Programmatic Strategy Synthesis

Resolving Nondeterminism in Probabilistic Programs

Tobias Winkler
based on a paper with Kevin Batz, Tom Jannik Biskup, and Joost-Pieter Katoen

SynCoP 2024 — 06.04.2024

Parametric ~~Programmatic~~ Strategy Synthesis

Resolving Nondeterminism in Probabilistic Programs

Tobias Winkler
based on a paper with Kevin Batz, Tom Jannik Biskup, and Joost-Pieter Katoen

SynCoP 2024 — 06.04.2024

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 €
- Objective: 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 €
- Objective: maximize winning probability

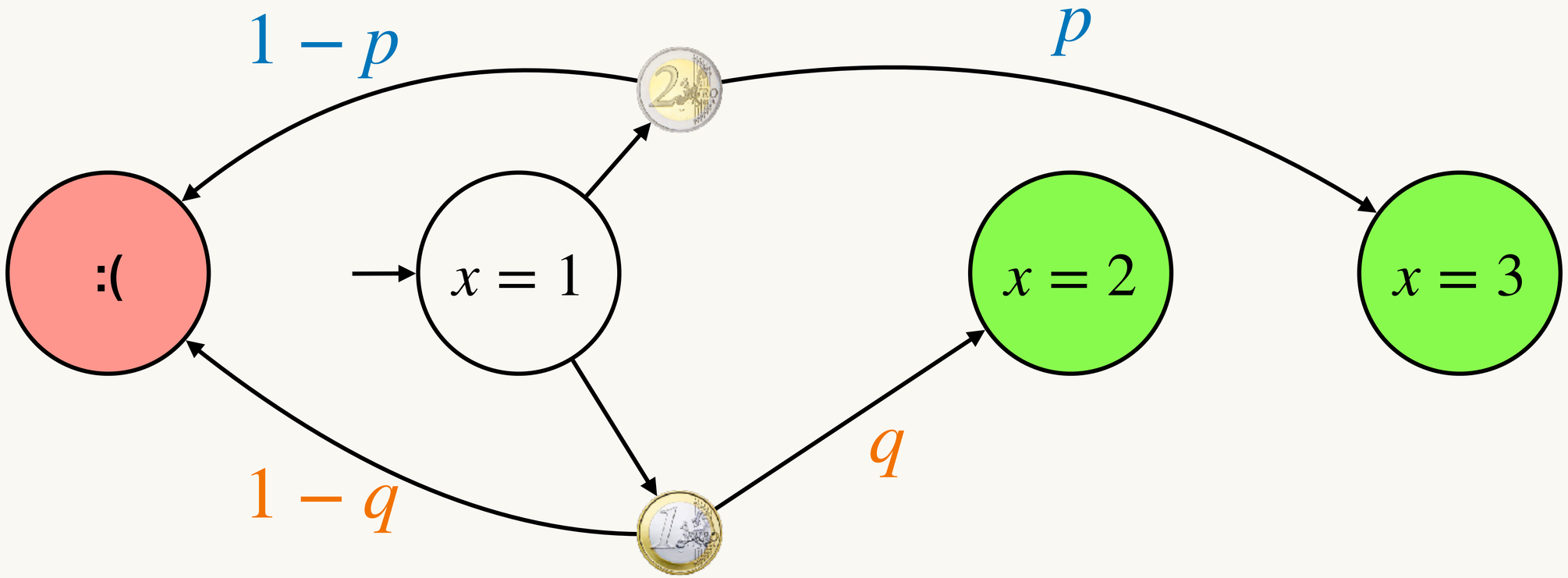
```
tails ≔ false ;
while (x<N ∧ !tails) ->
    if (true)
    ->   {x ≔ x+1} [q] {tails ≔ true}
    [] (true)
    ->   {x ≔ x+2} [p] {tails ≔ true}
    end
end
```

A Gamble with Two Coins

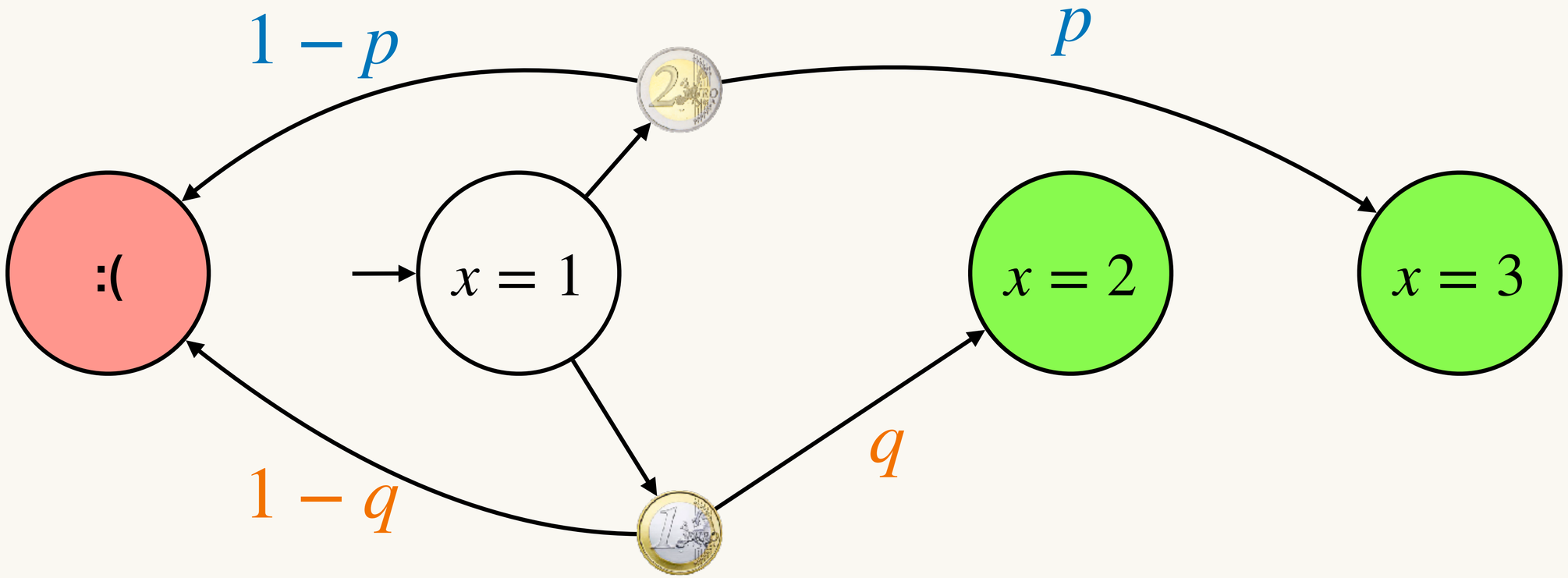
- Two coins with bias q (1 €) and p (2 €)
- Repeatedly select a coin and flip it
  ➡ Get € (heads) or game over (tails)
- Start with x €
- Win once we have at least N €
- Objective: maximize winning probability

```
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]
```

