Programmatic Strategy Synthesis

Resolving Nondeterminism in Probabilistic Programs

Tobias Winkler
with Kevin Batz, Tom Jannik Biskup, and Joost-Pieter Katoen

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

DFG Research Training Group 2236

erc

POPL 2024 — 18.01.2024

A Gamble with Two Coins

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)
- Repeatedly select a coin and flip it
  ➡ Get £ (heads) or game over (tails)

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)
- Repeatedly select a coin and flip it
  ➡ Get £ (heads) or game over (tails)
- Start with x £

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)
- Repeatedly select a coin and flip it
  ➡ Get £ (heads) or game over (tails)
- Start with x £
- Win once we have at least N £

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)
- Repeatedly select a coin and flip it
  ➡ Get £ (heads) or game over (tails)
- Start with x £
- Win once we have at least N £
- Task: maximize winning probability

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)
- Repeatedly select a coin and flip it
  ➡ Get £ (heads) or game over (tails)
- Start with x £
- Win once we have at least N £
- Task: maximize winning probability

```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (true)
    ->    {x ≔ x+1} [q] {tails ≔ true}
    [] (true)
    ->    {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

A Gamble with Two Coins

- Two coins with bias q (1£) and p (2£)
- Repeatedly select a coin and flip it
  ➡ Get £ (heads) or game over (tails)
- Start with x £
- Win once we have at least N £
- Task: maximize winning probability

```
tails ≔ false ;
while (x<N ∧ !tails) ->
     if (true)
     ->   {x ≔ x+1} [q] {tails ≔ true}
     [] (true)
     ->   {x ≔ x+2} [p] {tails ≔ true}
     end
end
// [x ≥ N  ∧  ¬tails]
```

A Gamble with Two Coins

Optimal Solution

```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

// $[x \geq N \ \wedge \ \neg tails]$

A Gamble with Two Coins

Optimal Solution

- Goal: find these predicates

```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N ∧ ¬tails]
```

A Gamble with Two Coins

Optimal Solution

- Goal: find these predicates
- Transformed program = strategy

```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N  ∧  ¬tails]
```

A Gamble with Two Coins

Optimal Solution

- Goal: find these predicates
- Transformed program = strategy
- Strategies are permissive & parametric

```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N ∧ ¬tails]
```

A Gamble with Two Coins

Optimal Solution

- Goal: find these predicates
- Transformed program = strategy
- Strategies are permissive & parametric
- Loops: rely on `@invariant` annotations

```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N-x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N-x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N  ∧  ¬tails]
```

Weakest Preconditions

[Dijkstra ’75]

