Toward Formally Verified Finance

Lessons from Model Checking

Quinn Dougherty

Casper Association

2024-05-07

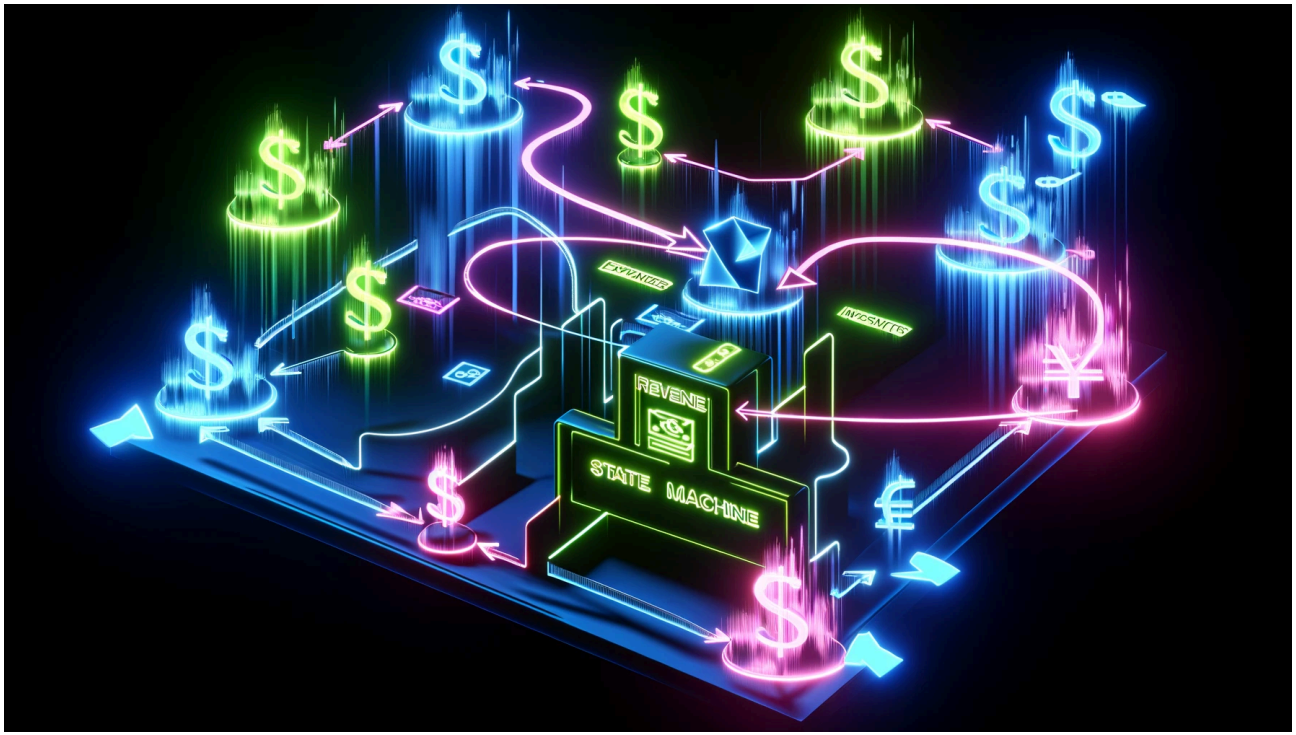
1. ACTUS
2. Logic and quality assurance
3. Logic and time
4. Automata
5. Model Checking
6. Conclusion

Overview

- ACTUS: finance down to state machine abstraction
- Formal verification: way better quality assurance
- Temporal logic: specify behaviors with time
- Automata: execution abiding by temporal logic specifications
- Reactive systems and model checking

Algorithmic Contract Types Unified Standard

ACTUS simulates *cash flows* with a *state machine* abstraction.