The Gamble with $N = 2$



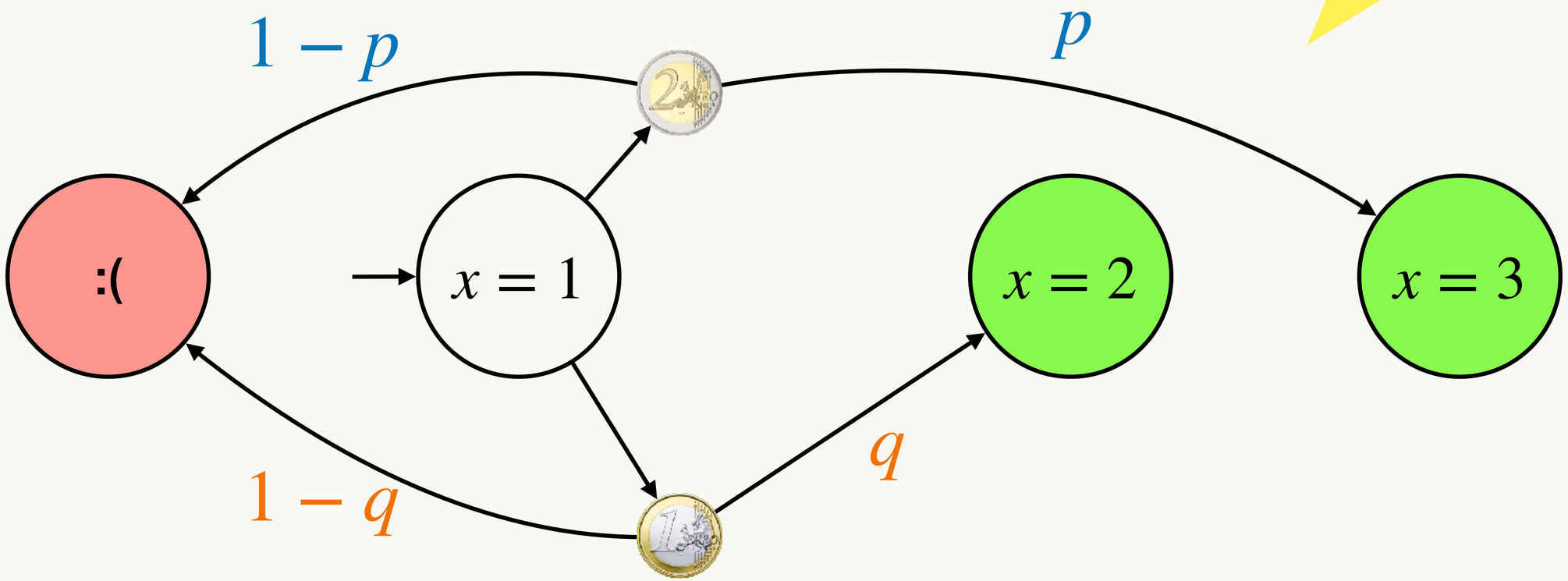
The Gamble with $N = 2$



- Best strategy: If $p > q$ flip  , if $p < q$ flip  , if $p = q$: don't care

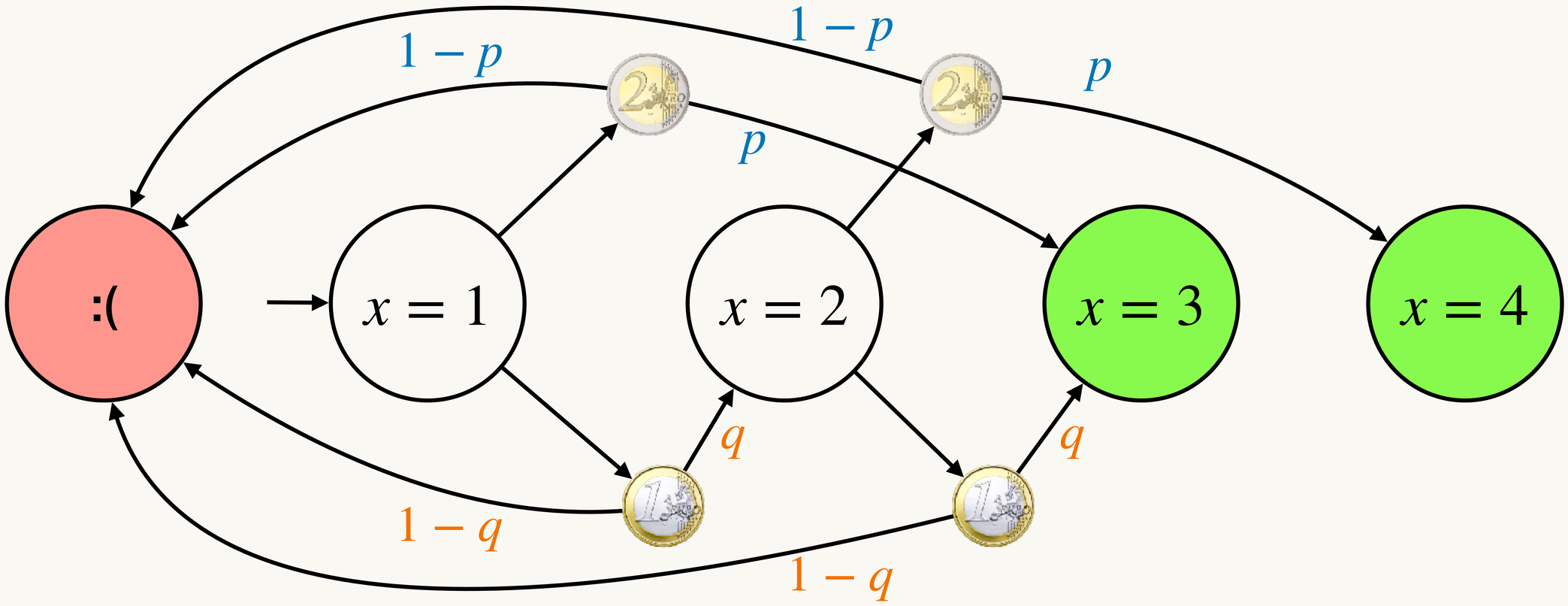
The Gamble with $N = 2$

Markov Decision Process (MDP)

