Automata for smart contracts...and more

Emilio Tuosto @ GSSI

joint work with

Maurizio Murgia @GSSI

Elvis Gerardin Konjoh Selabi @GSSI & UniCam

Antonio Ravara @NOVA

A tutorial @ FORTE 2025, Lille

2025-01-04

Automata for smart contracts...and more

Emilio Tuosto @ GSSI

joint work with

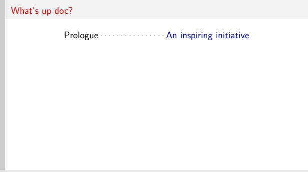
Maurizio Murgia @GSSI

Elvis Gerardin Konjoh Selabi @GSSI & UniCam

Antonio Ravara @NOVA

A tutorial @ FORTE 2025, Lille

What’s up doc?



What’s up doc?

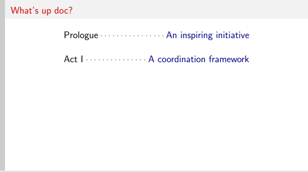
Prologue · · · · · · · · · · · · · · · · An inspiring initiative

What's up doc?

2025-01-04

Automata for smart contracts...and more

└─What's up doc?

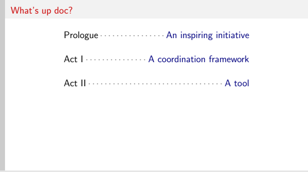


What's up doc?

Prologue · · · · · · · · · · · · · · · · An inspiring initiative

Act I · · · · · · · · · · · · · · · A coordination framework

What's up doc?



What's up doc?

Prologue · · · · · · · · · · · · · · · · · An inspiring initiative

Act I · · · · · · · · · · · · · · · · A coordination framework

Act II · A tool

What's up doc?

2025-01-04

Automata for smart contracts...and more

└What's up doc?

What's up doc?

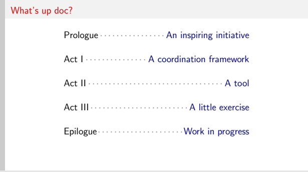
Prologue · · · · · · · · · · · · · · · · · An inspiring initiative

Act I · · · · · · · · · · · · · · · · A coordination framework

Act II · A tool

Act III · A little exercise

What's up doc?



What's up doc?

Prologue · · · · · · · · · · · · · · · · · An inspiring initiative

Act I · · · · · · · · · · · · · · · · A coordination framework

Act II · A tool

Act III · A little exercise

Epilogue · Work in progress

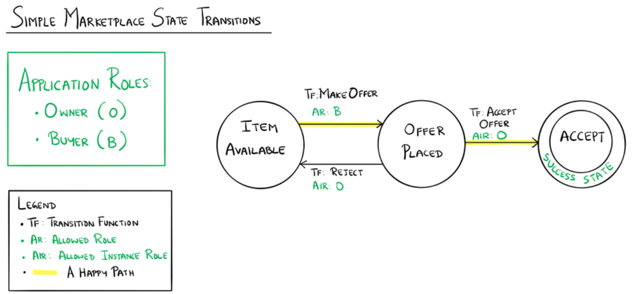
– Prologue –

[An inspiring initiative]

Automata for smart contracts...and more

A nice sketch! [?, ?]

A smart contract among Owners and Buyers



initially buyers can make offers

then

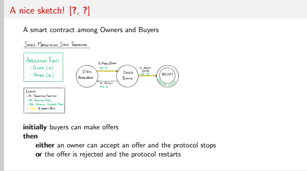
  either an owner can accept an offer and the protocol stops

  or the offer is rejected and the protocol restarts

Automata for smart contracts...and more

2025-01-04

└A nice sketch! [?, ?]



A nice sketch! [?, ?]

A smart contract among Owners and Buyers

initially buyers can make offers

then

either an owner can accept an offer and the protocol stops

or the offer is rejected and the protocol restarts

What did we just see?

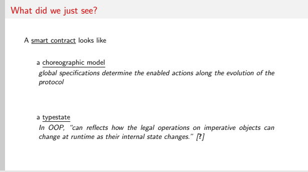
A smart contract looks like

a choreographic model

global specifications determine the enabled actions along the evolution of the protocol

a typestate

*In OOP, "can reflects how the legal operations on imperative objects can change at runtime as their internal state changes." [**?**]*


What did we just see?

A smart contract looks like

a choreographic model

global specifications determine the enabled actions along the evolution of the protocol

a typestate

In OOP, "can reflects how the legal operations on imperative objects can change at runtime as their internal state changes." [?]

