Reasoning About Almost-Sure Termination

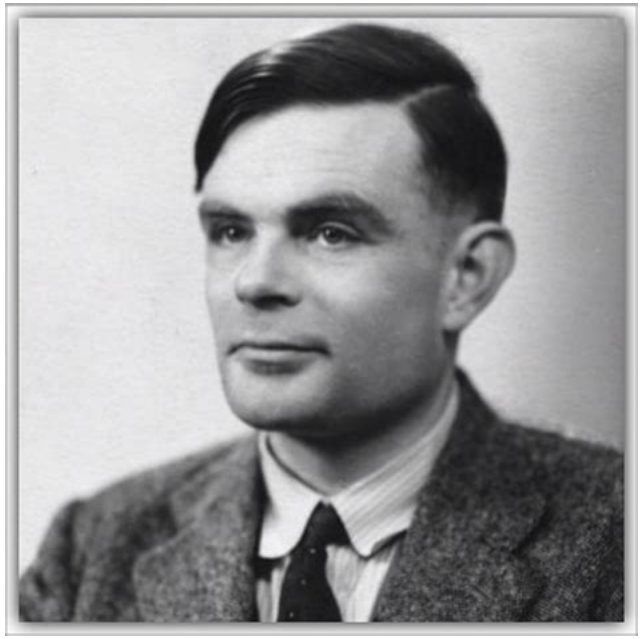
Rupak Majumdar

(Joint work with V.R. Sathiyanarayana)

Max Planck Institute for Software Systems

Termination, a.k.a. The Halting Problem

Given a program P, does P halt?



Algorithmically undecidable!

… But at the core of program analysis and verification
Roughly, termination ≈ reasoning about liveness

In order to assist the checker, the programmer should make assertions about the various states that the machine can reach. These assertions may be tabulated as in fig.2. Assertions are only made for the states when certain particular quantities are in control, corresponding to the ringed letters in the flow diagram. One column of the table is used for each such situation of the control. Other quantities are also needed to specify the condition of the machine completely: in our case it is sufficient to give r and s. The upper part of the table gives the various contents of the store lines in the various conditions of the machine, and restrictions on the quantities s, r (which we may call inductive variables). The lower part tells us which of the conditions will be the next to occur.

The checker has to verify that the columns corresponding to the initial condition and the stopped condition agree with the claims that are made for the routine as a whole. In this case the claim is that if we start with control in condition D and with n in line 29 we shall find a quantity in line 31 when the machine stops which is r (provided this is less than 2^{40}, but this condition has been ignored).

He has also to verify that each of the assertions in the lower half of the table is correct. In doing this the columns may be taken in any order and quite independently. Thus for column B the checker would argue, "From the flow diagram we see that after B the box $v^1 = u$ applies. From the upper part of the column for B we have $u = r$. Hence $v^1 = r$ i.e. the entry for v i.e. for line 31 in C should be r. The other entries are the same as in B".

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops. To the pure mathematician it is natural to give an ordinal number. In this problem the ordinal might be $(n - r)\,w^2 + (r - s)\,w + k$. A less highbrow form of the same thing would be to give the integer $2^{80}(n - r) + 2^{40}(r - s) + k$. Taking the latter case and the step from B to C there would be a decrease from $2^{80}(n - r) + 2^{40}(r - s) + 5$ to $2^{80}(n - v) + 2^{40}(r - s) + 4$. In the step from F to B there is a decrease from $2^{80}(n - r) + 2^{40}(r - s) + 1$ to $2^{80}(n - r + 1) + 2^{40}(r + 1 - s) + 5$.

In the course of checking that the process comes to an end the time involved may also be estimated by arranging that the decreasing quantity represents an upper bound to the time till the machine stops.

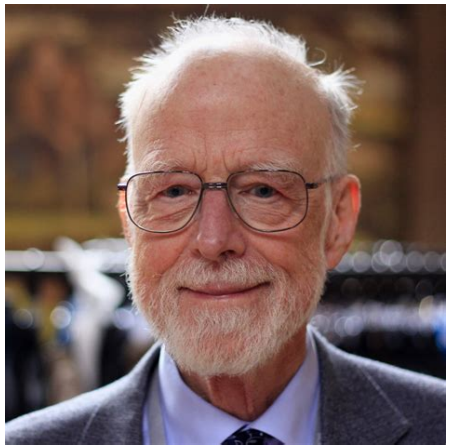
Alan Turing: 'Checking a large routine'.
Talk on 24 June 1949 at the
Inaugural conference of the EDSAC
computer at the Mathematical Laboratory,
Cambridge

Safety: Invariants and assertions

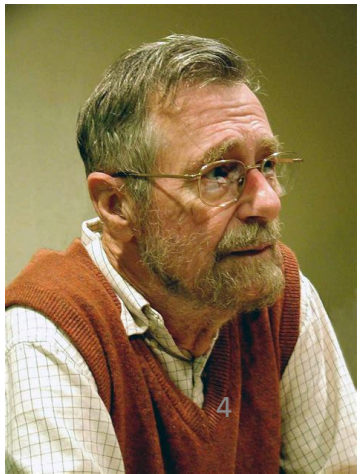
Termination: Ranking functions

How do we Prove Programs Correct?

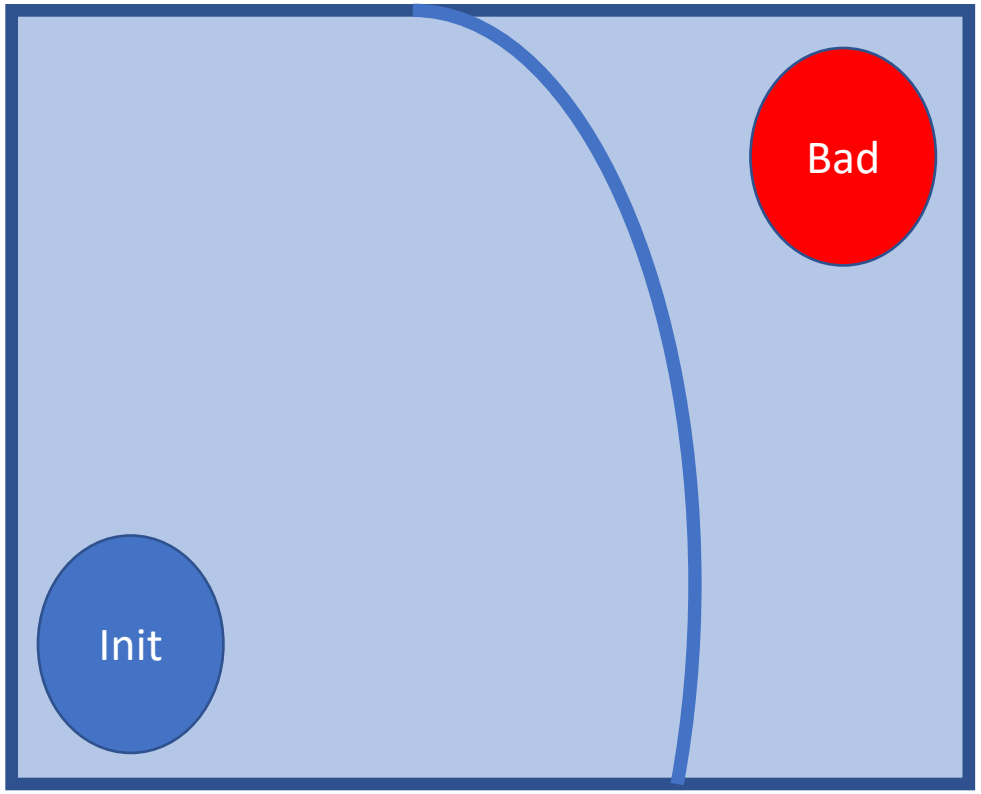
Program Logics: Reduce reasoning about programs to validity questions in logic



"In the old days, programmers would just twiddle with programs till they seemed to work,"
says Professor Emeritus of The Art of Computer Programming Donald Knuth.
"Floyd showed that there was a way to prove programs would work."
His approach of marrying math with computer science was "a revelation to the field," Knuth says.
One of the hottest topics in computer science at the time was the language of computer programming.
Says Knuth: "There were only four good papers on the topic -- all by Floyd."