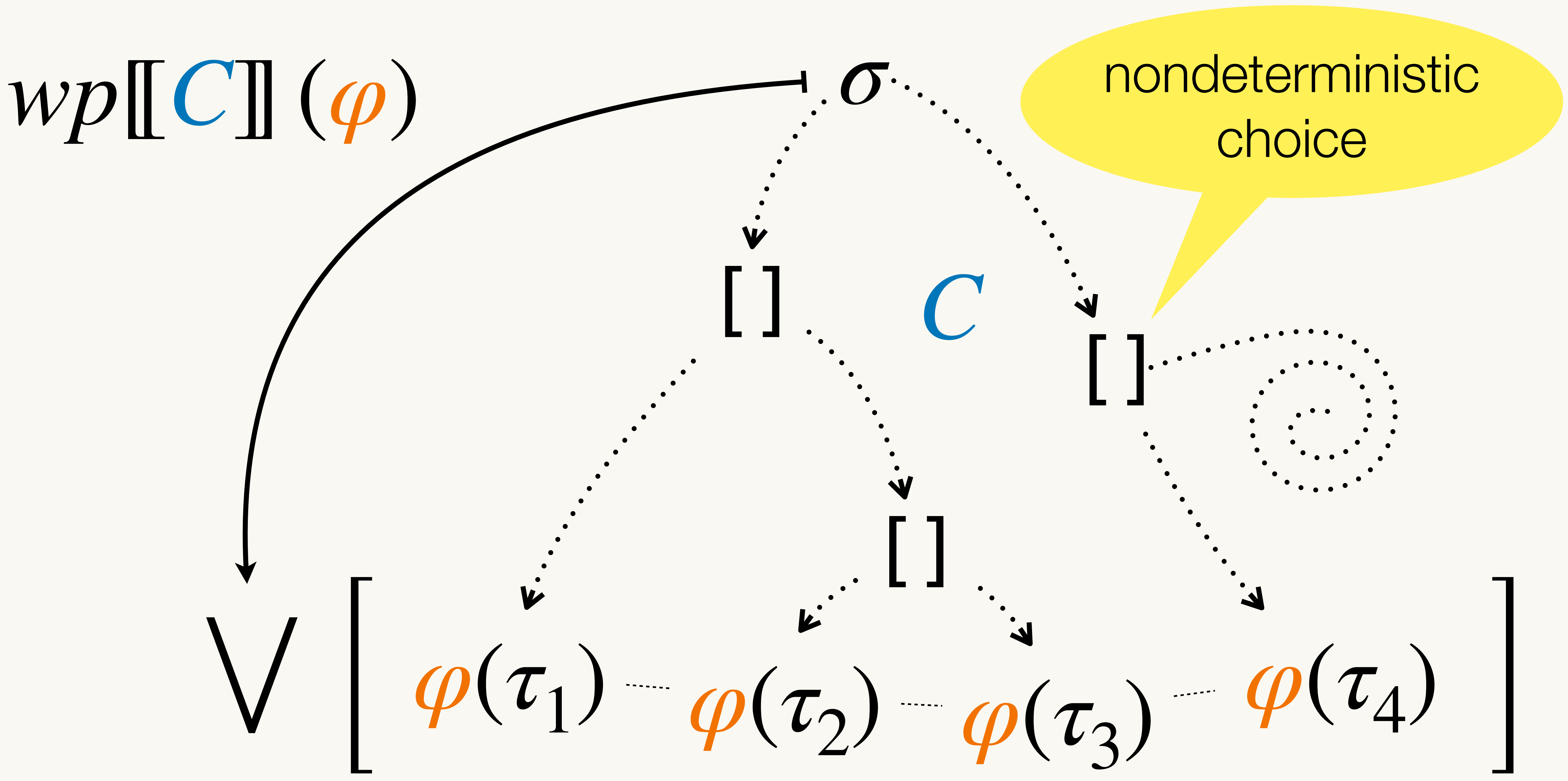
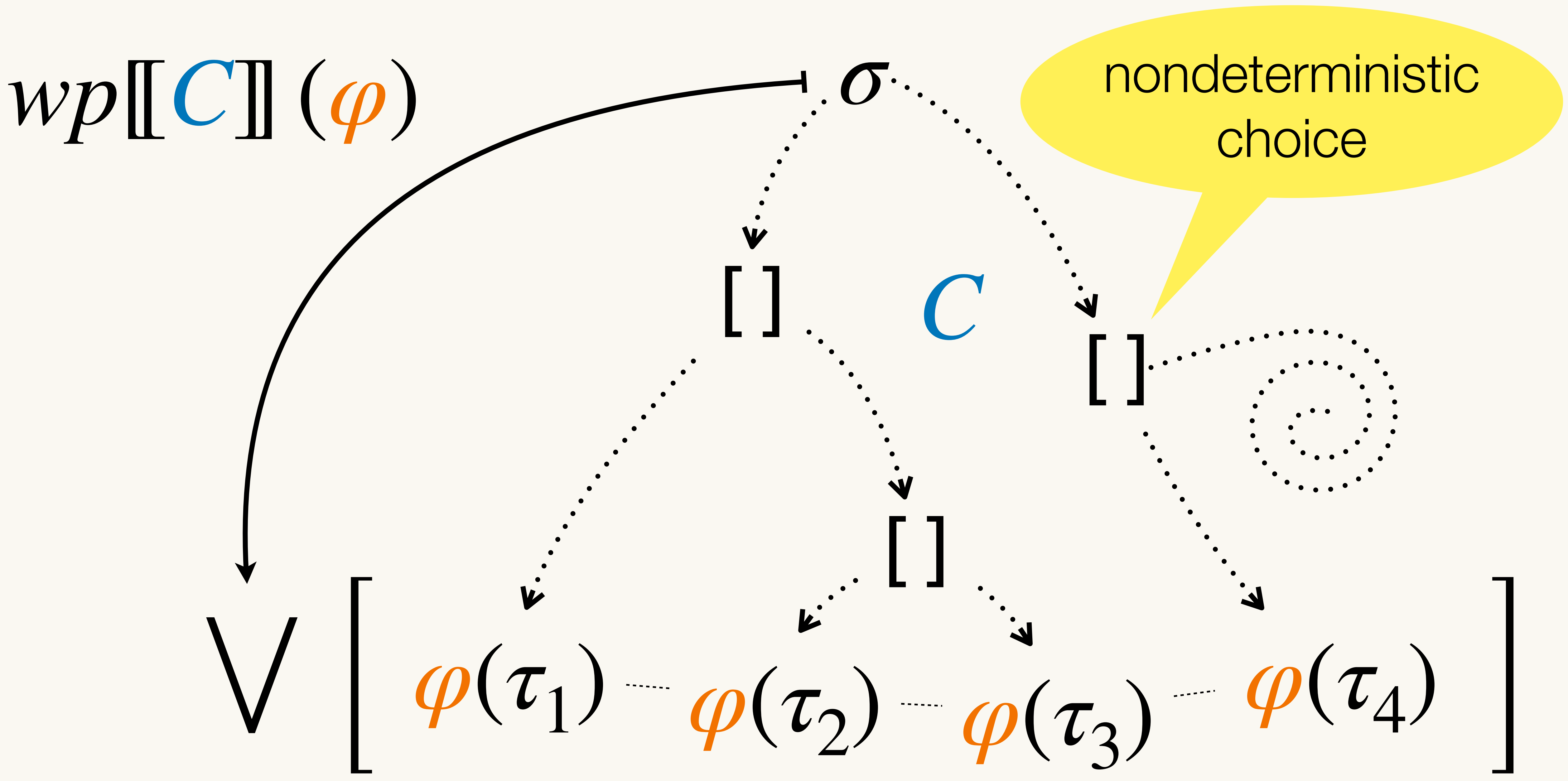


Illustration based on [Batz et al. OOPSLA ’22]

Weakest Preconditions

[Dijkstra ’75]



// $y \geq z \;=\; wp[\![C]\!](x = 2)$

```
if (y ≤ z) -> {x ≔ 1}
[] (y ≥ z) -> {x ≔ 2}
end
```

// $x = 2$

Illustration based on [Batz et al. OOPSLA ’22]

Weakest Pre-*expectations*

For Probabilistic Programs

[Kozen ‘83, McIver & Morgan ’05, Kaminski ‘19]