ACTUS state machine

Each contract comes equipped with

- $T : S \times E \to S$
- $P : E \to \mathbb{R}$

where T := Transition; P := Payoff; S := State; E := Event

ACTUS state machine

Each contract comes equipped with

- $T : S \times E \to S$
- $P : E \to \mathbb{R}$

where T := Transition; P := Payoff; S := State; E := Event

When you evaluate a state at an event e, a counterparty receives $P(e)$ payout (may be negative)

The study of argument structure

A logical formula is some propositions connected by the operators

- To "prove" a formula is to convince a skeptic that the formula is "true" or "valid"
- Formulae are types, proofs are programs

Modus ponens

If it is raining, then the ground is wet. It is raining. Therefore, the ground is wet

Modus tollens

If it is snowing, then it is cold outside. It is not cold outside. Therefore, it is not snowing

Modus ponens

$$(p \to q \land p) \to q$$

Modus tollens

$$(p \to q \land \neg q) \to \neg p$$

Formal Verification

Logic can set itself on any kind of mathematical object or phenomenon

When we use logic to study software, we're doing "formal verification"

Formal verification is kin with ordinary software testing, but much stronger

Providing steeper assurances

Baby testing: units

Come up with the cases you have time to enumerate

```
def is_even(x: int) -> bool:
  ...

assert is_even(2)
assert is_even(4)
```

Intermediate testing: properties

Procedurally generate unit tests (100-10000 cases)

```
def is_even(x: int) -> bool:
  if x == 2: return True
  elif x == 4: return True
  else: return False

@hypothesis.given(hypothesis.strategies.integers())
def is_even_agrees(x: int) -> bool:
  is_even(x) == (x % 2 == 0)
# Test fails with generated counterexample x = 6
```

Ascended testing: formal verification

A type checker in a total language can **exploit the structure** of datatypes and functions to prove over a whole type without exhaustively checking every value

Truth when?

We can make logical formulae specify properties involving time by introducing new operators called *modal operators*

- $\Box p$:= always p (p holds regardless of timestep)
- $\Diamond p$:= eventually p (there exists some timestamp later when p holds)
- $p U q$:= p until q (p holds continuously until q holds)

Traffic light

A traffic light should eventually turn green in all directions

A traffic light should never be green in all directions

Traffic light

A traffic light should eventually turn green in all directions

$$\Box\Diamond N \wedge \Box\Diamond S \wedge \Box\Diamond E \wedge \Box\Diamond W$$

