A Global Specification Model for Data-Aware Coordination (*towards smarter smart contracts*)

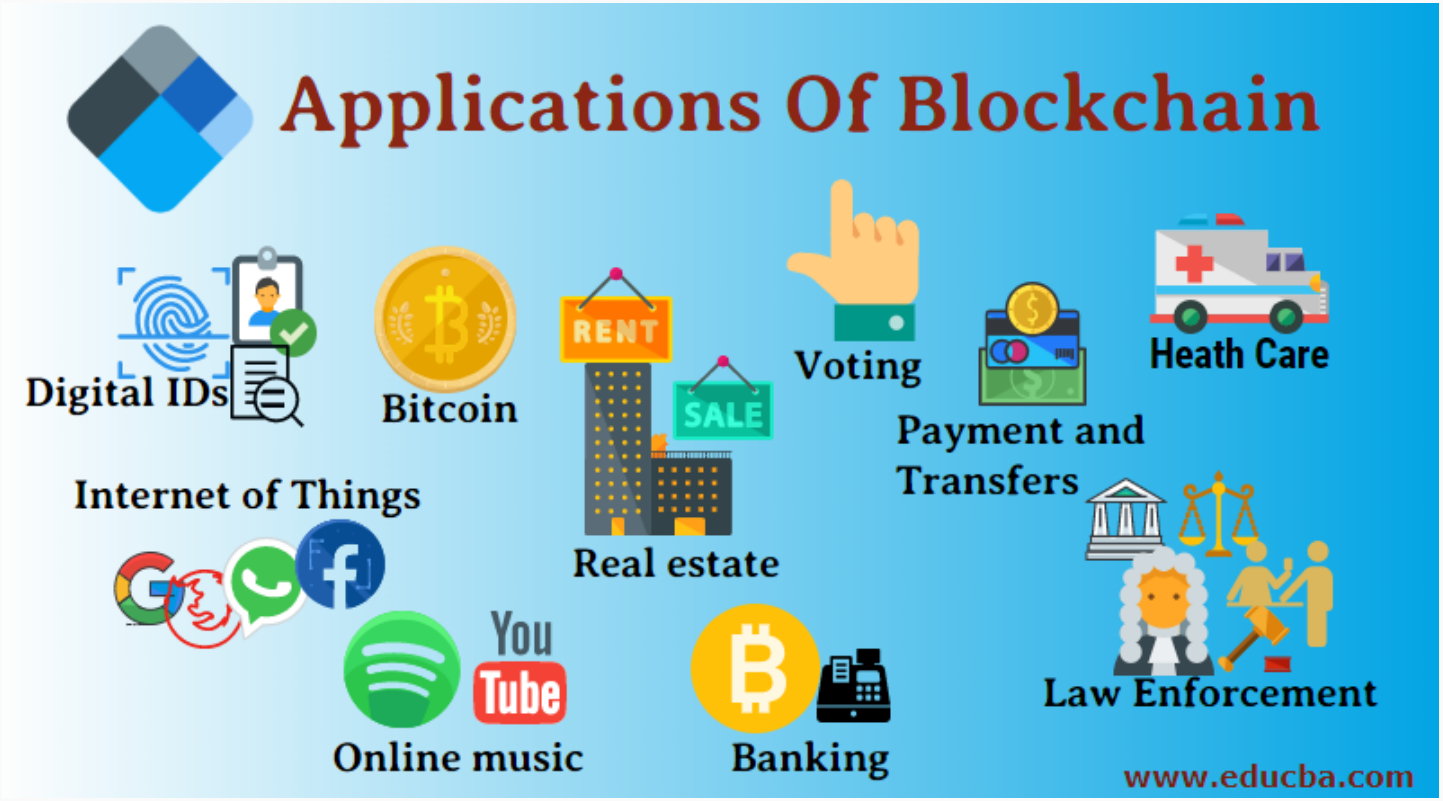
António Ravara (with Elvis K. Selabi, Maurizio Murgia, and Emilio Tuosto)
NOVA Laboratory of Computer Science and Informatics, Portugal