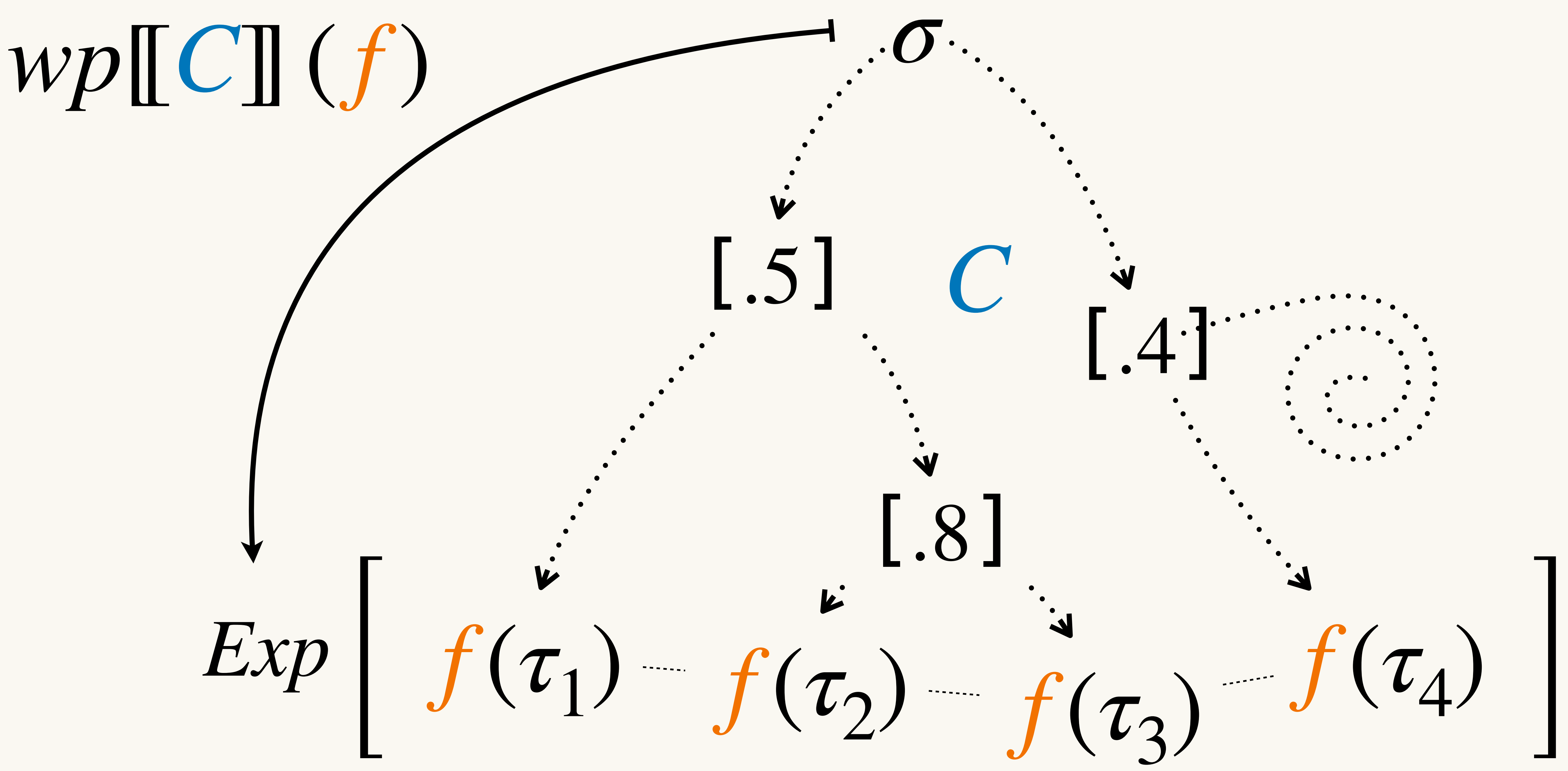
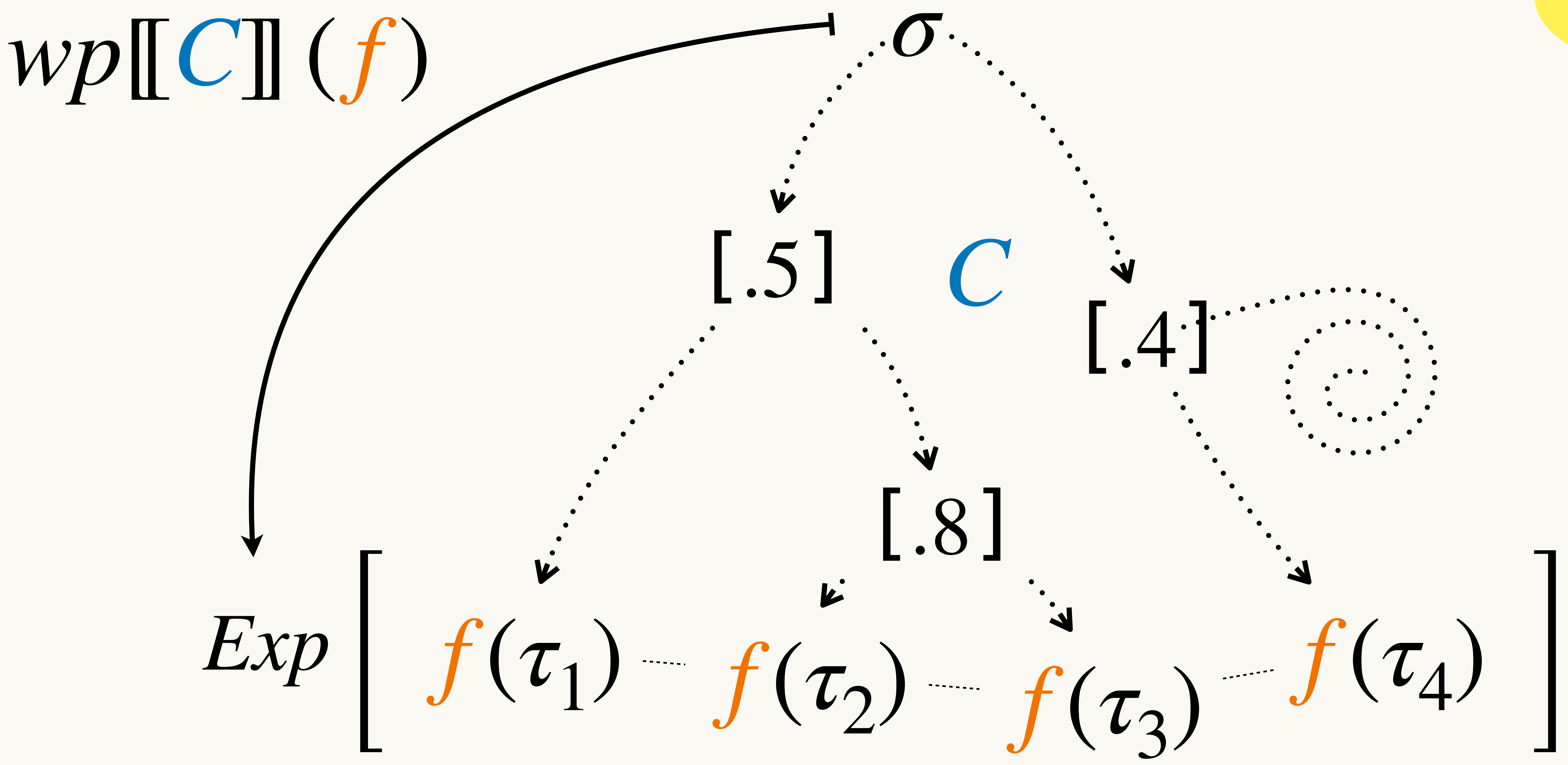