A traffic light should never be green in all directions

$$\Box(N \wedge S \rightarrow \neg(E \vee W))$$

Temporal logic

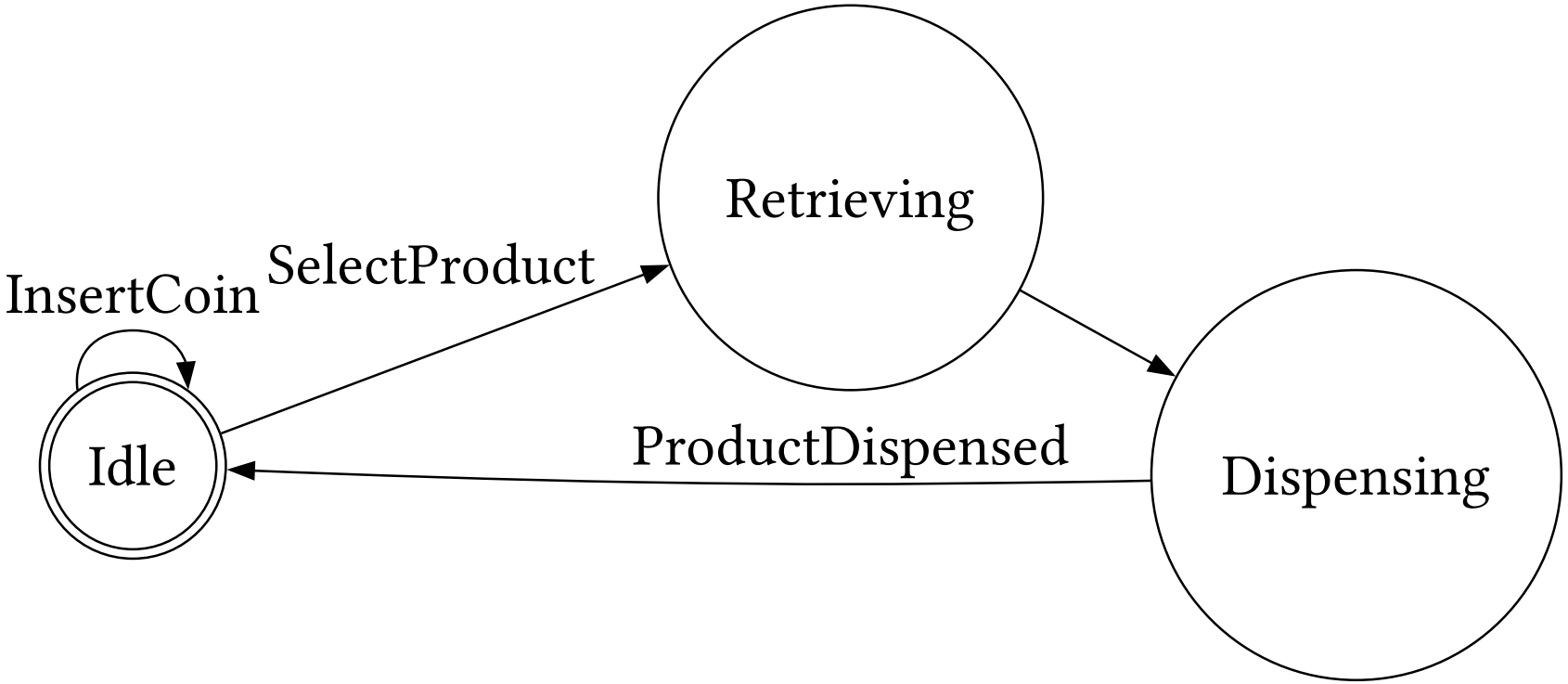
Capture what you care about

- Temporal logic **specifies properties** of computation over time.

Verify

- A **state machine** can be drawn from some implementation and **checked** that it matches a temporal logic **spec**.

Automata


Retrieving
InsertCoin
SelectProduct
Idle
ProductDispensed
Dispensing

Automata theory for finance

It turns out

ACTUS' notion of state machine is a special case of what's studied in *automata theory*

Automata theory for finance

It turns out

ACTUS' notion of state machine is a special case of what's studied in *automata theory*

Automata form semantics for temporal formulae

Automata theory for finance

It turns out

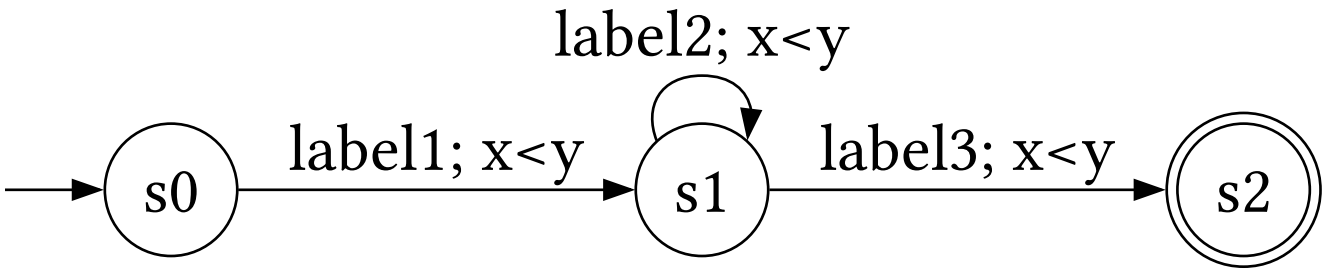
ACTUS' notion of state machine is a special case of what's studied in *automata theory*

