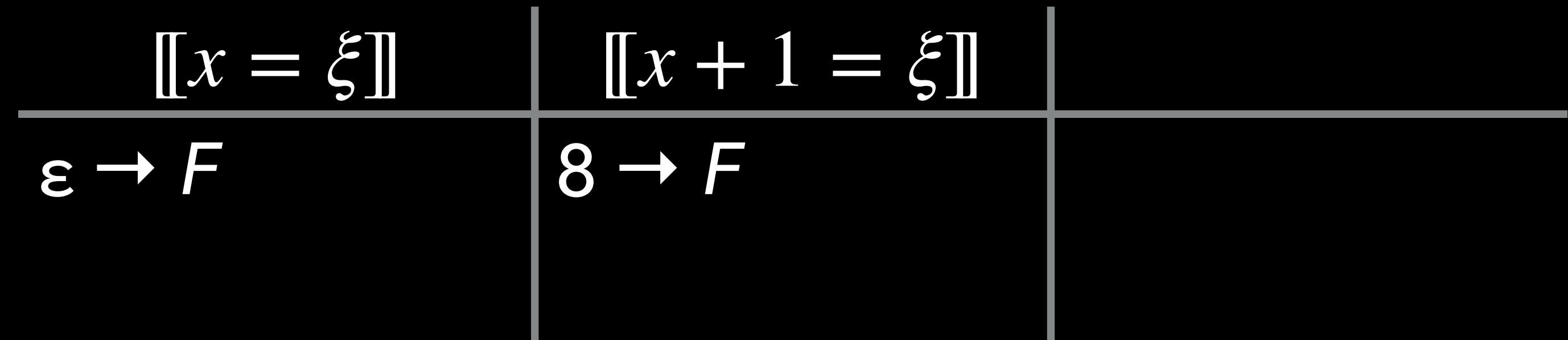
$$\frac{[[x = \xi]]}{\varepsilon \rightarrow F}$$



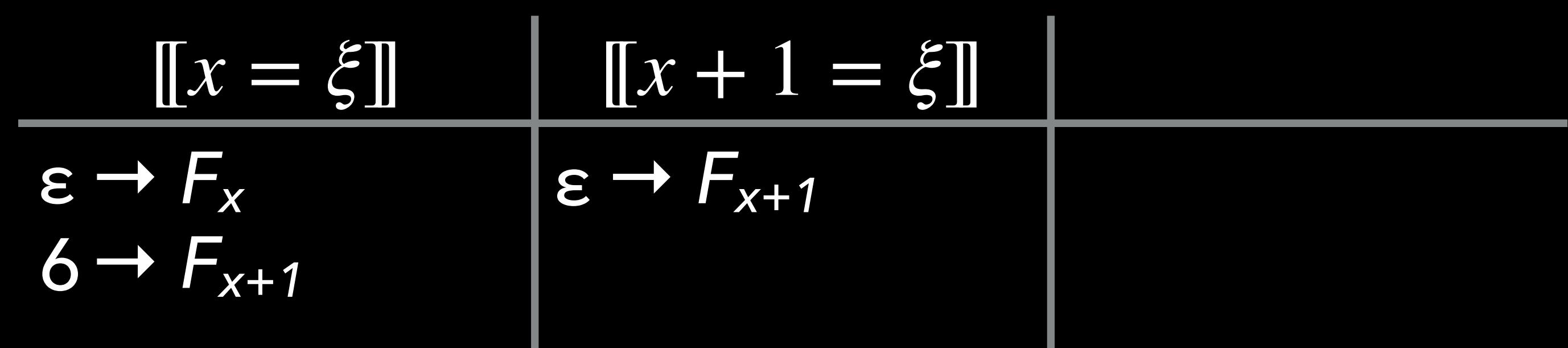
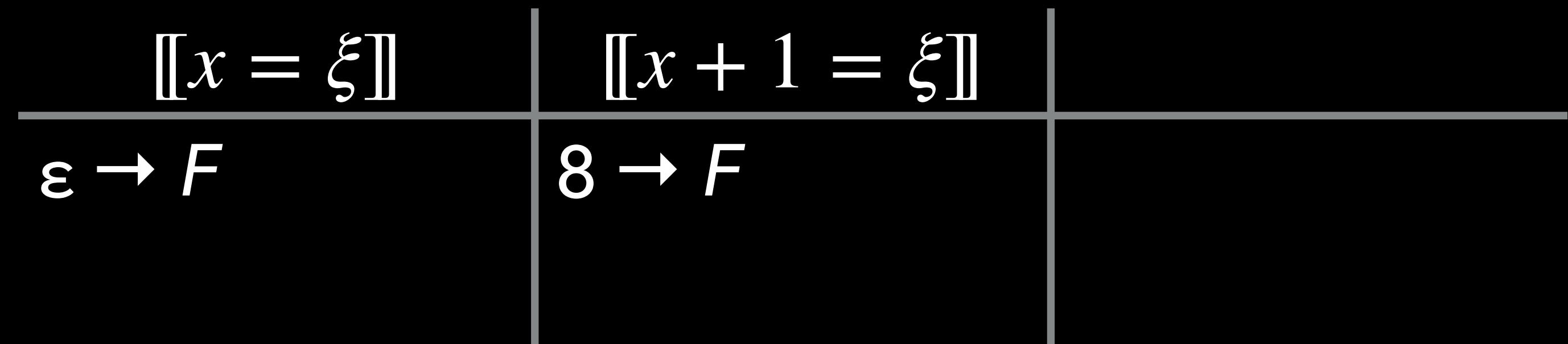
```
1 int x = 10
2 if prob(0.3) {
3     x -= 1
4 } else {
5     x = 20
6     x -= 1
7 }
8 x += 1
```



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



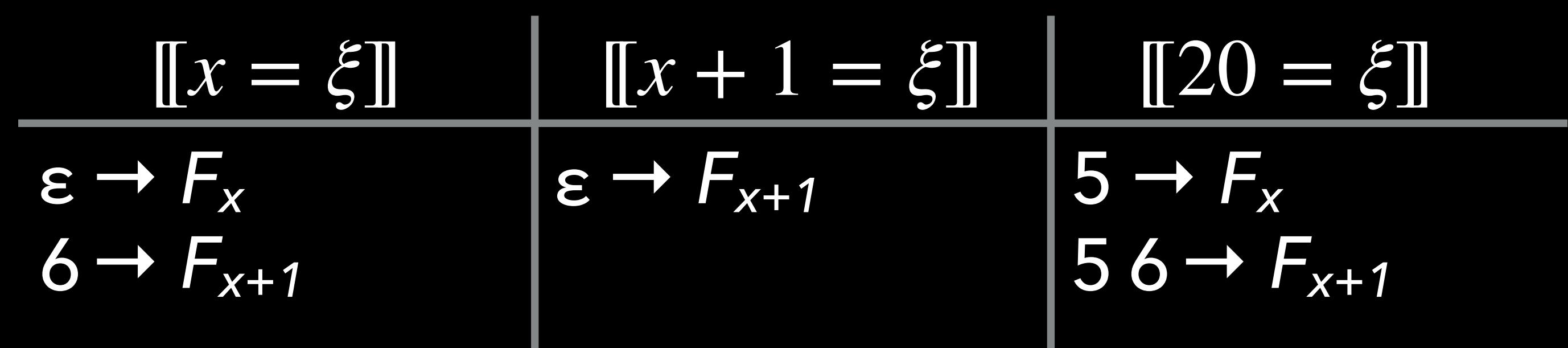
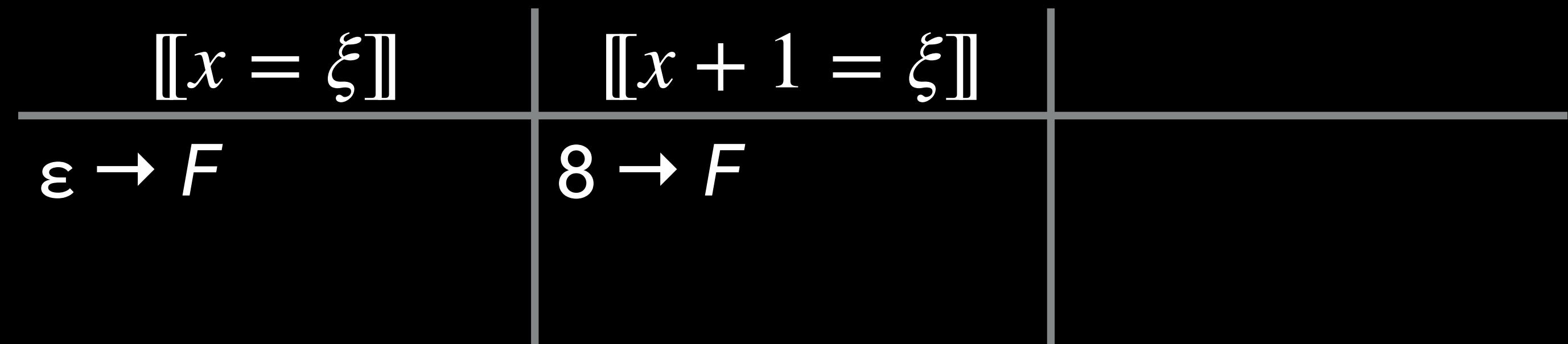
```
1 int x = 10
2 if prob(0.3) {
3     x -= 1
4 } else {
5     x = 20
6     x -= 1
7 }
8 x += 1
```



