Simulation vs. Formal: “Absorb what is useful; reject what is useless”

Bruce Lee

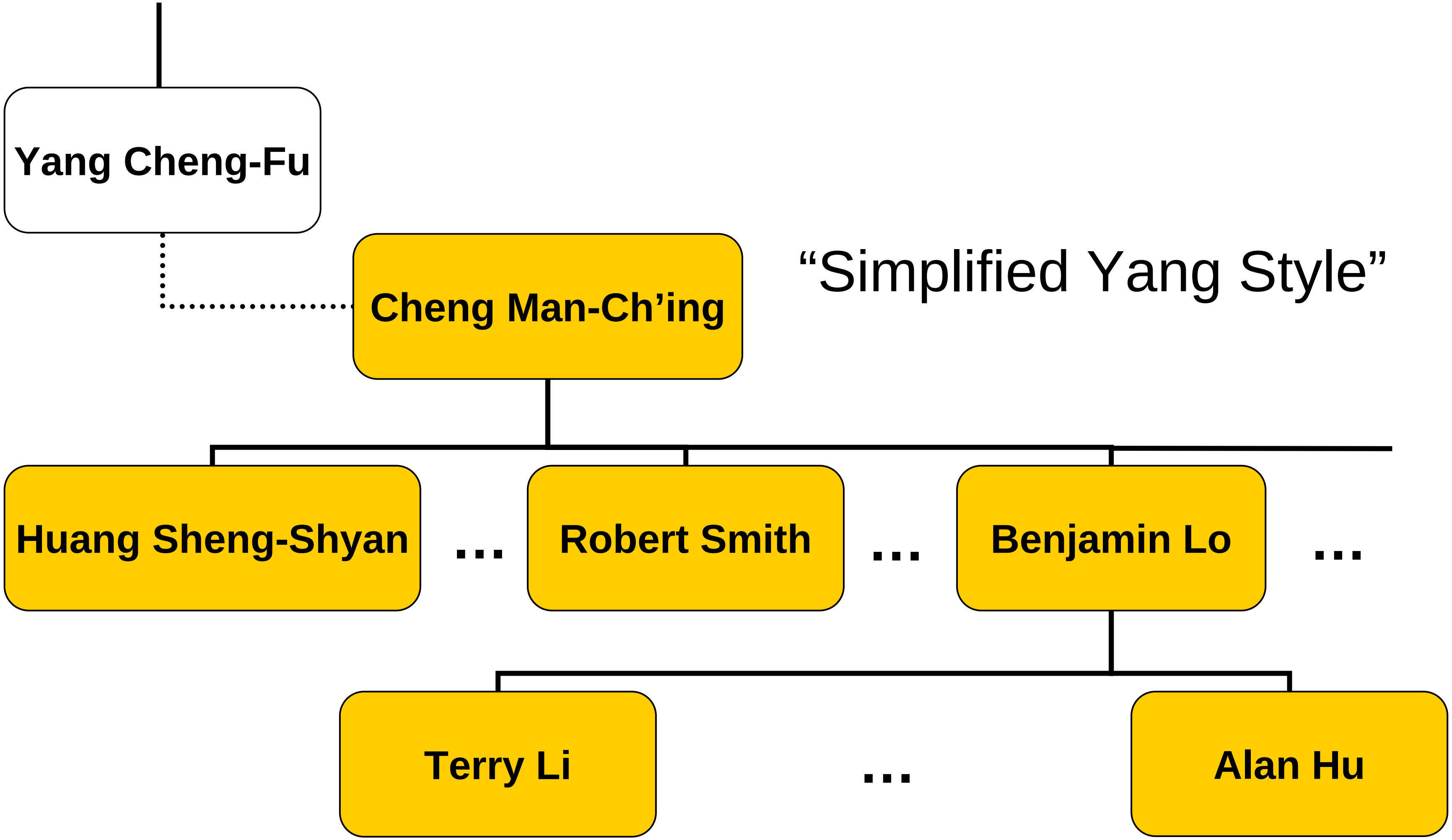
Approach to Verification

Alan J. Hu

University of British Columbia

Absorb what is useful.
Reject what is useless.
-- Bruce Lee

My T’ai Chi Ch’uan Genealogy



Yang Cheng-Fu

Cheng Man-Ch’ing

“Simplified Yang Style”

Huang Sheng-Shyan …

Robert Smith …

Benjamin Lo …

Terry Li …

Alan Hu

Characteristics of Traditional Martial Arts Instruction

- Study in a school led by the master.
- Introductory classes are in groups, with syllabus set by the master.
- Advanced study is one-on-one with master.
- Interact almost exclusively with fellow students and the master.
- Travel to tournaments/workshops. Compete/interact with others of same style.
- Read books, articles, papers by masters of your own style.
- Denigration of other styles.

Problem?

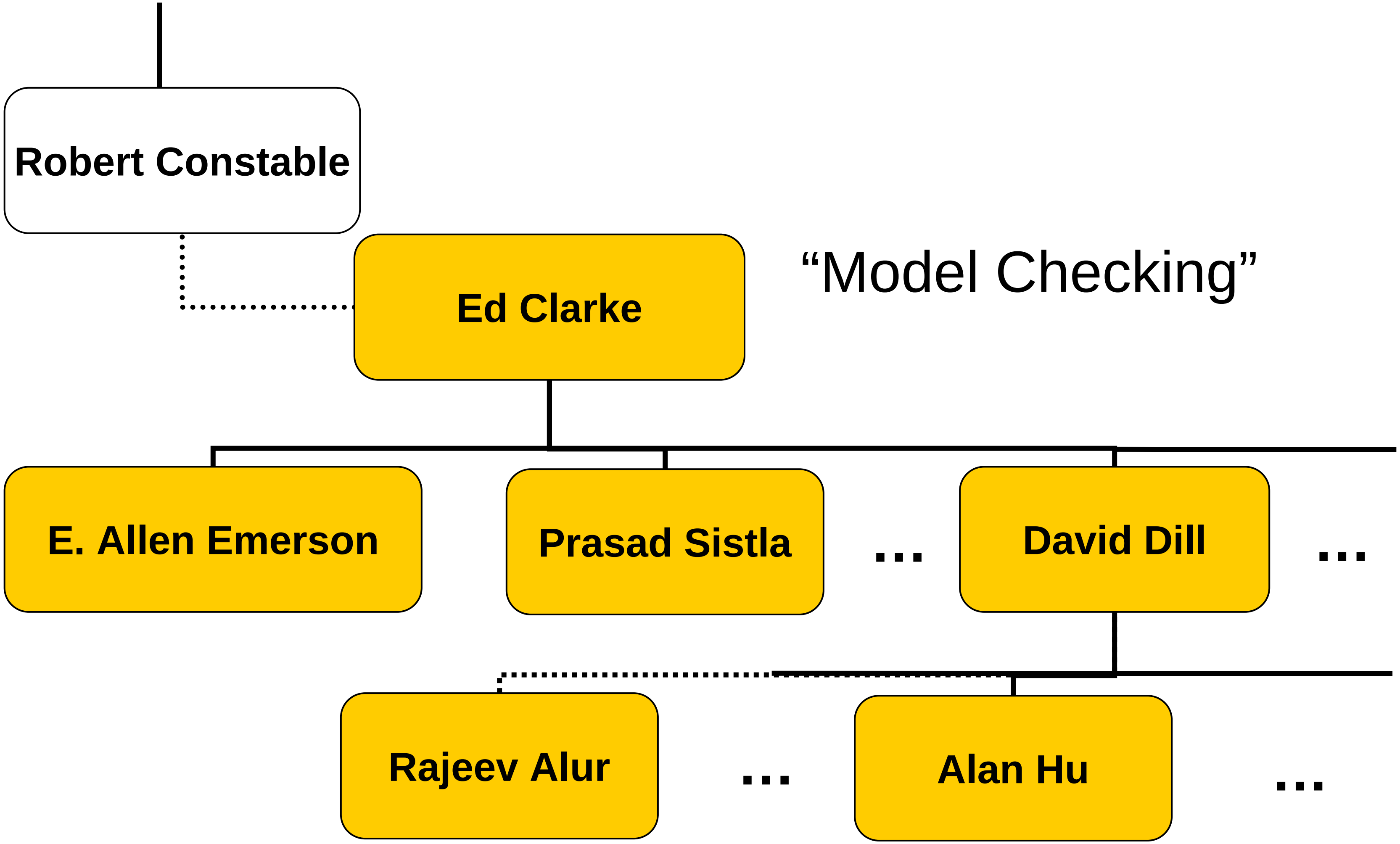
Characteristics of Traditional Martial Arts Instruction

- Study in a school led by the master.
- Introductory classes are in groups, with syllabus set by the master.
- Advanced study is one-on-one with master.
- Interact almost exclusively with fellow students and the master.
- Travel to tournaments/workshops. Compete/interact with others of same style.
- Read books, articles, papers by masters of your own style.
- Denigration of other styles.

Characteristics of Traditional EE/CS Instruction

- Study in a school led by the professors.
- Introductory classes are in groups, with syllabus set by the professor.
- Advanced study is one-on-one with professor.
- Interact almost exclusively with fellow students and the professor.
- Travel to conferences/workshops. Compete/interact with others of same style.
- Read books, articles, papers by masters of your own style.
- Denigration of other styles.

My Verification Genealogy


Robert Constable

Ed Clarke

“Model Checking”

E. Allen Emerson

Prasad Sistla

…

David Dill

…

Rajeev Alur

…

Alan Hu

…

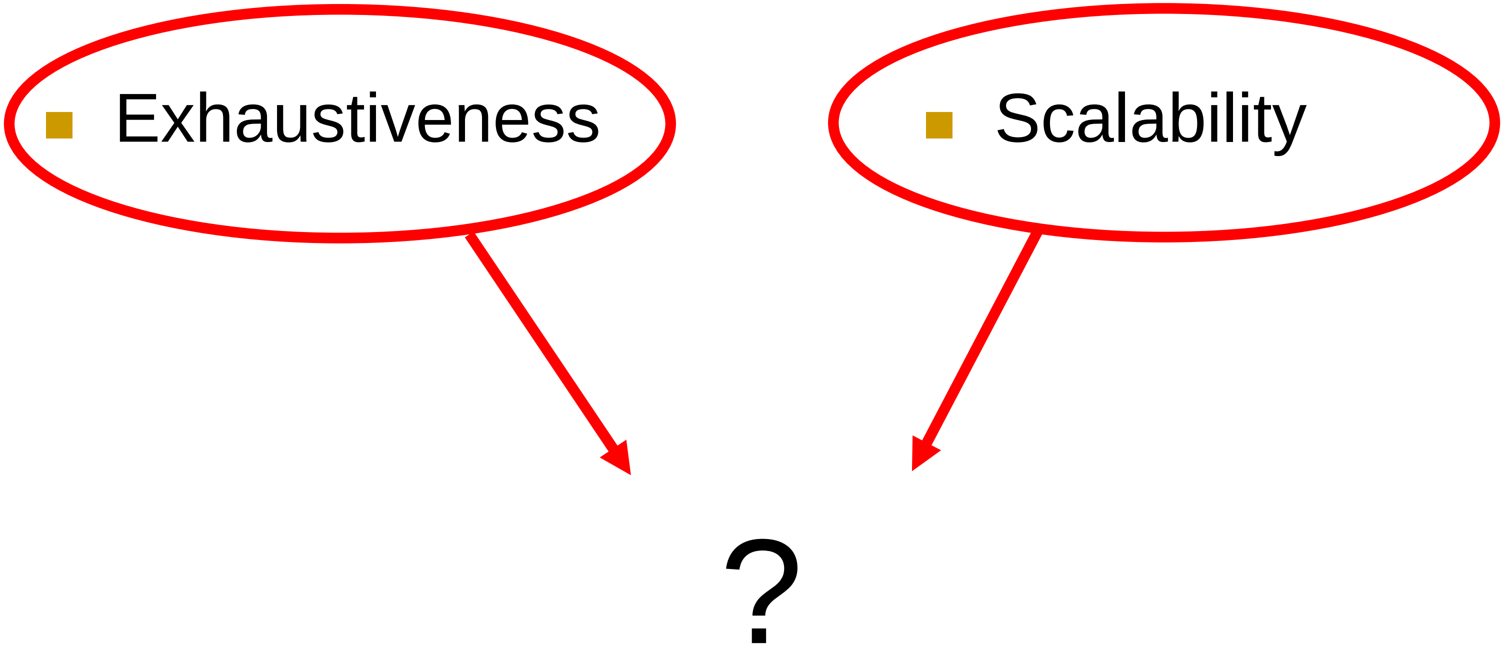
Problem?