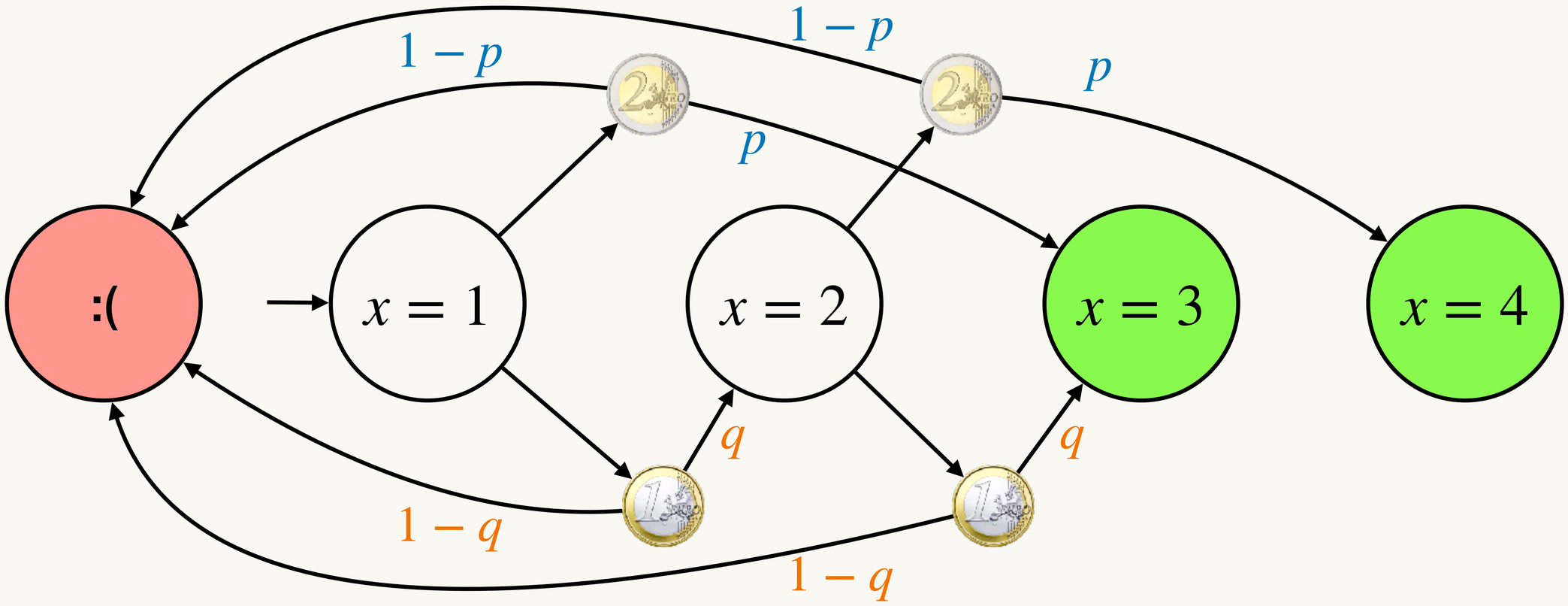


- Best strategy: If $p > q$ flip  , if $p < q$ flip  , if $p = q$: don't care

The Gamble with $N = 3$

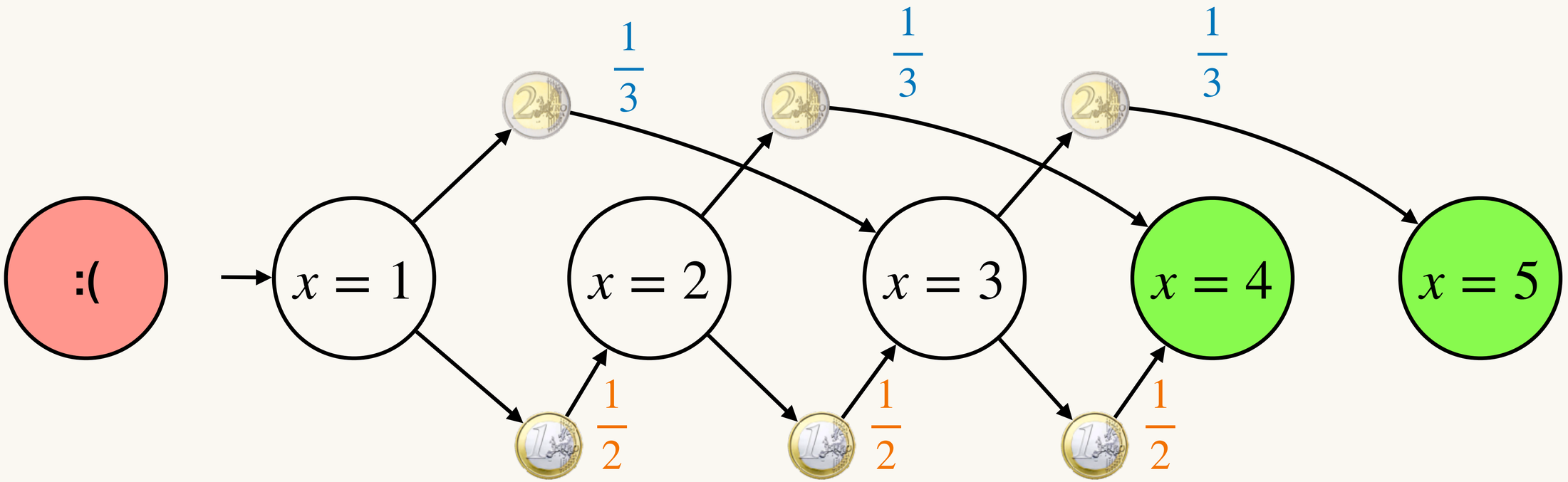


The Gamble with $N = 3$

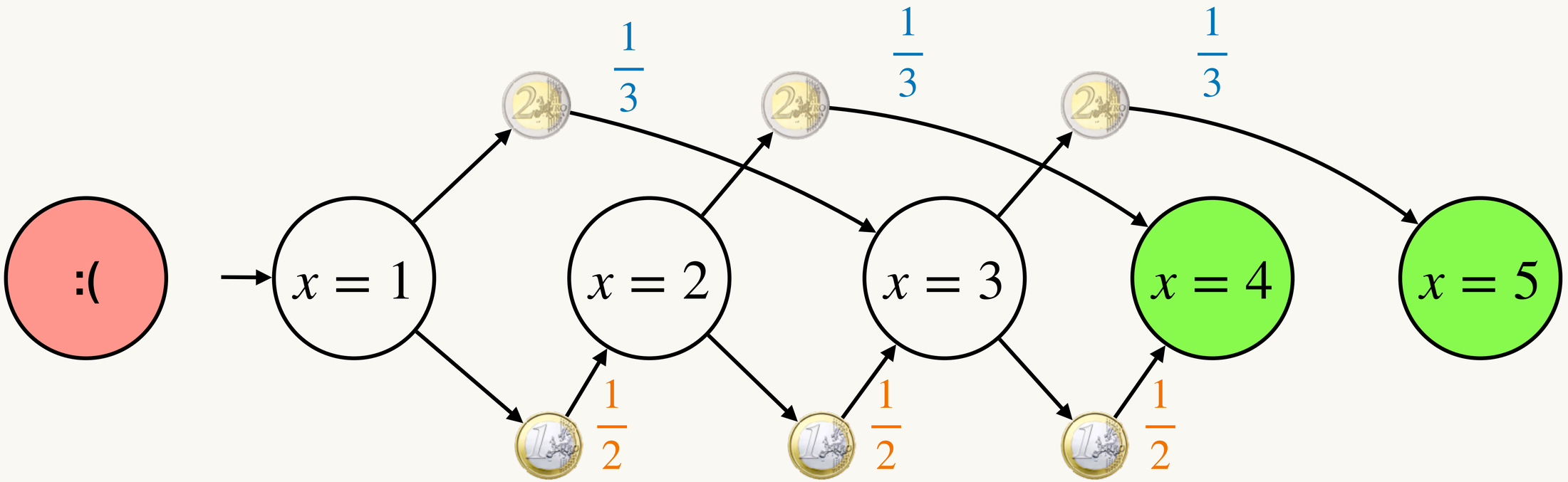


- Best strategy: If $p > q^2$ flip  , if $p < q^2$ flip  , if $p = q^2$: don't care

The Gamble with $N = 4$, $p = 1/3$, $q = 1/2$



The Gamble with $N = 4$, $p = 1/3$, $q = 1/2$



- Best strategy: First  , then  (or vice versa)

A Gamble with Two Coins

Optimal Solution


```
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 ≥ N ∧ ¬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

  = parametric strategy in N, p, q