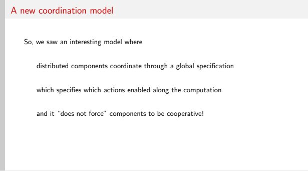
A new coordination model

So, we saw an interesting model where

distributed components coordinate through a global specification

which specifies which actions enabled along the computation

and it “does not force” components to be cooperative!



A new coordination model

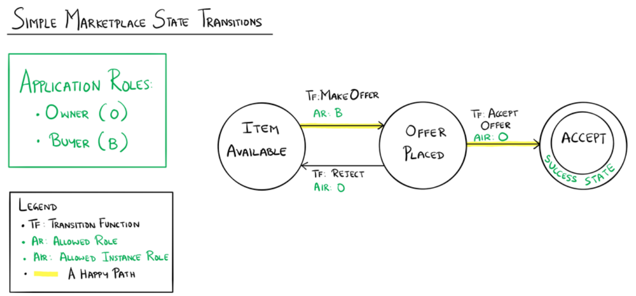
So, we saw an interesting model where

distributed components coordinate through a global specification

which specifies which actions enabled along the computation

and it “does not force” components to be cooperative!

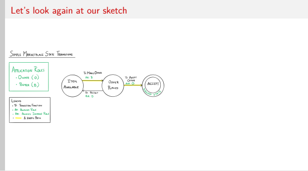
Let’s look again at our sketch



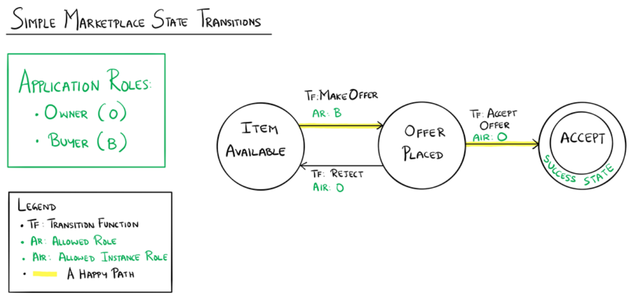
Automata for smart contracts...and more

2025-01-04

└Let’s look again at our sketch



Let's look again at our sketch



but...

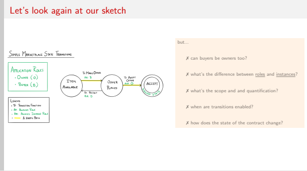
✗ can buyers be owners too?

✗ what's the difference between <u>roles</u> and <u>instances</u>?

✗ what's the scope and and quantification?

✗ when are transitions enabled?

✗ how does the state of the contract change?



ok

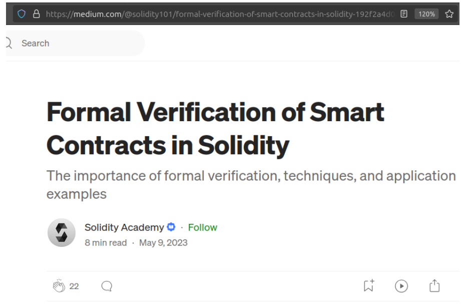
ok

from [?]: "The transitions between the Item Available and the Offer Placed states can continue until the owner is satisfied with the offer made." so, after a rejection, the new offer must be from the original buyer or a new one?

ok

should the price of the item remain unchanged when the owner invokes the Reject?

...and by the way


https://medium.com/@solidity101/formal-verification-of-smart-contracts-in-solidity-192f2a4d 120%

Search

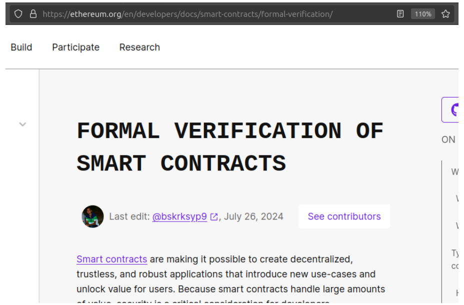
Formal Verification of Smart Contracts in Solidity

The importance of formal verification, techniques, and application examples

Solidity Academy · Follow
8 min read · May 9, 2023

22

https://medium.com/@solidity101/formal-verification-of-smart-contracts-in-solidity-192f2a4d0abd


https://ethereum.org/en/developers/docs/smart-contracts/formal-verification/ 110%

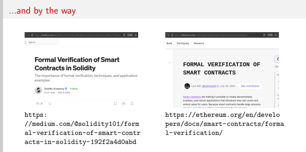
Build Participate Research

FORMAL VERIFICATION OF SMART CONTRACTS

Last edit: @bskrksyp9, July 26, 2024 See contributors

Smart contracts are making it possible to create decentralized, trustless, and robust applications that introduce new use-cases and unlock value for users. Because smart contracts handle large amounts

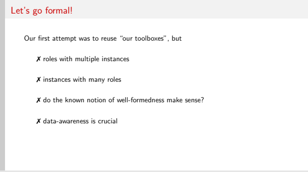
https://ethereum.org/en/developers/docs/smart-contracts/formal-verification/



Let's go formal!