- Slow spread of good ideas
- Failing to notice one’s own assumptions, blind spots

Formal vs. Simulation

- Exhaustiveness
- Scalability

Formal vs. Simulation



- Exhaustiveness
- Scalability

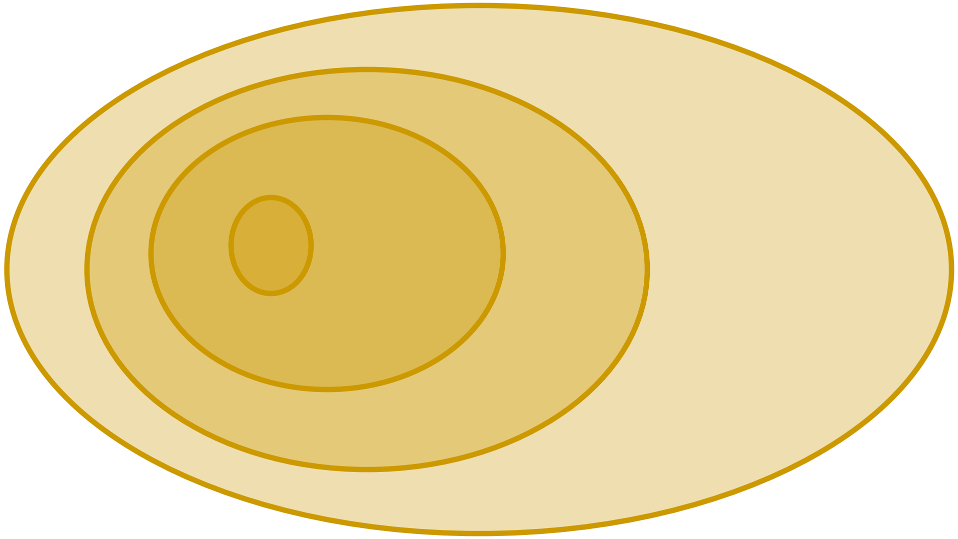
?

Mixing Formal and Simulation

- Methodological Combination:
  - Use formal wherever it can work.
  - Everywhere else, simulate.
- Semi-Formal:
  - Use a bit of formal while simulating.
  - Under-approximate images during formal verification.

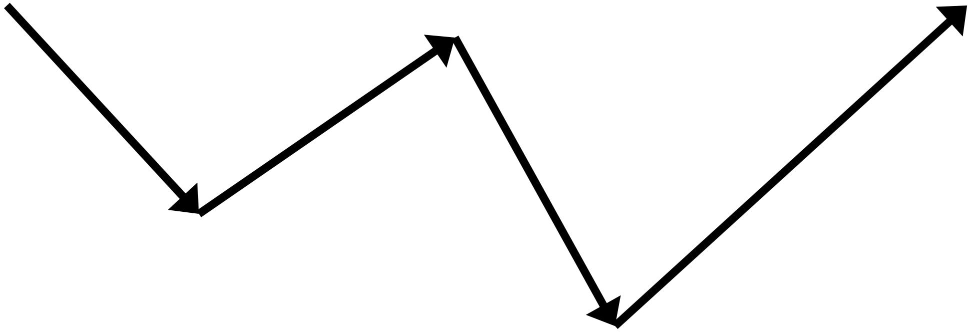
Semi-Formal: 1st Generation

- Formal (Symbolic Model Checking):



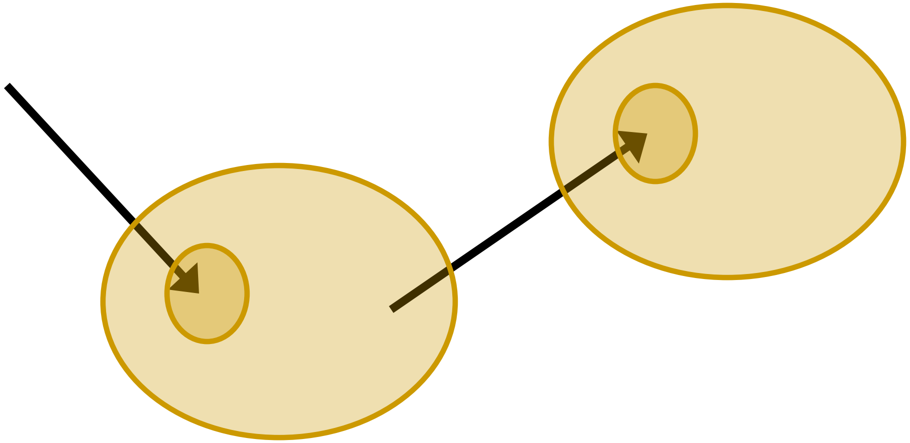
Semi-Formal: 1st Generation

- Dynamic Verification (Simulation/Emulation):



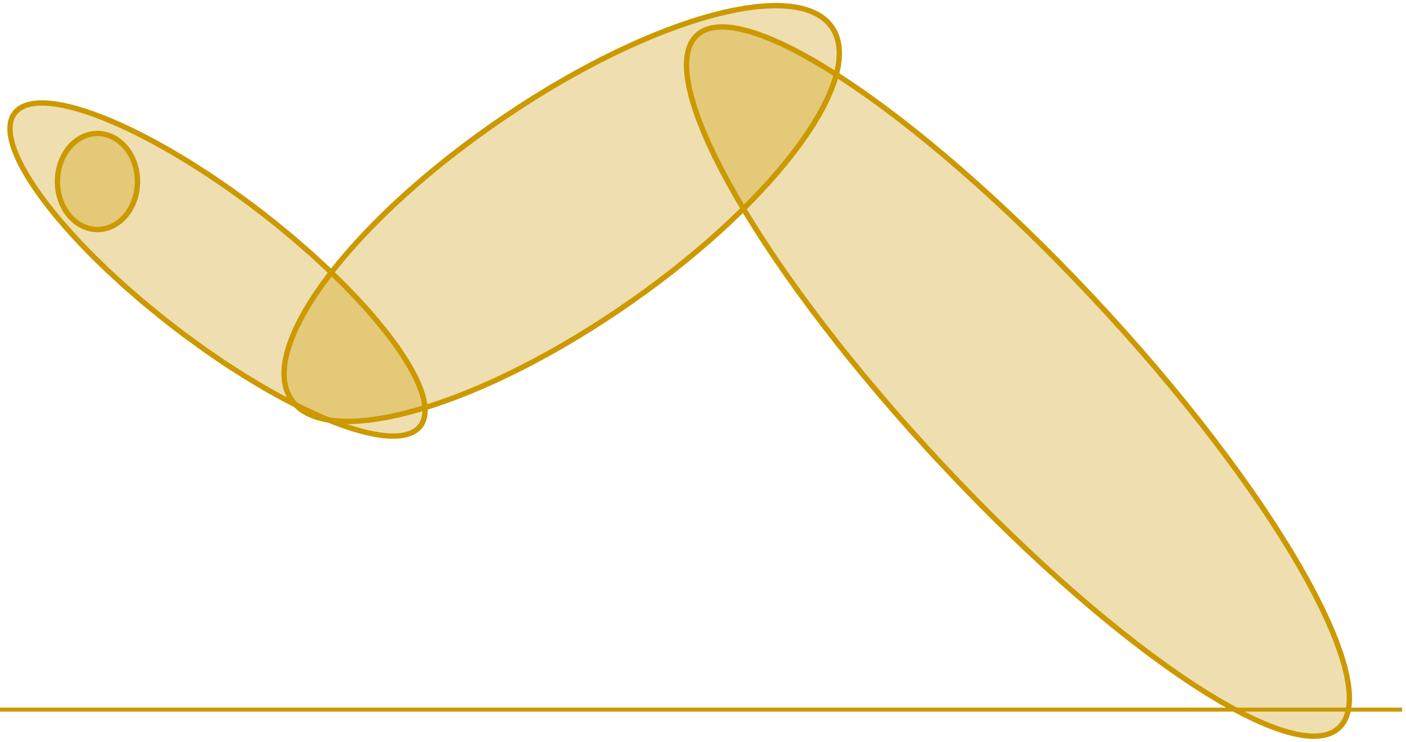
Semi-Formal: 1st Generation

- Semi-Formal (a bit of formal while simulating):



Semi-Formal: 1st Generation

- Semi-Formal (Under-Approximate Images):



Formal vs. Simulation

- Exhaustiveness

- Scalability

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability

Formal vs. Simulation

- Exhaustive analysis is useful!
    - Smart, brute force (BDDs, SAT, SMT, etc.)
    - Abstraction
- Machine-readable specifications.

- Scalability

E.g., Constrained Test Generation

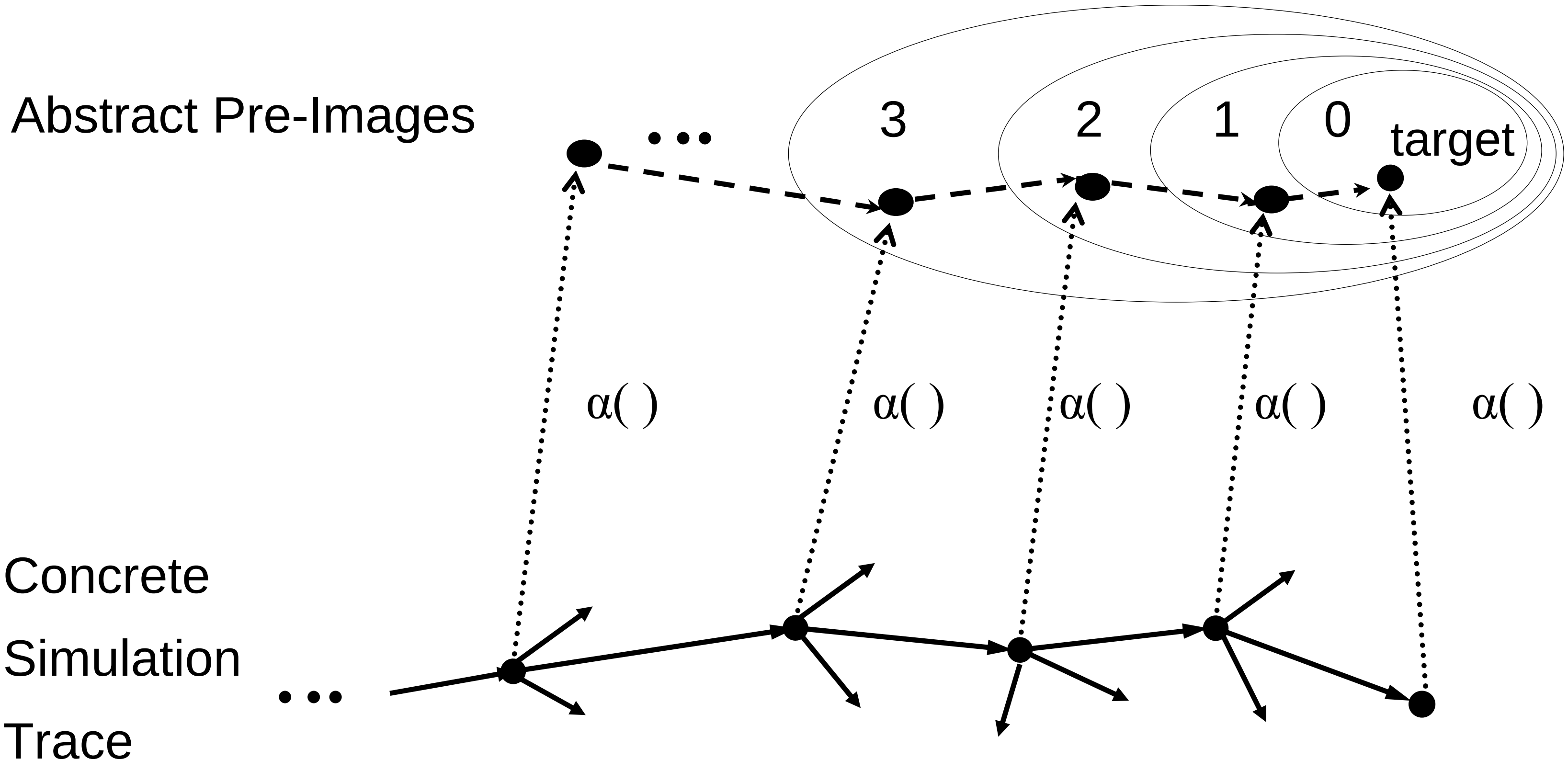
Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