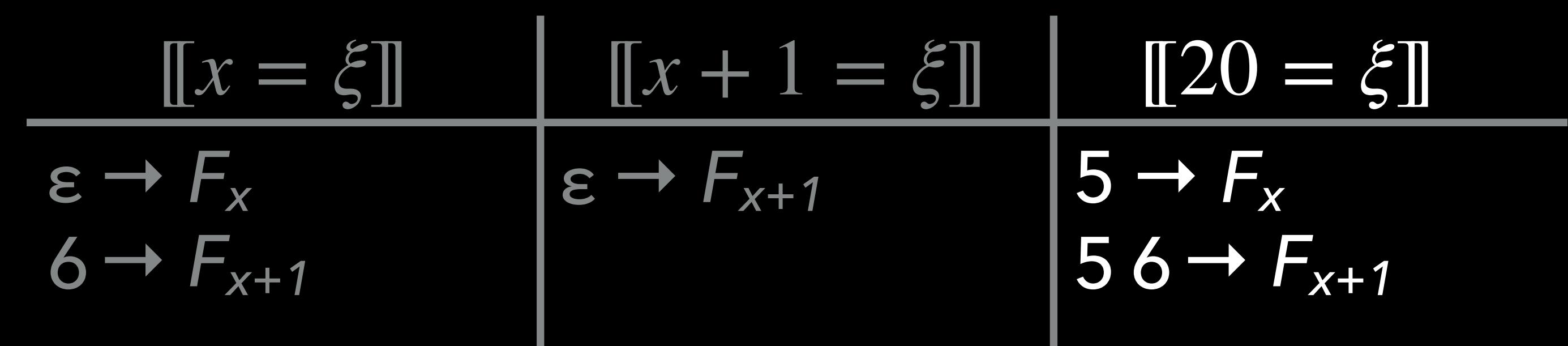
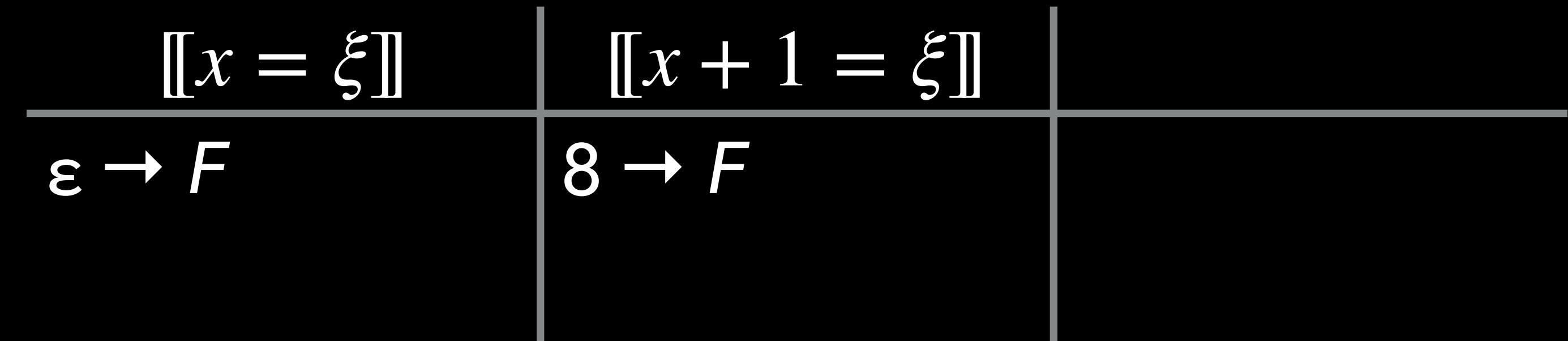
6 $x -= 1$



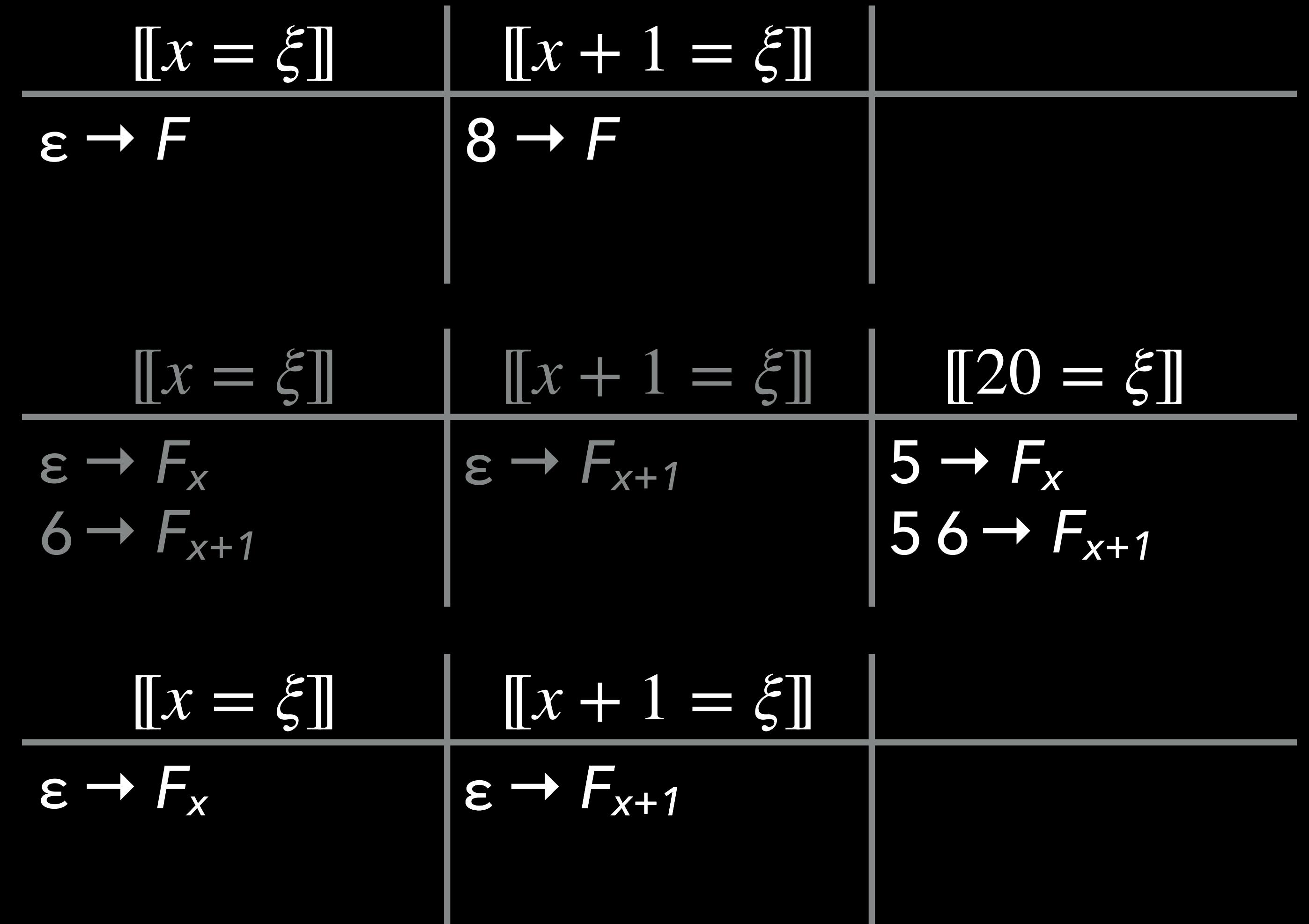
```
1 int x = 10
2 if prob(0.3) {
3     x -= 1
4 } else {
5     x = 20
6     x -= 1
7 }
8 x += 1
```