```
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

  = parametric strategy in N, p, q
- Strategies are *permissive*

```
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

  = parametric strategy in N, p, q
- Strategies are *permissive*
- Semi-automatic approach

  ➡ loops need `@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

(without probabilities)

[Dijkstra ’75]

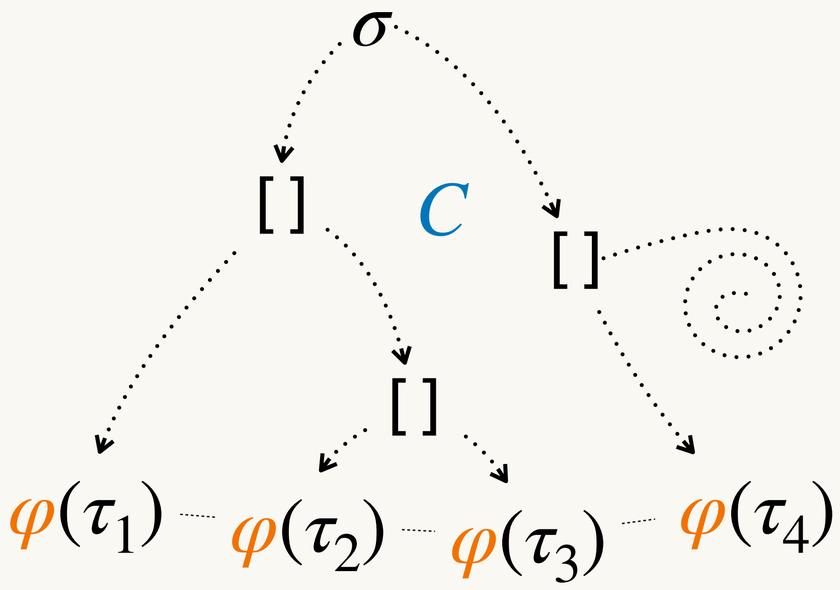