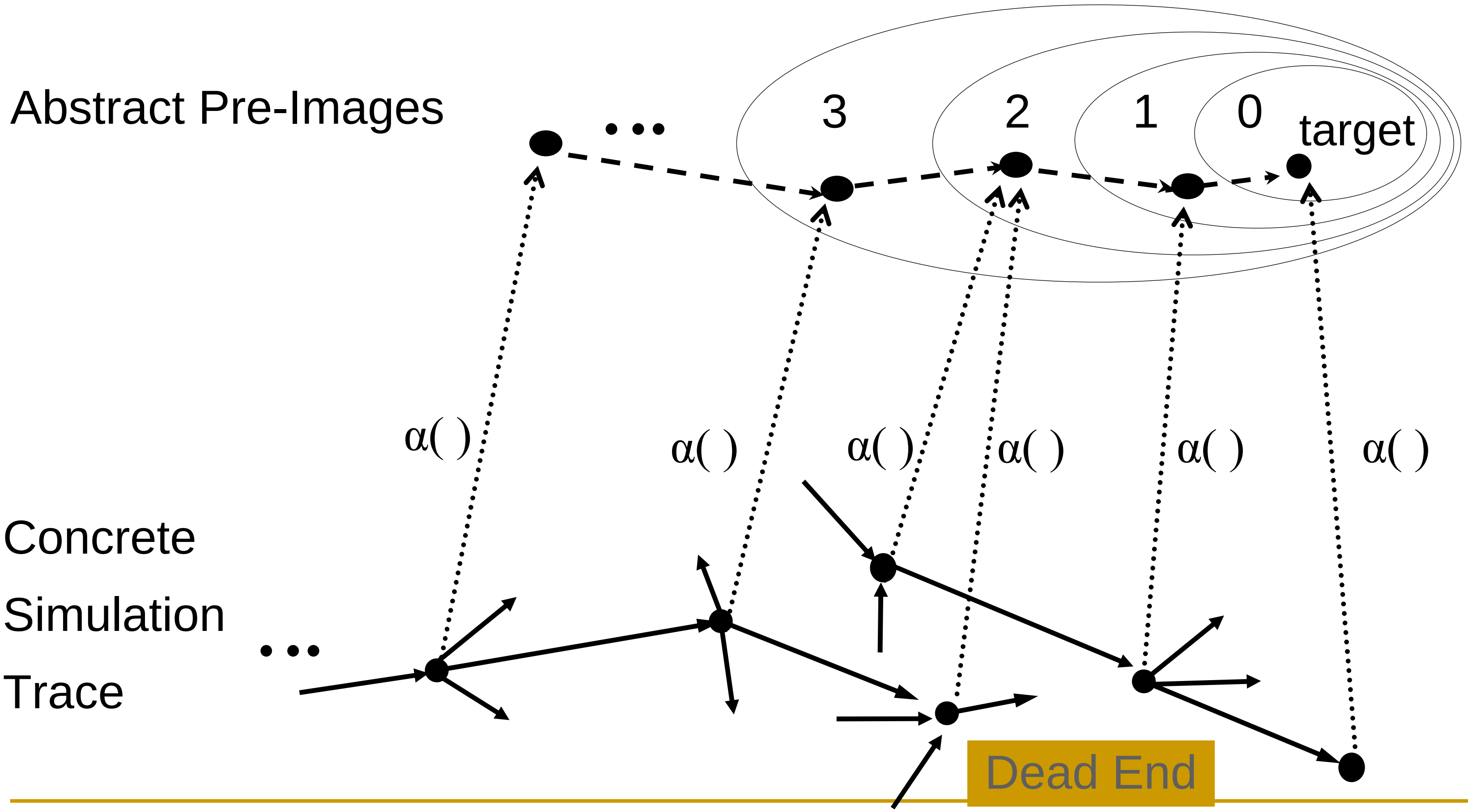
- Scalability

E.g., Abstraction-Guided Simulation

Abstraction-Guided Simulation



Leaky Abstractions



Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability

E.g., Abstraction-Guided Simulation

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability

E.g., Assertion-Based Verification

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability

Formal vs. Simulation

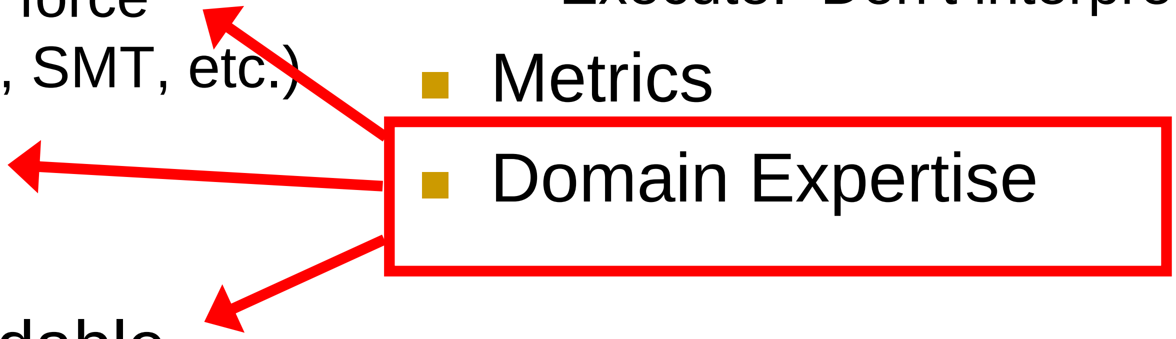
- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don't interpret.
- Metrics
- Domain Expertise

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise



Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don't interpret.
- Metrics
- Domain Expertise

E.g., Coverage for Formal Specs

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

E.g., Predicate Complete Testing

Predicate Complete Testing

- Predicate Abstraction: Use a set of n predicates as abstraction function.

$$\alpha : C \rightarrow \mathrm{B}^n$$

- Heuristic: Use all conditions in program as predicates.

- Predicate Complete Testing: Use the abstract state space as coverage metric for simulation.

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

E.g., Predicate Complete Testing

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don't interpret.
- Metrics
- Domain Expertise

E.g., SATOMETER, Semi-Formal BMC

SATOMETER, Semiformal BMC

- SAT solvers use learned clauses to track how much of the solution space has been explored.
- Collect these clauses in a ZBDD.
- Report fraction to user.

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

E.g., SATOMETER, Semi-Formal BMC

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

Maximizing Processor Performance

- Anything that reduces instruction-level parallelism costs performance:
  - Control Dependence – Branch mispredict penalty 10-20 cycles.
  - Data Dependence – L2 miss penalty 50-100 cycles.
- Unpredictable branch costs 10s of instructions.
- Random memory access costs 100s of instructions.
- Parallelizable instructions are free.

Compiled Code Simulation

- Conventional Logic Simulator:
  - Stores circuit in memory
  - Walks that data structure
  - Interprets gates/operators
  - Hundreds of instructions per gate.
- Compiled Code Simulator:
  - Compiles circuit into machine instructions.
  - Executes those instructions. No interpretation.
  - Few branches. Fewer memory accesses.
  - A few instructions per gate.

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

E.g., SATOMETER, Semi-Formal BMC

Conventional SAT Solver for BMC

- SAT solver stores circuit (as clauses) in memory.
- SAT solver walks this data structure.
- SAT solver interprets clauses.
- Even values on wires are encoded into data structure.
- Poor performance! (90% of time is BCP.)

Compiled Circuit SAT Solver

- Compile verification problem into a program.
- Brute Force:

```
while (vector = pick_a_vector()) {
    if (compiled_simulate(vector))
        return SAT;
    record_unsuccessful_trial(vector);
}
return UNSAT;
```

Compiled Circuit SAT Solver

- Compile verification problem into a program.
- New Idea:

```
while (vector = pick_a_vector()) {
    if (smart_compiled_simulate(vector))
        return SAT;
    record_unsuccessful_trial(vector.learned);
}
return UNSAT;
```

Compiled Circuit SAT Solver

- It sometimes works great.
- E.g., 2n by n bit Radix-2 SRT divider.

n	Chaff	Compiled
4	1.2	0.4
5	7.5	4.7
6	98.1	56.8
7	2848.4	735.2
8	time	time(0.7737)

- Also reports progress made.
- (Published as “Semiformal Bounded Model Checking”)

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don't interpret.
- Metrics
- Domain Expertise

E.g., SATOMETER, Semi-Formal BMC

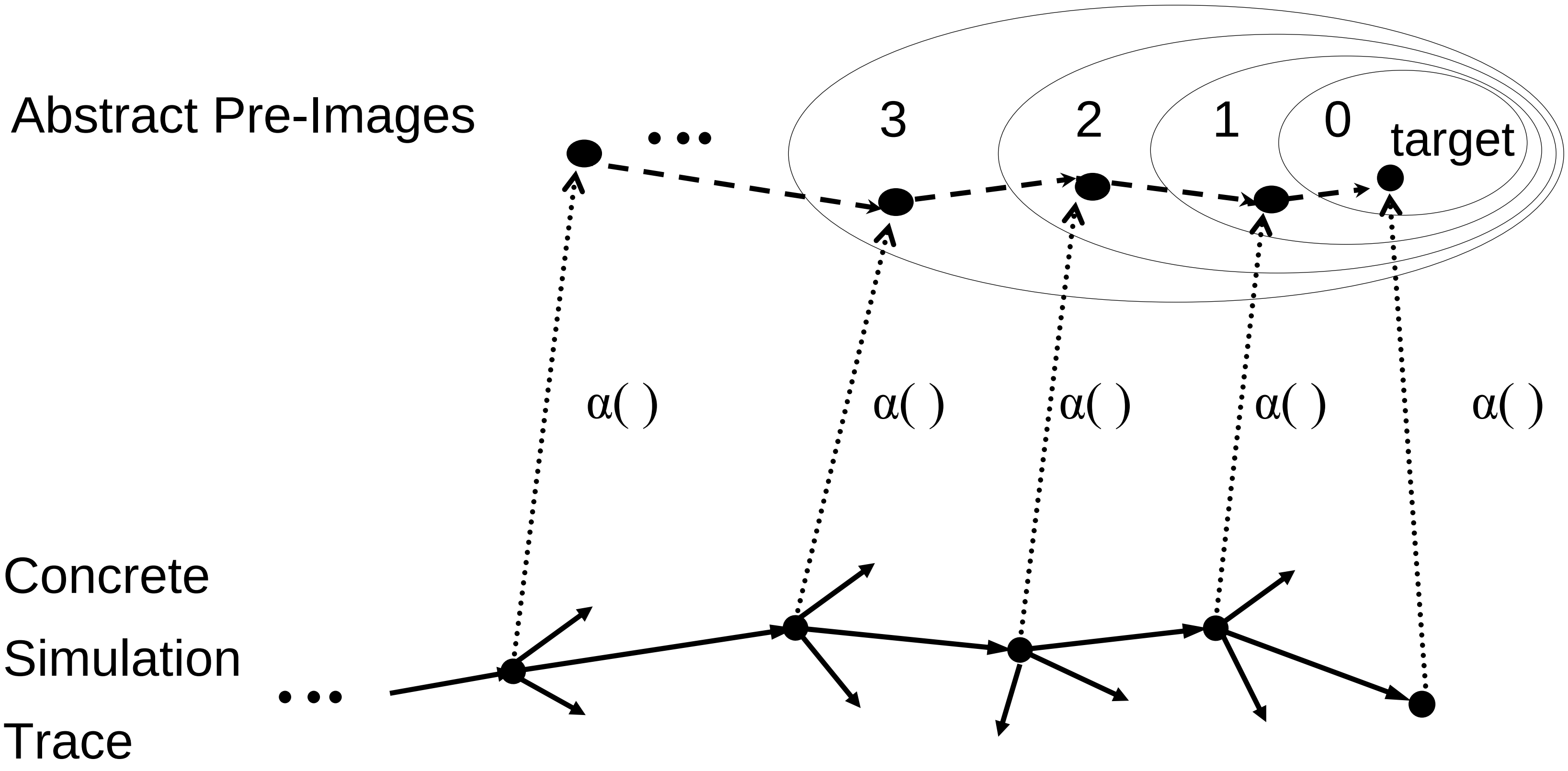
Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