Illustration based on [Batz et al. OOPSLA ’22]

Weakest Pre-*expectations*

For Probabilistic Programs

[Kozen ‘83, McIver & Morgan ’05, Kaminski ‘19]



expected value of x after termination

$// \; 2x \;\; = \;\; wp[\![C]\!](x)$

```
{x ≔ 4x} [.5] {x ≔ 0}
```

$// \; x$

Illustration based on [Batz et al. OOPSLA ’22]

Weakest Pre-*expectations*

For Probabilistic Programs *with Nondeterminism*

[McIver & Morgan '05]

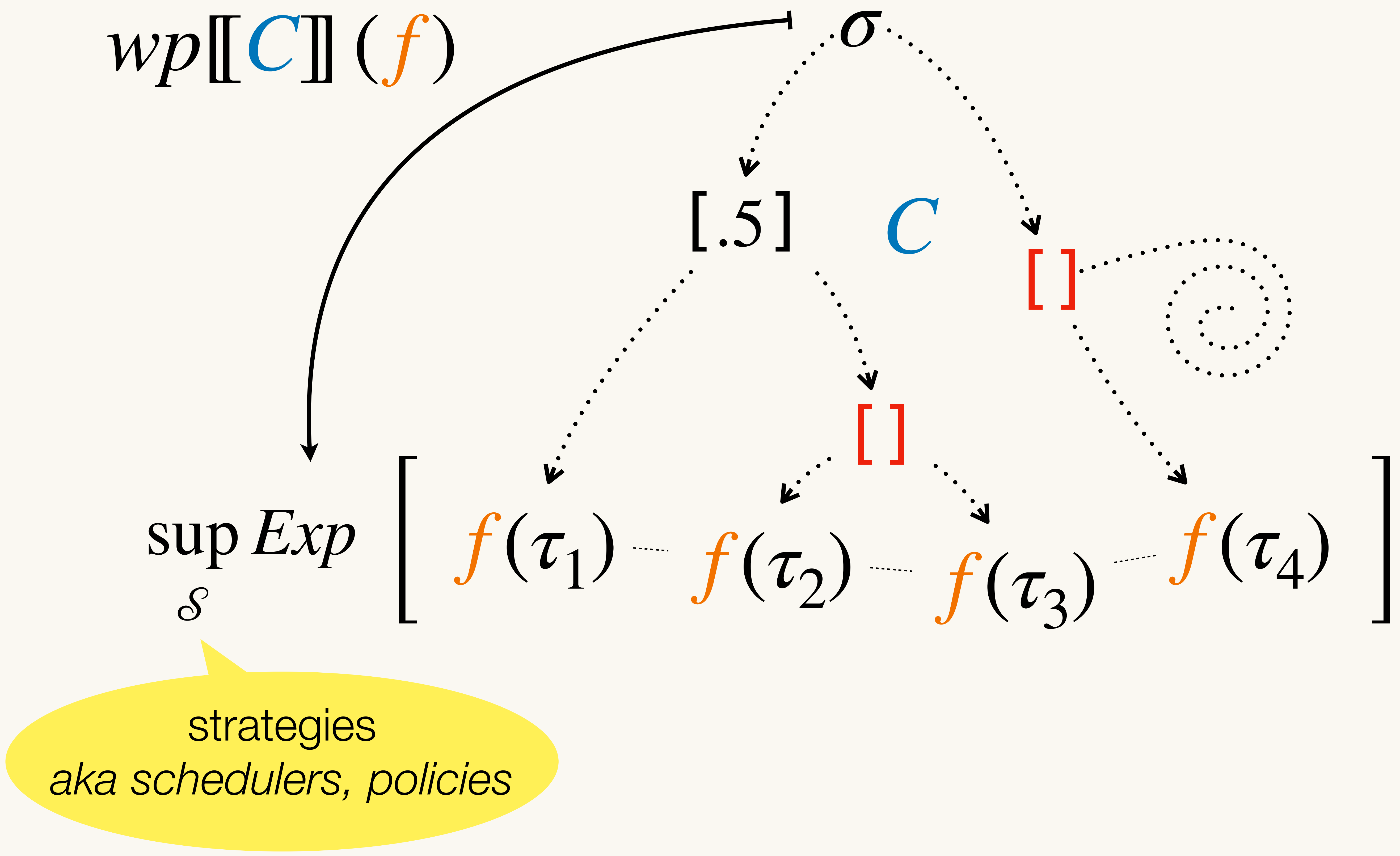
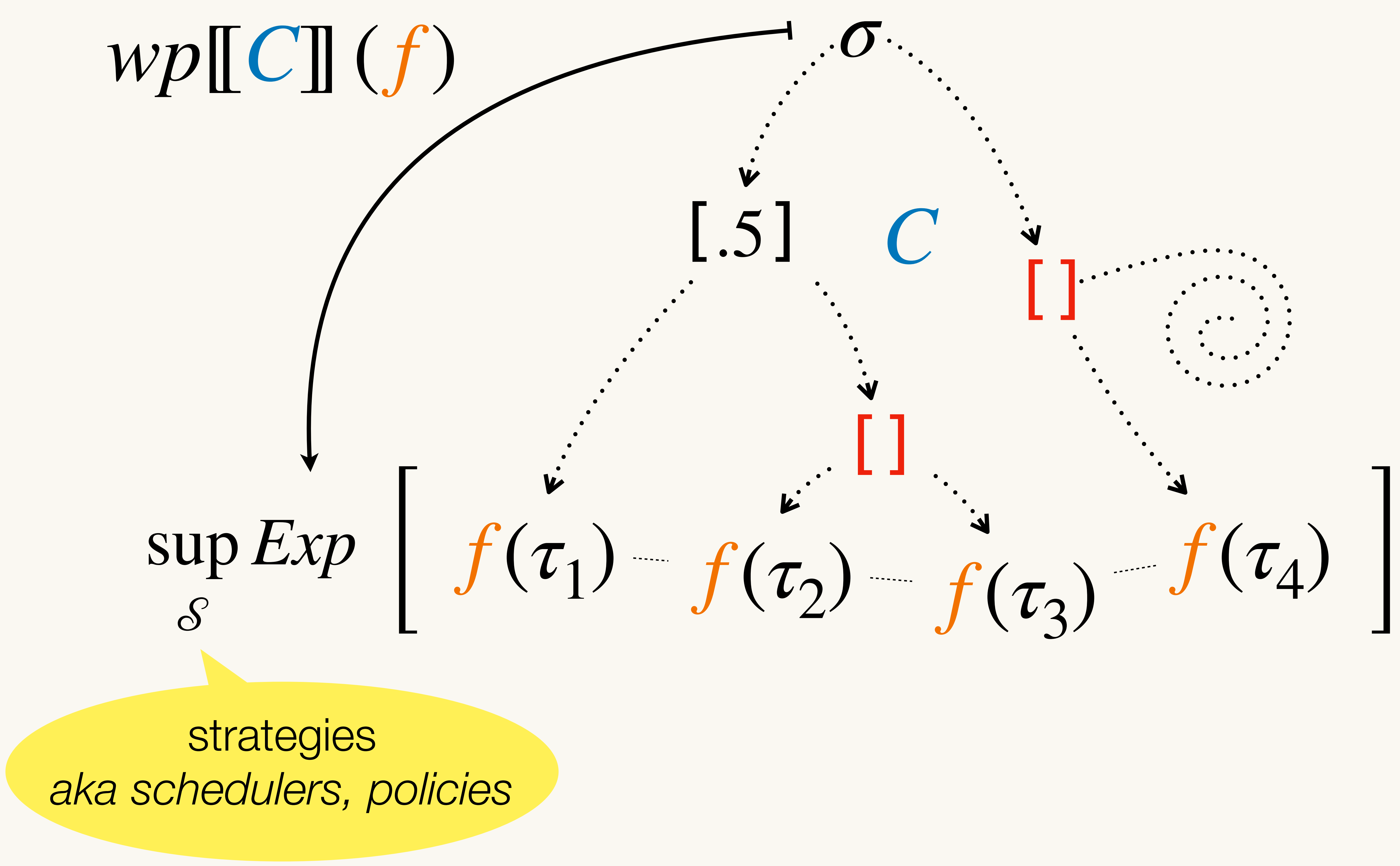