September 19, 2025

Context

The blockchain explosion

from cryptocurrencies to omni-present distributed apps

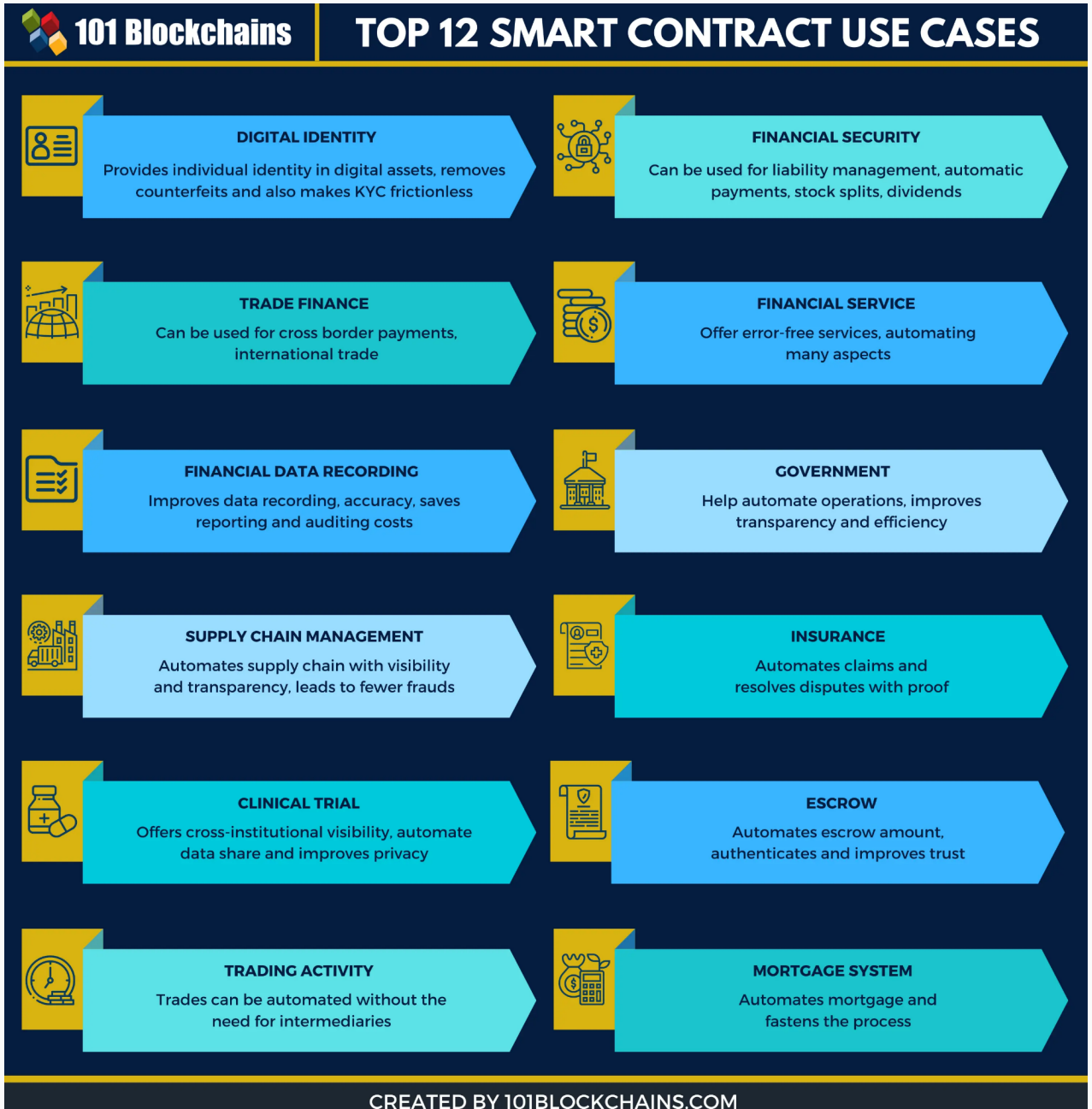


blockchain as a global computer

Services available via Contracts

- Scripts ready to run on real-time on a blockchain
- Eliminate central authorities – fully distributed peer-to-peer
- All parties accountable

Smart Contracts everywhere



Problem

Exploitable vulnerabilities

Code is law, so bugs are features

Nitesh Dhanjani
Jan 24 · 12 min read

Smart Contract Attacks: Hundred Million Dollar Heists, Rug-pullers, Front Runners, NFT Snipers, and Uninformed Auditors

The nefarious tactics being employed in the crypto universe are a useful study because they shed a light on the current state of risk and what needs to be improved in terms of trust. This write-up covers some recent security incidents, the analysis of root causes, and helps set the stage for an understanding of what's to come in terms of the players involved and their future incentives.