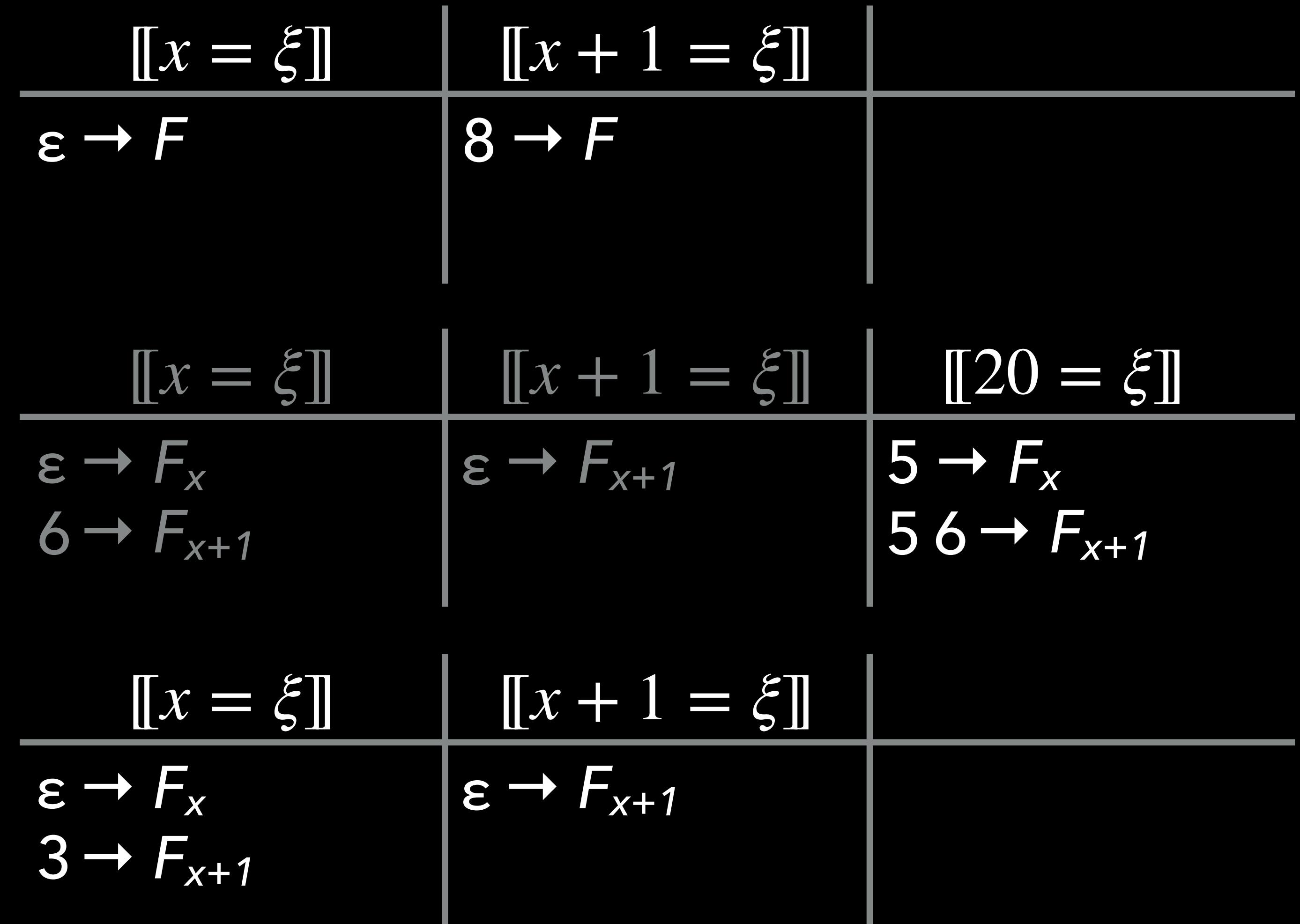
```
1 int x = 10
2 if prob(0.3) {
3     x -= 1
4 } else {
5     x = 20
6     x -= 1
7 }
8 x += 1
```



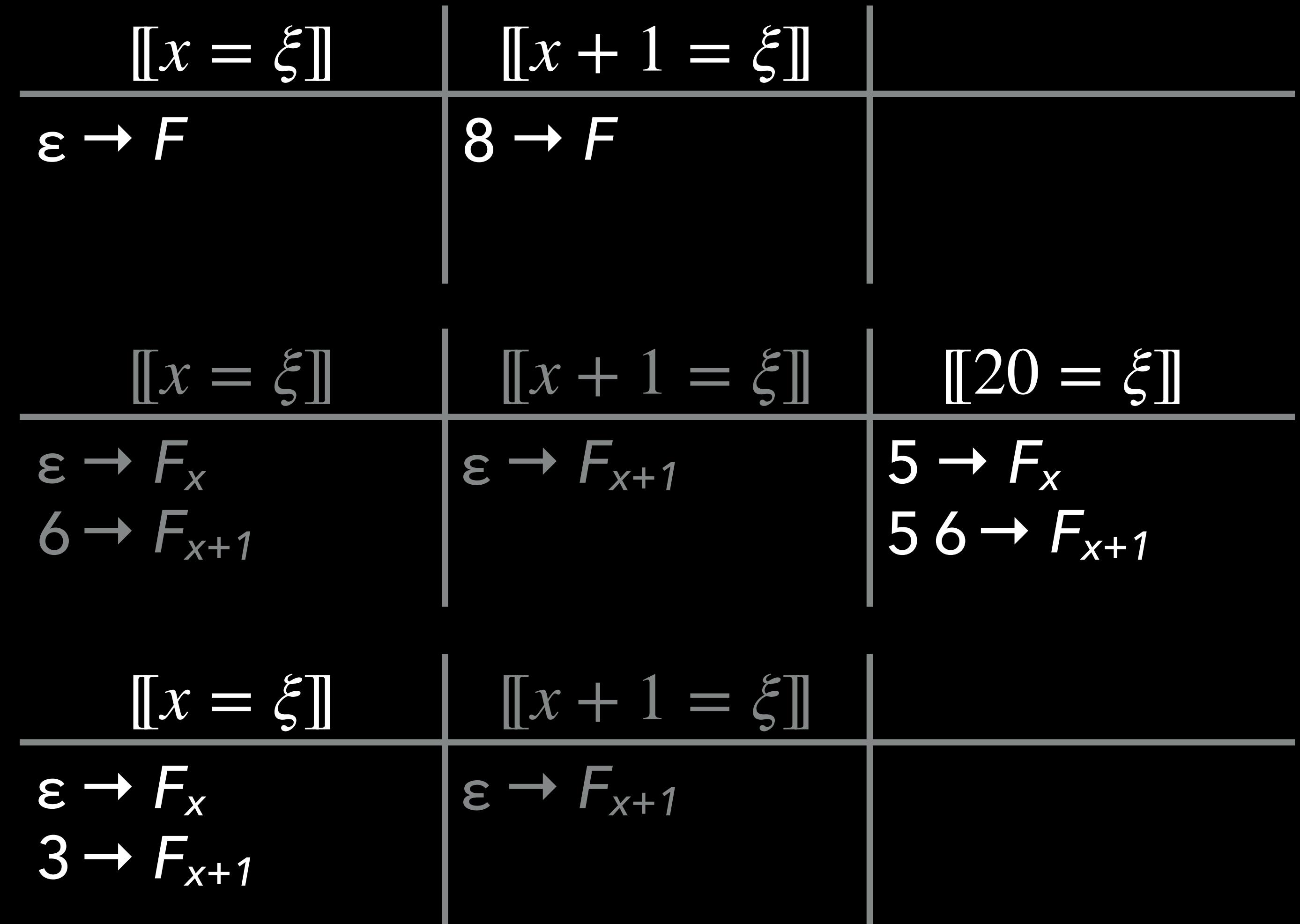
```
1 int x = 10
2 if prob(0.3) {
3     x -= 1
4 } else {
5     x = 20
6     x -= 1
7 }
8 x += 1
```



```
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
```



```
1 int x = 10
```

```
2 if prob(0.3) {
```

```
3     x -= 1
```