Illustration based on [Batz et al. OOPSLA '22]

Weakest Pre-*expectations*

For Probabilistic Programs *with Nondeterminism*

[McIver & Morgan ‘05]




```
// [y ≥ z] · 2 + [y < z] · 0
if (y ≤ z) -> {x ≔ 0}
[] (y ≥ z) -> {x ≔ 1}
end ;
{x ≔ 4x} [.5] {x ≔ 0}
// x
```


Illustration based on [Batz et al. OOPSLA ’22]

Inductive Definition of *wp*

[McIver & Morgan '05]

C	$wp[\![C]\!](f)$
`x ≔` $Expr$	
C_1 `;` C_2	
`{`C_1`} [`p`] {`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	
`while` φ `->` C' `end`	

Inductive Definition of *wp*

[McIver & Morgan ‘05]

C	$wp[\![C]\!](f)$
`x :=` $Expr$	“substitute x by $Expr$ in f”
C_1 `;` C_2	
`{`C_1`} [`p`] {`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	
`while` φ `->` C' `end`	

Inductive Definition of *wp*

[McIver & Morgan ‘05]

C	$wp[\![C]\!](f)$
`x ≔` $Expr$	“substitute x by $Expr$ in f”
C_1 `;` C_2	$wp[\![C_1]\!]\big(wp[\![C_2]\!](f)\big)$
`{`C_1`} [`p`] {`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	
`while` φ `->` C' `end`	

Inductive Definition of *wp*

[McIver & Morgan '05]

C	$wp[\![C]\!](f)$
`x ≔` $Expr$	"substitute x by $Expr$ in f"
C_1 `;` C_2	$wp[\![C_1]\!]\big(wp[\![C_2]\!](f)\big)$
`{`C_1`} [`p`] {`C_2`}`	$p \cdot wp[\![C_1]\!](f) \; + \; (1-p) \cdot wp[\![C_2]\!](f)$
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	
`while` φ `->` C' `end`	

Inductive Definition of *wp*

[McIver & Morgan '05]

C	$wp[\![C]\!]\,(f)$
`x := ` $Expr$	"substitute x by $Expr$ in f"
C_1 `;` C_2	$wp[\![C_1]\!]\big(wp[\![C_2]\!](f)\big)$
`{`C_1`} [`p`] {`C_2`}`	$p \cdot wp[\![C_1]\!]\,(f) \;+\; (1-p) \cdot wp[\![C_2]\!]\,(f)$
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	$\max\left\{ [\varphi_1] \cdot wp[\![C_1]\!]\,(f),\; [\varphi_2] \cdot wp[\![C_2]\!]\,(f) \right\}$
`while` φ `->` C' `end`	

Inductive Definition of *wp*

[McIver & Morgan '05]

C	$wp[\![C]\!](f)$
`x := ` $Expr$	"substitute x by $Expr$ in f"
C_1 `;` C_2	$wp[\![C_1]\!]\big(wp[\![C_2]\!](f)\big)$
`{`C_1`} [`p`] {`C_2`}`	$p \cdot wp[\![C_1]\!](f) \ + \ (1-p) \cdot wp[\![C_2]\!](f)$
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	$\max\left\{ [\varphi_1] \cdot wp[\![C_1]\!](f), \ [\varphi_2] \cdot wp[\![C_2]\!](f) \right\}$
`while` φ `->` C' `end`	$\mathrm{lfp}\ Y.\ [\varphi] \cdot wp[\![C']\!](Y) \ + \ [\overline{\varphi}] \cdot f$

Computing *wp*

```
if ( true   ) -> {      x ≔ y      }
[] ( true   ) -> {      x ≔ z      }
end ;



{      x ≔ 4x     } [.5] {     x ≔ 0     }
```

Computing *wp*

```
if ( true   ) -> {      x ≔ y      }
[] ( true   ) -> {      x ≔ z      }
end ;


{      x ≔ 4x      } [.5] {      x ≔ 0      }
// x
```