Illustration based on [Batz et al. OOPSLA ’22]

Weakest Preconditions

(without probabilities)

[Dijkstra ’75]

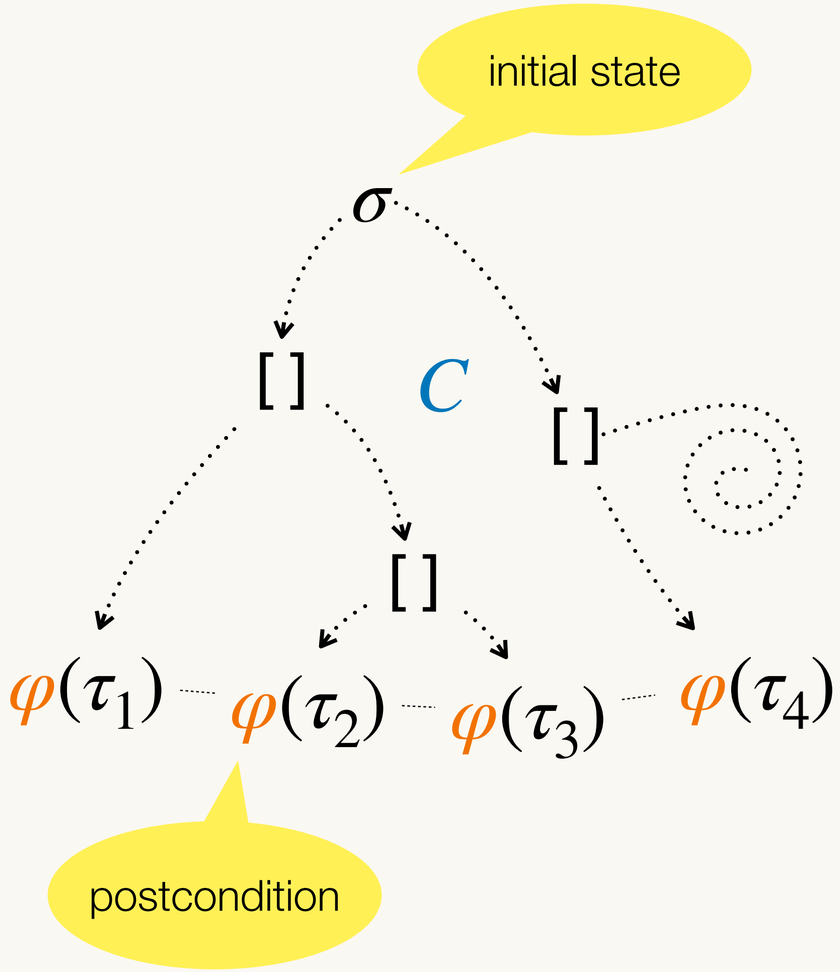


Illustration based on [Batz et al. OOPSLA ’22]

Weakest Preconditions

(without probabilities)

[Dijkstra ’75]

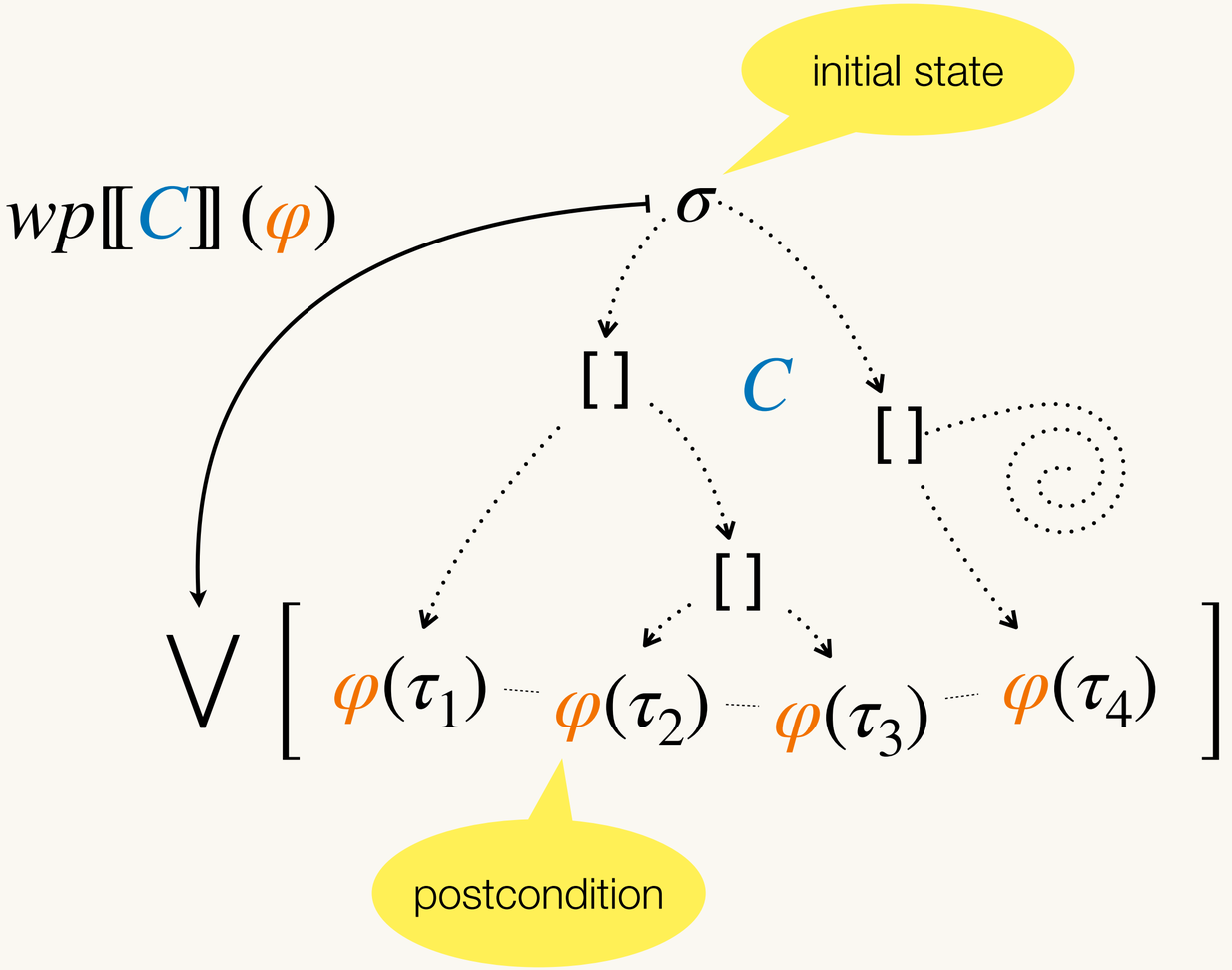
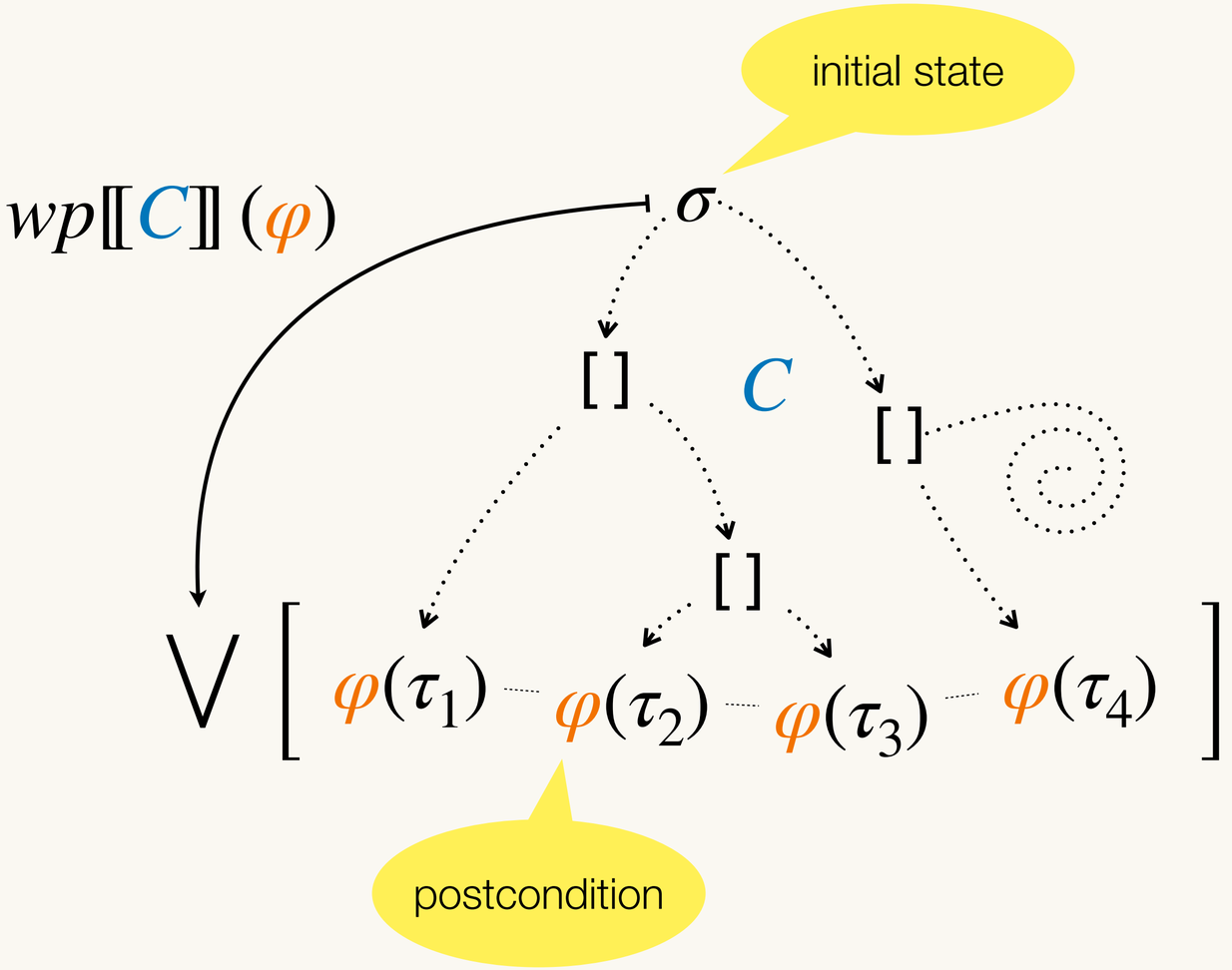


Illustration based on [Batz et al. OOPSLA ’22]

Weakest Preconditions

(without probabilities)

[Dijkstra ’75]