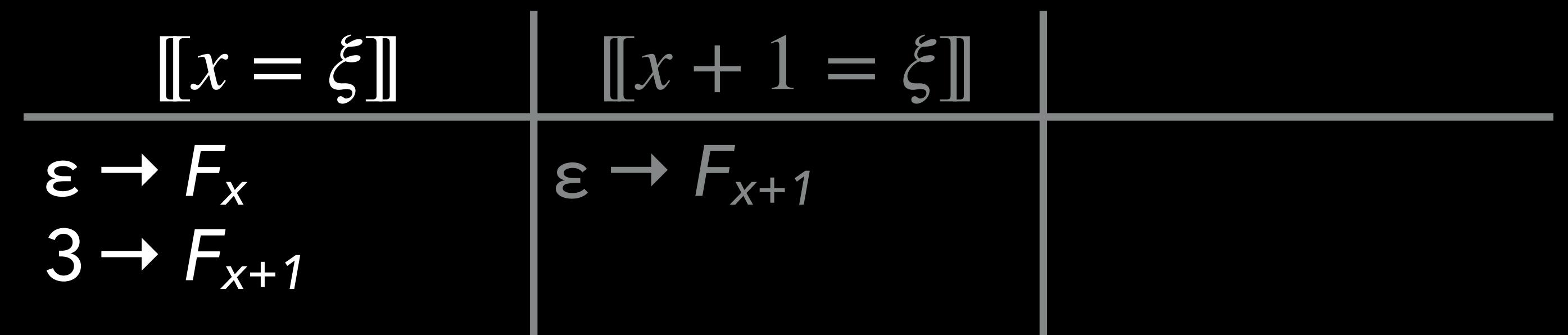
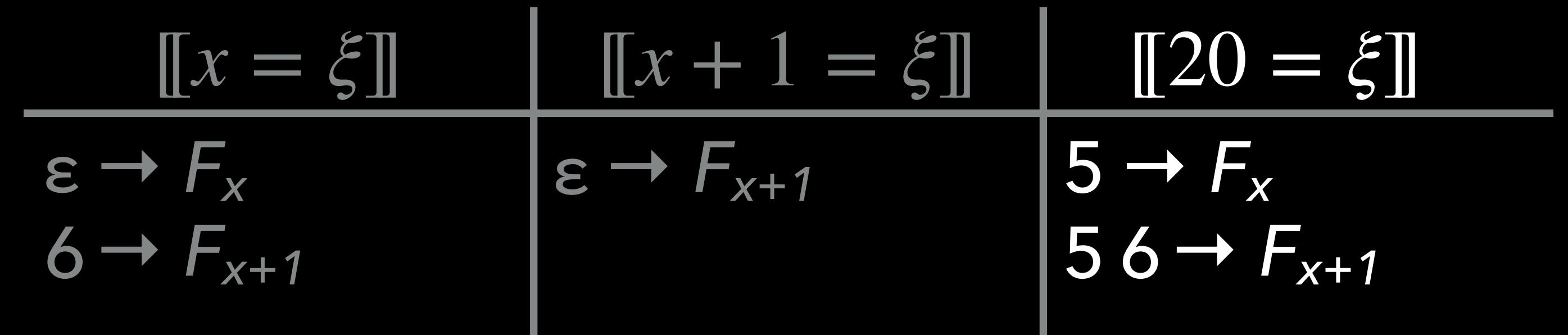
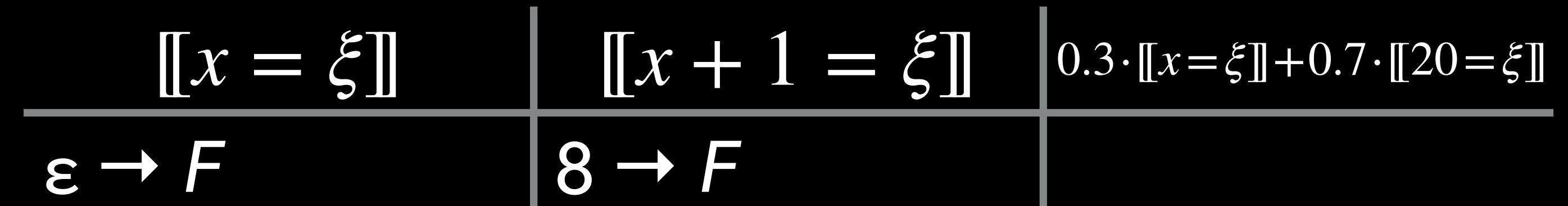
```
4 } else {
```

```
5     x = 20
```

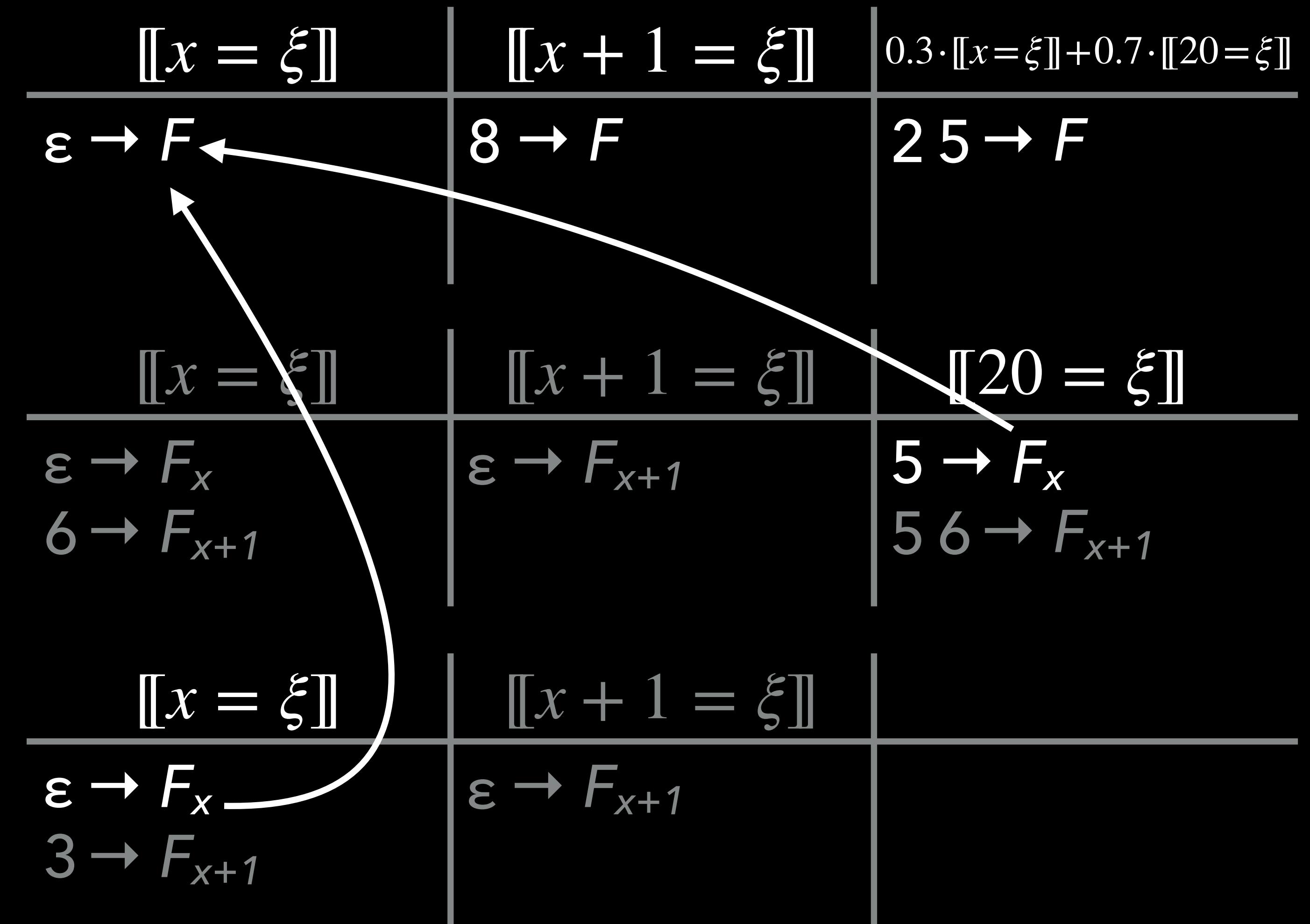
```
6     x -= 1
```

```
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
```