Computing *wp*

```
if ( true   ) -> {      x ≔ y      }
[] ( true   ) -> {      x ≔ z      }
end ;



{      x ≔ 4x // x } [.5] {     x ≔ 0 // x }
// x
```

Computing *wp*

```
if ( true  ) -> {     x ≔ y     }
[] ( true  ) -> {     x ≔ z     }
end ;
```

```
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

Computing *wp*

```
if ( true   ) -> {      x ≔ y      }
[] ( true   ) -> {      x ≔ z      }
end ;
```

// $0.5 \cdot 4x + 0.5 \cdot 0$

`{` // $4x$ `x ≔ 4x` // x `} [.5] {` // 0 `x ≔ 0` // x `}`

// x

Computing *wp*

```
if ( true   ) -> {      x ≔ y      }
[] ( true   ) -> {      x ≔ z      }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x  x ≔ 4x // x } [.5] { // 0  x ≔ 0 // x }
// x
```

Computing *wp*

```
if ( true  ) -> {      x ≔ y // 2x }
[] ( true  ) -> {      x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

Computing *wp*

```
if ( true  ) -> { // 2y x ≔ y // 2x }
[] ( true  ) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

Computing *wp*

```
// max { [true] · 2y , [true] · 2z }
if ( true  ) -> { // 2y  x ≔ y  // 2x  }
[] ( true  ) -> { // 2z  x ≔ z  // 2x  }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x  x ≔ 4x  // x  } [.5] { // 0  x ≔ 0  // x  }
// x
```

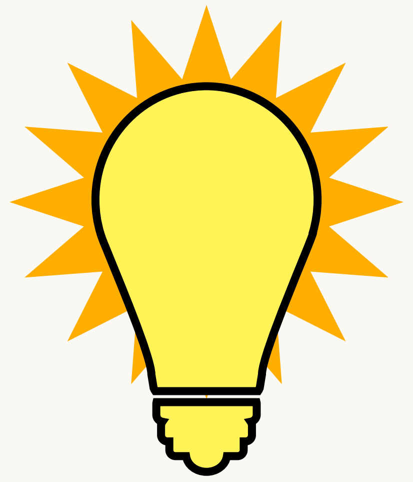
Computing *wp*

```
// 2 · max{y, z}
// max { [true] · 2y , [true] · 2z }
if ( true ) -> { // 2y x ≔ y // 2x }
[] ( true ) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

From *wp* Computation to Strategies

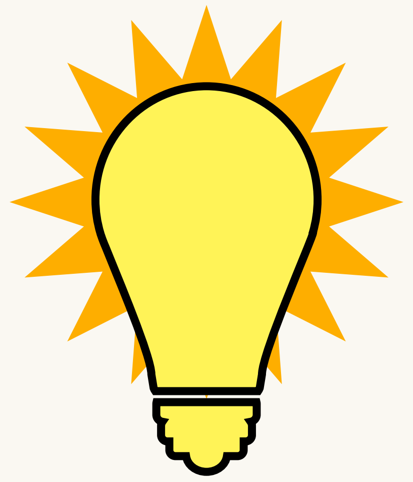
```
if (          ) -> { // 2y  x ≔ y  // 2x }
[] (          ) -> { // 2z  x ≔ z  // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x  x ≔ 4x  // x } [.5] { // 0  x ≔ 0  // x }
// x
```

From *wp* Computation to Strategies



```
if (2y ≥ 2z) -> { // 2y x ≔ y // 2x }
[] (       ) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

From *wp* Computation to Strategies

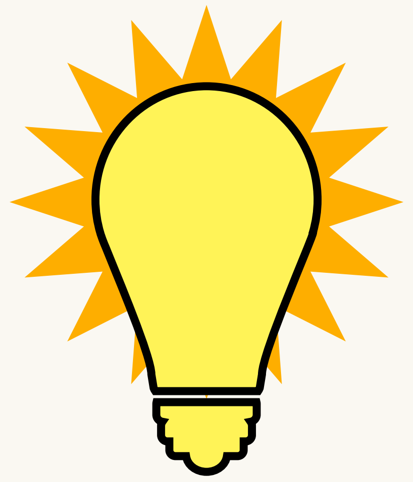


```
if (2y ≥ 2z) -> { // 2y x ≔ y // 2x }
[] (2y ≤ 2z) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

From *wp* Computation to Strategies

```
if ( y ≥  z) -> { // 2y x ≔ y // 2x }
[] ( y ≤  z) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

From *wp* Computation to Strategies



```
// max { [y ≥ z] · 2y , [y ≤ z] · 2z }
if ( y ≥  z) -> { // 2y  x ≔ y  // 2x  }
[] ( y ≤  z) -> { // 2z  x ≔ z  // 2x  }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{  // 4x  x ≔ 4x  // x  } [.5] {  // 0  x ≔ 0  // x  }
// x
```

From *wp* Computation to Strategies

```
// 2 · max{y, z}
//  max { [y ≥ z] · 2y , [y ≤ z] · 2z }
if ( y ≥  z) -> { // 2y x ≔ y // 2x }
[] ( y ≤  z) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

From *wp* Computation to Strategies

same as before!

```
// 2 · max{y, z}
//  max { [y ≥ z] · 2y , [y ≤ z] · 2z }
if ( y ≥  z) -> { // 2y x ≔ y // 2x }
[] ( y ≤  z) -> { // 2z x ≔ z // 2x }
end ;
// 2x
// 0.5 · 4x + 0.5 · 0
{ // 4x x ≔ 4x // x } [.5] { // 0 x ≔ 0 // x }
// x
```

Strategy Synthesis as a Program Transform

C	$trans(C, f)$
`x ≔` $Expr$	
C_1 `;` C_2	
`{`C_1`} [`p`] {`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	

Strategy Synthesis as a Program Transform

“objective function”

C	$trans(C, f)$
`x ≔` $Expr$	
C_1 `;` C_2	
`{`C_1`} [`p`] {`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	

Strategy Synthesis as a Program Transform

"objective function"

C	$trans(C, f)$
`x ≔` $Expr$	`x ≔` $Expr$
C_1 `;` C_2	
`{`C_1`}` `[`p`]` `{`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	

Strategy Synthesis as a Program Transform

"objective function"