Automata form semantics for temporal formulae

Automata provide the execution environment for model checking

Automata theory for finance

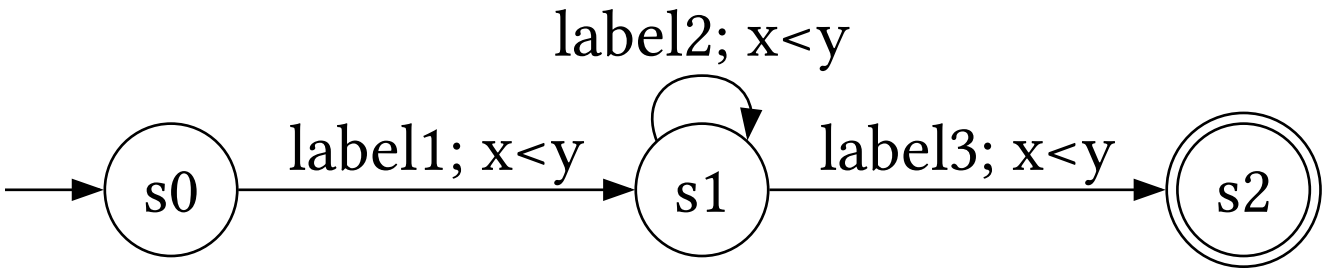
An automaton is an abstraction of computation consisting of states connected by events



Automata theory for finance

An automaton is an abstraction of computation consisting of states connected by events

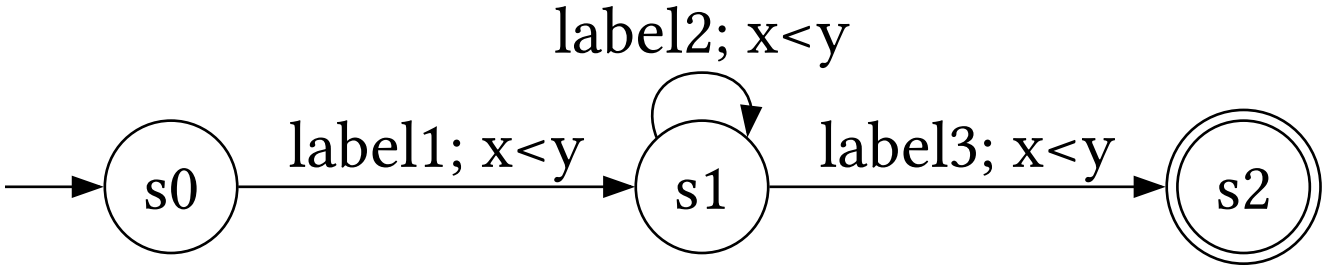
- In a *finite* automaton, some states are distinguished as “final”



Automata theory for finance

An automaton is an abstraction of computation consisting of states connected by events

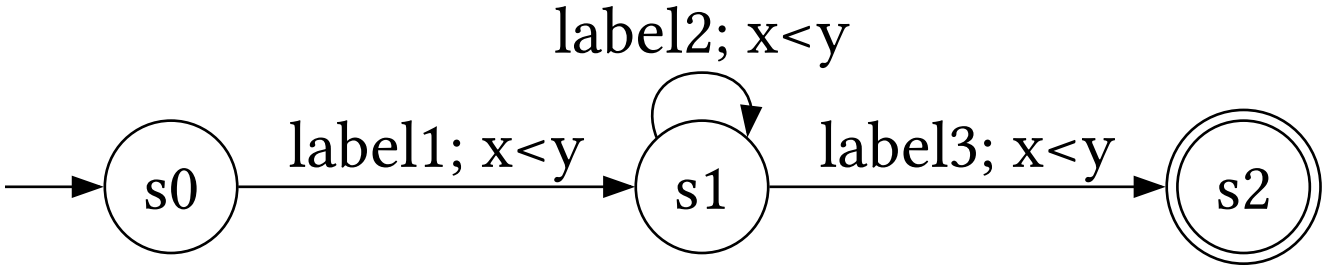
- In a *finite* automaton, some states are distinguished as "final"
- In a *timed* automaton, transitions (traversing along events) increment some "clocks", and events can only fire if "guard conditions" on those clocks are met



