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

Automata for smart contracts...and more

What's up doc?

What's up doc?

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

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?

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 –

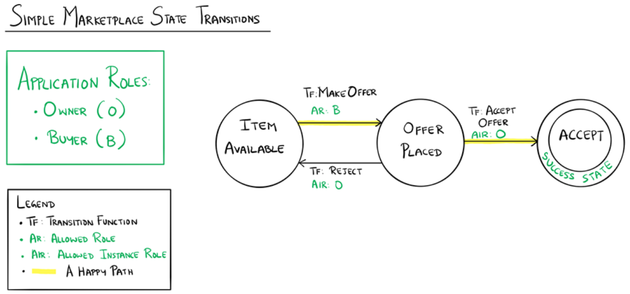
[An inspiring initiative]

– Prologue –

[An inspiring initiative]

A nice sketch! [3, 4]

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

A nice sketch! [3, 4]
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." [2]

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." [2]

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

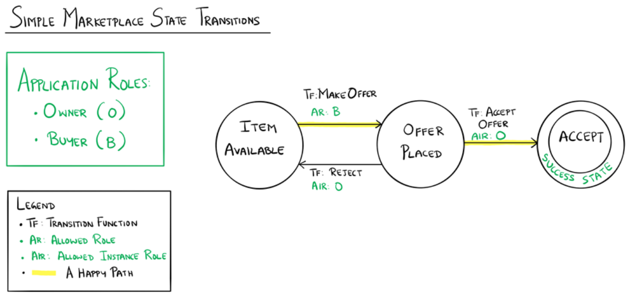
So, we saw an interesting model where

distributed components coordinate through a global specification

which specifies which actions enabled along the computation

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

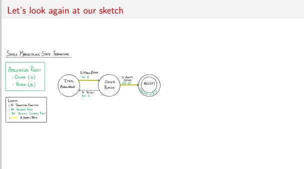
Let's look again at our sketch



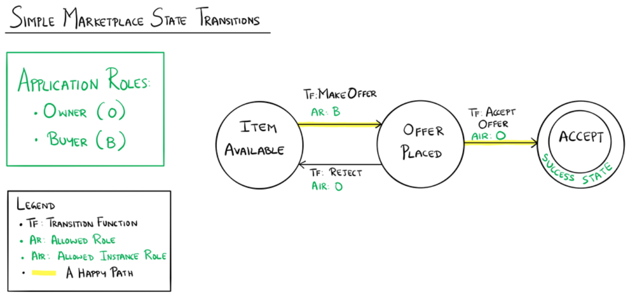
2025-01-06

Automata for smart contracts...and more

└Let's look again at our sketch



Let's look again at our sketch



but...

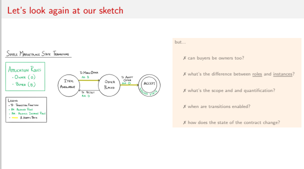
✗ can buyers be owners too?

✗ 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?



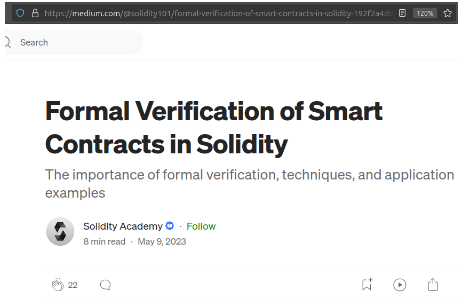
ok

ok

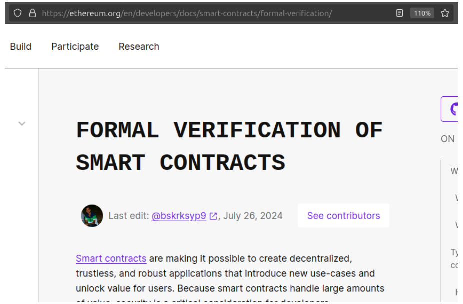
from [4]: "The transitions between the Item Available and the Offer Placed states can continue until the owner is satisfied with the offer made." so, after a rejection, the new offer must be from the original buyer or a new one?

ok

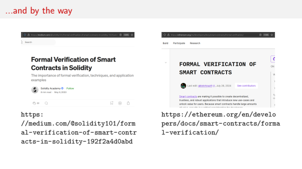
should the price of the item remain unchanged when the owner invokes the Reject?



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



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]

– Act I –
[A coordination framework]

Basic concepts and notation

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

Basic concepts and notation
Participants p, p′, . . .

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 **??**.