E.g., EverLost

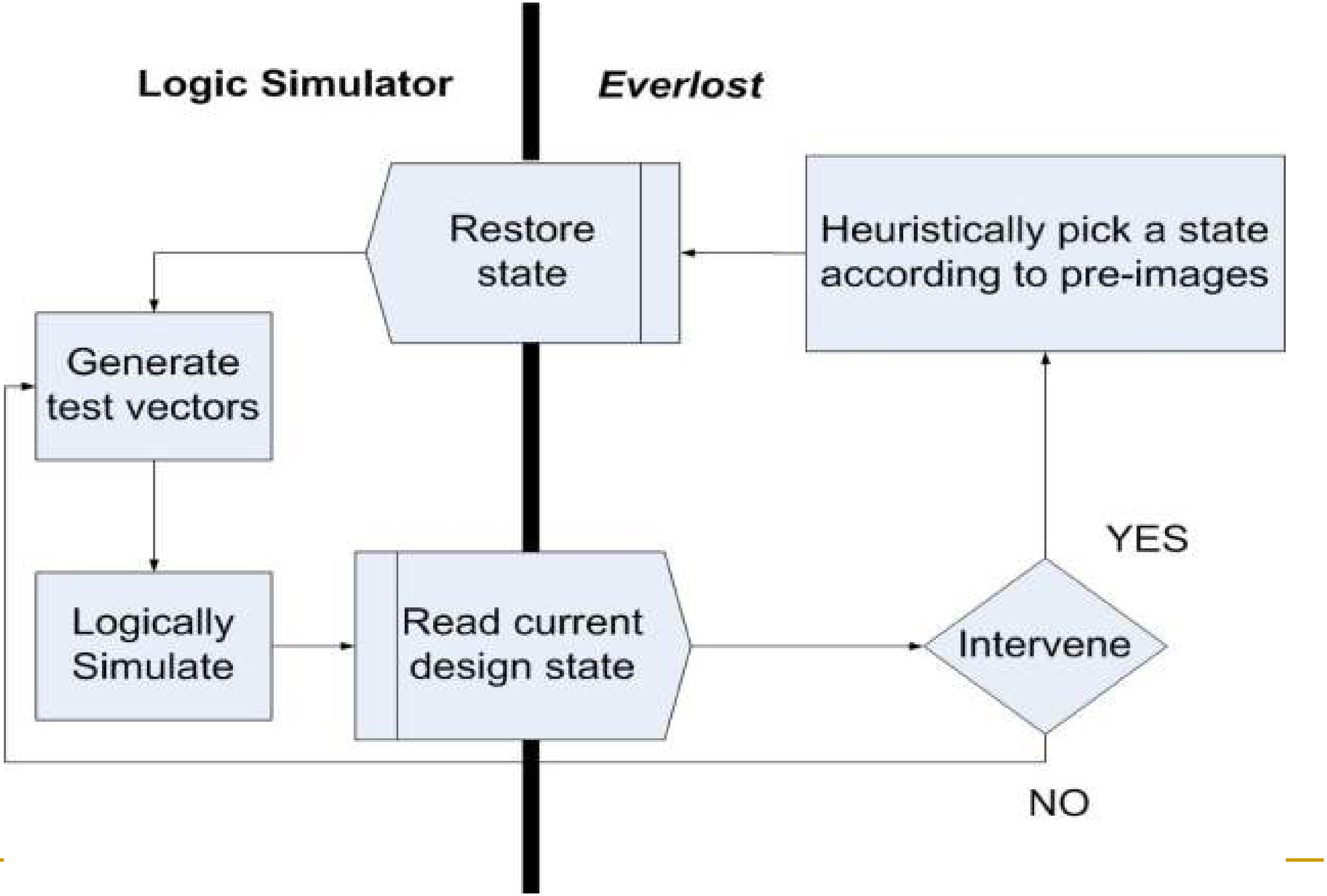
Abstraction-Guided Simulation



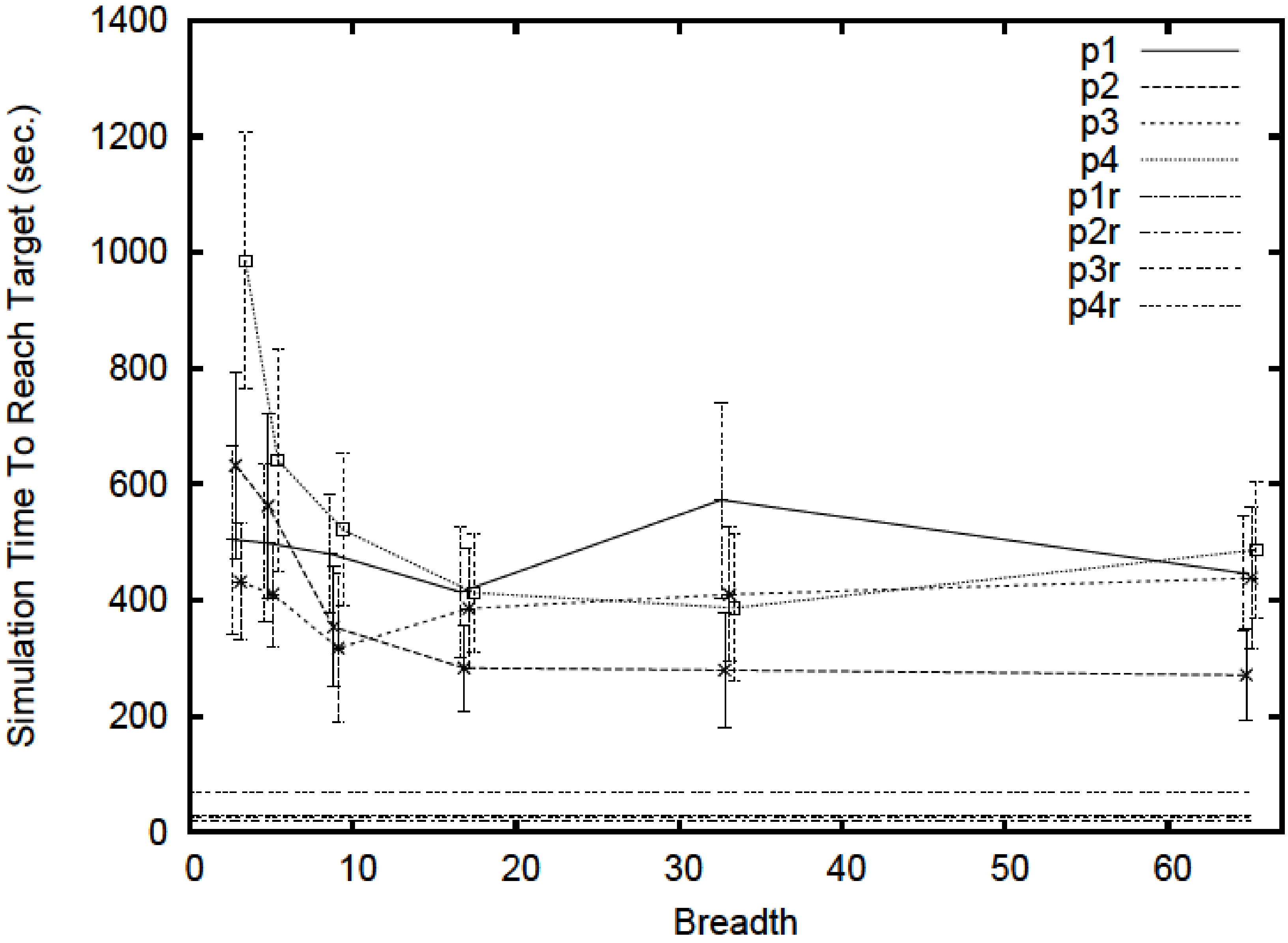
Abstract Pre-Images

Concrete Simulation Trace

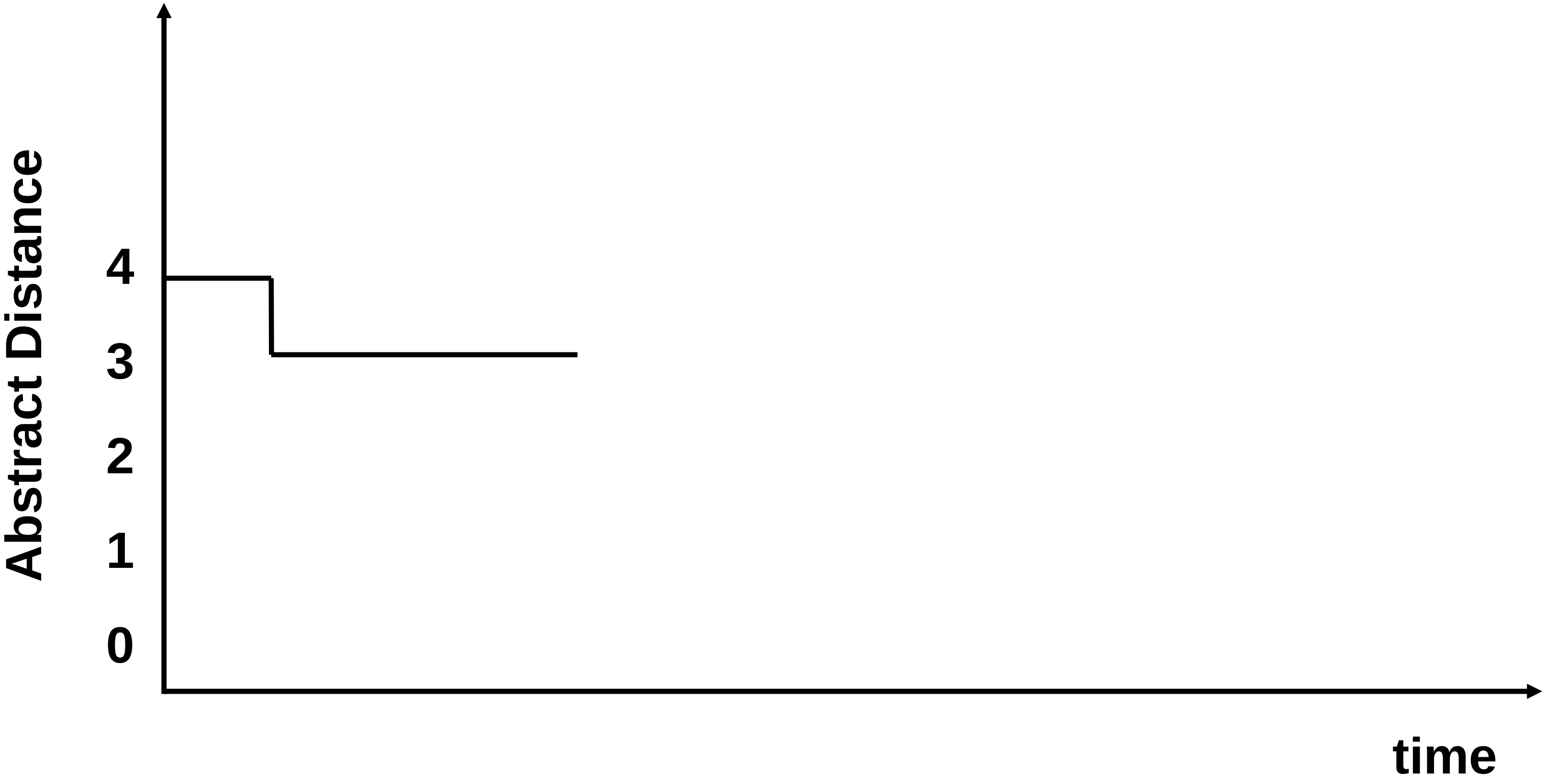
Platform: EverLost



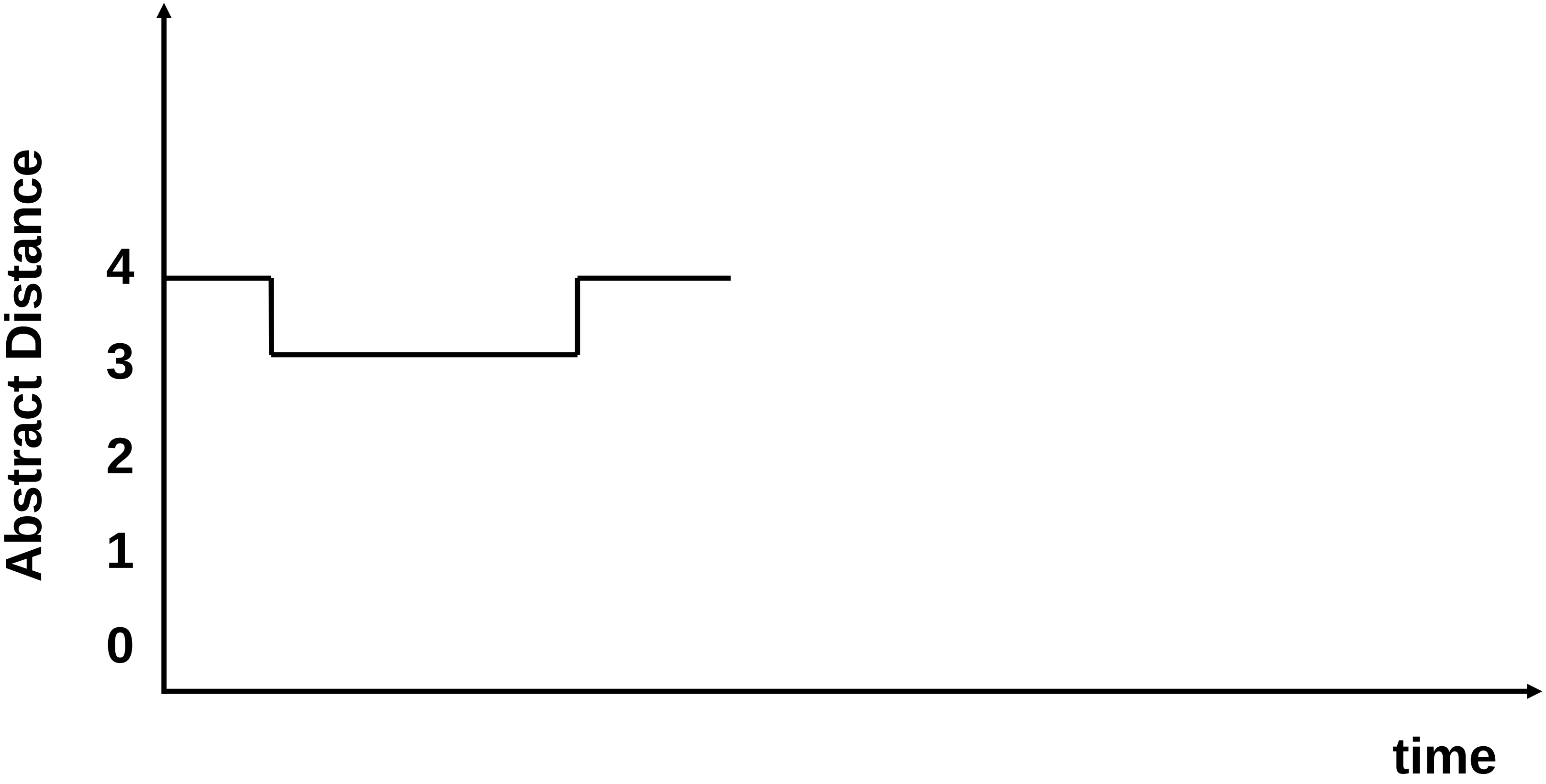
Simulation Time for SimSearch



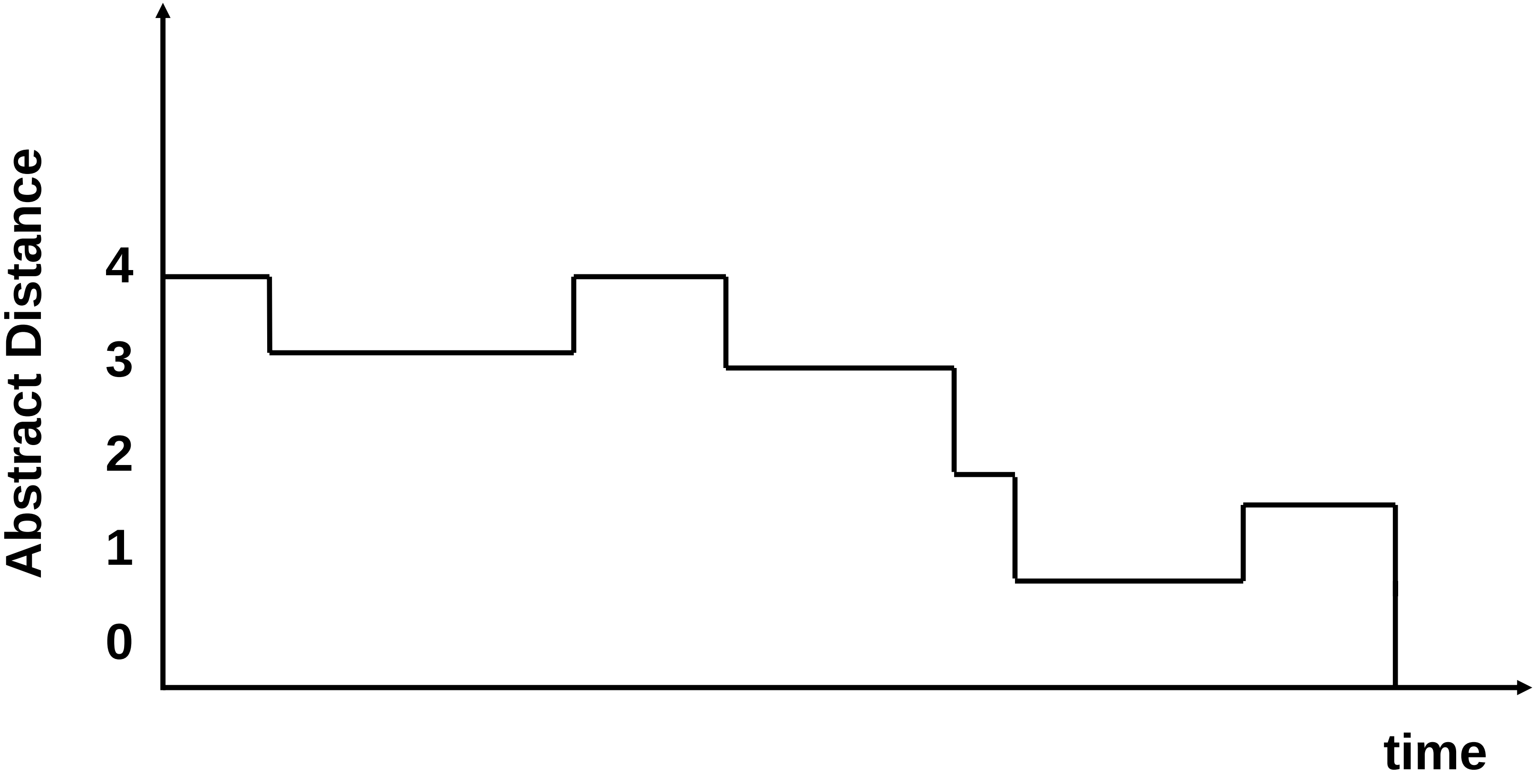
Simulation Trace: Expected Results



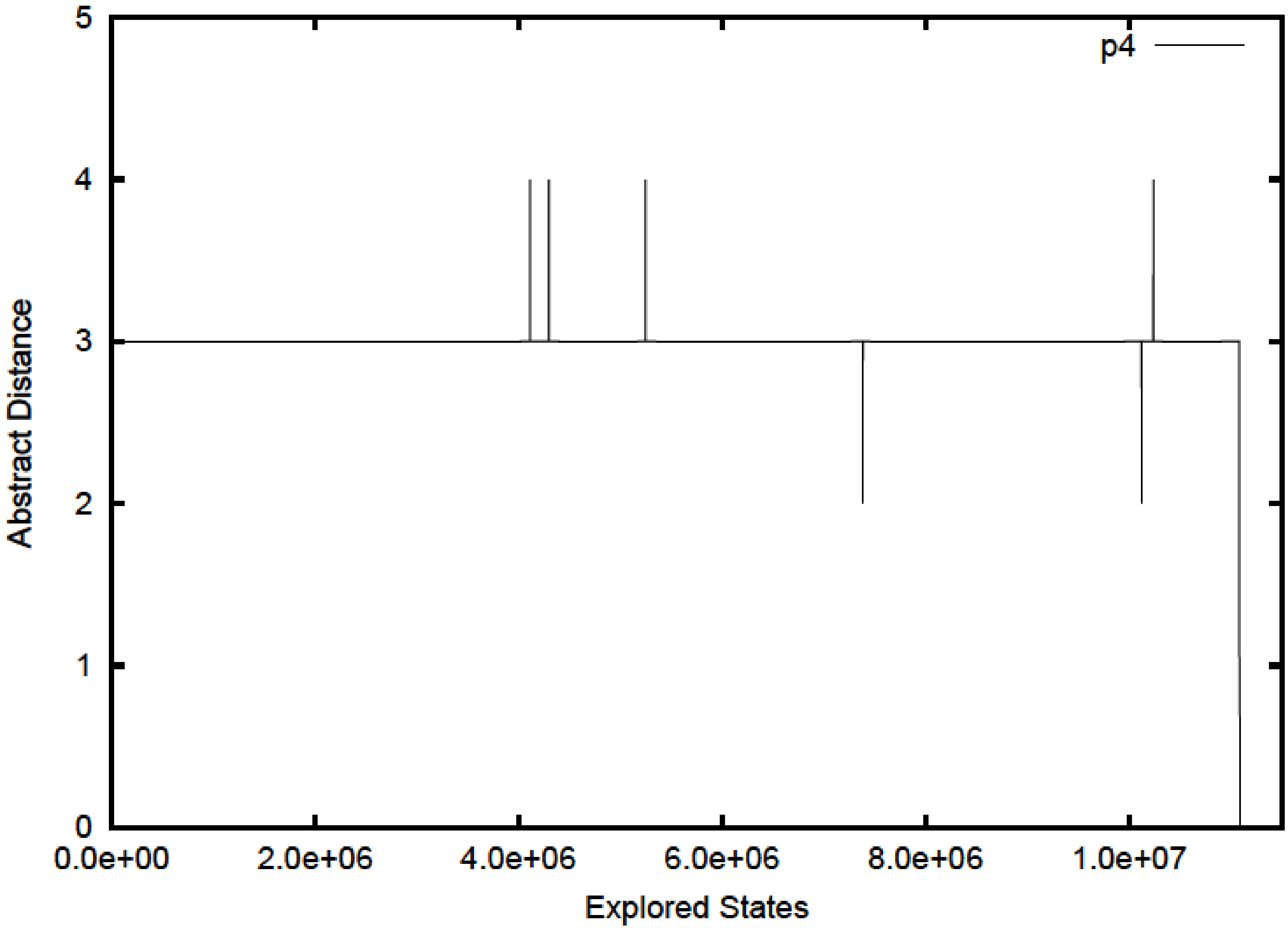
Simulation Trace: Expected Results



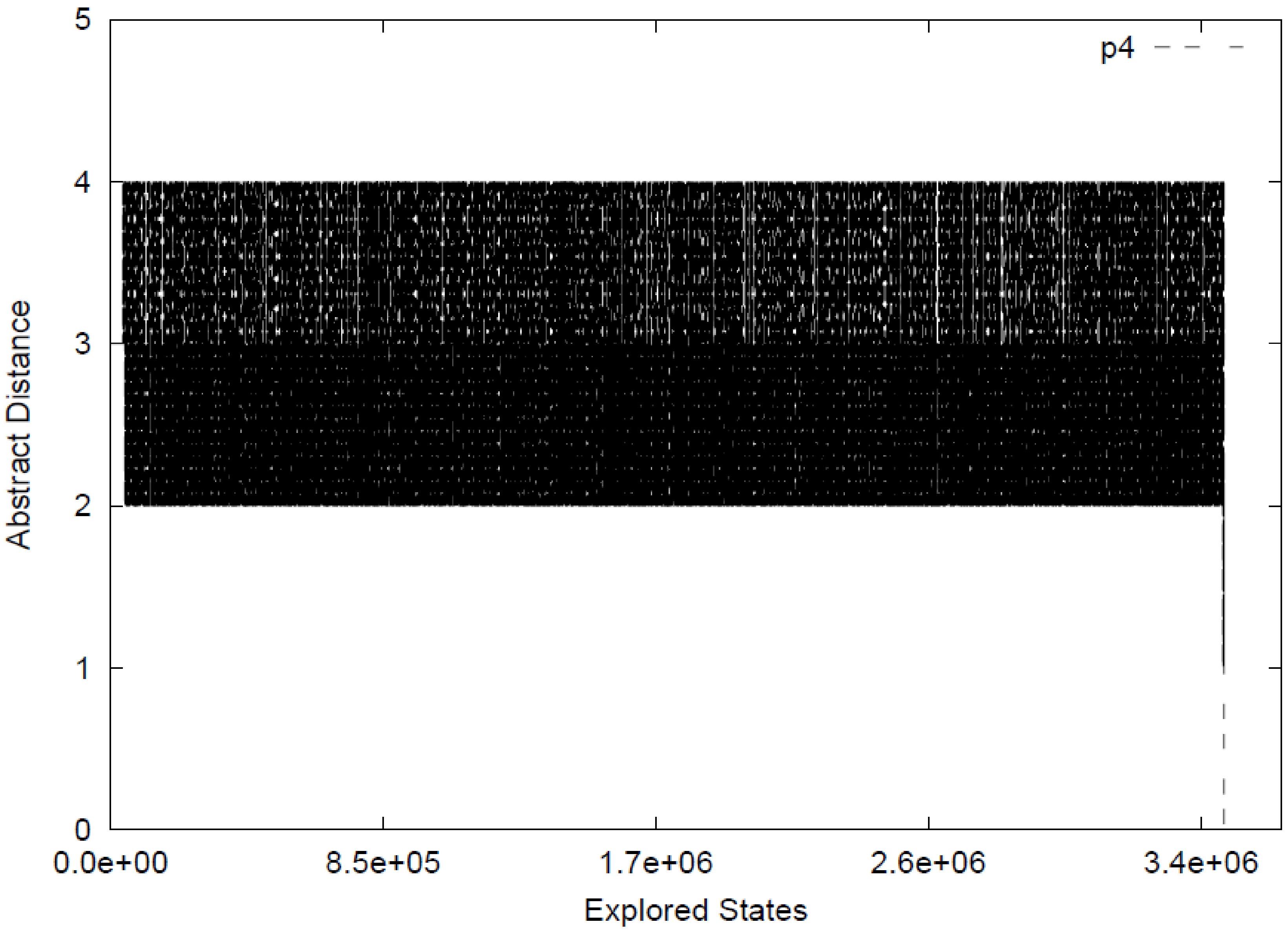
Simulation Trace: Expected Results



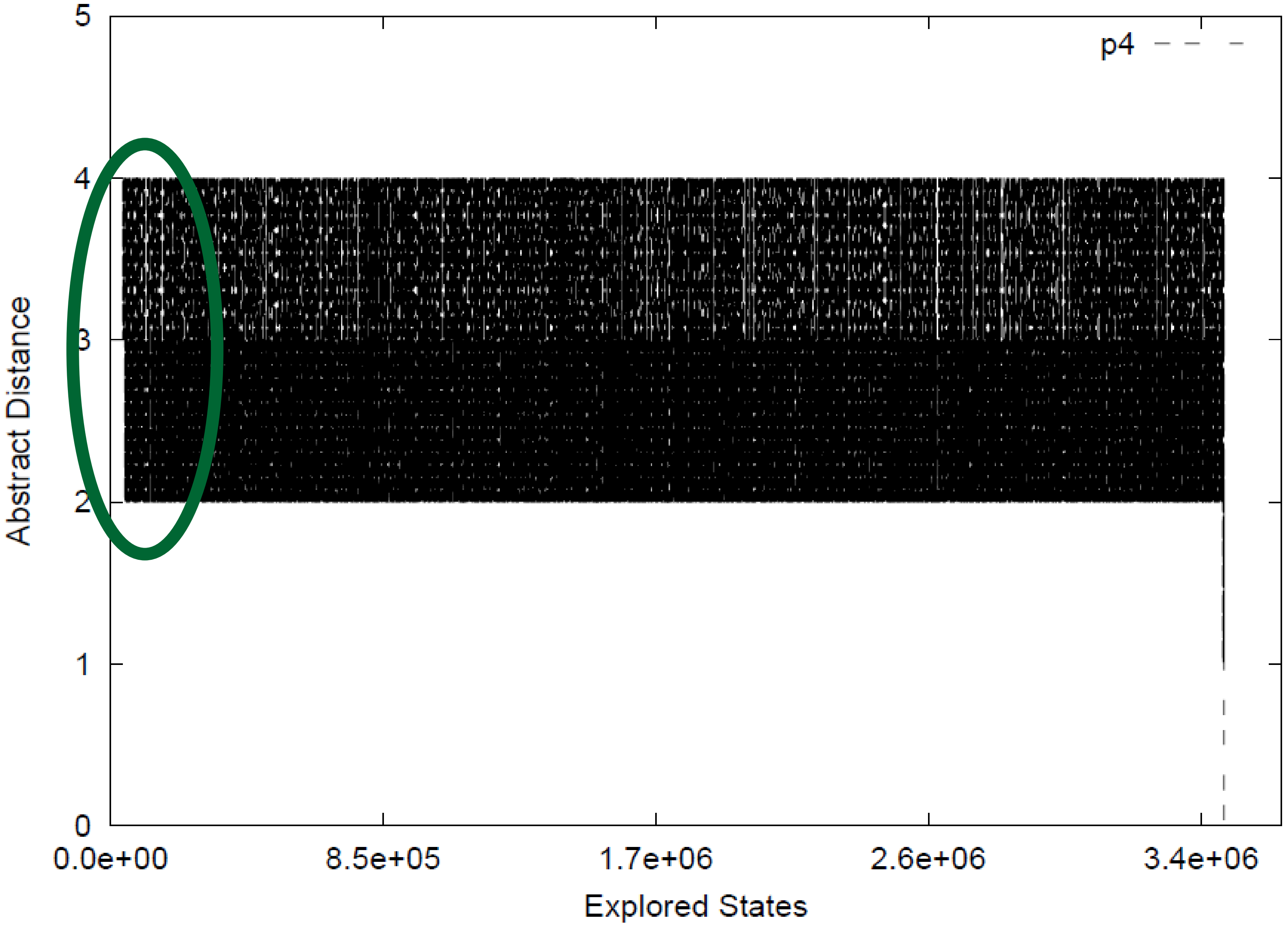
Hard Gains, Easy Losses



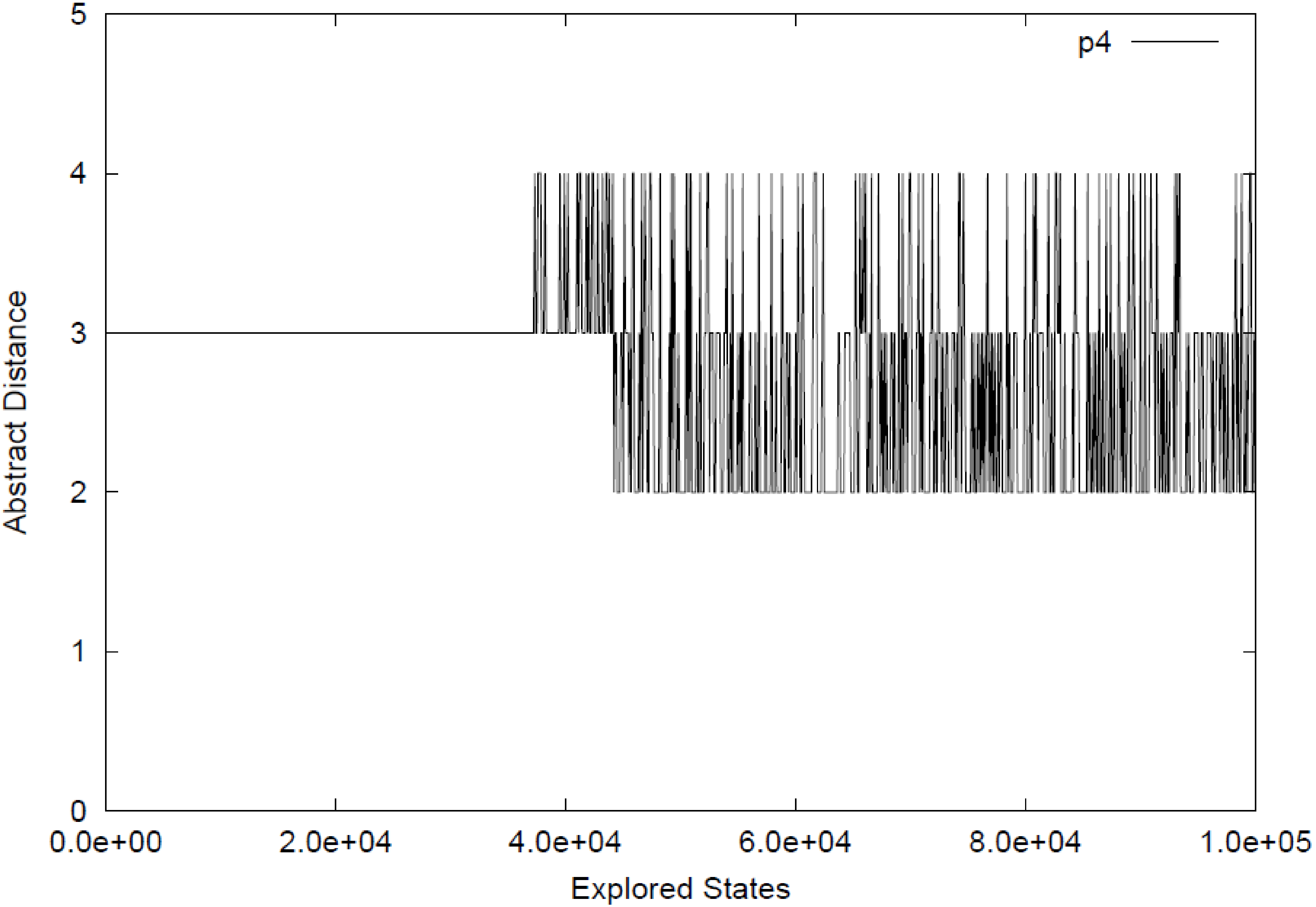
Simulation Trace for New



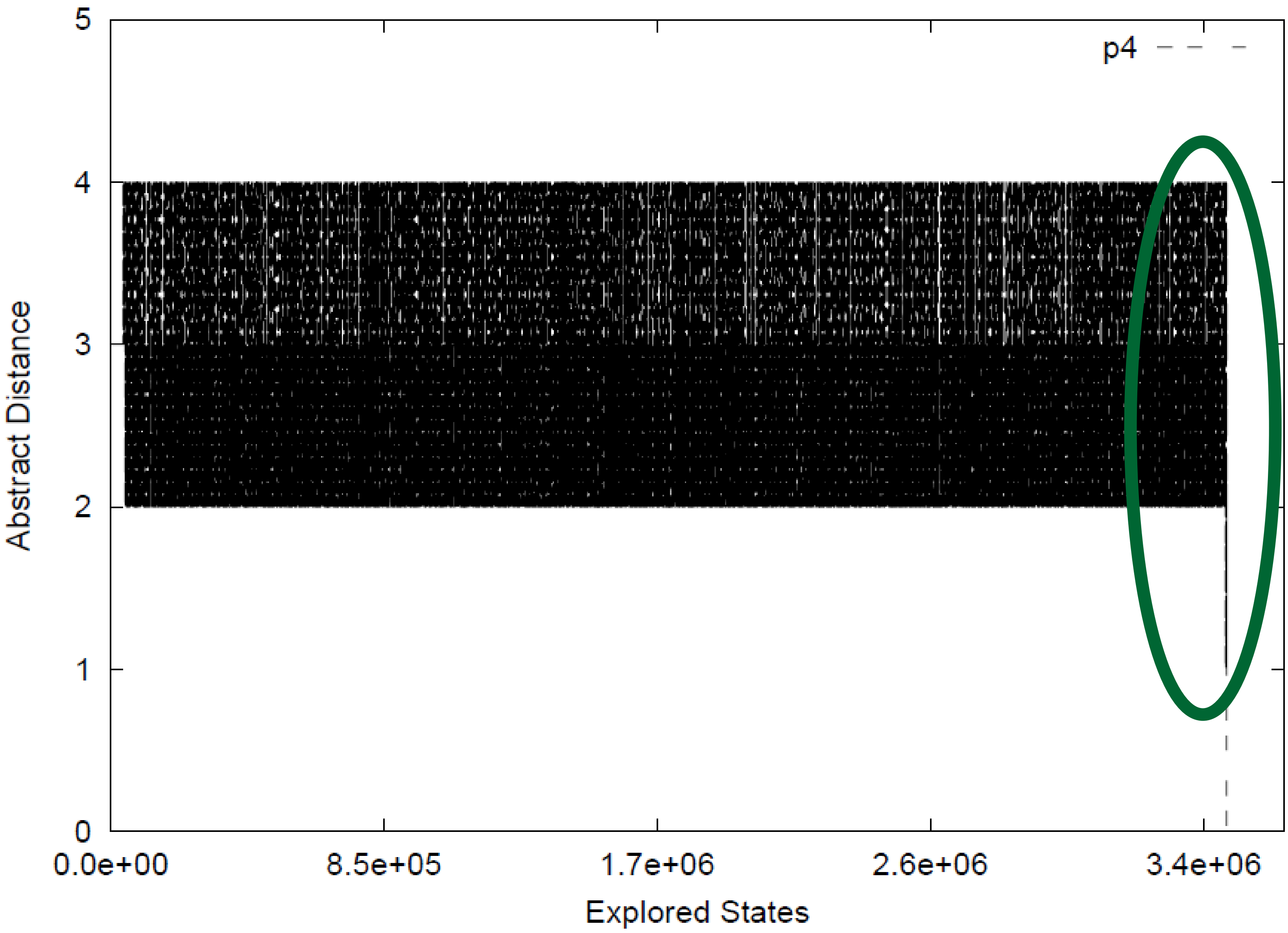