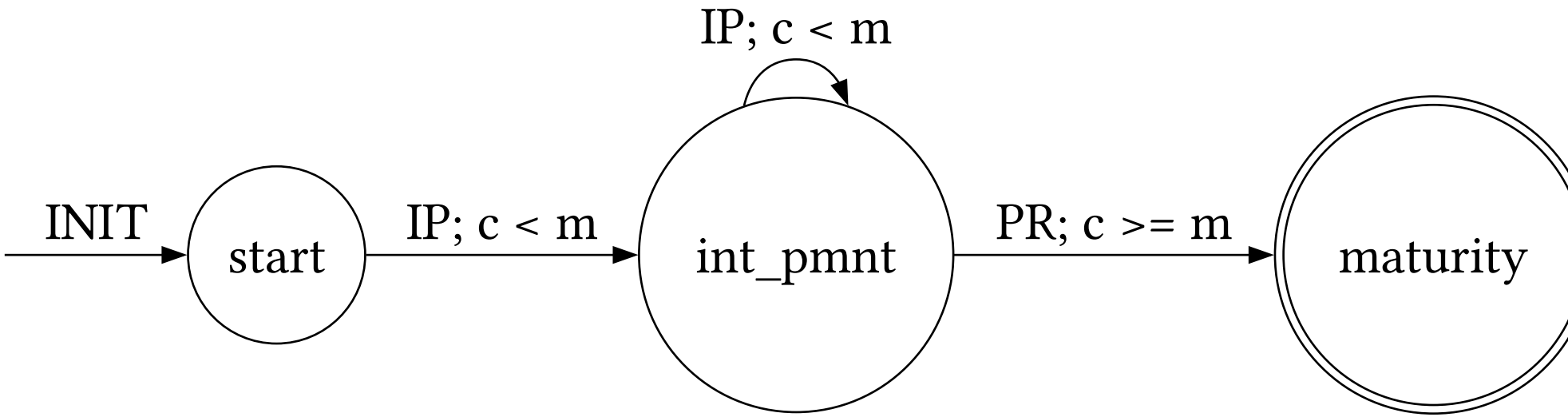
Automata theory for finance

An automaton is an abstraction of computation consisting of states connected by events

- In a *finite* automaton, some states are distinguished as "final"
- In a *timed* automaton, transitions (traversing along events) increment some "clocks", and events can only fire if "guard conditions" on those clocks are met
- In our case, the *guard conditions* are *event labels*

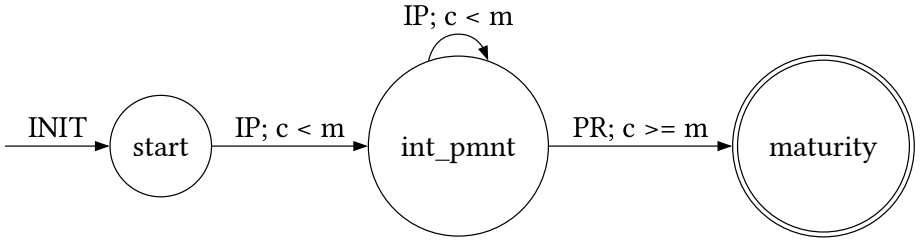


Principal at Maturity (PAM)



- c: *clock*
- m: *maturity date*
- IP: interest payment event
- PR: principal repayment event
- All events increment clock c by 1

Principal At Maturity (PAM)



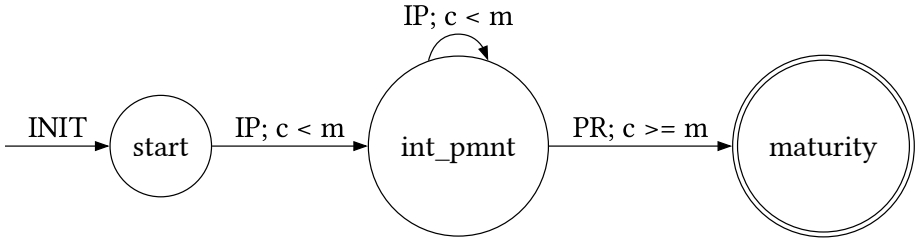
We can *run* PAM as a timed finite automaton to elicit a *trace*