```
// y ≥ 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]

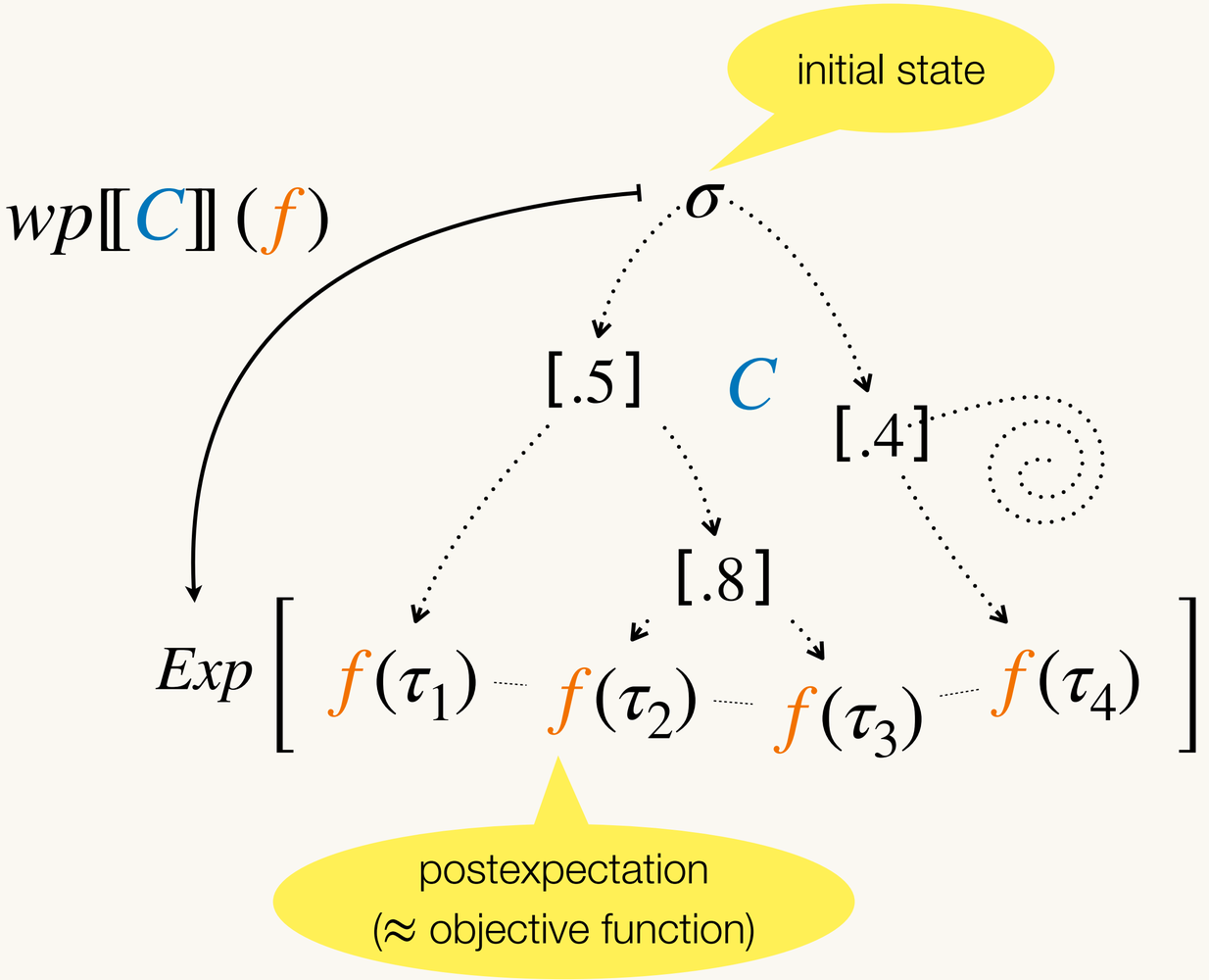
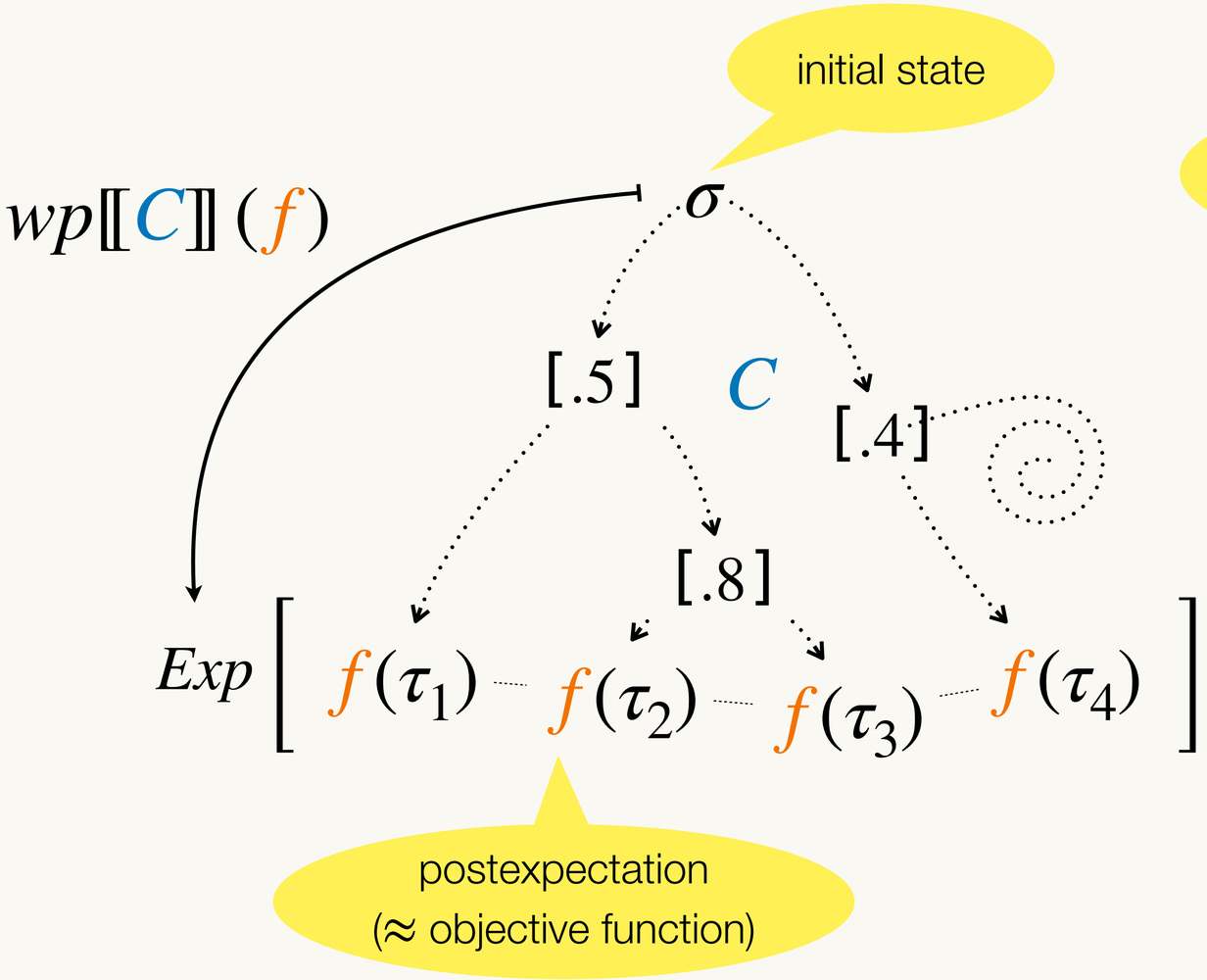