Proof Rule for Safety: Inductive Invariants



To show Bad is unreachable:

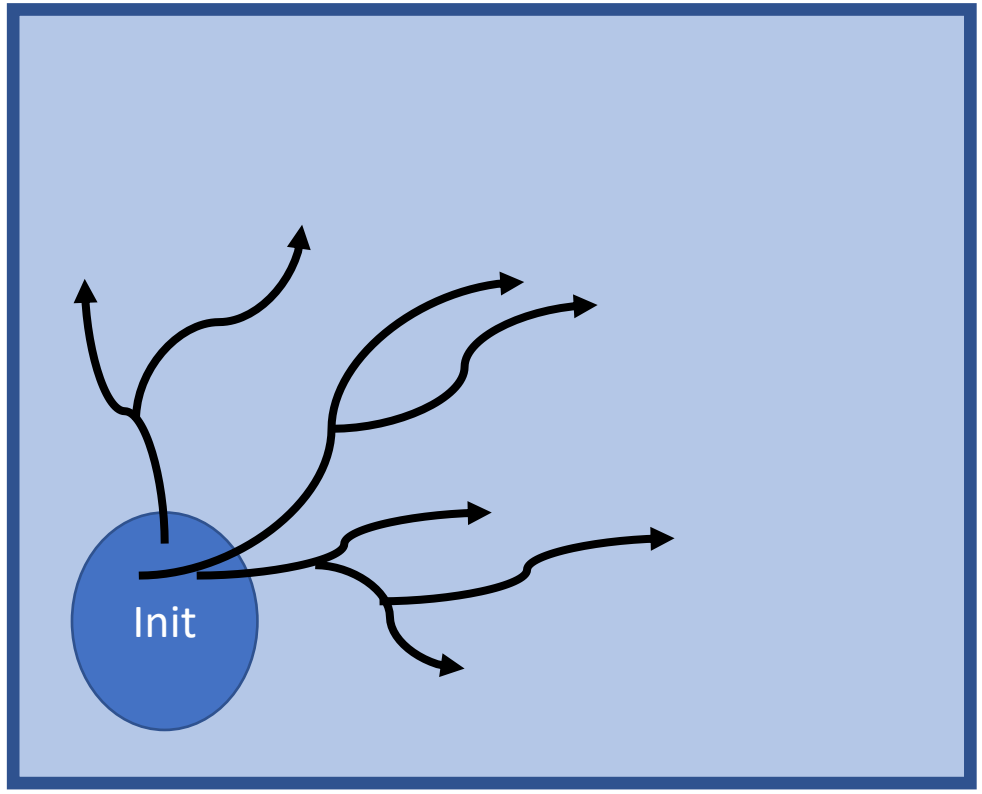
Find a certificate Inv such that:

1. Init ⊆ Inv

2. Post(Inv) ⊆ Inv
   [Inv closed under the transition relation]

3. Bad ∩ Inv = ∅

Proof Rule for Termination: Ranks



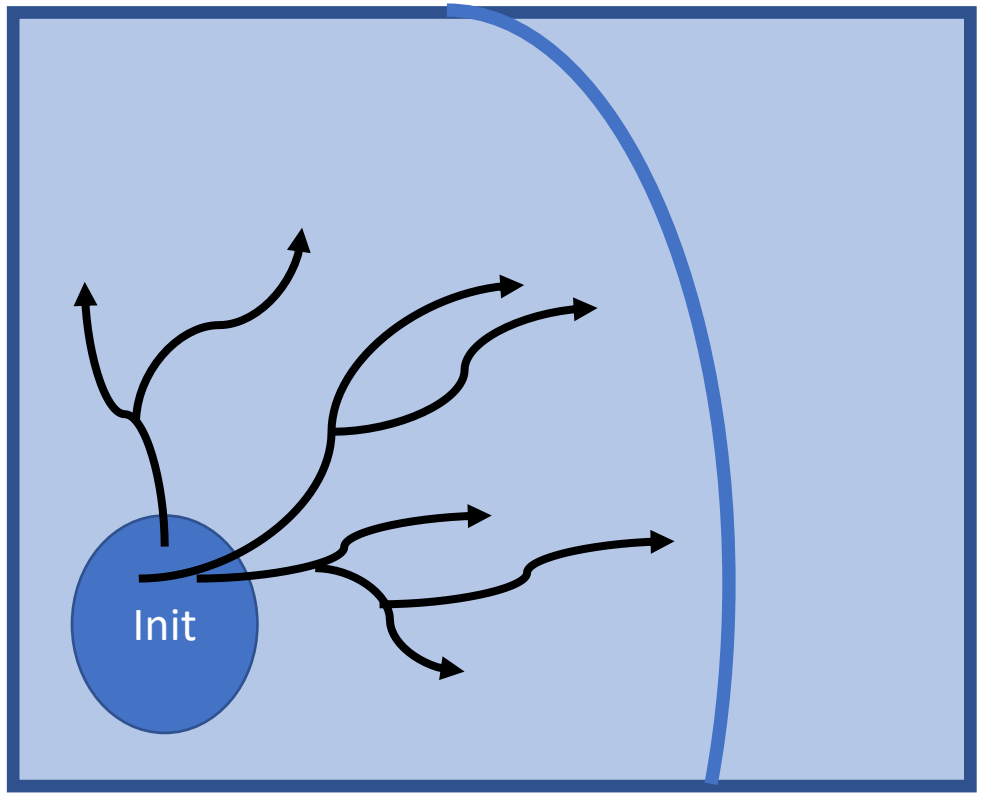
To show P is terminating:

Find a *rank* U mapping states to ordinals:

whenever s → t, we have

U(s) > U(t)

Proof Rule for Termination: Ranks

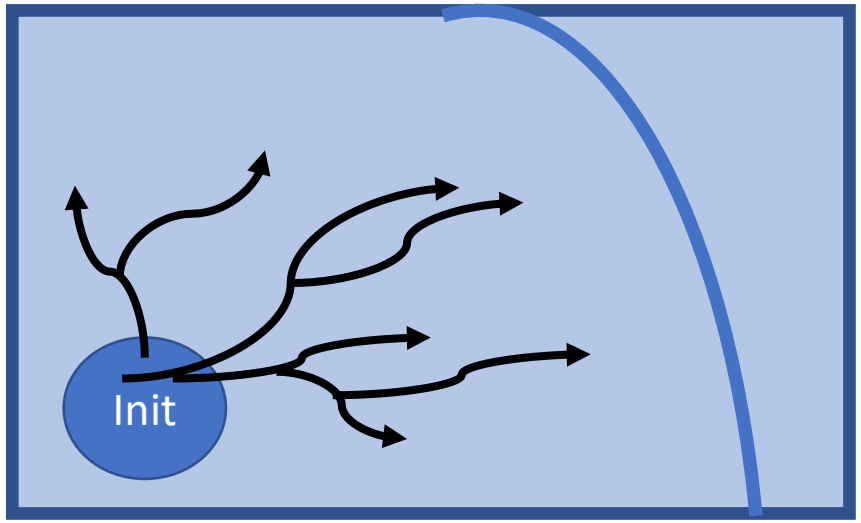


To show P is terminating:

Find an invariant Inv
and a *rank* U mapping
states to ordinals:

whenever *Inv(s)* and s → t,
we have:
U(s) > U(t)

Proof Rule for Termination: Ranks



To show P is terminating:
Find an invariant Inv
and a *rank* mapping states to ordinals:

whenever *Inv(s)* and s → t,
we have r(s) > r(t)

Example: Ranking functions

Find a mapping $U: \Sigma \to \mathbb{Z}^{\geq 0}$ that decreases in each execution step.

```
while (x > 0) {
    x -= 1
}
```

$$U(x) \triangleq x$$

Soundness and Completeness

- *Soundness:* Whenever our proof system says something about a program, this statement is true for the program

- *Completeness:* Whenever a program is safe or is terminating, the proof system is able to prove this

Completeness is relative to the underlying logic [Cook]

Soundness

- *Safety: Argue by induction that all reachable states are in Inv*
- *Termination: Argue by well-foundedness that there cannot be an infinite execution*

Completeness

- *Safety: Take Inv to be the set of reachable states*
- *Termination: Take rank to be the number of states reachable from s*

Cook: We can encode Inv and rank in the language of arithmetic

Fast Forward to Today….

Many groundbreaking results in formal methods and program verification, algorithmic methods, and tools….

Are there any open problems left?

Probabilistic Programs:
Algorithms that toss coins

Turing machines + Randomness

- Why?
- Randomization can often lead to efficient algorithms

  Polynomial Identity Testing: Best algorithms are randomized

- Randomization can overcome barriers

  Randomization allows symmetric solutions to dining philosophers

  Randomized asynchronous consensus avoids FLP impossibility

Language

- Imperative language
- Bounded nondeterministic choice
- Bounded probabilistic choice $c_1 \oplus_p c_2$
  - Toss a coin. With probability p, execute c_1 and with 1-p, execute c_2

Almost Sure Safety and Termination

Almost sure = “Happens with probability 1”

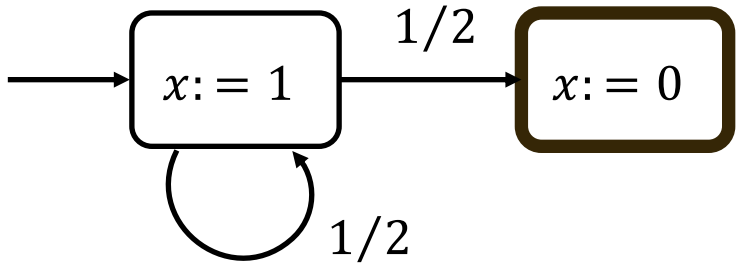
Almost sure safety = Safety

But:

Almost sure termination ≠ Termination

Example: Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```



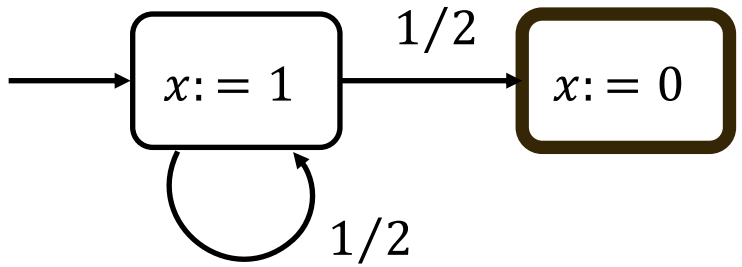
Almost-Sure Termination (AST)

- Natural probabilistic generalization of classical termination
- Requires probability of termination $= 1$

Example: Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

- Complexity of AST is Π_2^0-complete [Kaminski & Katoen 2015].
  - Harder than classical halting problem [Turing 1943]



What are proof rules for AST?