The trace (for m = 2):

0. Start with empty trace

[]

Principal At Maturity (PAM)



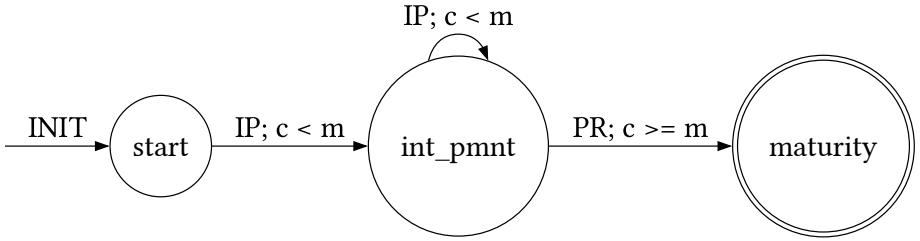
We can *run* PAM as a timed finite automaton to elicit a *trace*

The trace (for m = 2):

1. Enter contract at `start` state
   - c = 0
   - push `INIT` to trace

Result: [`INIT`]

Principal At Maturity (PAM)



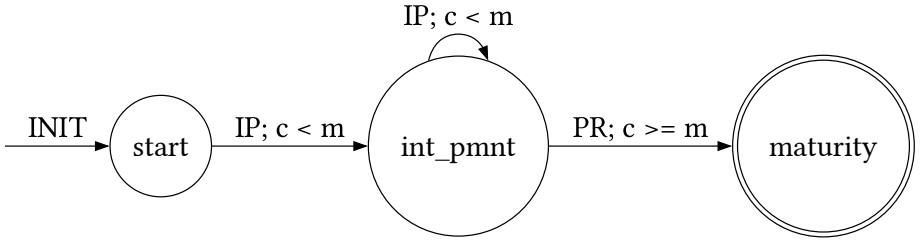
We can *run* PAM as a timed finite automaton to elicit a *trace*

The trace (for m = 2):

2. Apply an interest payment with the `IP` event, evaluating the guard $0 < 2$ to enter `int_pmnt` state
   - c = 1
   - push `IP` to trace

Result: [`INIT`, `IP`]

Principal At Maturity (PAM)



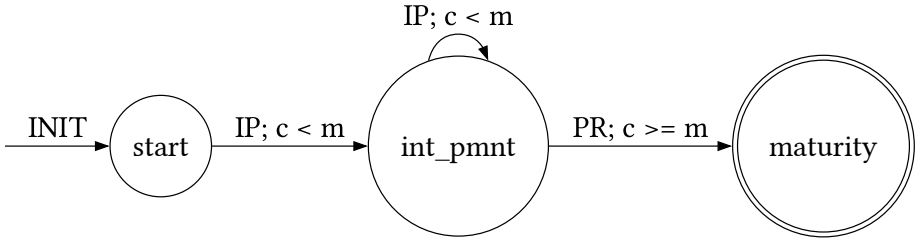
We can *run* PAM as a timed finite automaton to elicit a *trace*

The trace (for m = 2):

3. Apply an interest payment with the `IP` event, evaluating the guard $1 < 2$ to enter `int_pmnt` state
   - c = 2
   - push `IP` to trace

Result: [`INIT`, `IP`, `IP`]

Principal At Maturity (PAM)