Illustration based on [Batz et al. OOPSLA ’22]

Weakest Pre-*expectations*

For Probabilistic Programs

[Kozen ‘83]



expected value of x
after termination

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

```
{x ≔ 4x} [.5] {x ≔ 0}
```

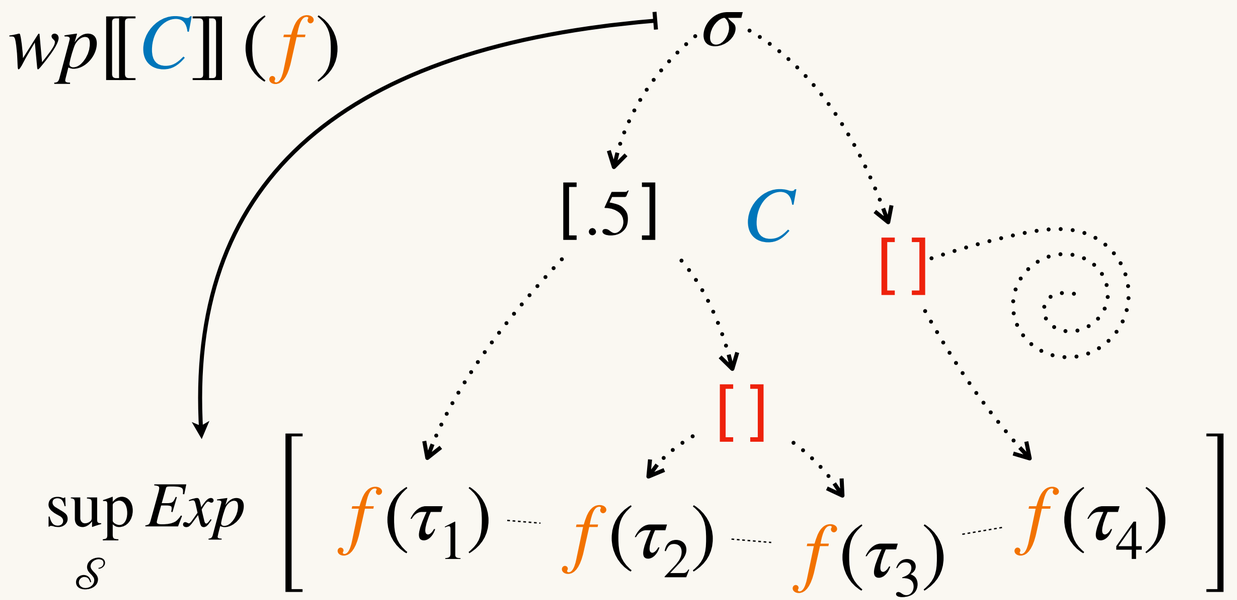
$// \; x$

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

Weakest Pre-*expectations*

For Probabilistic Programs *with Nondeterminism*

[McIver & Morgan ‘05]