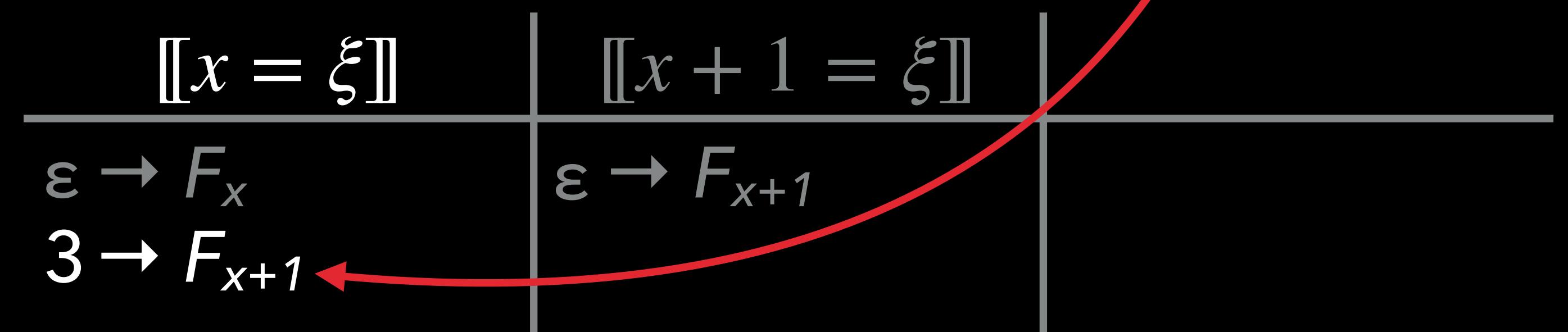
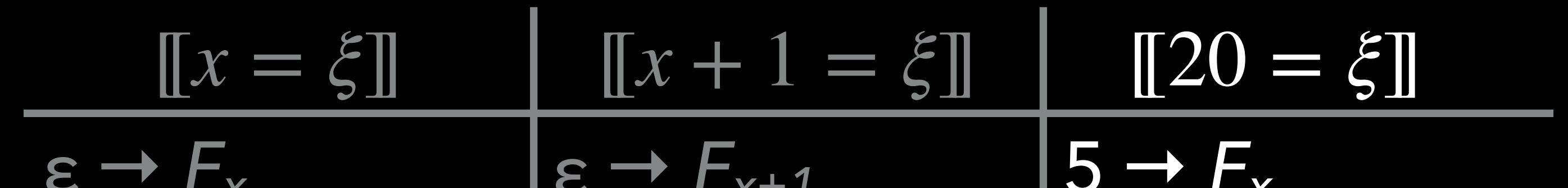
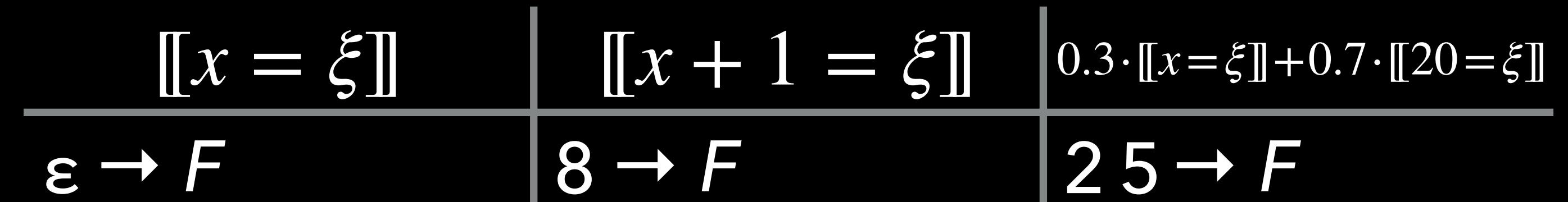
```
4 } else {
```

```
5     x = 20
```

```
6     x -= 1
```

```
7 }
```

```
8 x += 1
```



```
1 int x = 10
```

```
2 if prob(0.3) {
```

```
3   x -= 1
```

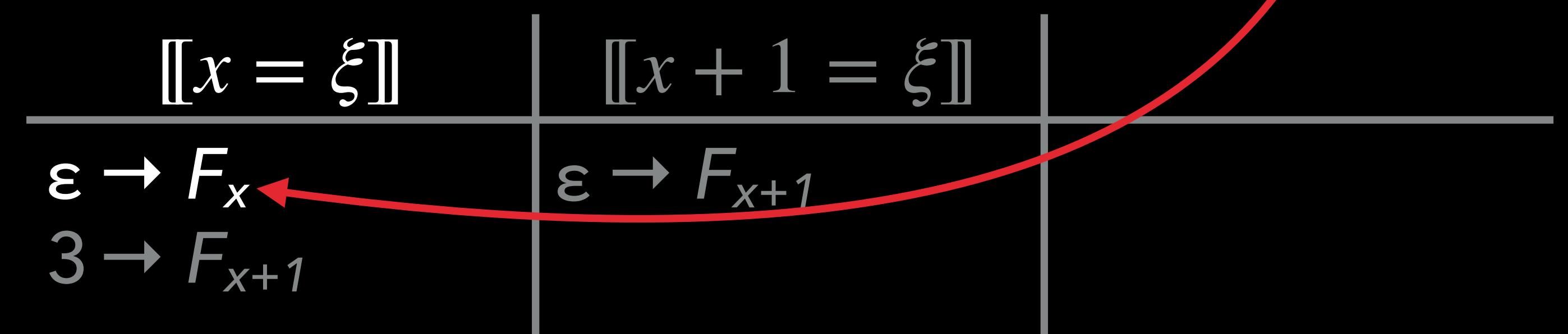
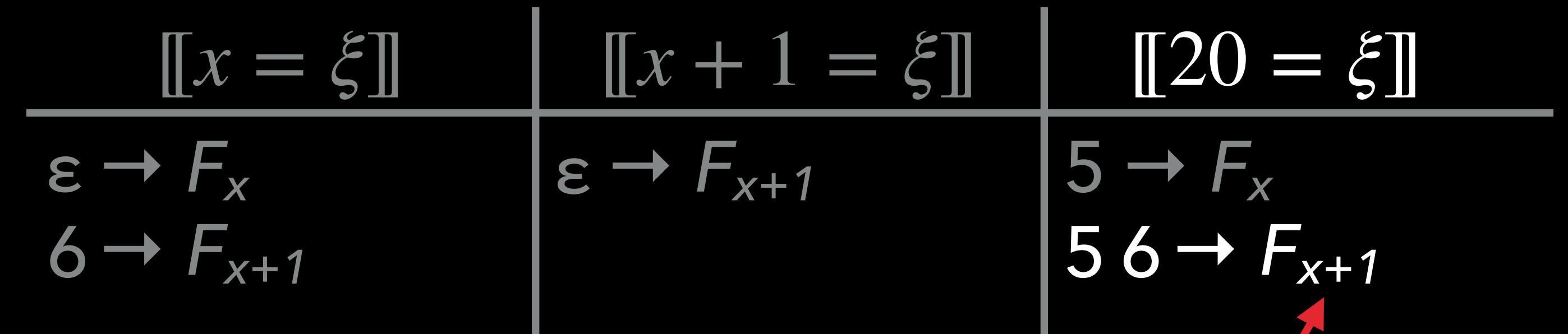
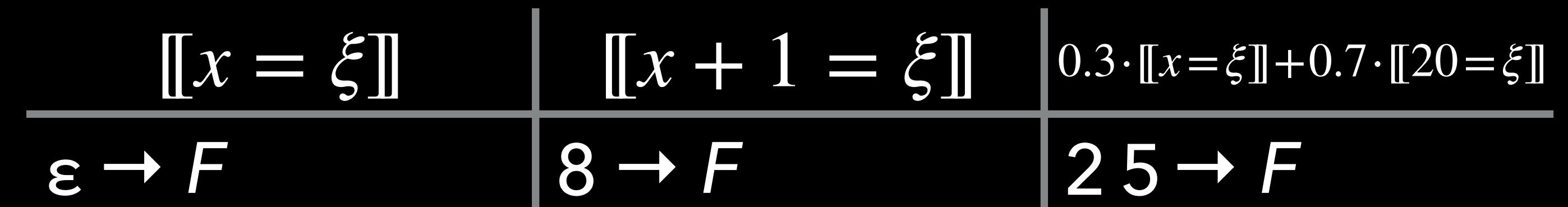
```
4 } else {
```

```
5   x = 20
```

```
6   x -= 1
```

```
7 }
```