Simulation Trace for New



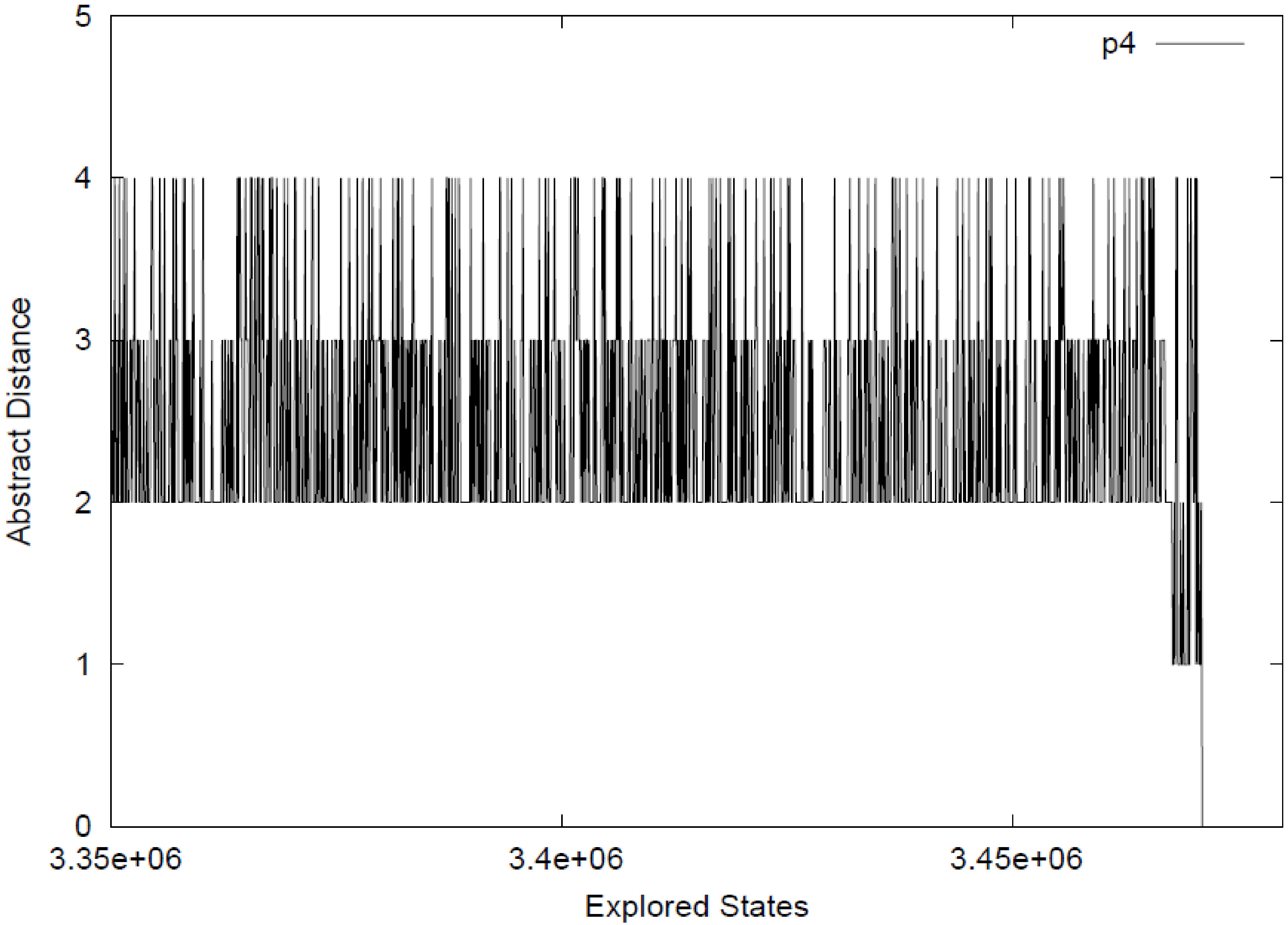
Enlarging Simulation Trace



Simulation Trace for New



Enlarging Simulation Trace



Random vs. Guided: p0

Run	Min (sec)	95% Confidence Interval			Max (sec)
		Low (sec)	Average (sec)	High (sec)	
R	27.5	656.8	1011.3	1365.8	3999.3
G	0.4	1.2	1.4	1.7	2.9

Random vs. Guided: p1

Run	Min (sec)	95% Confidence Interval			Max (sec)
		Low (sec)	Average (sec)	High (sec)	
R	106.8	2224.2	3,510.1	4,795.9	10885.5
G	150.8	4015.6	6,681.6	9,347.7	28865.0

Random vs. Guided: p2

Run	Min (sec)	95% Confidence Interval			Max (sec)
		Low (sec)	Average (sec)	High (sec)	
R	Timed Out (>100hrs) 22/22 Trials				
G	481	6,110	10,586	15,061	51,444

Random vs. Guided: p3