- Similar in spirit to proof rules for termination
- Find ***certificates*** of termination
  - Invariants, ranks
- Hard to find, but “easy” to check
- Easy = Validity question in logic

Example: Ranking functions

Find a mapping $U : \Sigma \to \mathbb{Z}^{\geq 0}$ that decreases in each execution step.

```
while (x > 0) {
    x -= 1 ⊕1/2 x -= 2;
}
```

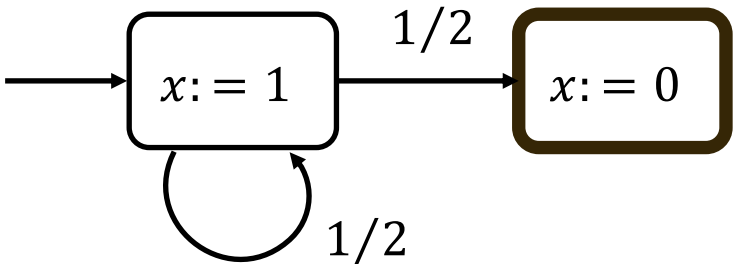
$$U(x) \triangleq x$$

What are proof rules for **AST**?

- Similar in spirit to proof rules for termination
- Find ***certificates*** of termination
  - Invariants, ranks
- Hard to find, but “easy” to check
- Easy = Validity question in logic

Example: Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```



Non-probabilistic ranking functions do not work

Main Theorem

A Sound and Complete proof rule for almost sure termination

This “closes” a line of research, since the early 80s, that successively found better and better *sound* proof rules

Outline

- A Program Logic (and why you can forget about it)
- The Rule
- Soundness
- Completeness

For non-probabilistic programs [Dijkstra 1975]

Program Logic: Weakest Precondition Calculus

Predicate over program states

This is what the calculus defines syntactically

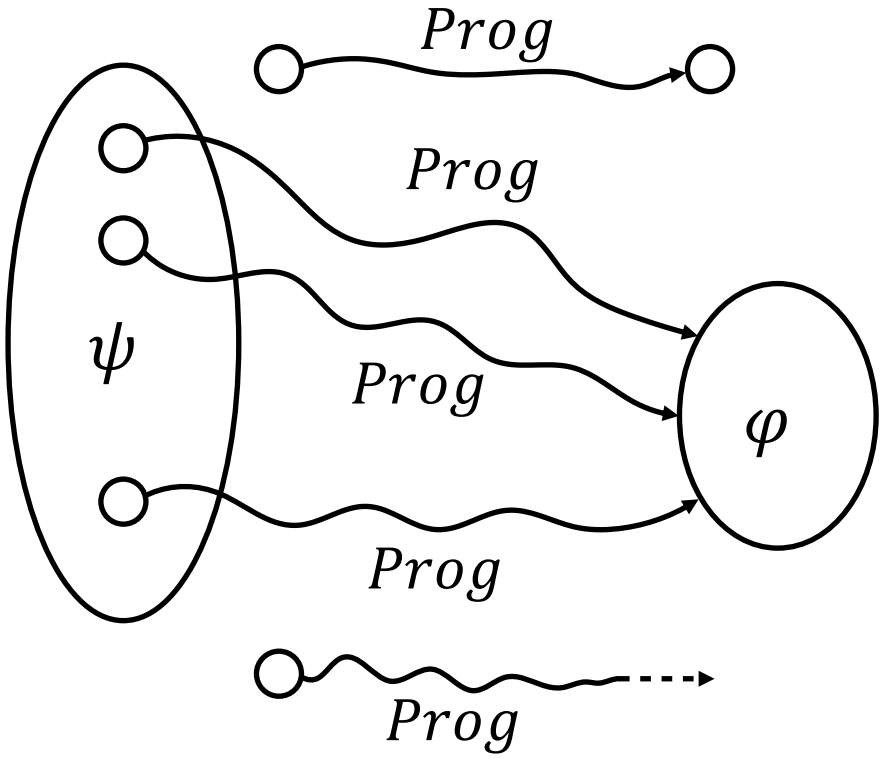
For a predicate φ and a program $Prog$, the function wp. $Prog$. φ outputs the *weakest condition* ψ such that the execution of $Prog$ from ψ *ends in* φ.

"Most general" predicate, i.e., largest set of states

$Prog$ terminates in φ

Program Logic: Weakest Precondition Calculus

For a predicate φ and a program $Prog$, the function $\mathrm{wp}.Prog.\varphi$ outputs the *weakest condition* ψ such that the execution of $Prog$ from ψ *ends in* φ.



Example

$Prog :=$

```
while (x != 0) {
    x -= 1;
}
```

$\mathrm{wp}.Prog.\top = x \geq 0$

For probabilistic programs [Kozen 1985]

Probabilities: Weakest Pre-Expectation Calculus

Mapping from program states to $\mathbb{R}^{\geq 0}$

Generalizes boolean predicates

For an ***expectation*** f and a program $Prog$, the calculus wp. $Prog$. f outputs the **expectation*** g such that g assigns to each state σ the expected value of f after the execution of $Prog$ from σ.

Because $Prog$ may not terminate with probability 1

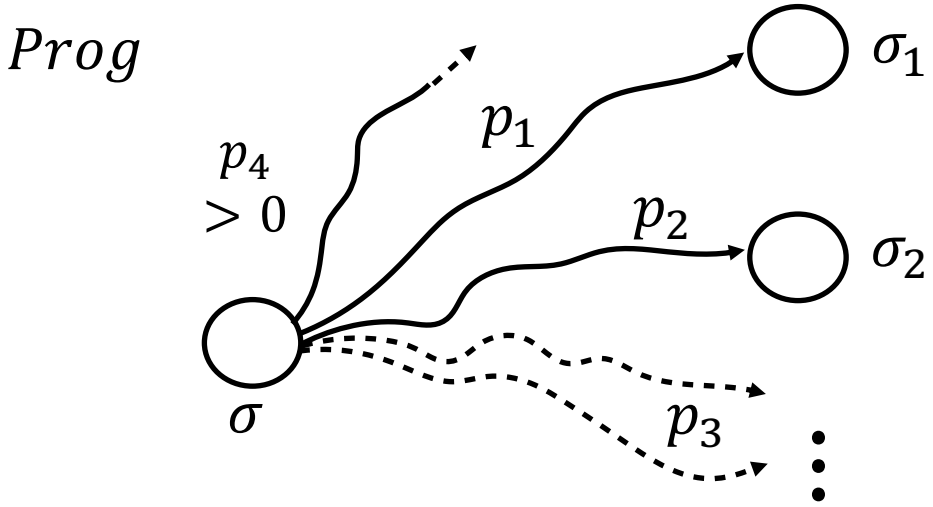
Gives a probability **sub-distribution** over the state space

*Only when there isn't nondeterminism!

For probabilistic programs [Kozen 1985]

Probabilities: Weakest Pre-Expectation Calculus

For an ***expectation*** f and a program $Prog$, the calculus wp.$Prog$.f outputs the **expectation*** g such that g assigns to each state σ the expected value of f after the execution of $Prog$ from σ.



$(\text{wp}.Prog.f)(\sigma)$
$= p_1 f(\sigma_1) + p_2 f(\sigma_2) + \cdots$
$p_1 + p_2 + p_3 < 1$ is possible!

Example

$Prog :=$

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

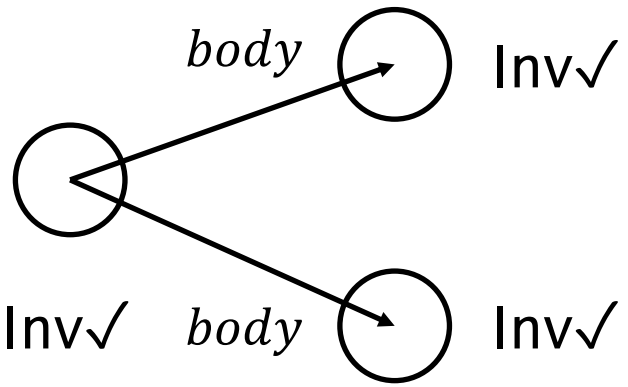
$\text{wp}.Prog.x = 0$

Loop Invariants in this Notation

For the loop $prog = while(\varphi)\{body;\}$, the predicate Inv is an **invariant** of $prog$ if its truth ***doesn't change*** after an execution of $body$.

In our notation: wp. $body$. [Inv] $\geq$ [$\varphi \wedge$ Inv]

The brackets [] around Inv makes a Boolean characteristic function into an expectation



Loop Invariants

For the loop
$prog = while(\varphi)\{body;\},$
the predicate Inv is an invariant of $prog$ if its truth ***doesn't change*** after an execution of $body$.