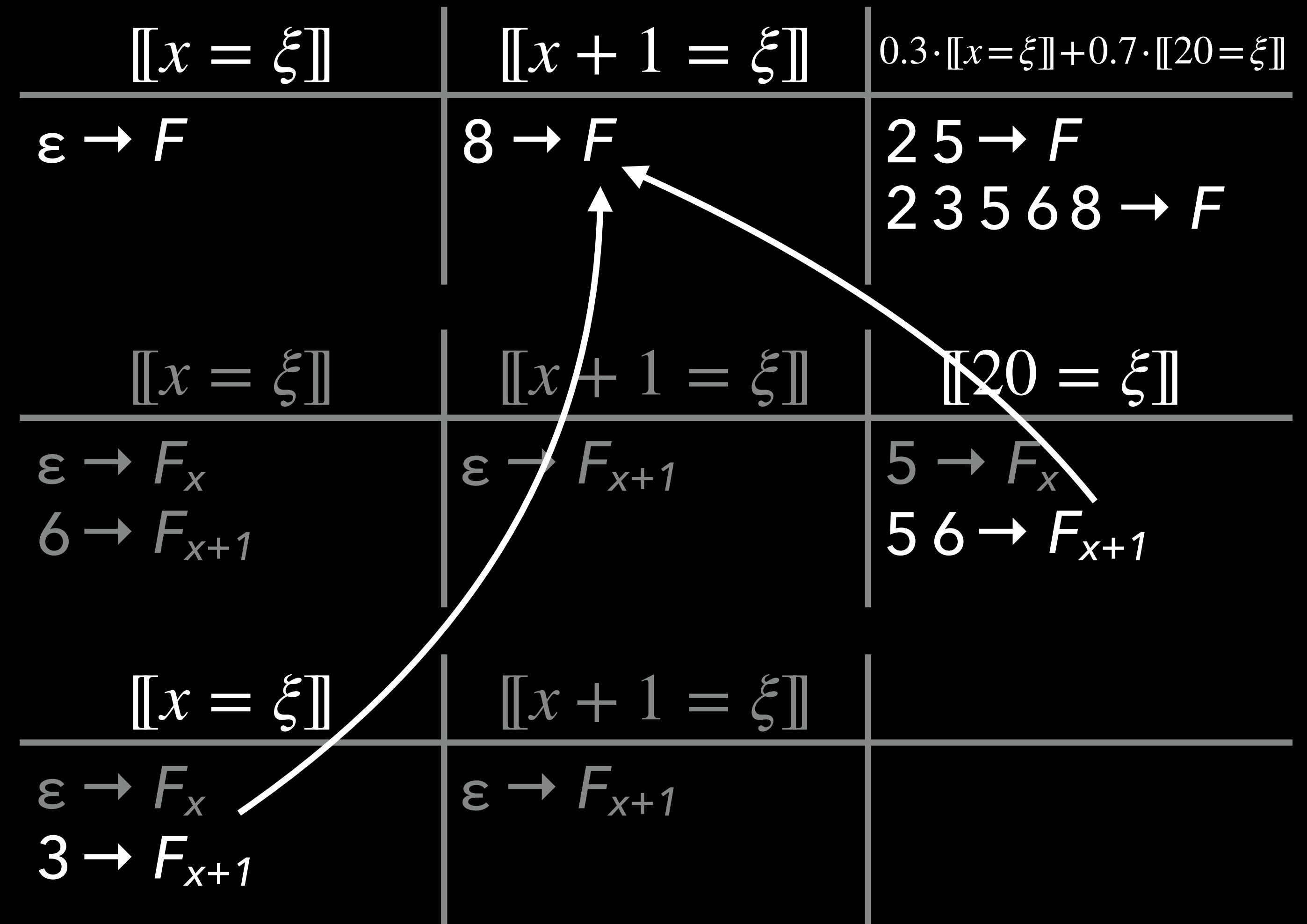
```
8 x += 1
```



```

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
```

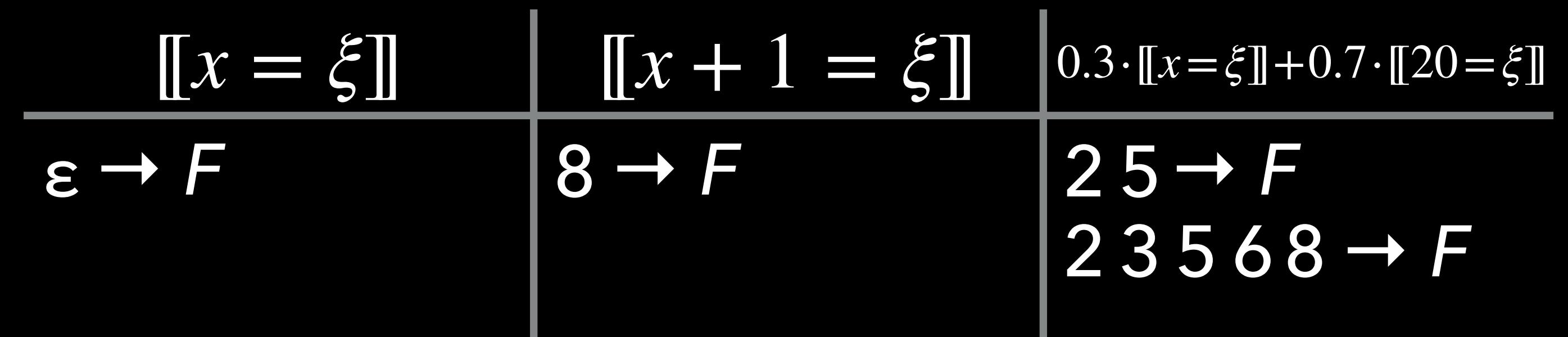
```
4 } else {
```

```
5     x = 20
```

```
6     x -= 1
```

```
7 }
```

```
8 x += 1
```



```
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 }
```

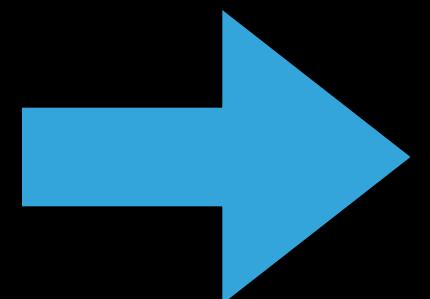
→ Slice w.r.t. expectation-triple instead of the fractional value if
 $\text{wp}(P,1)$ and $\text{woip}(P,1)$ of bodies doesn't match



```
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 }
```