Run	Min (sec)	95% Confidence Interval			Max (sec)
		Low (sec)	Average (sec)	High (sec)	
R	Timed Out (> 150 hours) 16/16 Trials				
G	4,424	53,805	71,687	89,570	224,963

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction
- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don’t interpret.
- Metrics
- Domain Expertise

E.g., EverLost

Formal vs. Simulation

- Exhaustive analysis is useful!
  - Smart, brute force (BDDs, SAT, SMT, etc.)
  - Abstraction

- Machine-readable specifications.

- Scalability
  - Compiled code -- Execute. Don't interpret.
- Metrics
- Domain Expertise

E.g., Importance of handling imperative as well as declarative specifications?

Future?

- Absorbing useful ideas from other “styles” generates good results.
- Mix and match!

- E.g., I conjecture a strong underlying connection between abstraction and coverage.

Conclusion

- Train hard in your own style.
  Expertise and depth are your foundation.
- Cross-Train: Friendly sparring with other styles to illuminate your assumptions.
- Learn from other masters, too.
  - This doesn’t apply just to verification!

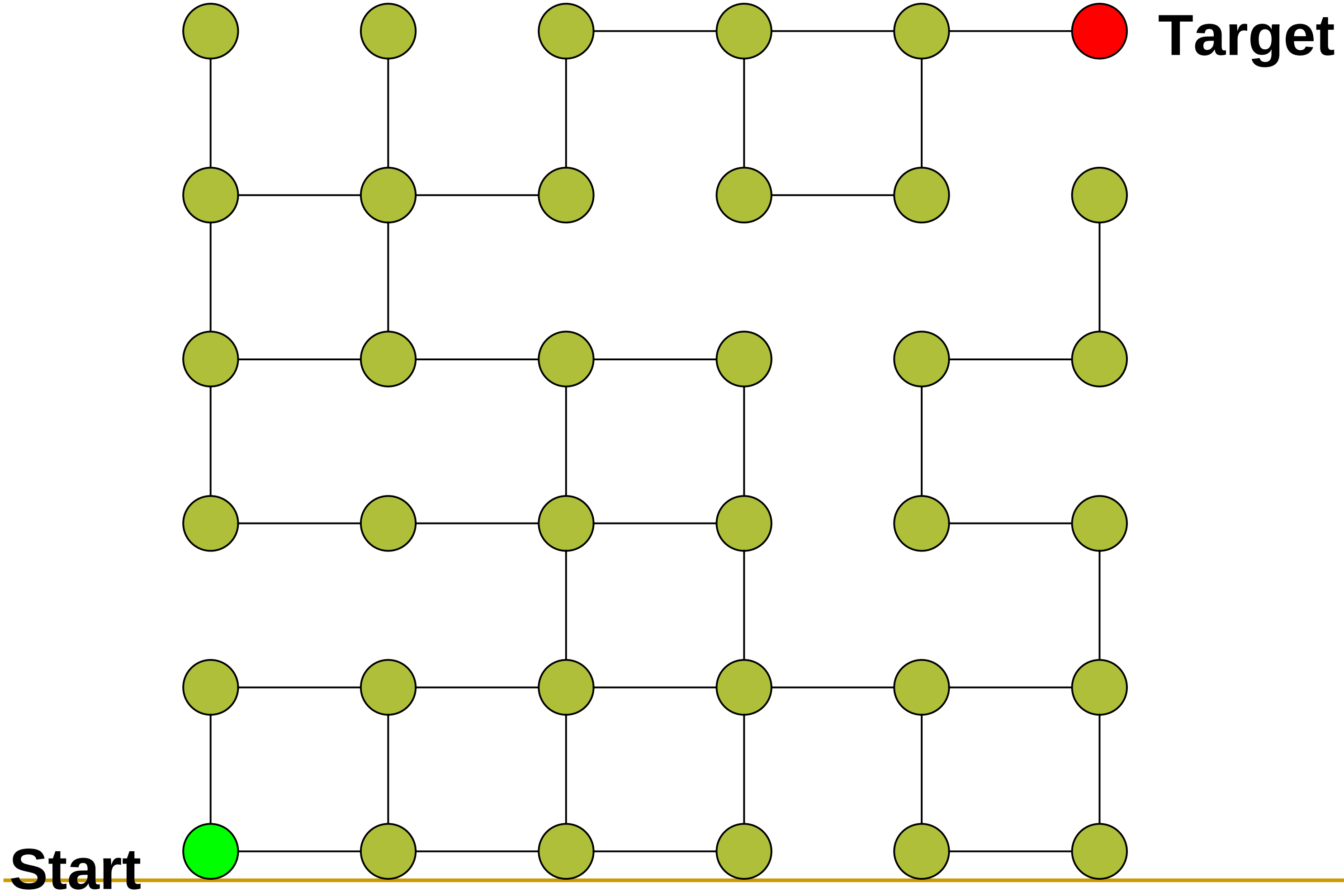
- You can become the Bruce Lee of verification!

Conclusion

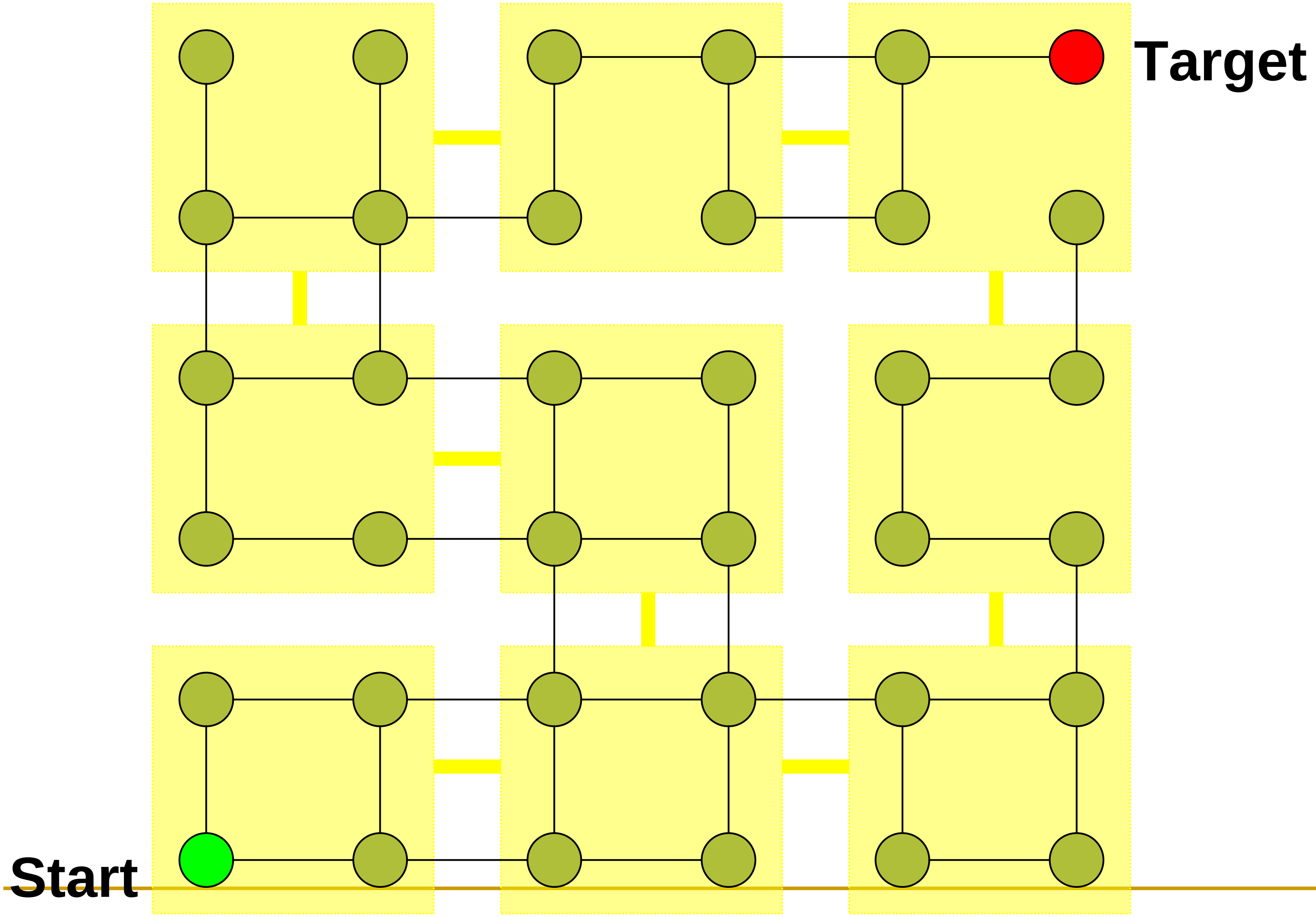
- Train hard in your own style.
  Expertise and depth are your foundation.
- Cross-Train: Friendly sparring with other styles to illuminate your assumptions.
- Learn from other masters, too.
  - This doesn't apply just to verification!