Example

$Prog :=$
```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

Inv: $= x = 0 \vee x = 1$

Goal: AST Proof Rule

Programs without loops always terminate

Our goal is to show a loop terminates almost surely from all states satisfying the loop invariant

i.e., for a loop

$$Prog = while(\varphi)\{body\}$$

with a loop invariant Inv, that

$$[\text{Inv}] \leq \text{wp}.Prog.1$$

Outline

- A Program Logic (and why you can forget about it)
- The Rule
- Soundness
- Completeness

Let's build the proof rule…

…using 3 simple examples of loops

Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

AST

Symmetric 1DRW

```
while (x != 0) {
    x++ ⊕1/2 x--;
}
```

AST

Asymmetric 1DRW

```
while (x != 0) {
    x++ ⊕1/3 x--;
}
```

Not AST

Why is the coin flipper **AST**?

Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

Each state is one loop iteration

away from termination

Why is the coin flipper **AST**?

[McIver & Morgan 2005]

Use an existing proof rule for this!

Look for a **bounded** integer-valued mapping U and a positive ϵ such that $body$ reduces U with probability $\geq \epsilon$

$$U(x) \triangleq \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases}$$

Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

Each state is one loop iteration away from termination

Generalize ranks: Ranks decrease with positive probability in each step

Formal Proof Rule

(***Variant rule*** of McIver and Morgan, 2005)

To prove that $[\mathsf{Inv}] \leq \mathsf{wp}.Prog.1$, find an integer valued expectation U and two positive values ϵ and H such that

- $\mathsf{Inv} \wedge \varphi \Longrightarrow H > U > 0$ and $\neg\varphi \Longrightarrow U = 0$
- $\epsilon[\mathsf{Inv} \wedge U = n] \leq \mathsf{wp}.body.[U < n]$ for all n

U is bounded by H and 0 at terminal states

ϵ minimum probability of reduction

Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

$$U(x) \triangleq \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases}$$

Sound for AST

Complete for finite-state programs!

Variant Rule is incomplete!

Symmetric 1DRW

```
while (x != 0) {
    x++ ⊕1/2 x--;
}
```

AST, but cannot be proved using only bounded variants!

To prove that [Inv] $\geq$ wp. $Prog.\,1$, find an integer valued expectation U and two positive values ϵ and H such that

- Inv $\wedge\, \varphi \implies H > U > 0$ and $\neg\varphi \implies U = 0$
- $\epsilon[\text{Inv} \wedge U = n] \leq \text{wp.}\, body.\, [U < n]$ for all n

Observation: The candidate $U(x) = x$ is unbounded!

Maybe we make the variant unbounded?

Symmetric 1DRW

```
while (x != 0) {
    x++ ⊕1/2 x--;
}
```

AST, but cannot be proved using only bounded variants!

To prove that $[\text{Inv}] \geq \text{wp}.\, Prog.\, 1$, find an integer valued expectation U and two positive values ϵ ~~and H~~ such that

- $\text{Inv} \wedge \varphi \implies$ ~~$H >$~~ $U > 0$ and $\neg\varphi \implies U = 0$
- $\epsilon[\text{Inv} \wedge U = n] \leq \text{wp}.\, body.\, [U < n]$ for all n

Observation: The candidate $U(x) = x$ is unbounded!

Unbounded variants aren't sound

Asymmetric 1DRW

```
while (x != 0) {
    x++ ⊕1/3 x--;
}
```

Not AST, but the same unbounded candidate exists

To prove that [Inv] ≥ wp. $Prog.\,1$, find an integer valued expectation U and two positive values ϵ ~~and H~~ such that

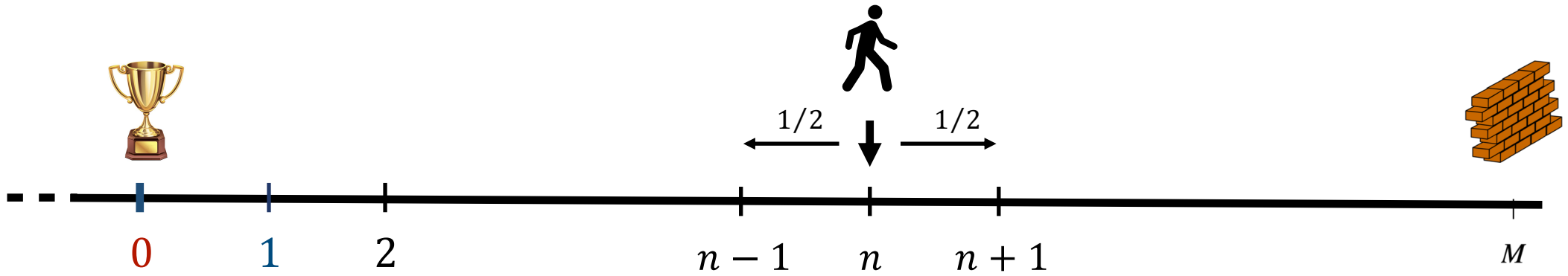
- Inv ∧ $\varphi \implies$ ~~$H >$~~ $U > 0$ and $\neg\varphi \implies U = 0$
- ϵ[Inv ∧ $U = n$] ≤ wp. $body.\,[U < n]$ for all n

Observation: The candidate $U(x) = x$ is unbounded!

Look deeper into the Symmetric 1DRW

Observation: The probability of hitting the wall grows smaller the further away the wall is!

```
while (x != 0 && x <= M) {
    x++ ⊕ 1/2 x--;
}
```



The candidate $U(x) = x$ is now bounded! $\Longrightarrow$ Walled 1DRW is terminating

Look deeper into the Symmetric 1DRW

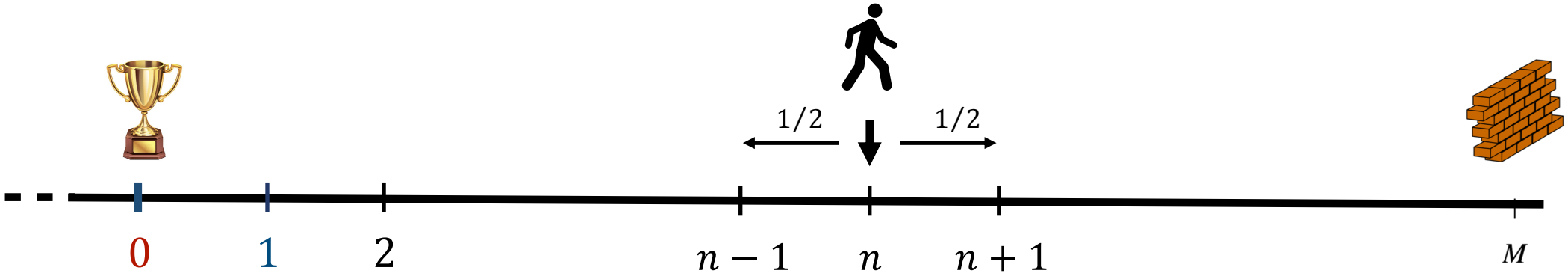
More precisely, for any $0 < \epsilon < 1$,

there is an M far away

such that probability of hitting the wall

at M before hitting  is $< \epsilon$

```
while (x != 0 ∧ x <= M) {
    x++ ⊕1/2 x--;
}
```



The candidate $U(x) = x$ is now bounded! $\Longrightarrow$ Walled 1DRW is terminating

Look deeper into the Symmetric 1DRW

More precisely, for any $0 < \epsilon < 1$,

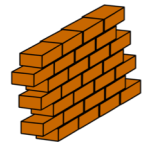
there is an M far away

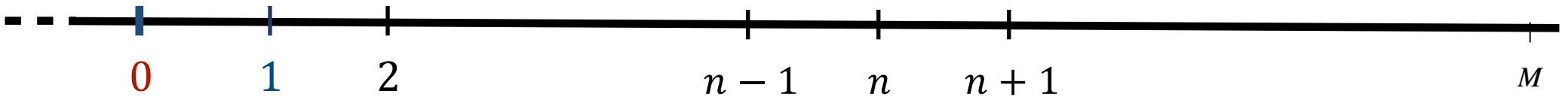
such that probability of hitting the wall

at M before hitting  is $< \epsilon$