Syntax highlighting



Static semantic functionality



Program execution



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

Debugger



Recording-based debugger



Slicing

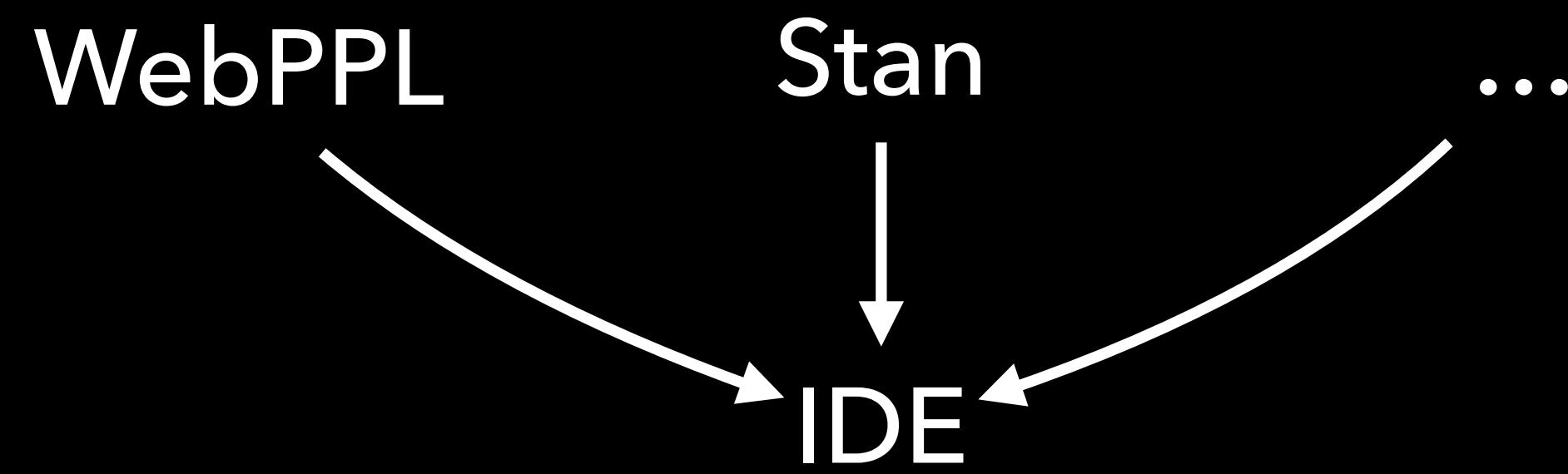


Support for multiple languages



Implementation



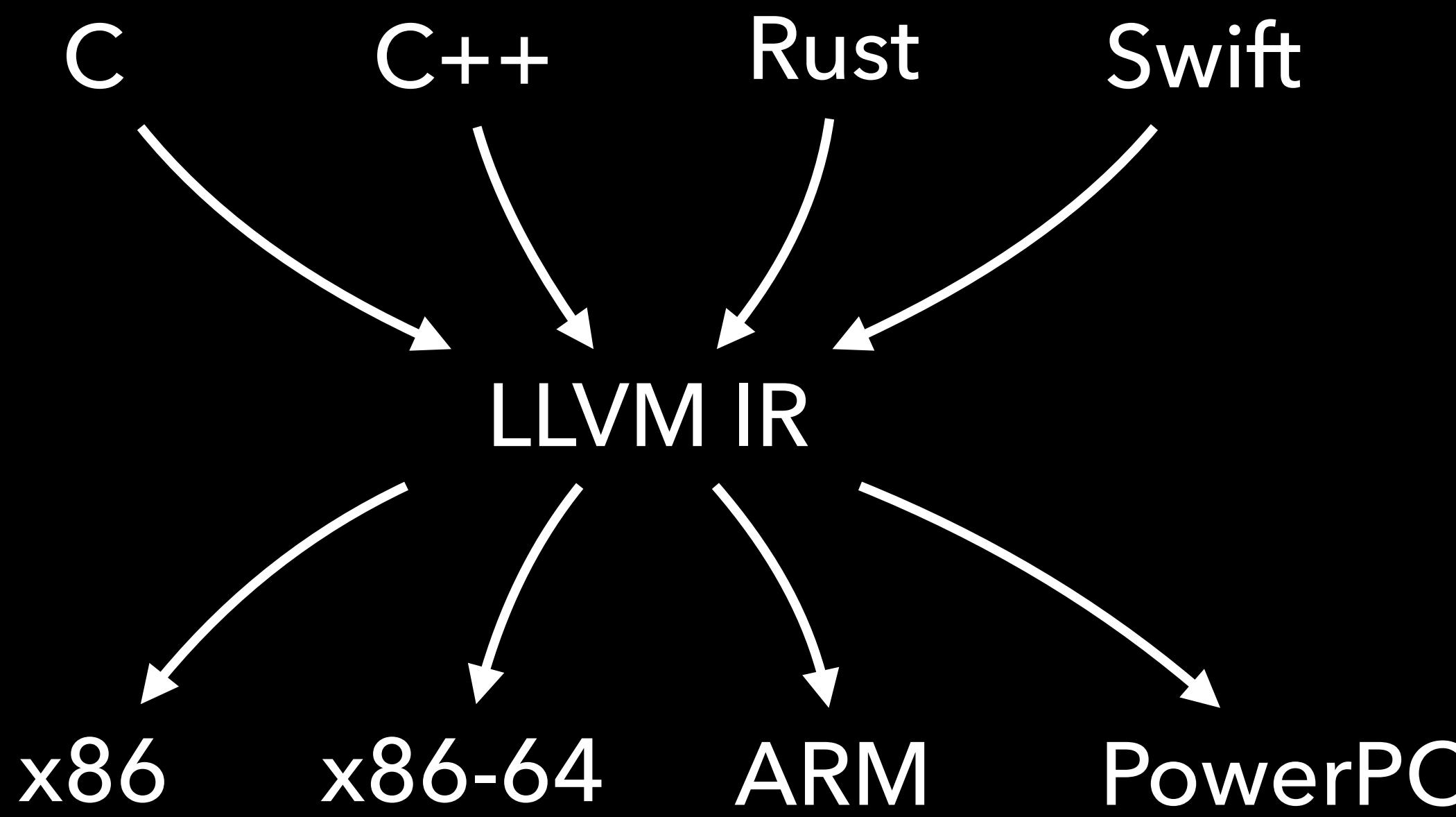
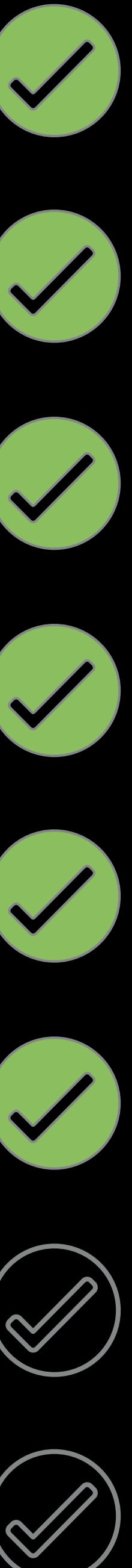


- Syntax highlighting 
- Static semantic functionality 
- Program execution 
- Debugger 
- Recording-based debugger 
- Slicing 
- Support for multiple languages 
- Implementation 