$31 Million USD Stolen Because Someone Forgot an if Statement

On November 30, 2021, $31 Million USD was stolen from the MonoX Protocol smart contracts deployed on the Ethereum and Polygon network.

Business

Badger DAO Protocol Suffers $120M Exploit

The hacker or hackers may have targeted the platform's user interface.

By Andrew Thurman · Dec 2, 2021 at 4:51 a.m. GMT · Updated Dec 2, 2021 at 2:40 p.m. GMT ·

Sep 4, 2021, 06:02am EDT

They're Not Smart And They're Not Contracts

David G.W. Birch Contributor
Fintech
Author, advisor and global commentator on digital financial services.

But they are the building blocks of a new financial infrastructure.

Goal

The case for generating correct-by-construction code

Turing award in 1972: "The humble programmer"

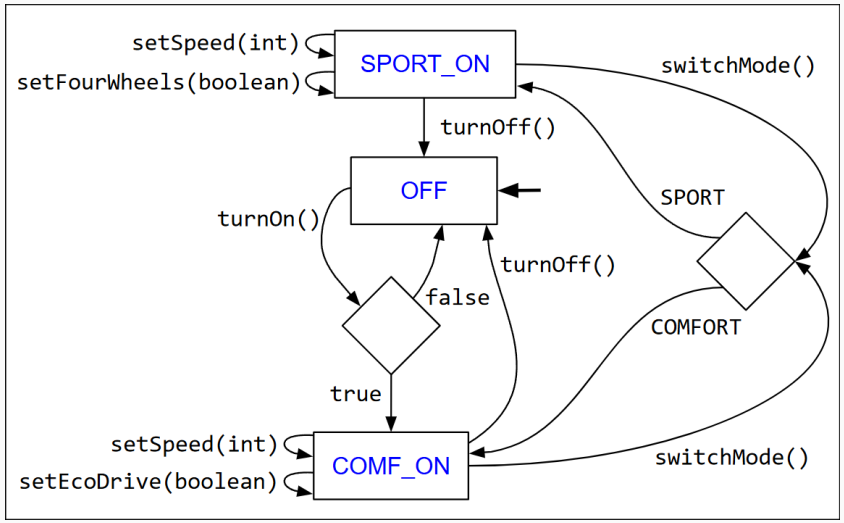
"If debugging is the process of removing bugs from the code, programming must be the process of putting them there!"

The Verified Software Initiative

"We envision a world in which computer programs are always the most reliable component of any system or device that contains them" [Hoare & Misra]

"We propose an ambitious and long-term research program toward the construction of error-free software systems."

Our (modest) goal: from models to code




```
typestate SUV {
  OFF = {
    boolean turnOn():
      <true:COMF_ON, false:OFF>,
    drop: end
  }

  COMF_ON = {
    void turnOff(): OFF,
    void setSpeed(int): COMF_ON,
    Mode switchMode():
      <SPORT:SPORT_ON,COMFORT:COMF_ON>,
    void setEcoDrive(boolean): COMF_ON
  }

  SPORT_ON = {
    void turnOff(): OFF,
    void setSpeed(int): SPORT_ON,
    Mode switchMode():
      <SPORT:SPORT_ON,COMFORT:COMF_ON>,
    void setFourWheels(boolean): SPORT_ON
  }
}
```

Contribute with modelling tools and suitable programming abstractions

- State-Machines via Lightweight (graphical) annotations
- Sound code generation from scribbled specifications
- Verification of compliance *at compile-time* when the developer changes the code directly

How difficult is the situation now?

December 2, 2021:

Really Stupid 'Smart Contract' Bug Let Hackers Steal $31 Million In Digital Coin

User could send tokens to themselves and increase their balance!

```
// swap from tokenIn to tokenOut with fixed tokenIn amount.
function swapIn (address tokenIn, address tokenOut, address from, address to,
    uint256 amountIn) internal lockToken(tokenIn) returns(uint256 amountOut)  {
```

Someone forgot an if statement: *tokenIn != tokenOut*

How difficult is the situation now?

December 2, 2021:

Really Stupid 'Smart Contract' Bug Let Hackers Steal $31 Million In Digital Coin

User could send tokens to themselves and increase their balance!

```
// swap from tokenIn to tokenOut with fixed tokenIn amount.
function swapIn (address tokenIn, address tokenOut, address from, address to,
    uint256 amountIn) internal lockToken(tokenIn) returns(uint256 amountOut)  {
```

Someone forgot an if statement: *tokenIn != tokenOut*

"Someone forgot an if statement" – really?!