```
while (x != 0 ∧ x <= M) {
    x++ ⊕1/3 x--;
}
```

Not true for the
Asymmetric 1DRW





The candidate $U(x) = x$ is now bounded! $\Longrightarrow$ Walled 1DRW is terminating

Supermartingales: A Notion of Boundedness over Time

Supermartingales: Mappings from states to real numbers that don't increase in expectation:

$$E[V(x')|x] \leq V(x)$$

Probability of increasing for a non-negative supermartingale from a low value L to a high value H is bounded above by L/H

Example: *Symmetric* Random Walk

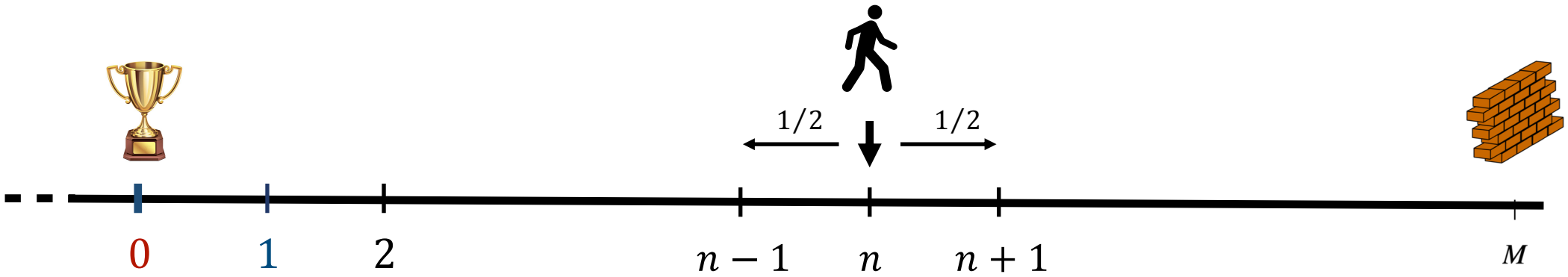
```
while (x != 0) {
    x++ ⊕1/2 x--;
}
```

$V(x) = x$ is a supermartingale

Symmetric 1DRW: Supermartingale

More precisely, for any $0 < \epsilon < 1$,
there is an M far away
such that probability of hitting the wall
at M before hitting  is $< \epsilon$

Probability of increasing *V(x) = x*
from a low value L to a high value
H is bounded above by L/H



$V(0) = 0$

$V(x) = x$ is a supermartingale

Probability of reaching wall $< n/M$

Note:
$U(x) = x$ is bounded when V(x) is bounded

We say U is *compatible with* V

Summarising

- Bounded variants imply AST
- Unbounded variants do not imply AST

- What if we have a supermartingale and a compatible variant?
  - We get AST!

Coin flipper

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

***Asymmetric* 1DRW**

```
while (x != 0) {
    x++ ⊕1/3 x--;
}
```

***Symmetric* 1DRW**

```
while (x != 0) {
    x++ ⊕1/2 x--;
}
```

$U(x) = x$ is an unbounded variant

$V(x) = x$ is a supermartingale

U is bounded when V is bounded

Proof Rule

To prove that $[\text{Inv}] \leq \text{wp}.\, Prog.\, 1$,

find an integer valued expectation U and a **real valued** expectation V such that

- $\text{Inv} \wedge \varphi \implies U > 0 \wedge V > 0$,
- $\neg\varphi \implies U = 0 \wedge V = 0$,

V and U are zero only at terminal states

- $V \geq \text{wp}.\, body.\, ([\text{Inv}]V)$,

The Supermartingale property

- For all $r \in \mathbb{R}^{>0}$, there exists an ϵ_r and a H_r such that
  - $V \leq r \implies U \leq H_r$,
  - $\epsilon_r[\text{Inv} \wedge U = n \wedge V \leq r] \leq \text{wp}.\, body.\, [U < n]$ for all n

Variant is compatible with supermartingale

Variant decreases with positive probability

Proof Rule

To prove that $[\text{Inv}] \leq \text{wp}.\, Prog.\, 1$,

find an integer valued expectation U and a **real valued** expectation V such that

- $\text{Inv} \wedge \varphi \Longrightarrow U > 0 \wedge V > 0$,
- $\neg\varphi \Longrightarrow U = 0 \wedge V = 0$,
- $V \geq \text{wp}.\, body.\, ([\text{Inv}]V)$,
- For all $r \in \mathbb{R}^{>0}$, there exists an ϵ_r and a H_r such that
  - $V \leq r \Longrightarrow U \leq H_r$,
  - $\epsilon_r[\text{Inv} \wedge U = n \wedge V \leq r] \leq \text{wp}.\, body.\, [U < n]$ for all n

***Symmetric* 1DRW**