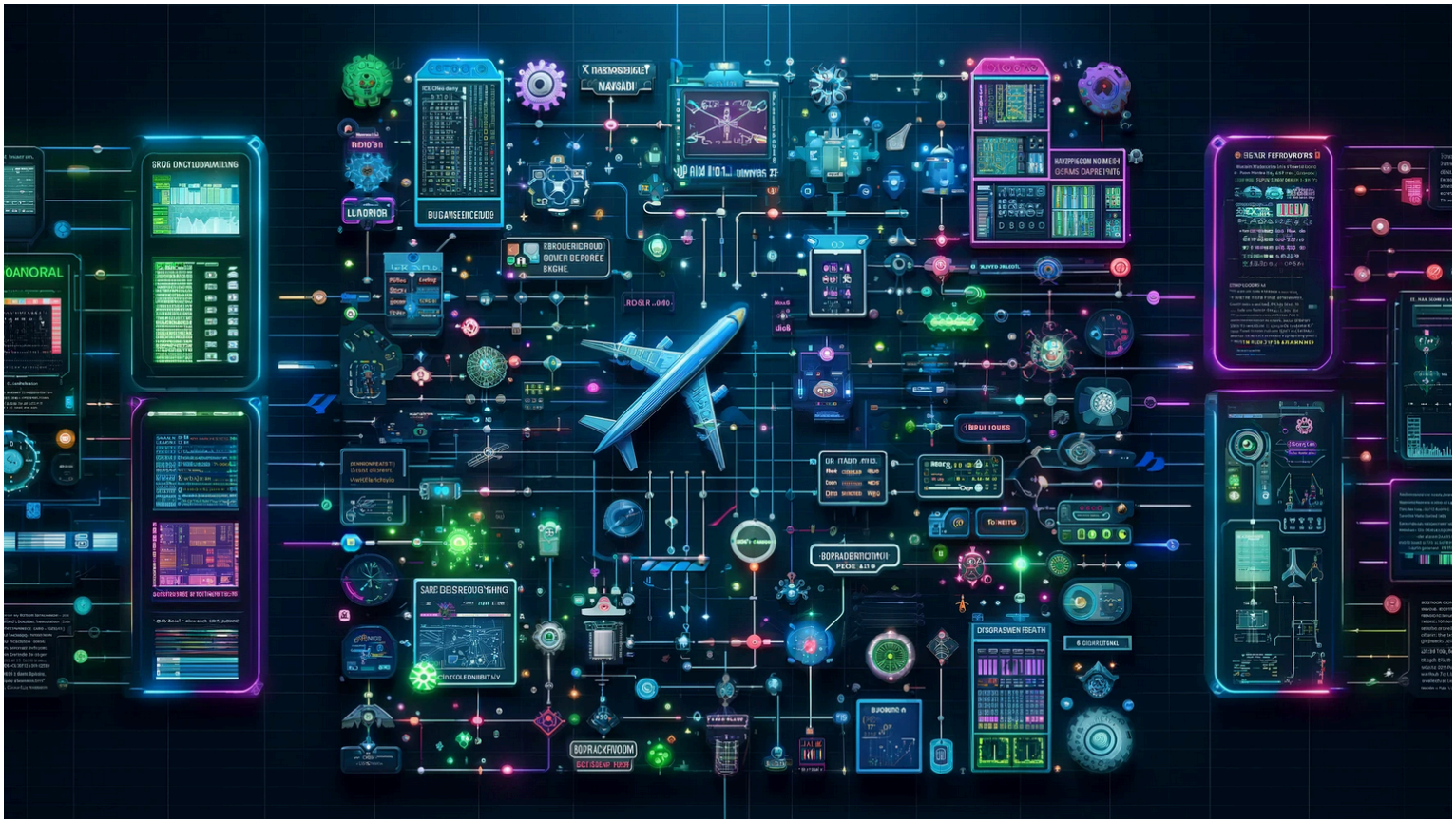
We can *run* PAM as a timed finite automaton to elicit a *trace*

The trace (for m = 2):

4. $2 \not< 2$, but $2 \geq 2$, so we take the `PR` (principal repayment) event instead, entering the `maturity` state which is final
   - c = 3 (doesn't matter)
   - push `PR` to trace