Isn't it a problem of lack of abstractions?

How difficult is the situation now?

December 2, 2021:

Really Stupid 'Smart Contract' Bug Let Hackers Steal $31 Million In Digital Coin

User could send tokens to themselves and increase their balance!

```
// swap from tokenIn to tokenOut with fixed tokenIn amount.
function swapIn (address tokenIn, address tokenOut, address from, address to,
    uint256 amountIn) internal lockToken(tokenIn) returns(uint256 amountOut)  {
```

Someone forgot an if statement: *tokenIn != tokenOut*

"Someone forgot an if statement" – really?!

Isn't it a problem of lack of abstractions?

Sender and receiver should be different roles,
with different permissions

What's out there that can help?

On Smart Contracts and State Machines

From the Solidity documentation

State Machine

Contracts often act as a state machine, which means that they have certain **stages** in which they behave differently or in which different functions can be called. A function call often ends a stage and transitions the contract into the next stage (especially if the contract models **interaction**). It is also common that some stages are automatically reached at a certain point in **time**.

On Smart Contracts and State Machines

From the Solidity documentation

State Machine

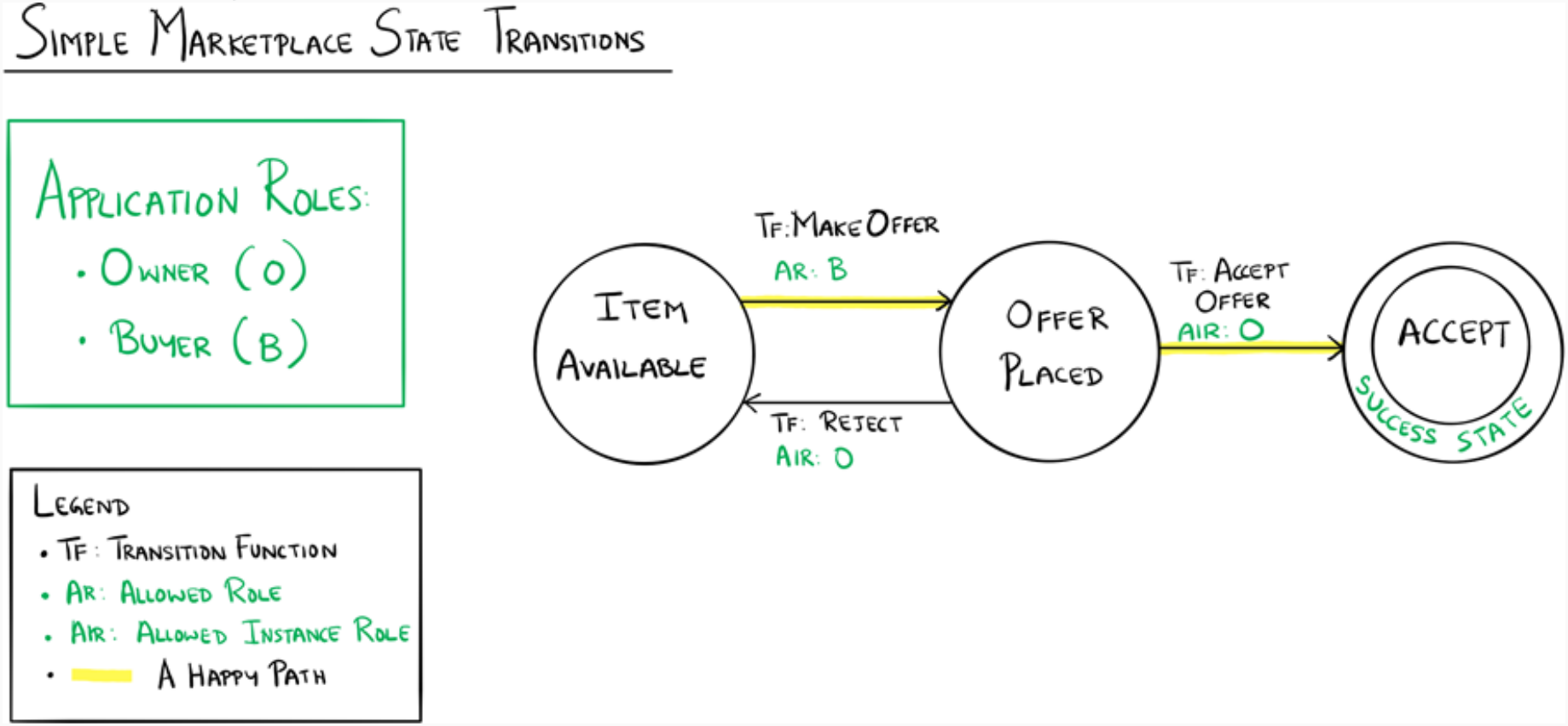
Contracts often act as a state machine, which means that they have certain **stages** in which they behave differently or in which different functions can be called. A function call often ends a stage and transitions the contract into the next stage (especially if the contract models **interaction**). It is also common that some stages are automatically reached at a certain point in **time**.