```
while (x != 0) {
    x = 0 ⊕1/2 x = 1;
}
```

V(x) = x

U(x) = x

Outline

- A Program Logic (and why you can forget about it)
- The Rule
- Soundness
- Completeness

Proof of Soundness

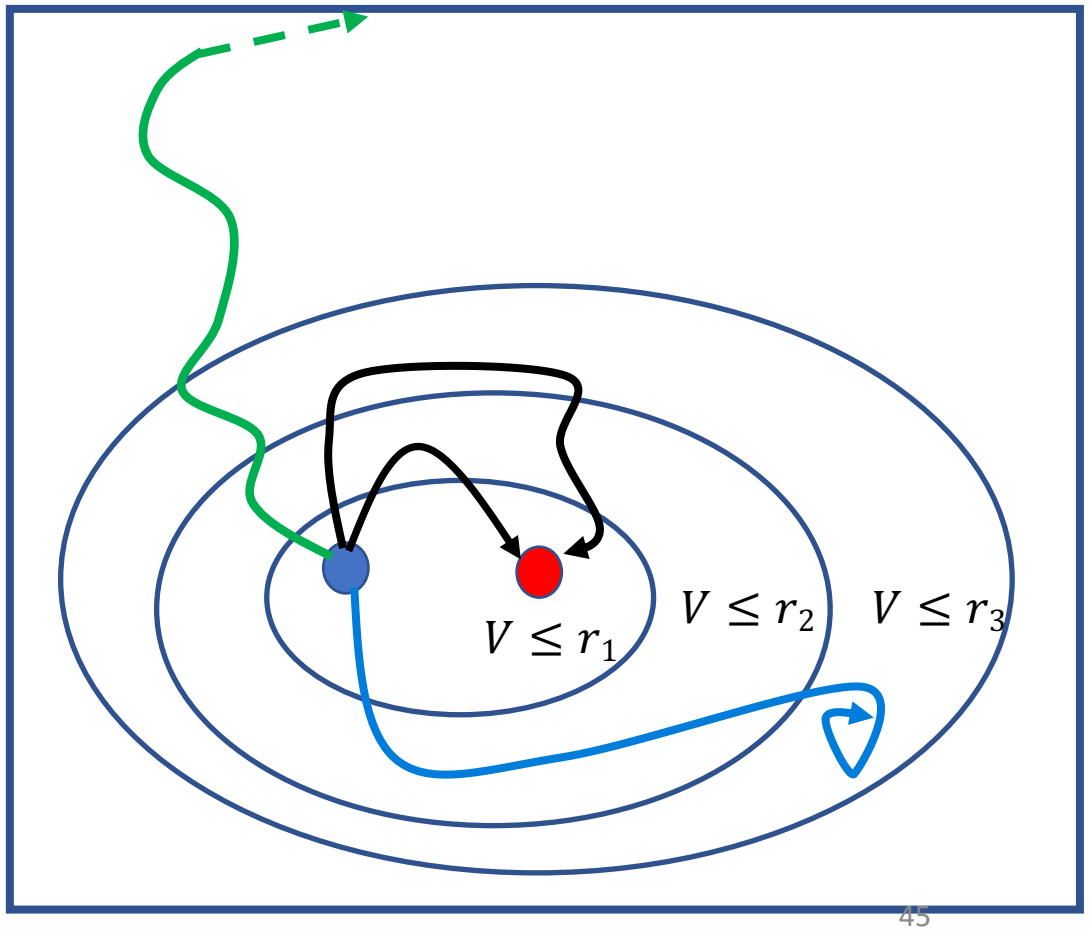
Fact #1 The probability of reaching $[V \geq r]$ shrinks to 0 as $r \to \infty$

The probability of executions for which $V \to \infty$ is 0

$Pr[\text{Green}] = 0$

Fact #2 Compatibility of Variant U implies that $while(\varphi \wedge V \leq r)\{body\}$ is AST

$Pr[\text{Blue}] = 0$



Drunken Men vs Drunken Birds

Symmetric Random Walks in d Dimensions

- Symmetric d-Dimensional Random Walk: Over points in $\mathbb{Z}^d$
- In each step, pick one of 2d neighbors u.a.r.

Theorem: From any starting point:

Pr[Reaching 0] = 1 if d=1,2 and < 1 if d>2

$$V(x, y) = \text{sqrt}(\log(x^2 + y^2))$$

Outline

- A Program Logic (and why you can forget about it)
- The Rule
- Soundness
- Completeness

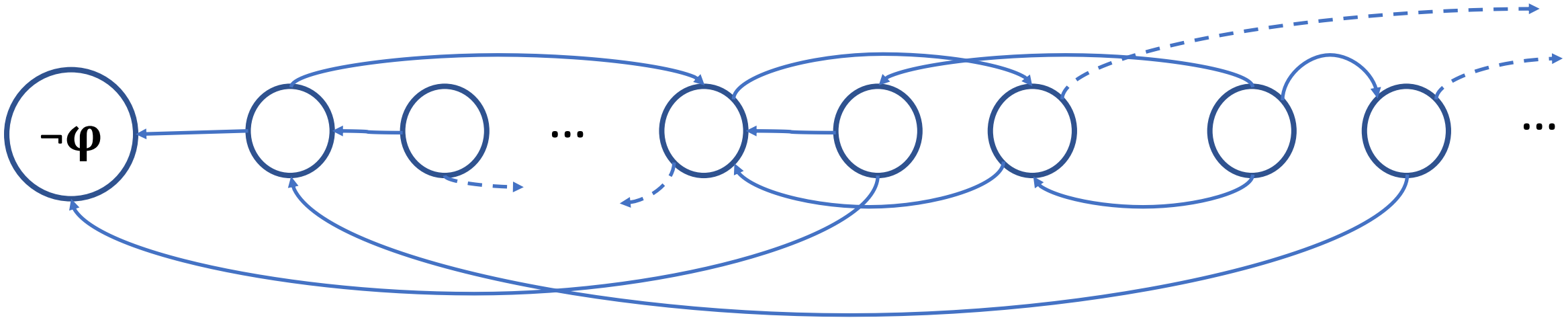
How do we prove completeness?

Given: An invariant Inv such that the loop $while(\varphi)\{body\}$ is AST from every state in Inv

To construct: a supermartingale V and a compatible variant U that is bounded when V is bounded

Given: All states terminate with prob 1

A transition system view of the loop



#1: Each transition step is a full execution of $body$

Meaning branching can be unbounded, but probability weights sum to 1

#2: All terminal states are lumped together

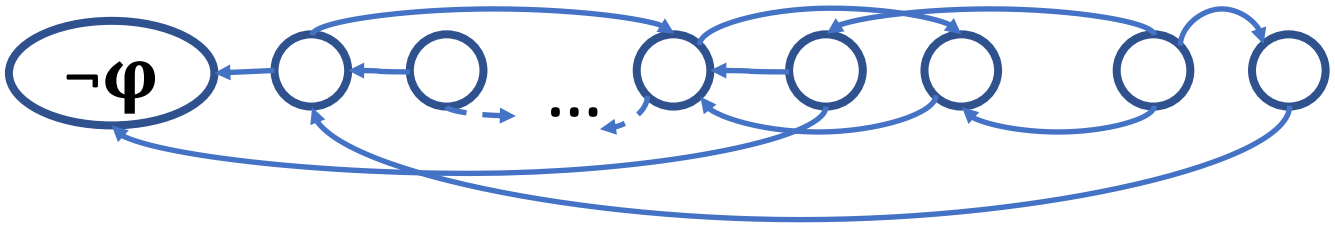
Building the variant

Given: All states terminate with prob 1

Let's start with McIver & Morgan's variant rule from 2005

To prove that $[\text{Inv}] \leq \text{wp}.Prog.1$, find an integer valued expectation U and two positive values ϵ and H such that

- $\text{Inv} \wedge \varphi \Longrightarrow H > U > 0$ and $\neg\varphi \Longrightarrow U = 0$
- $\epsilon[\text{Inv} \wedge U = n] \leq \text{wp}.body.[U < n]$ for all n



Complete for finite-state programs!

A function that assigns to σ ***the smallest number of executions of*** $body$ to get non-zero probability of termination

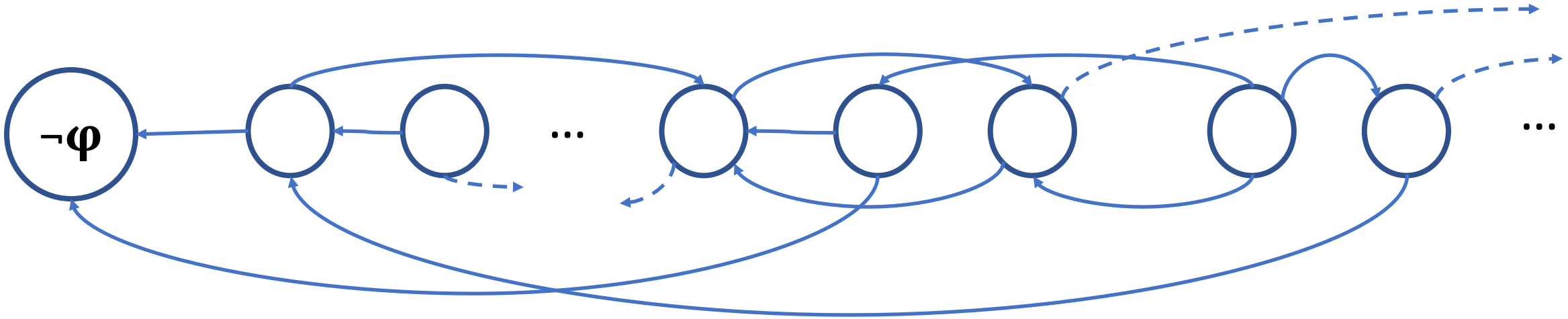
Define the variant U: state σ ↦length of shortest path from σ to $\neg\varphi$

For finite graphs, U is trivially bounded

Given: All states terminate with prob 1

Building the variant

What if the state space in unbounded?



Define the variant U: state $\sigma \mapsto$ length of shortest path from σ to $\neg\varphi$

U can be unbounded (e.g., 1DRW)

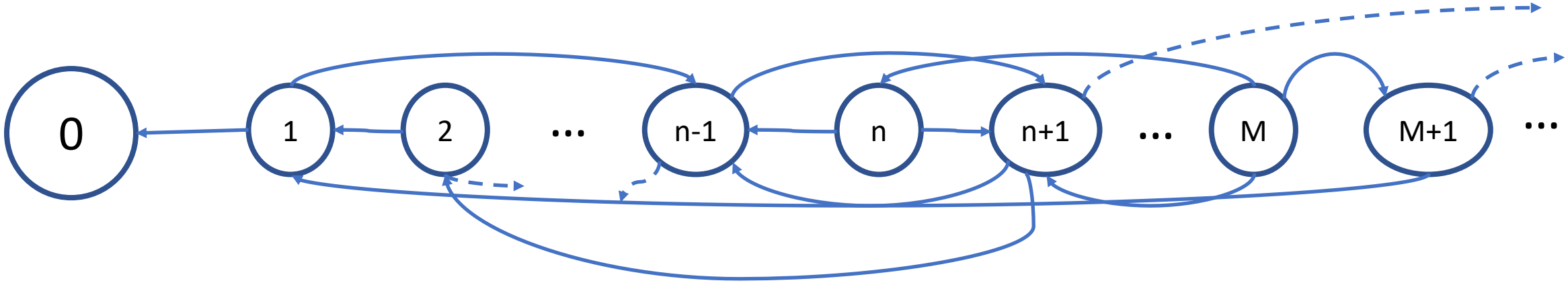
From now on, ***assume*** U *is unbounded*

Given: All states terminate with prob 1

Building the supermartingale

Enumeration

Step 1: Arbitrarily enumerate all nonterminal states.



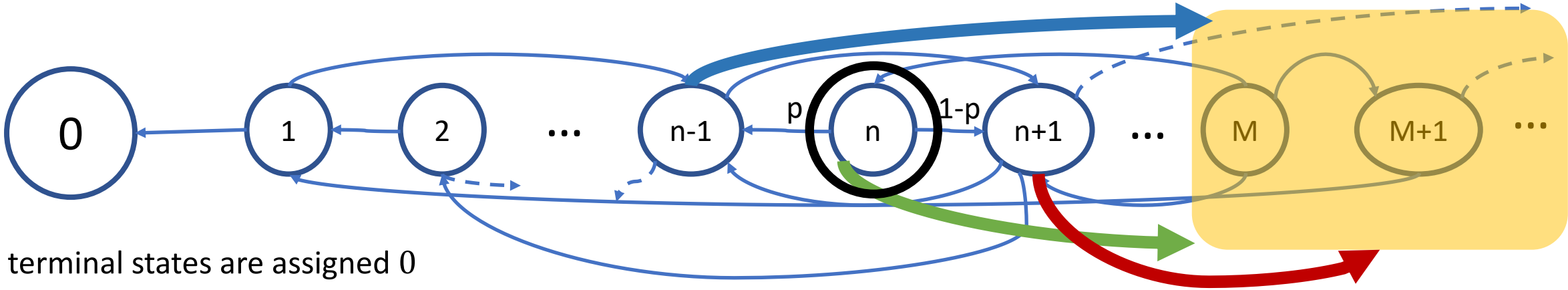
All terminal states are assigned 0

Given: All states terminate with prob 1

Building the supermartingale

For this enumeration, construct a bounded supermartingale

Step 2: For any M, the mapping $V_M : n \mapsto$ Probability of reaching $\{M, M+1, \ldots\}$ before reaching 0 is a bounded martingale



All terminal states are assigned 0

$V_M(M) = 1$

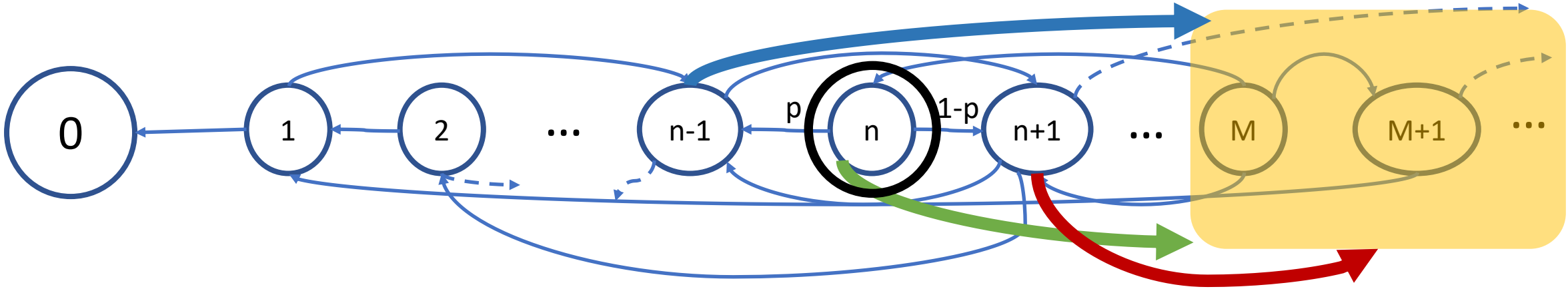