C	$trans(C, f)$
`x ≔` $Expr$	`x ≔` $Expr$
C_1 `;` C_2	$trans\big(C_1, wp[\![C_2]\!](f)\big)$ `;` $trans(C_2, f)$
`{`C_1`} [`p`] {`C_2`}`	
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	

Strategy Synthesis as a Program Transform

"objective function"

C	$trans(C, f)$
`x ≔` $Expr$	`x ≔` $Expr$
C_1 `;` C_2	$trans\left(C_1, wp[\![C_2]\!](f)\right)$ `;` $trans(C_2, f)$
`{`C_1`}` `[`p`]` `{`C_2`}`	`{`$trans(C_1, f)$`}` `[`p`]` `{`$trans(C_2, f)$`}`
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	

Strategy Synthesis as a Program Transform

"objective function"

C	$trans(C, f)$
`x ≔` $Expr$	`x ≔` $Expr$
C_1 `;` C_2	$trans\big(C_1, wp[\![C_2]\!](f)\big)$ `;` $trans(C_2, f)$
`{`C_1`} [`p`] {`C_2`}`	`{`$trans(C_1, f)$`} [`p`] {`$trans(C_2, f)$`}`
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	`if` $\varphi_1 \wedge \big(\varphi_2 \implies wp[\![C_1]\!](f) \geq wp[\![C_2]\!](f)\big)$ `->` $trans(C_1, f)$ `[]` $\varphi_2 \wedge \big(\varphi_1 \implies wp[\![C_1]\!](f) \leq wp[\![C_2]\!](f)\big)$ `->` $trans(C_2, f)$ `end`

Soundness of Loop-free Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C}}\ \underbrace{\text{of } trans(C, f)} \quad \Longrightarrow \quad \underbrace{wp[\![\tilde{C}]\!](f)}\ =\ \underbrace{wp[\![C]\!](f)}\ .$$

Soundness of Loop-free Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)} \ .$$

Resolving the remaining nondeterminism in $\mathit{trans}(C, f)$ *arbitrarily* … *yields maximum expected value of* f *after termination.*

Soundness of Loop-free Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)} .$$

Resolving the remaining nondeterminism in $\mathit{trans}(C, f)$ *arbitrarily* … *yields maximum expected value of* f *after termination.*

$\mathit{trans}(C, f)$

```
if (y ≥ z) -> { // 2y x ≔ y // 2x }
[] (y ≤ z) -> { // 2z x ≔ z // 2x } end ; …
```

Soundness of Loop-free Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)} .$$

Resolving the remaining nondeterminism in $\mathit{trans}(C, f)$ *arbitrarily* … *yields maximum expected value of* f *after termination.*

$\tilde{C}$

```
if (y ≥ z) -> { // 2y x ≔ y // 2x }
[] (y < z) -> { // 2z x ≔ z // 2x } end ; …
```

Soundness of Loop-free Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \textit{trans}(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)} \;.$$

Resolving the remaining nondeterminism in $\textit{trans}(C, f)$ *arbitrarily* … *yields maximum expected value of* f *after termination.*

$\tilde{C}$

```
if (y > z) -> { // 2y x ≔ y // 2x }
[] (y ≤ z) -> { // 2z x ≔ z // 2x } end ; …
```

Soundness of Loop-free Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)}\ .$$

Resolving the remaining nondeterminism in $\mathit{trans}(C, f)$ arbitrarily … *yields maximum expected value of f after termination.*

$\tilde{C}$

```
if (y > z) -> { // 2y x ≔ y // 2x }
[] (y ≤ z) -> { // 2z x ≔ z // 2x } end ; …
```

Moreover, $\mathit{trans}(C, f)$ is effectively constructible*.

*If we fix a suitable syntax to represent expectations, e.g. [Batz et al. POPL '21].

Probabilistic Loop Invariants

$$\frac{\{\varphi \wedge I\} \quad C \quad \{I\}}{\{I\} \quad \texttt{while}\ \varphi\ \texttt{->}\ C\ \texttt{end} \quad \{\overline{\varphi} \wedge I\}}$$ (classic partial correctness)

Probabilistic Loop Invariants

$$\frac{\{\varphi \wedge I\} \quad C \quad \{I\} \qquad \wedge \qquad \textit{termination}}{\{I\} \quad \texttt{while}\ \varphi\ \texttt{->}\ C\ \texttt{end} \quad \{\overline{\varphi} \wedge I\}}$$ (classic total correctness)

Probabilistic Loop Invariants

$$\frac{\{\varphi \wedge I\} \quad C \quad \{I\} \qquad \wedge \qquad \textit{termination}}{\{I\} \quad \texttt{while } \varphi \texttt{ -> } C \texttt{ end} \quad \{\overline{\varphi} \wedge I\}}$$ (classic total correctness)

rewrite in terms of wp

$$\frac{\varphi \wedge I \implies wp[\![C]\!](I) \qquad \wedge \qquad \textit{termination}}{I \implies wp\left[\!\left[\texttt{while } \varphi \texttt{ -> } C \texttt{ end}\right]\!\right]\left(\overline{\varphi} \wedge I\right)}$$

Probabilistic Loop Invariants

$$\frac{\{\varphi \land I\} \quad C \quad \{I\} \qquad \land \qquad \textit{termination}}{\{I\} \quad \texttt{while } \varphi \texttt{ -> } C \texttt{ end} \quad \{\overline{\varphi} \land I\}}$$

(classic total correctness)

rewrite in terms of wp

$$\frac{\varphi \land I \implies wp[\![C]\!](I) \qquad \land \qquad \textit{termination}}{I \implies wp\left[\!\left[\texttt{while } \varphi \texttt{ -> } C \texttt{ end}\right]\!\right]\left(\overline{\varphi} \land I\right)}$$

boolean → probabilistic

$$\frac{[\varphi] \cdot I \leq wp[\![C]\!](I) \qquad \land \qquad \textit{side conditions}}{I \leq wp\left[\!\left[\texttt{while } \varphi \texttt{ -> } C \texttt{ end}\right]\!\right]\left([\overline{\varphi}] \cdot I\right)}$$

(lower bound on wp) [McIver & Morgan '05]