So, a smart contract looks like

- a choreographic model
  global specification determining enabled actions along the protocol
- a typestate (declares non-uniform component behaviour)
  "reflects how the legal operations ... can change at runtime as their internal state changes."

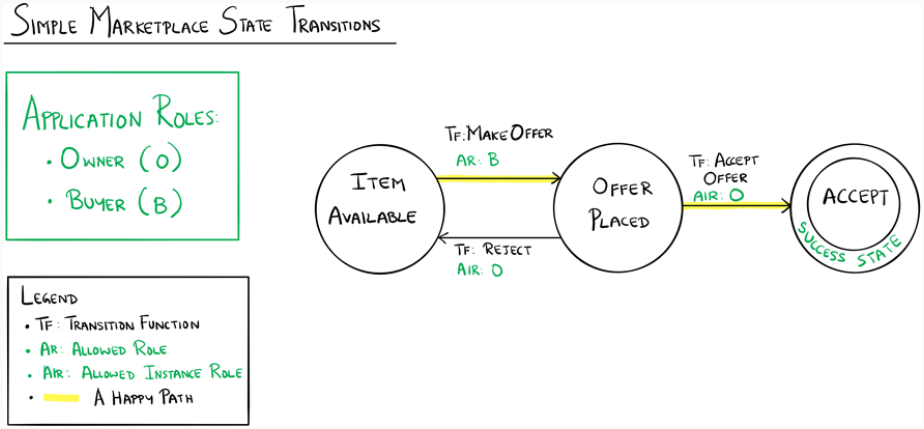
From the Microsoft Azure Blockchain Workbench

No attempt to be formal...



https://github.com/Azure-Samples/blockchain/blob/master/blockchain-workbench

From the Microsoft Azure Blockchain Workbench




```
function AcceptOffer() public {
    if ( msg.sender != InstanceOwner ) { revert(); }
    State = StateType.Accepted;
}
```

Code snippet is bugged: AcceptOffer does not check the state

Our proposal: Data-Aware Finite State Machines

Ingredients for a model to cope with the scenario

a global specification to

- coordinate distributed components
- declare how actions are enabled along the computation
- not force component cooperation

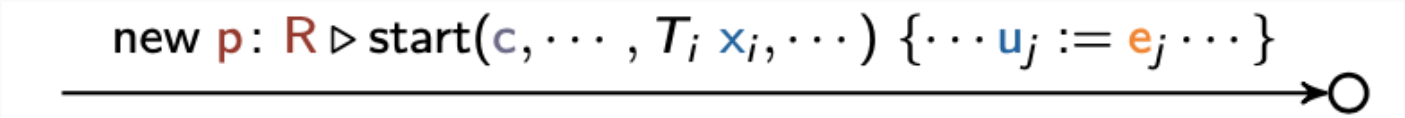
Ingredients for a model to cope with the scenario

a global specification to

- coordinate distributed components
- declare how actions are enabled along the computation
- not force component cooperation

A Data-Aware FSM (DAFSM) c

on state variables $u_1, \ldots, u_n$ is deployed by participant p:



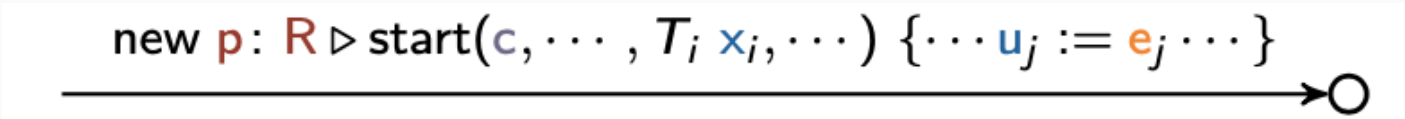
Ingredients for a model to cope with the scenario

a global specification to

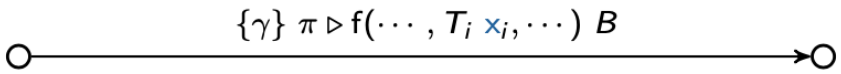
- coordinate distributed components
- declare how actions are enabled along the computation
- not force component cooperation

A Data-Aware FSM (DAFSM) c

on state variables $u_1, \ldots, u_n$ is deployed by participant p:



and has transitions like



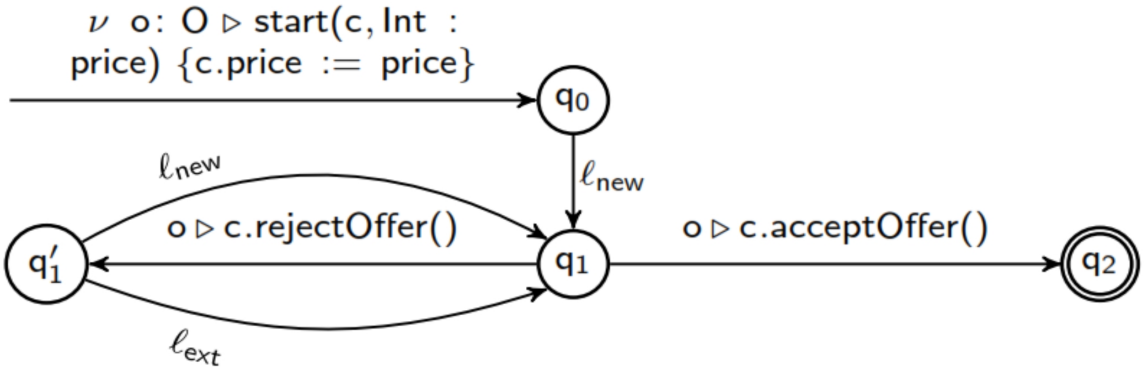
where γ is a guard (ie a boolean expression) and

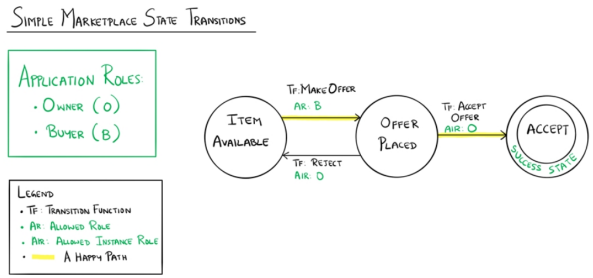
π ::= new p: R | any p: R | p

is a qualified participant calling f with parameters x_i

state variables are reassigned according to B if the invocation is successful

Back to the Azure Workbench example





$$\ell_{\text{new}} = \{\text{offer} > 0\}\ \nu\ \text{b}\colon \text{B} \triangleright \text{c.makeOffer}(\text{Int} : \text{offer})\ \{\text{c.offer} := \text{offer}\}$$

$$\ell_{\text{ext}} = \{\text{offer} > 0\}\ \text{any b}\colon \text{B} \triangleright \text{c.makeOffer}(\text{Int} : \text{offer})\ \{\text{c.offer} := \text{offer}\}$$

Can we tackle all the Azure Workbench examples?

	ICI	BI	PP	RR	MPR
Hello Blockchain	⊖	✓	⊖	⊖	⊖
Simple Marketplace	⊖	✓	⊖	↗	⊖
Bazaar	x	✓	⊖	⊖	⊖
Ping Pong	x	✓	⊖	⊖	⊖
Defective Component Counter	⊖	✓	✓	⊖	⊖
Frequent Flyer Rewards Calculator	⊖	✓	✓	⊖	⊖
Room Thermostat	⊖	⊖	✓	⊖	⊖
Asset Transfer	⊖	✓	✓	↗	⊖
Basic Provenance	⊖	✓	✓	↗	⊖
Refrigerated Transport	⊖	✓	✓	↗	↗
Digital Locker	⊖	✓	✓	↗	↗