$$V_M(n) \triangleq pV_M(n-1) + (1-p)V_M(n+1)$$

Given: All states terminate with prob 1

Building the supermartingale

For this enumeration, construct a bounded supermartingale

Step 2: For any M, the mapping $V_M : n \mapsto$ Probability of reaching $\{M, M+1, \ldots\}$ before reaching 0 is a bounded martingale



$$V_M(n) \triangleq pV_M(n-1) + (1-p)V_M(n+1)$$

$$V_M(M) = 1$$

Problem: Compatibility: *Every state satisfies* $V_M(M) \leq 1$*, but U is unbounded*

Given: All states terminate with prob 1

Building the supermartingale

For this enumeration, construct a bounded supermartingale

We have:

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \ldots\}$ before reaching 0

$U: n \mapsto$ length of shortest path to 0

Problem: Compatibility: *Every state satisfies $V_M(M) \leq 1$, but U is unbounded*

Can we define V = $\sum V_M$

This would diverge

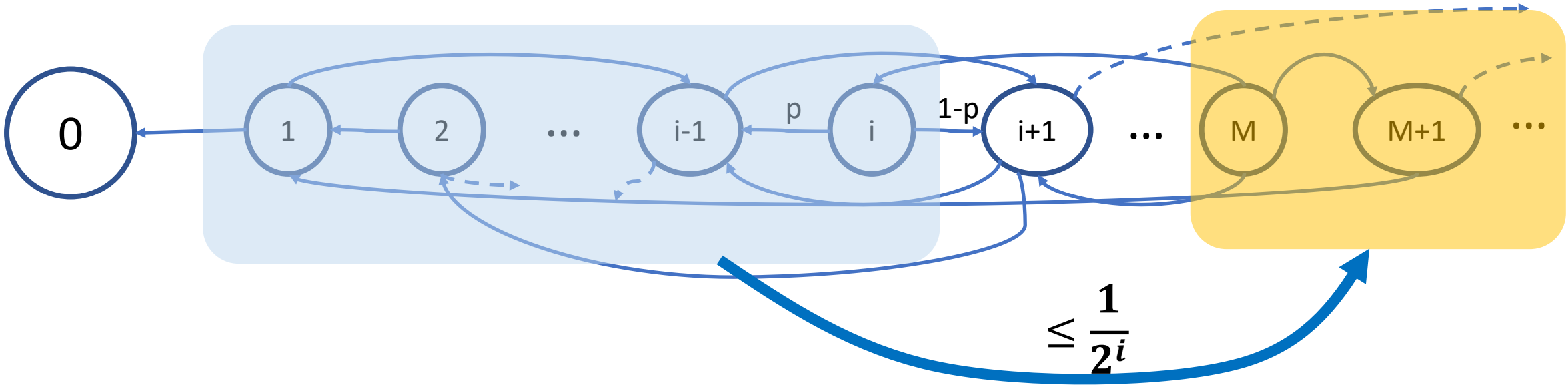
Building the supermartingale

Defining V by diagonalization

Given: All states terminate with prob 1

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \dots\}$ before reaching 0

Step 3: Define the sequence $(n_i)_{i \in \mathbb{N}}$ such that n_i is the *smallest* number such that $V_{n_i}(j) \leq 1/2^i$ for all $j \leq i$.



Claim: *If the sequence exists, then* $\sum V_{n_i}$ *converges*

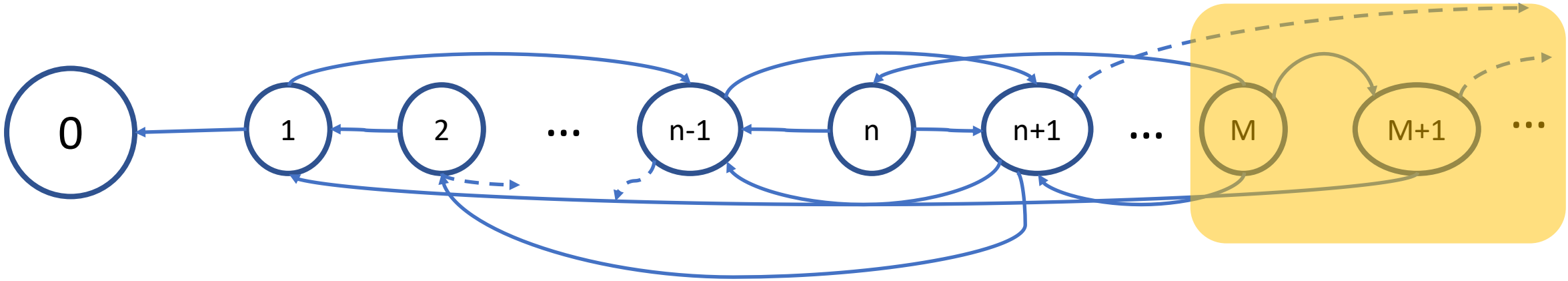
Building the supermartingale

Showing the sequence exists

Given: All states terminate with prob 1

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \ldots\}$ before reaching 0

Step 4 [Left leaning lemma]: For each fixed n, the probability of reaching {M, M+1, …} goes to 0 as M → ∞



Fact: for every p < 1,
there is a *finite* collection of *terminal (finite)* runs from n of probability weight p
These runs only see a finite part of the state space

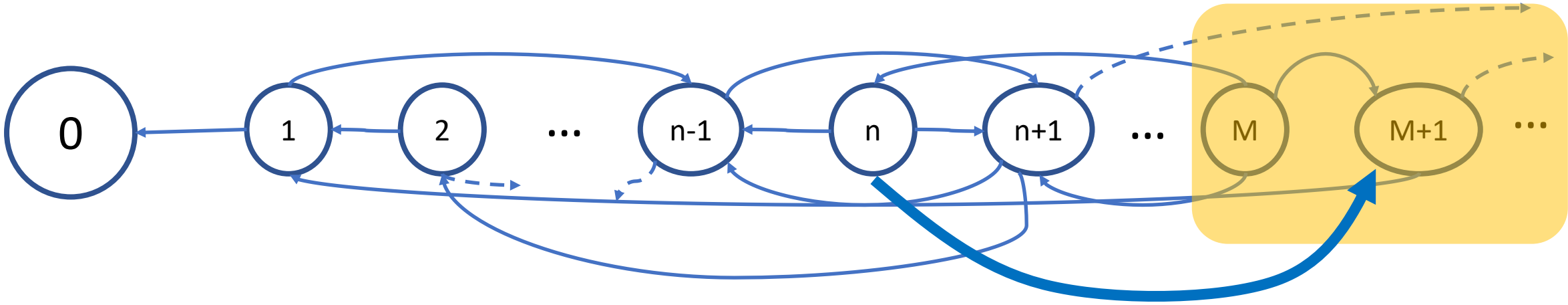
Building the supermartingale

Showing the sequence exists

Given: All states terminate with prob 1

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \ldots\}$ before reaching 0

Step 4 [Left leaning lemma]: For each fixed n, the probability of reaching {M, M+1, …} goes to 0 as M → ∞



Fact: for every p < 1, there is a *finite* collection of *terminal (finite)* runs from n of probability weight p

$\leq 1 - p$

As p increases, M gets further away

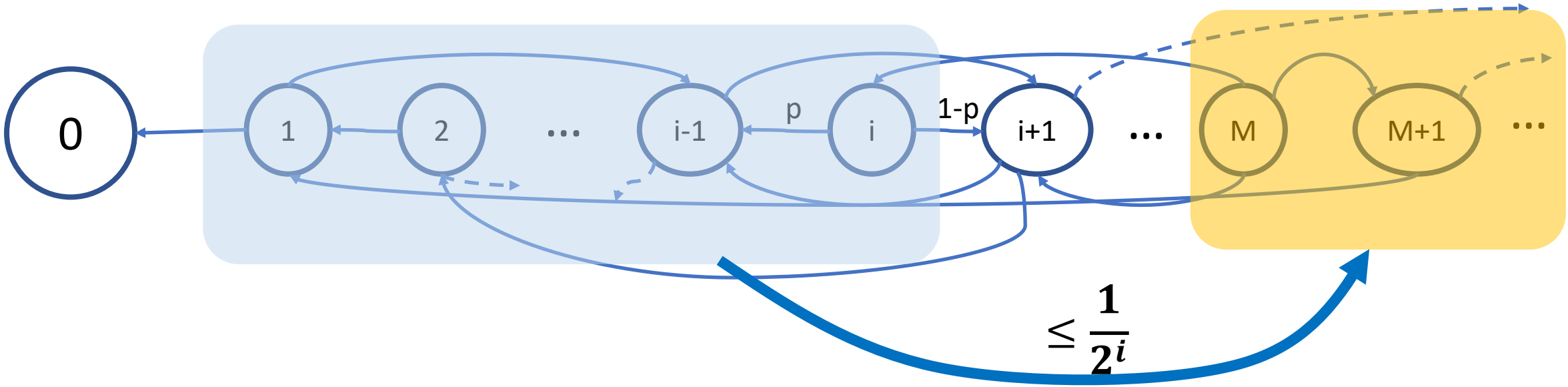
Building the supermartingale

Defining V by diagonalization

Given: All states terminate with prob 1

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \dots\}$ before reaching 0

Step 3: Define the sequence $(n_i)_{i \in \mathbb{N}}$ such that n_i is the *smallest* number such that $V_{n_i}(j) \leq 1/2^i$ for all $j \leq i$



Claim: *If the sequence exists, then* $\sum V_{n_i}$ *converges*

The left leaning lemma implies the sequence exists

Building the supermartingale

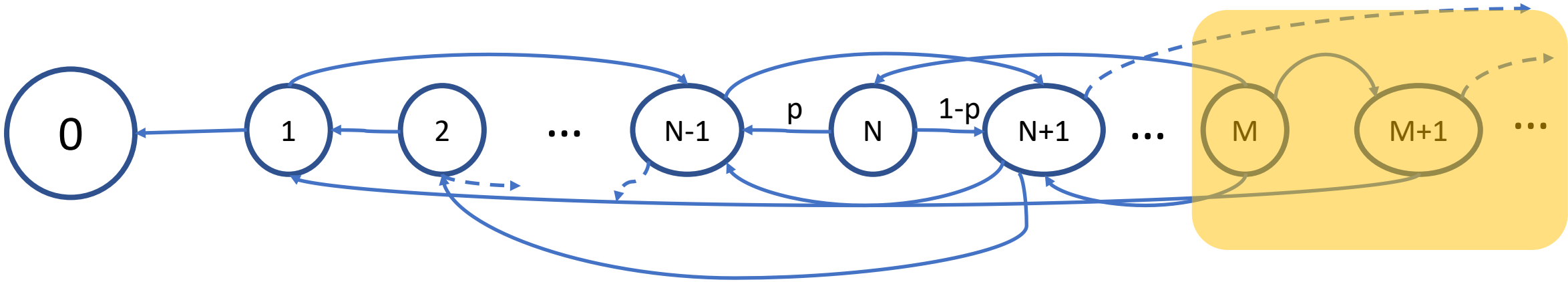
Given: All states terminate with prob 1

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \dots\}$ before reaching 0

Step 5: Prove $\sum V_{n_i}$ *converges*

Define the sequence $(n_i)_{i \in \mathbb{N}}$ such that n_i is the *smallest* number such that $V_{n_i}(j) \leq 1/2^i$ for all $j \leq i$