Program Transformation with Loops

C	$trans(C, f)$
`x` $\coloneqq$ $Expr$	`x` $\coloneqq$ $Expr$
C_1 `;` C_2	$trans\big(C_1, wp[\![C_2]\!](f)\big)$ `;` $trans(C_2, f)$
$\{C_1\}$ $[p]$ $\{C_2\}$	$\{trans(C_1, f)\}$ $[p]$ $\{trans(C_2, f)\}$
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	`if` $\varphi_1 \wedge \big(\varphi_2 \implies wp[\![C_1]\!](f) \leq wp[\![C_2]\!](f)\big)$ `->` $trans(C_1, f)$ `[]` $\varphi_2 \wedge \big(\varphi_1 \implies wp[\![C_1]\!](f) \geq wp[\![C_2]\!](f)\big)$ `->` $trans(C_2, f)$ `end`
`while` φ `->` C' `@`I `end`	

externally provided invariant annotation

Program Transformation with Loops

C	$trans(C, f)$
`x := ` $Expr$	`x := ` $Expr$
C_1 `;` C_2	$trans\big(C_1, wp[\![C_2]\!](f)\big)$ `;` $trans(C_2, f)$
`{`C_1`} [`p`] {`C_2`}`	`{`$trans(C_1, f)$`} [`p`] {`$trans(C_2, f)$`}`
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	`if` $\varphi_1 \wedge \big(\varphi_2 \implies wp[\![C_1]\!](f) \leq wp[\![C_2]\!](f)\big)$ `->` $trans(C_1, f)$ `[]` $\varphi_2 \wedge \big(\varphi_1 \implies wp[\![C_1]\!](f) \geq wp[\![C_2]\!](f)\big)$ `->` $trans(C_2, f)$ `end`
`while` φ `->` C' `@`I `end`	`while` φ `->` $trans(C', I)$ `end`

externally provided invariant annotation

Program Transformation with Loops

C	$trans(C, f)$
`x := ` $Expr$	`x := ` $Expr$
C_1 `;` C_2	$trans\big(C_1, wp[\![C_2]\!](f)\big)$ `;` $trans(C_2, f)$
$\{C_1\}\ [p]\ \{C_2\}$	$\{trans(C_1, f)\}\ [p]\ \{trans(C_2, f)\}$
`if` φ_1 `->` C_1 `[]` φ_2 `->` C_2 `end`	`if` $\varphi_1 \wedge \big(\varphi_2 \implies wp[\![C_1]\!](f) \leq wp[\![C_2]\!](f)\big)$ `->` $trans(C_1, f)$ `[]` $\varphi_2 \wedge \big(\varphi_1 \implies wp[\![C_1]\!](f) \geq wp[\![C_2]\!](f)\big)$ `->` $trans(C_2, f)$ `end`
`while` φ `->` C' `@` I `end`	`while` φ `->` $trans(C', I)$ `end`

externally provided
invariant annotation

& generate VCs:

$$[\varphi] \cdot I \leq wp[\![C']\!](I)$$
$$\wedge\ [\overline{\varphi}] \cdot I \leq f$$
$\wedge$ *side conditions*

Soundness of Transformation with Loops

Main Result

Let $C =$ `while` φ `->` C' `@`I `end`.

If all VCs generated during construction of $trans(C, f)$ are satisfied, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } trans(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) \geq I} .$$

Soundness of Transformation with Loops

Main Result

Let $C =$ `while` φ `->` C' `@`I `end`.

If all VCs generated during construction of $\mathit{trans}(C, f)$ are satisfied, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)} \implies \underbrace{wp[\![\tilde{C}]\!](f) \geq I} \,.$$

Resolving the remaining nondeterminism in $\mathit{trans}(C, f)$ *arbitrarily* … *yields at least the expected value "promised" by the invariant.*

Soundness of Transformation with Loops

Main Result

Let $C =$ `while` φ `->` C' `@` I `end`.

If all VCs generated during construction of $\mathit{trans}(C, f)$ are satisfied, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)} \quad \Longrightarrow \quad \underbrace{\mathit{wp}[\![\tilde{C}]\!](f) \;\geq\; I} \,.$$

Resolving the remaining nondeterminism in $\mathit{trans}(C, f)$ *arbitrarily* … *yields at least the expected value "promised" by the invariant.*

If a suitable I is given, then $\mathit{trans}(C, f)$ is effectively constructible.

Summary

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N ∧ ¬tails]
```

Summary

☑ We derive strategies for optimizing expected values after termination

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N-x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N-x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

// $[x \geq N \;\wedge\; \neg tails]$

Summary

- ☑ We derive strategies for optimizing expected values after termination
- ☑ Loop-free programs: fully mechanizable

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N-x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N-x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

// $[x \geq N \ \wedge \ \neg tails]$

Summary

☑ We derive strategies for optimizing expected values after termination

☑ Loop-free programs: fully mechanizable

☑ Loopy programs: strategy synthesis reduces to invariant synthesis

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

// $[x \geq N \ \wedge \ \neg tails]$

Summary

- ☑ We derive strategies for optimizing expected values after termination
- ☑ Loop-free programs: fully mechanizable
- ☑ Loopy programs: strategy synthesis reduces to invariant synthesis
- ☑ In paper: connection to MDP theory, minimization, upper bounds

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N  ∧  ¬tails]
```

Summary

- ☑ We derive strategies for optimizing expected values after termination
- ☑ Loop-free programs: fully mechanizable
- ☑ Loopy programs: strategy synthesis reduces to invariant synthesis
- ☑ In paper: connection to MDP theory, minimization, upper bounds
- ☐ Stochastic 2-player games

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

// $[x \geq N \ \wedge \ \neg tails]$

Summary

- ☑ We derive strategies for optimizing expected values after termination
- ☑ Loop-free programs: fully mechanizable
- ☑ Loopy programs: strategy synthesis reduces to invariant synthesis
- ☑ In paper: connection to MDP theory, minimization, upper bounds
- ☐ Stochastic 2-player games

```
tails ≔ false;
while (x<N ∧ !tails) ->
    if (p≤q² ∨ (q²<p<q ∧ N−x is odd))
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (q≤p ∨ p=q² ∨ (q²<p<q ∧ N−x≥2))
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
// [x ≥ N ∧ ¬tails]
```

Thank you! ✉ *tobias.winkler@cs.rwth-aachen.de*