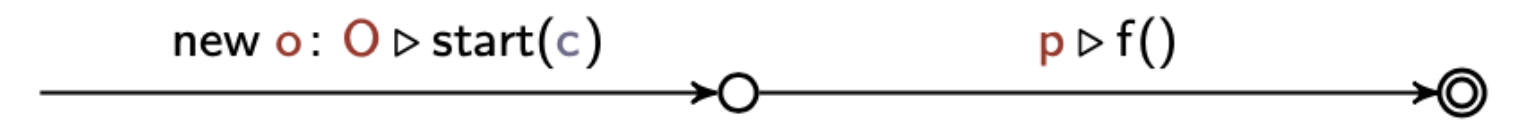
ICI Inter-contracts interactions

BI and PP New participants By-Invocation or Parameter Passing

RR and MPR Role Revocation and Multiple Participant Roles

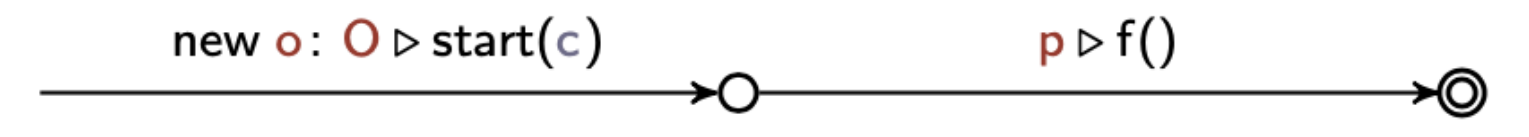
Not all DAFSMs make sense

Names' freeness

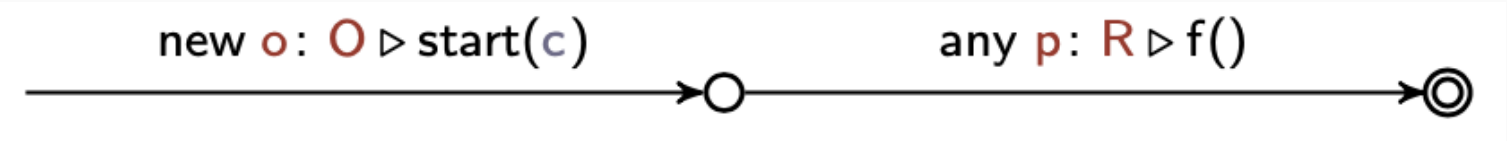


Not all DAFSMs make sense

Names' freeness

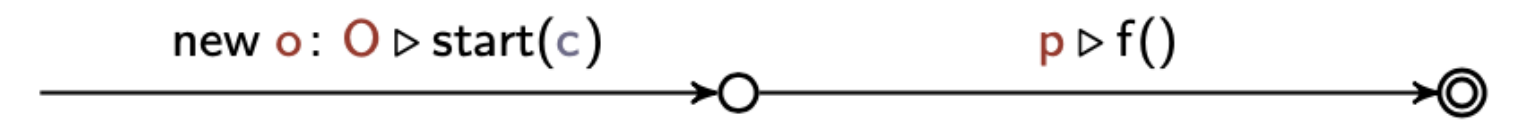


Role emptyness

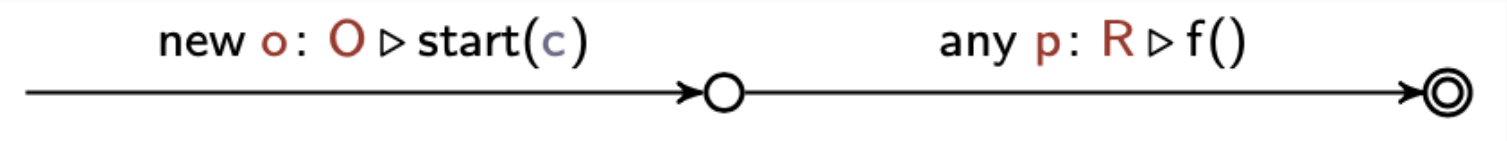


Not all DAFSMs make sense

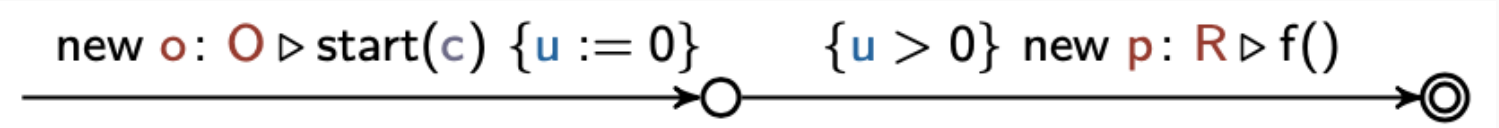
Names' freeness



Role emptyness

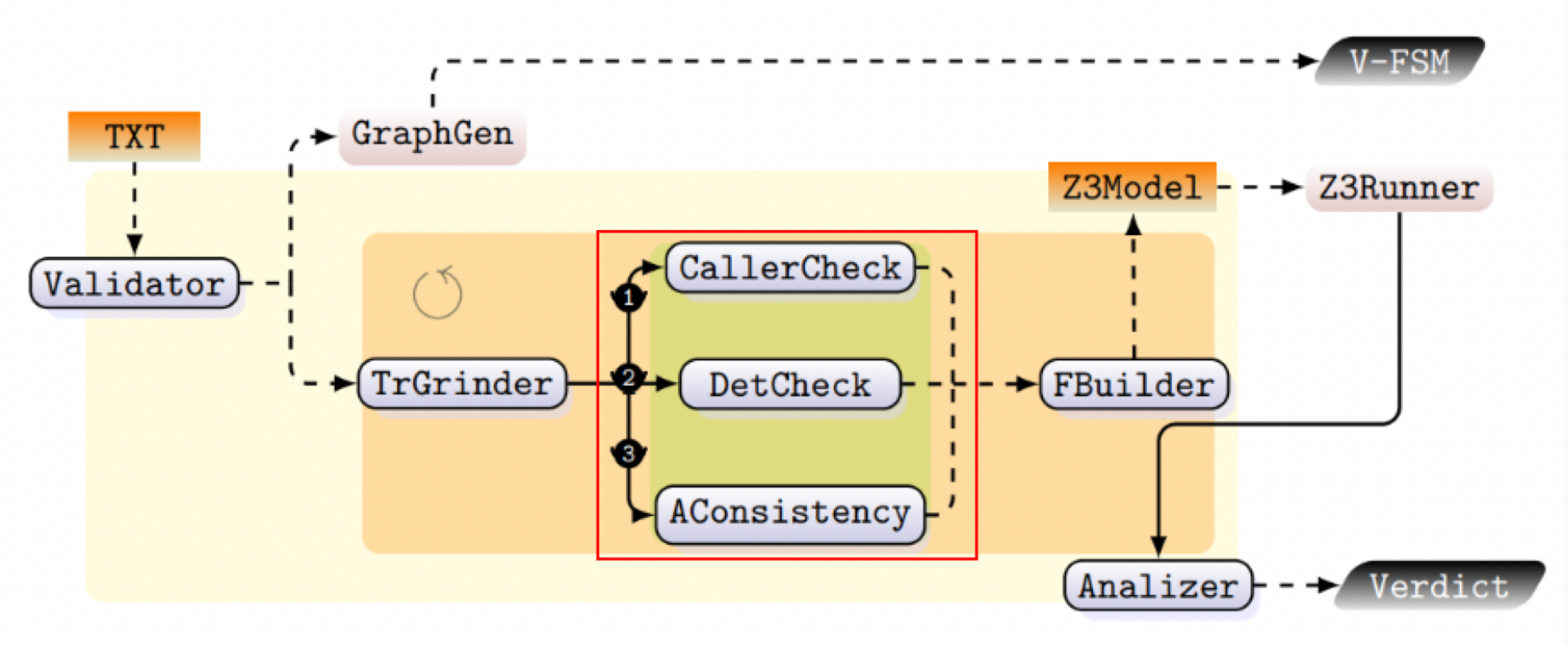


No progress



TRAC: model and ensure well-formed DAFSMs

Paper and Artefact at Coordination'24



https://github.com/loctet/TRAC

Concluding

The role of a choreographic model

Smart Contracts: code is law, bugs are features

How to make sure to deploy correct ones and defend them from exploits?

- Automatically get correct-by-construction ones
- Protect from malicious clients blocking insidious requests (using monitors, for instance)

The role of a choreographic model

Smart Contracts: code is law, bugs are features

How to make sure to deploy correct ones and defend them from exploits?

- Automatically get correct-by-construction ones
- Protect from malicious clients blocking insidious requests (using monitors, for instance)

Our contribution

- A model for global choreographies: Data-Aware Finite State Machines
- A tool to define and check their well-formedness: TRAC
- A prototype to generate Solidity code

Thanks!

For now...