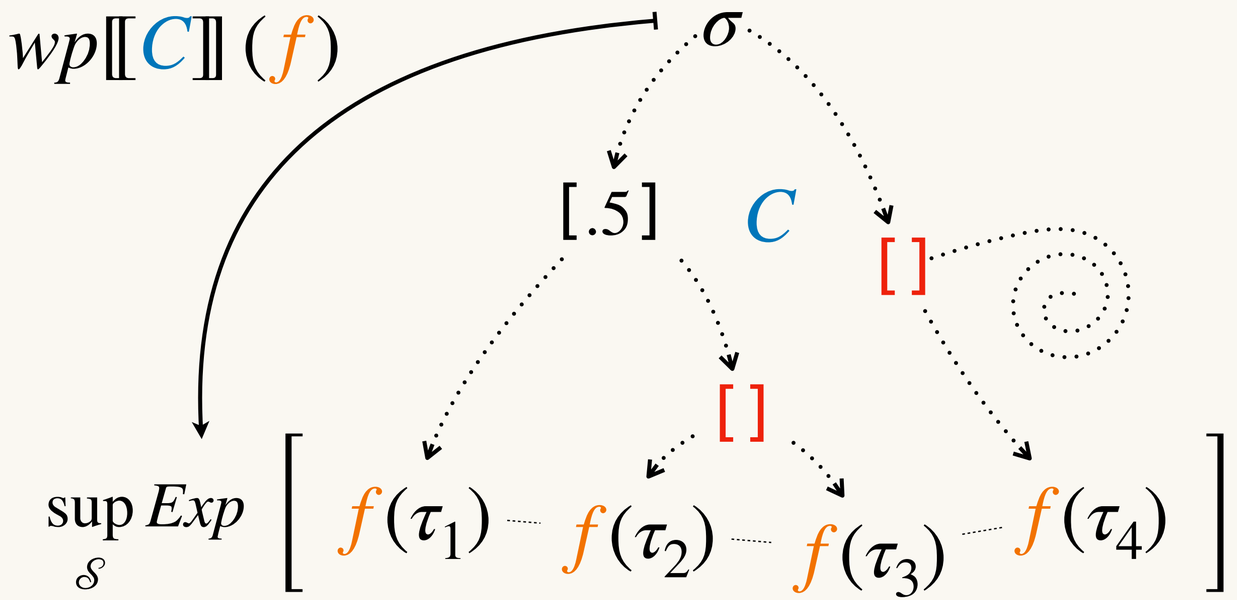
strategies
aka schedulers, policies

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

Weakest Pre-*expectations*

For Probabilistic Programs *with Nondeterminism*

[McIver & Morgan ‘05]



strategies
aka schedulers, policies

greatest possible expected value of x after termination

```
// [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]

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 · 4x + 0.5 · 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

postexpectation
(≈ 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

postexpectation
(≈ 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

postexpectation
(≈ 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

postexpectation
(≈ 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	

Strategy Synthesis as a Program Transform

postexpectation
(≈ 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`

Strategy Synthesis as a Program Transform

postexpectation
(≈ objective functior

tldr;
for loop-free program C we can **automatically** construct the "strategy program" $trans(C, f)$

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 Transformation

If C is loop-free, then

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

Soundness of Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)}_{\text{Resolving the remaining nondeterminism in } \mathit{trans}(C, f) \text{ arbitrarily}} \quad \Longrightarrow \quad \underbrace{\mathit{wp}[\![\tilde{C}]\!](f) = \mathit{wp}[\![C]\!](f)}_{\text{yields maximum expected value of } f \text{ after termination.}} .$$

Soundness of Transformation

If C is loop-free, then

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

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 Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)}_{\text{Resolving the remaining nondeterminism in } \mathit{trans}(C, f) \text{ arbitrarily}} \quad \Longrightarrow \quad \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)}_{\text{yields maximum expected value of } f \text{ after termination.}} .$$

$\tilde{C}$

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

Soundness of Transformation

If C is loop-free, then

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)}_{\text{Resolving the remaining nondeterminism in } \mathit{trans}(C, f) \text{ arbitrarily}} \quad \Longrightarrow \quad \underbrace{wp[\![\tilde{C}]\!](f) = wp[\![C]\!](f)}_{\text{yields maximum expected value of } f \text{ after termination.}} .$$

…

$\tilde{C}$

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

What about loops?

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

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

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

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

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

boolean → probabilistic

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

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

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`	

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

externally provided invariant annotation

& generate VCs:

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

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

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

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

$$\underbrace{\forall \text{ determinizations } \tilde{C} \text{ of } \mathit{trans}(C, f)}_{\text{Resolving the remaining nondeterminism in } \mathit{trans}(C, f) \text{ arbitrarily } \ldots} \implies \underbrace{\mathit{wp}[\![\tilde{C}]\!](f) \geq I}_{\text{yields at least the expected value “promised” by the invariant.}} .$$

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

☑ Programmatic 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 ≥ N ∧ ¬tails]
```

Summary

- ☑ Programmatic 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 ≥ N ∧ ¬tails]
```

Summary

- ☑ Programmatic 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 ≥ N ∧ ¬tails]
```

Summary

- ☑ Programmatic 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]
```

Future Work


```
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]
```

Future Work

- Generalization to stochastic 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]
```

Future Work

- Generalization to stochastic games
- Automatic 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 ≥ N ∧ ¬tails]
```

Future Work

- Generalization to stochastic games
- Automatic invariant synthesis
- Most permissive strategies?


```
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]
```

Future Work

- Generalization to stochastic games
- Automatic invariant synthesis
- Most permissive strategies?

Link to full paper

```
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]
```

Future Work

- Generalization to stochastic games
- Automatic invariant synthesis
- Most permissive strategies?

```
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]
```

Link to full paper ➡

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

Backup: Invariant for Running Example

```
while c < N && a = false ->
    if true ->
        { c := c+1 } [q] { c := true }
    [] true ->
        { c := c+2 } [p] { c := true }
    endif
endwhile
```

$$I_{Gamb} \;:=\; [a=0]\cdot\Big(1 \;-\; \Big([p\le q]\cdot q^{\lceil (N-c)/2\rceil} \;+\; \big[q\le p^2\big]\cdot p^{N-c} \;+\; \big[p^2<q<p\big]\cdot p^{[N-c>0][N-c\ \textit{is odd}]}\cdot q^{\lfloor (N-c)/2\rfloor}\Big)\Big) \;+\; [a=1]$$