- You can become the Bruce Lee of verification!

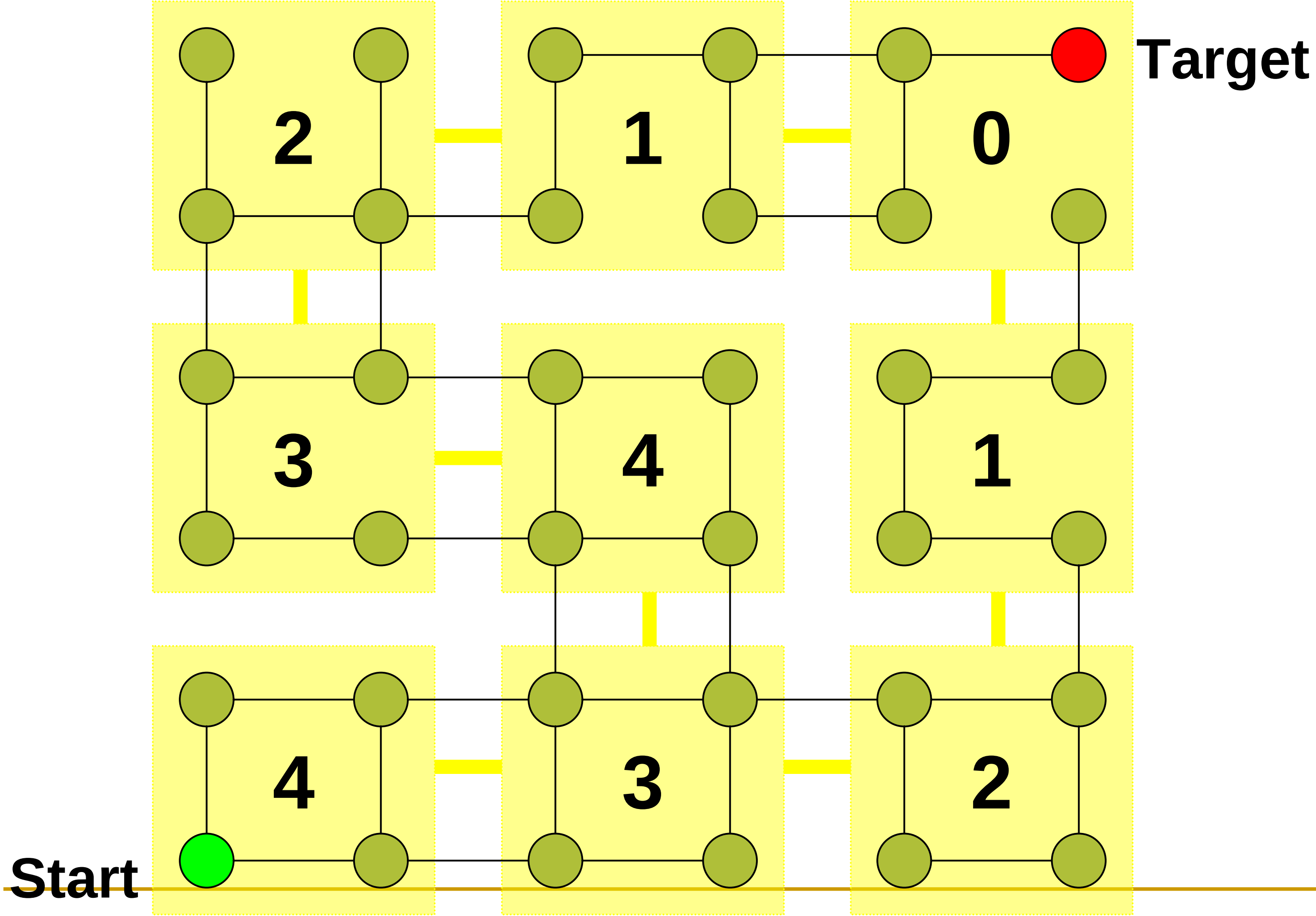
Simulation is search on concrete state space…



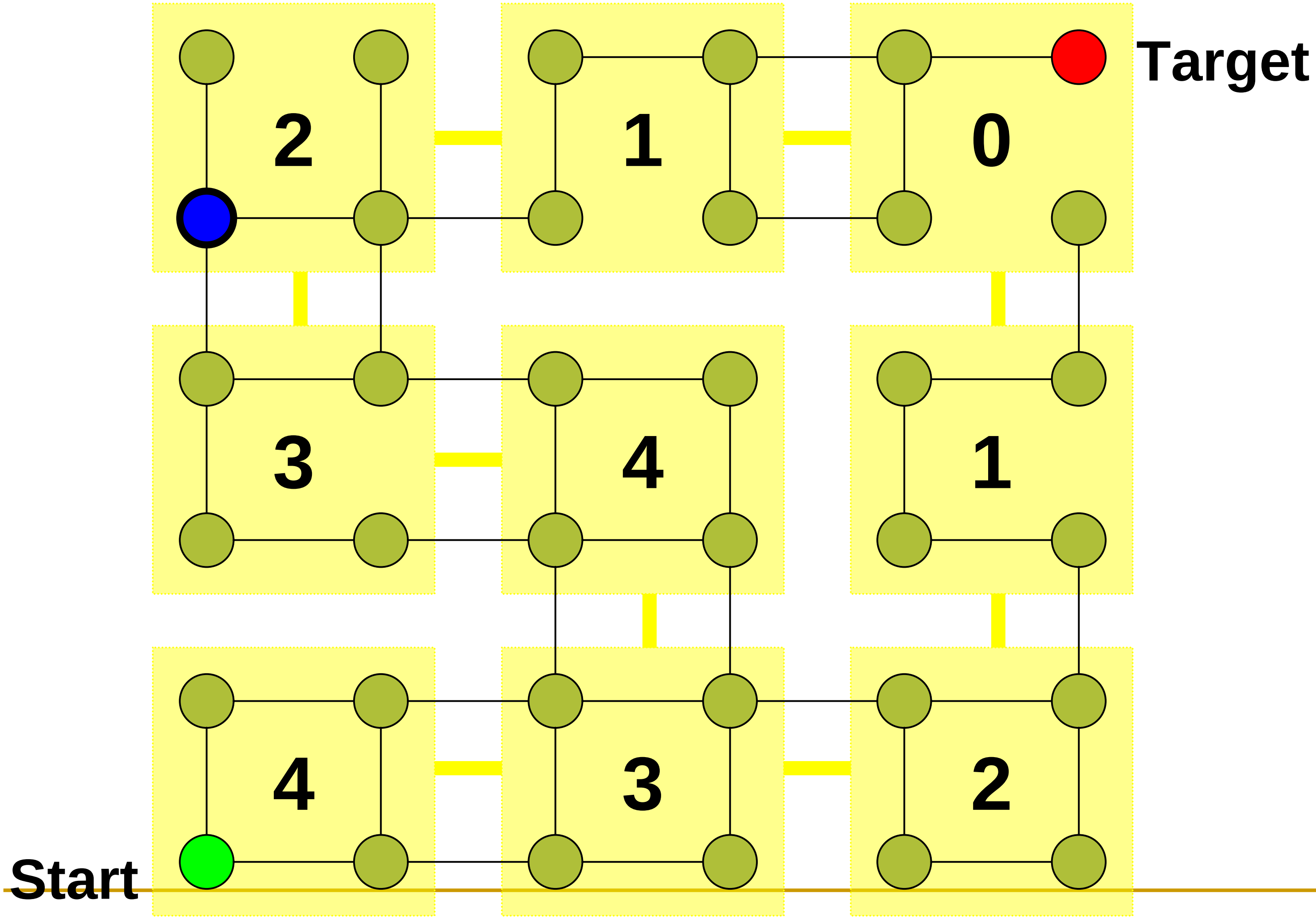
Abstraction groups states together, reducing state space by losing information…



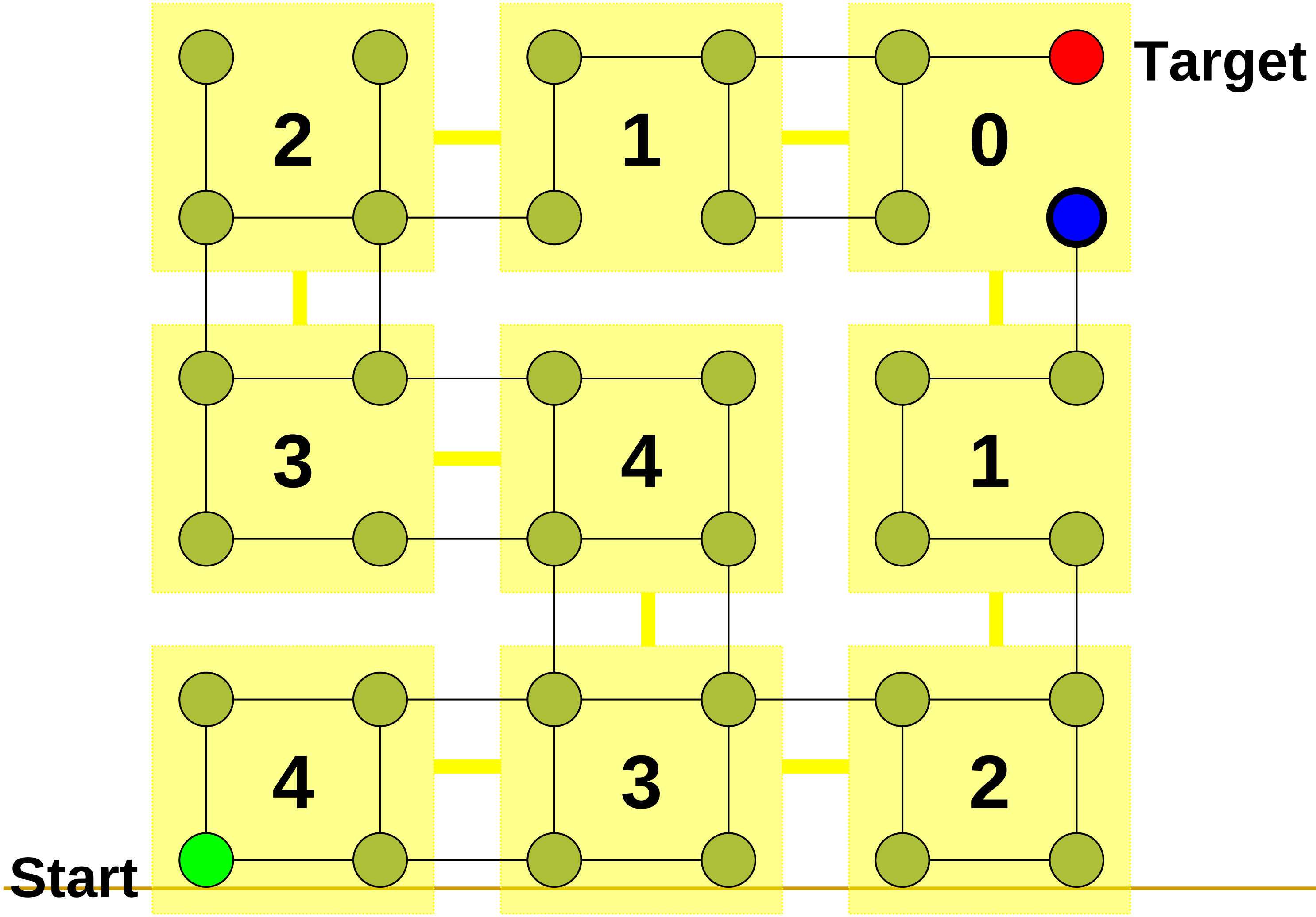
Abstract state space can be model checked. Abstract states can be ranked by abstract distance.



Use abstract distances to guide simulation…



Leaky Abstractions…


Target
2
1
0
3
4
1
4
3
2
Start