Fix N, vary V_{n_i} in $\sum V_{n_i}(N)$



"Small" n_i

$$N \geq n_i \Rightarrow V_{n_i}(N) = 1$$

"Medium" n_i

$$i < N < n_i \Rightarrow V_{n_i}(N) \leq 1$$

"Large" n_i

$$N \leq i \Rightarrow V_{n_i}(N) \leq \frac{1}{2^i}$$

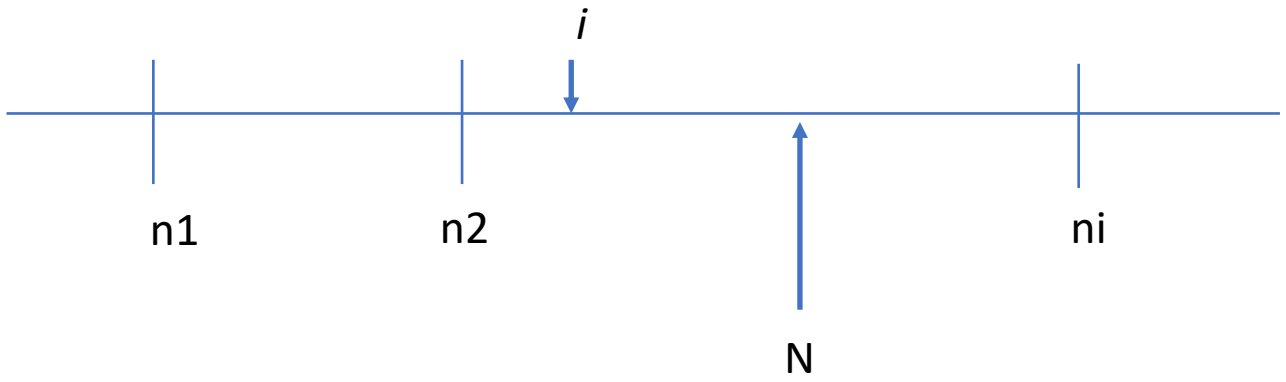
Building the supermartingale

Given: All states terminate with prob 1

$V_M: n \mapsto$ Probability of reaching $\{M, M+1, \dots\}$ before reaching 0

Step 5: Prove $\sum V_{n_i}$ *converges*

Define the sequence $(n_i)_{i \in \mathbb{N}}$ such that n_i is the *smallest* number such that $V_{n_i}(j) \leq 1/2^i$ for all $j \leq i$



Fix N, vary V_{n_i} in $\sum V_{n_i}(N)$

"Small" n_i $\quad N \geq n_i \Longrightarrow V_{n_i}(N) = 1$

"Medium" n_i $\quad i < N < n_i \Longrightarrow V_{n_i}(N) \leq 1$

"Large" n_i $\quad N \leq i \Longrightarrow V_{n_i}(N) \leq \frac{1}{2^i}$

$$\left(\sum V_{n_i}\right)(N) \leq M_N + \sum_{j \geq i_N} \frac{1}{2^j} \leq M_N + 1$$

Building the supermartingale

Defining an unbounded V by diagonalization

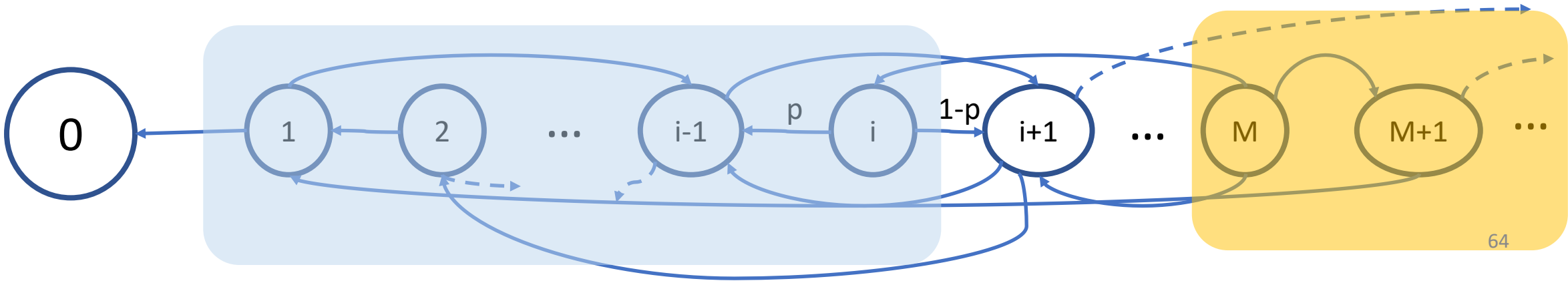
Given: All states terminate with prob 1

$V_M : n \mapsto$ Probability of reaching $\{M, M+1, \dots\}$ before reaching 0

Define the sequence $(n_i)_{i \in \mathbb{N}}$ such that n_i is the *smallest* number such that $V_{n_i}(j) \leq 1/2^i$ for all $j \leq i$

Step 6: Define the supermartingale $V : n \mapsto \sum_{i=1}^{\infty} V_{n_i}(n)$

Claim: V is well-defined, increasing, $[V \leq r]$ *finite* for each r



Given: All states terminate with prob 1

Summary

$V_M : n \mapsto$ Probability of reaching $\{M, M+1, \ldots\}$ before reaching 0

Define the sequence $(n_i)_{i \in \mathbb{N}}$ such that

n_i is the *smallest* number such that $V_{n_i}(j) \leq 1/2^i$ for all $j \leq i$

Define the supermartingale $V : n \mapsto \sum_{i=1}^{\infty} V_{n_i}(n)$

Define the variant U: state $\sigma \mapsto$ length of shortest path from σ to $\neg\varphi$

1. V is well-defined
2. V is a supermartingale
3. $[V \leq r]$ is finite for each r
4. U is a variant
5. U is compatible with V

Outline

- A Program Logic (and why you can forget about it)
- The Rule
- Soundness
- Completeness
- Coffee

A Sound and Complete Proof Rule for AST

To prove that $[\mathsf{Inv}] \leq \mathsf{wp}.Prog.1$,
find an integer valued expectation U and a **real valued** expectation V such that

- $\mathsf{Inv} \wedge \varphi \Longrightarrow U > 0 \wedge V > 0$,
- $\neg\varphi \Longrightarrow U = 0 \wedge V = 0$,
- $V \geq \mathsf{wp}.body.([\mathsf{Inv}]V)$,

- For all $r \in \mathbb{R}^{>0}$, there exists an ϵ_r and a H_r such that
  - $V \leq r \Longrightarrow U \leq Hr$,
  - $\epsilon_r\ [\mathsf{Inv} \wedge U = n \wedge V \leq r] \leq \mathsf{wp}.body.[U < n]$ for all n

Where do we go next?

- Applications: Proving AST for randomized protocols
  - Research question: *How can you add fairness?*
- Theory: Proving AST for infinite-branching nondeterminism
  - Research question: *How can you combine ordinals and supermartingales?*
- Tools: First steps based on Caesar
  - Research question: *Automation*

Thank You!