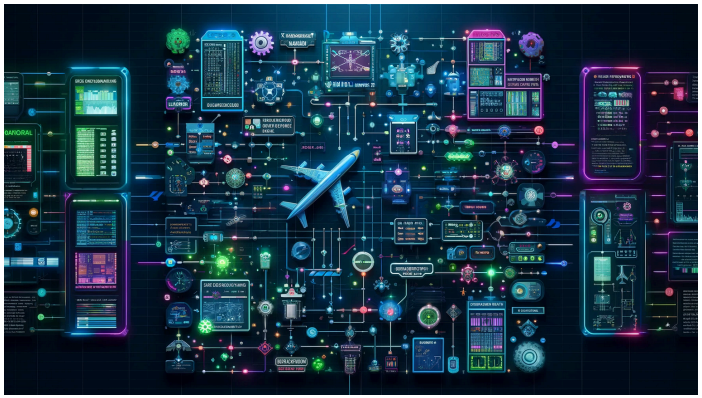
So the run produces trace: [`INIT`, `IP`, `IP`, `PR`]

Formal verification for reactive systems



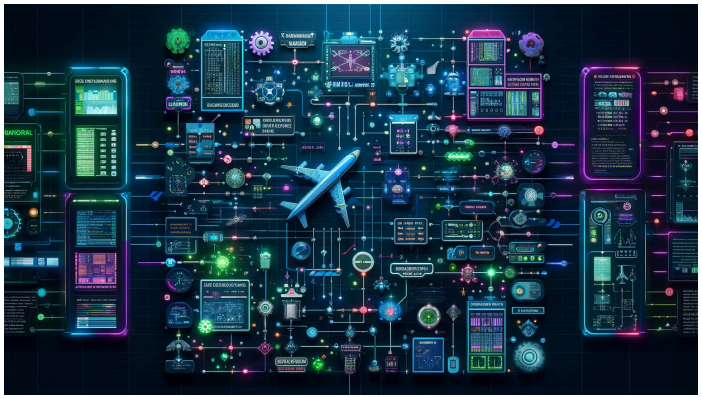
Reactive systems

- A **reactive system** is a software system embedded in an environment that responds to sensor input
  - often in continuous/infinite time horizon
  - often with actuator output effecting the environment



Reactive systems

- A **reactive system** is a software system embedded in an environment that responds to sensor input
  - often in continuous/infinite time horizon
  - often with actuator output effecting the environment
- Examples: traffic lights, airplane autopilot, fitness tracker on smartwatch, cruise control on a car



Formal verification for reactive systems

Recall that formal verification deals in mathematical proofs of software correctness

Formal verification for reactive systems

Recall that formal verification deals in mathematical proofs of software correctness

- A temporal logic forms a **specification language** in which normative constraints for reactive systems can be captured

Formal verification for reactive systems

Recall that formal verification deals in mathematical proofs of software correctness

- A temporal logic forms a **specification language** in which normative constraints for reactive systems can be captured
- **Model checking** is the discipline of turning programs into automata and showing that the automata is validated by a spec
  - Two key types of properties are *safety* properties and *liveness* properties

Safety

nothing bad ever happens

Liveness

something good eventually happens

Liveness property for PAM

Eventually maturity

$\Diamond$ Mat

Safety property for PAM

Maturity doesn't come before at least one interest payment

$\neg$ Mat U IP

Interest payments go continuously until maturity

$$\square(\text{IP} \rightarrow (\bigcirc\,(\text{IP} \vee \text{PR}))U\ \text{Mat})$$

PAM as a temporal logic spec

$$\Diamond \text{ Mat} \wedge \neg \text{ Mat } U \text{ IP} \wedge \Box(\text{IP} \rightarrow (\bigcirc (\text{IP} \vee \text{PR}))U \text{ Mat})$$

Conclusion

- ACTUS represents financial interactions as a state machine
- Logic, especially temporal logic, can specify behavioral correctness properties of state machines
- Formal verification leverages this as a quality assurance process
- Model checking is a rich literature, underappreciated in finance

Thanks

Quinn Dougherty

Casper Association

2024-05-07