Our first attempt was to reuse "our toolboxes", but

- ✗ roles with multiple instances
- ✗ instances with many roles
- ✗ do the known notion of well-formedness make sense?
- ✗ data-awareness is crucial


Let's go formal!

Our first attempt was to reuse "our toolboxes", but

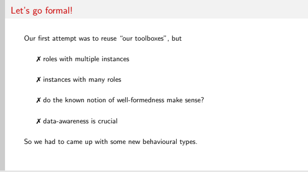
- ✗ roles with multiple instances
- ✗ instances with many roles
- ✗ do the known notion of well-formedness make sense?
- ✗ data-awareness is crucial

Let's go formal!

Our first attempt was to reuse "our toolboxes", but

- ✗ roles with multiple instances
- ✗ instances with many roles
- ✗ do the known notion of well-formedness make sense?
- ✗ data-awareness is crucial

So we had to came up with some new behavioural types.


Let's go formal!

Our first attempt was to reuse "our toolboxes", but

✗ roles with multiple instances

✗ instances with many roles

✗ do the known notion of well-formedness make sense?

✗ data-awareness is crucial

So we had to came up with some new behavioural types.

– Act I –

[A coordination framework]

2025-01-04

Automata for smart contracts...and more

– Act I –

[A coordination framework]

Basic concepts and notation

Participants $\mathsf{p}, \mathsf{p}', \ldots$

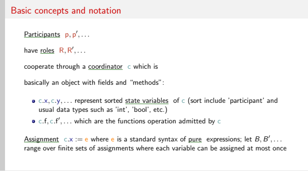
have roles $\mathsf{R}, \mathsf{R}', \ldots$

cooperate through a coordinator c which is

basically an object with fields and "methods":

- $\mathsf{c.x}, \mathsf{c.y}, \ldots$ represent sorted state variables of c (sort include 'participant' and usual data types such as 'int', 'bool', etc.)
- $\mathsf{c.f}, \mathsf{c.f}', \ldots$ which are the functions operation admitted by c

Assignment $\mathsf{c.x} := \mathsf{e}$ where e is a standard syntax of pure expressions; let $B, B', \ldots$ range over finite sets of assignments where each variable can be assigned at most once


Basic concepts and notation

Participants $\mathsf{p}, \mathsf{p}', \ldots$

have roles $\mathsf{R}, \mathsf{R}', \ldots$

cooperate through a coordinator c which is

basically an object with fields and "methods":

- $\mathsf{c.x}, \mathsf{c.y}, \ldots$ represent sorted state variables of c (sort include 'participant' and usual data types such as 'int', 'bool', etc.)
- $\mathsf{c.f}, \mathsf{c.f}', \ldots$ which are the functions operation admitted by c

Assignment $\mathsf{c.x} := \mathsf{e}$ where e is a standard syntax of pure expressions; let $B, B', \ldots$ range over finite sets of assignments where each variable can be assigned at most once

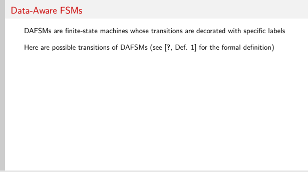
In every assignment $\mathsf{c.x} := \mathsf{e}$ data variables occurring in e must have the old qualifier to refer to their value before the assignments.

We adapt the mechanism based on the old keyword from the Eiffel language [**?**] which, as explained in [**?**] is necessary to render assignments into logical formulae since e.g., $x = x + 1 \Leftrightarrow$ False. This will be used in def:consistency.

Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs (see [**?**, Def. 1] for the formal definition)



Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs (see [?, Def. 1] for the formal definition)

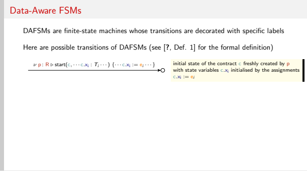
Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs (see [**?**, Def. 1] for the formal definition)

$$\xrightarrow{\nu\ \mathsf{p} : \mathsf{R} \triangleright \mathsf{start}(\mathsf{c}, \cdots \mathsf{c}.\mathsf{x}_\mathsf{i} : T_i \cdots)\ \{\cdots \mathsf{c}.\mathsf{x}_\mathsf{i} := \mathsf{e}_i \cdots\}} \bigcirc$$

initial state of the contract c freshly created by p with state variables $\mathsf{c}.\mathsf{x}_\mathsf{i}$ initialised by the assignments $\mathsf{c}.\mathsf{x}_\mathsf{i} := \mathsf{e}_i$



Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs (see [?, Def. 1] for the formal definition)

$\nu\ \mathsf{p} : \mathsf{R} \triangleright \mathsf{start}(\mathsf{c}, \cdots \mathsf{c}.\mathsf{x}_\mathsf{i} : T_i \cdots)\ \{\cdots \mathsf{c}.\mathsf{x}_\mathsf{i} := \mathsf{e}_i \cdots\}$

initial state of the contract c freshly created by p with state variables $\mathsf{c}.\mathsf{x}_\mathsf{i}$ initialised by the assignments $\mathsf{c}.\mathsf{x}_\mathsf{i} := \mathsf{e}_i$

each state variable is declared and initialises with type-consistent expressions

start is a “build-in” function name

Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

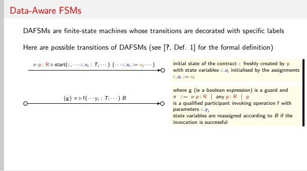
Here are possible transitions of DAFSMs (see [**?**, Def. 1] for the formal definition)

$$\xrightarrow{\nu\ \mathsf{p} : \mathsf{R} \triangleright \mathsf{start}(\mathsf{c}, \cdots \mathsf{c}.\mathsf{x_i} : T_i \cdots)\ \{\cdots \mathsf{c}.\mathsf{x_i} := \mathsf{e}_i \cdots\}} \circ$$

initial state of the contract c freshly created by p with state variables $\mathsf{c}.\mathsf{x_i}$ initialised by the assignments $\mathsf{c}.\mathsf{x_i} := \mathsf{e}_i$

$$\circ \xrightarrow{\{\mathsf{g}\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ$$

where g (ie a boolean expression) is a guard and
$\pi \ ::= \ \nu\ \mathsf{p} : \mathsf{R} \ \mid \ \text{any } \mathsf{p} : \mathsf{R} \ \mid \ \mathsf{p}$
is a qualified participant invoking operation f with parameters $\mathsf{c}.\mathsf{y_i}$
state variables are reassigned according to B if the invocation is successful



g predicates over state variables and formal parameters; like in Hoare triples, they are pre-conditions to be satisfied in order for the invocation to be enabled

ν p: R specifies that p must be a fresh participant with role R
any p: R qualifies p as an existing participant with role R
p we refer to a participant in the scope of a binder

the variables occurring in the right-hand side of assignments in B are either state variables or parameters of the invocation

Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs (see [**?**, Def. 1] for the formal definition)

$$\xrightarrow{\nu\, p : R \triangleright \mathsf{start}(c, \cdots c.x_i : T_i \cdots)\ \{\cdots c.x_i := e_i \cdots\}} \bigcirc$$

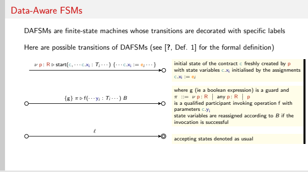
initial state of the contract c freshly created by p with state variables $c.x_i$ initialised by the assignments $c.x_i := e_i$

$$\bigcirc \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \bigcirc$$

where g (ie a boolean expression) is a guard and
$\pi \ ::= \ \nu\, p : R \ \mid \ \text{any } p : R \ \mid \ p$
is a qualified participant invoking operation f with parameters $c.y_i$
state variables are reassigned according to B if the invocation is successful

$$\bigcirc \xrightarrow{\ell} \circledcirc$$

accepting states denoted as usual


Data-Aware FSMs

DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs (see [?, Def. 1] for the formal definition)

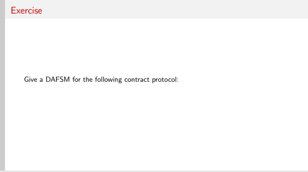
initial state of the contract c freshly created by p with state variables c.x_i initialised by the assignments c.x_i := e_i

where g (ie a boolean expression) is a guard and π ::= ν p : R | any p : R | p is a qualified participant invoking operation f with parameters c.y_i state variables are reassigned according to B if the invocation is successful

accepting states denoted as usual

Exercise

Give a DAFSM for the following contract protocol:


Exercise

Give a DAFSM for the following contract protocol:

let them play with qualified participants