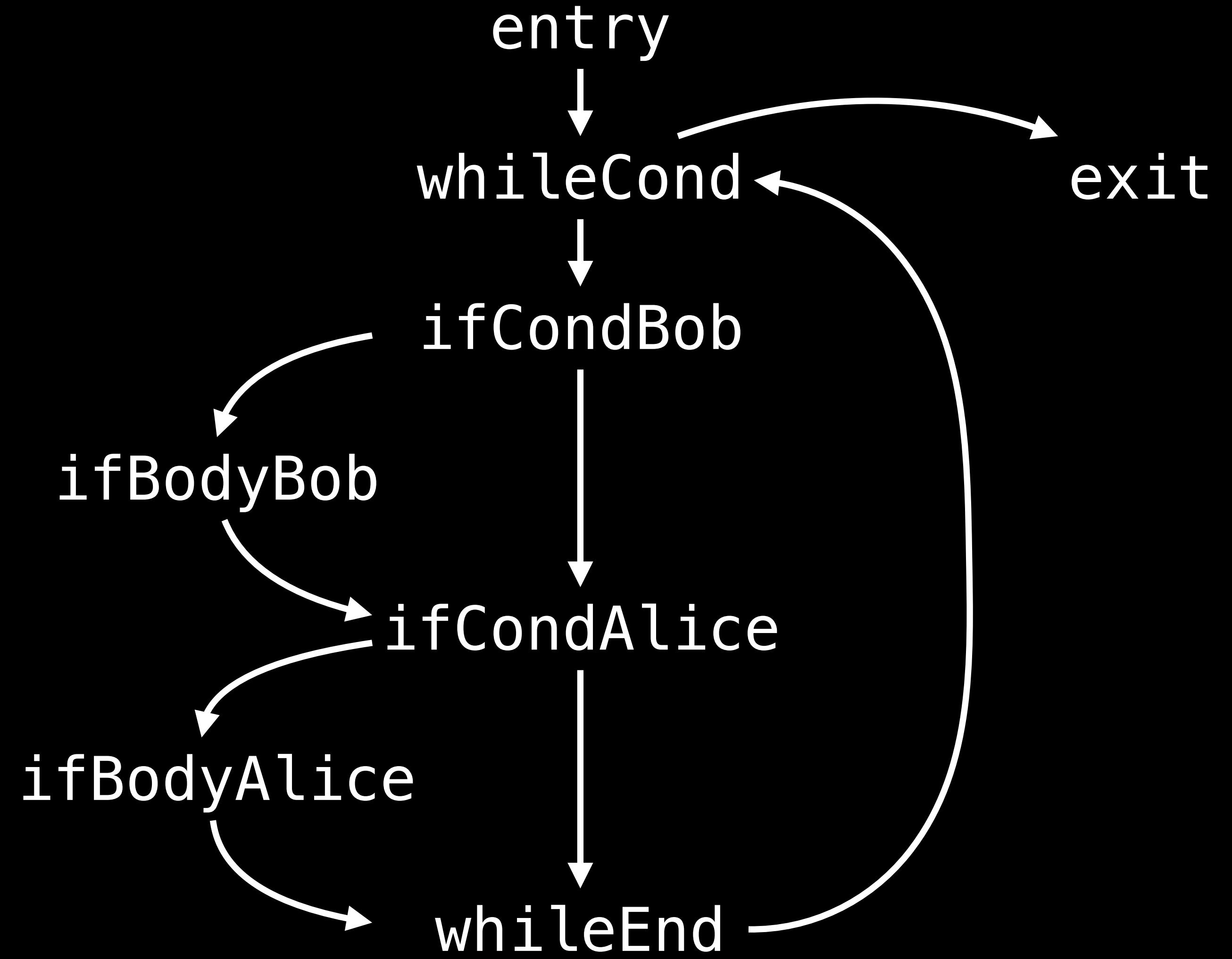
SUPPORT FOR MULTIPLE LANGUAGES

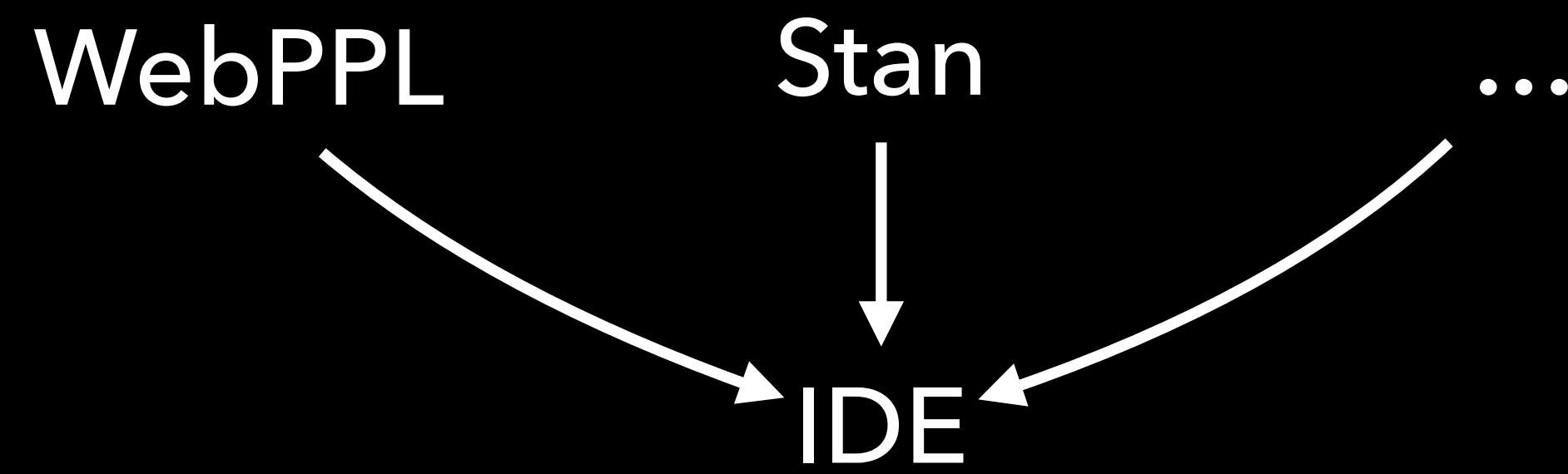
	Probability	
greta ^[46]	TensorFlow	R
pomegranate ^[47]	Python	Python
Lea ^[48]	Python	Python
WebPPL ^[49]	JavaScript	JavaScript
Picture ^[4]	Julia	Julia
Turing.jl ^[50]	Julia	Julia
Gen ^[51]	Julia	Julia
Low-level First-order PPL ^[52]	Python, Clojure, Pytorch	Various: Python, Clojure
Troll ^[53]		Moscow 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

ALEX HOPPEN



```
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



The screenshot shows a software interface with two main panes. The left pane is a tree view labeled 'Code' with nodes for 'int aliceInfections = 1', 'int bobInfected = 0', and a 'while' loop. The 'while' loop has children for 'iterations' (with 'iteration 1', 'iteration 2', and 'iteration 3') and a condition node. The condition node has two children: 'if discrete({0: 0.9, 1: 0.1}) == 1' and 'if discrete({0: 0.4, 1: 0.6}) == 1'. The right pane displays statistics for each node, including 'Samples' and 'Error'. For the condition node, the statistics are: Samples 16.0% and Error 0.0%. Below the statistics is a toolbar with icons for back, forward, and search, followed by a checkbox for 'Refine using WP' (which is checked) and a 'Distribute Error' button. At the bottom, there is a table showing variable values: 'aliceInfections' is 1.0 (Average) with 1: 100.0% (Values), and 'bobInfected' is 0.19 (Average) with 0: 81.0%, 1: 19.0% (Values). There are also 'Inspect' buttons for each variable.

Variable	Average	Values	Inspect
aliceInfections	1.0	1: 100.0%	(i)
bobInfected	0.19	0: 81.0%, 1: 19.0%	(i)

Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



DEMO

Syntax highlighting



Static semantic functionality



The screenshot shows a software interface with two main panes. The left pane is titled 'Code' and displays a C-like pseudocode program with annotations for 'Samples' and 'Error'. The right pane shows the source code with specific lines highlighted in green, indicating they are part of a slice or have been analyzed. A status bar at the bottom indicates 'Refine using WP' is checked.

Code	Samples	Error
int aliceInfections = 1	100.0%	0.0%
int bobInfected = 0	100.0%	0.0%
while aliceInfections == 1 {	100.0%	0.0%
if discrete({0: 0.9, 1: 0.1}) == 1 {	16.0%	0.0%
bobInfected = 1	16.0%	0.0%
if discrete({0: 0.4, 1: 0.6}) == 1 {	16.0%	0.0%
aliceInfections = 0	16.0%	0.0%
}	16.0%	0.0%
if discrete({0: 0.9, 1: 0.1}) == 1	16.0%	0.0%
if discrete({0: 0.4, 1: 0.6}) == 1	16.0%	0.0%
end	16.0%	0.0%
iteration 4	6.4%	0.0%
iteration 5	2.56%	0.0%
iteration 6	1.024%	0.0%
iteration 7	0.41%	0.0%
iteration 8	0.16%	0.0%
iteration 9	0.07%	0.0%
iteration 10	0.03%	0.0%
iteration 11	0.01%	0.0%
iteration 12	0.0%	0.0%
iteration 13	0.0%	0.0%
Exit states	0.0%	inf%
end	100.0%	=0%

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 $\left(\pm \frac{1 - \text{woip}_\beta(P,1)}{\text{wp}_\beta(P,1)} \right)$

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

Syntax highlighting



Static semantic functionality



Program execution



Debugger



Recording-based debugger



Slicing



Support for multiple languages



Implementation



- ▶ 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