Basic concepts and notation

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

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

Basic concepts and notation

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

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

In every assignment $\mathsf{c}.\mathsf{x} := \mathsf{e}$ 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 **??**.

Basic concepts and notation

Participants p, p′, . . .

have roles R, R′, . . .

cooperate through a coordinator c which is

Basic concepts and notation
Participants p, p′, . . .
have roles R, R′, . . .
cooperate through a coordinator c which is

In every assignment $\mathsf{c}.\mathsf{x} := \mathsf{e}$ 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 **??**.

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":

Basic concepts and notation
Participants p, p′, . . .
have roles R, R′, . . .
cooperate through a coordinator c which is
basically an object with fields and "methods":

In every assignment $\mathsf{c}.\mathsf{x} := \mathsf{e}$ 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 **??**.

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.)

Basic concepts and notation
Participants p, p′, . . .
have roles R, R′, . . .
cooperate through a coordinator c which is
basically an object with fields and "methods":
c.x, c.y, . . . represent sorted state variables of c (sort include 'participant' and usual data types such as 'int', 'bool', etc.)

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 **??**.

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":

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

Basic concepts and notation
Participants p, p′, . . .
have roles R, R′, . . .
cooperate through a coordinator c which is
basically an object with fields and "methods":
• c.x, c.y, . . . represent sorted state variables of c (sort include 'participant' and usual data types such as 'int', 'bool', etc.)
• c.f, c.f′, . . . which are the functions operation admitted by c

In every assignment $\mathsf{c}.\mathsf{x} := \mathsf{e}$ 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 **??**.

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

Basic concepts and notation
Participants p, p′, . . .
have roles R, R′, . . .
cooperate through a coordinator c which is
basically an object with fields and "methods":
c.x, c.y, . . . represent sorted state variables of c (sort include 'participant' and usual data types such as 'int', 'bool', etc.)
c.f, c.f′, . . . which are the functions operation admitted by c

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 **??**.

Basic concepts and notation

<u>Participants</u> $\mathsf{p}, \mathsf{p}', \dots$

have <u>roles</u> $\mathsf{R}, \mathsf{R}', \dots$

cooperate through a <u>coordinator</u> c which is

basically an object with fields and "methods":

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

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

Basic concepts and notation
Participants p, p′, ...
have roles R, R′, ...
cooperate through a coordinator c which is
basically an object with fields and "methods":
c.x, c.y, ... represent sorted state variables of c (sort include 'participant' and usual data types such as 'int', 'bool', etc.)
c.f, c.f′, ... which are the functions operation admitted by c
Assignment c.x := e where e is a standard syntax of pure expressions; let B, B′, ... 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 **??**.

Data-Aware FSMs

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

Here are possible transitions of DAFSMs[1]

[1] See [1, 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[1]
[1] See [1, 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[1]

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

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

[1] See [1, Def. 1] for the formal definition

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[1]

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

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

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

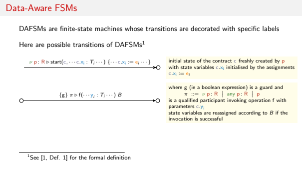
where g (ie a boolean expression) is a guard and

$$\pi \ ::= \ \nu\ \mathsf{p} : \mathsf{R} \ \mid \ \mathsf{any}\ \mathsf{p} : \mathsf{R} \ \mid \ \mathsf{p}$$

is a qualified participant invoking operation f with parameters $\mathsf{c}.\mathsf{y}_\mathsf{i}$

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

[1] See [1, Def. 1] for the formal definition



g predicates over state variables and formal parameters; guards have to be satisfied in order for the invocation to be enabled: an invocation that makes the guard false is rejected

$\nu\ \mathsf{p} : \mathsf{R}$ specifies that p must be a fresh participant with role R
any $\mathsf{p} : \mathsf{R}$ qualifies p as an existing participant with role R
p we refer to a participant in the scope of a binder
invocations from non-suitable callers are rejected

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[1]

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

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

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

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

$$\pi \ ::= \ \nu\ \mathsf{p} : \mathsf{R} \ \mid\ \mathsf{any}\ \mathsf{p} : \mathsf{R} \ \mid\ \mathsf{p}$$

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

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

accepting states denoted as usual

[1] See [1, Def. 1] for the formal definition

Exercise: modelling

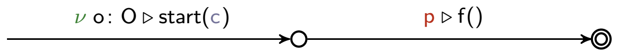
Give a DAFSM for the following contract protocol:

Exercise: modelling

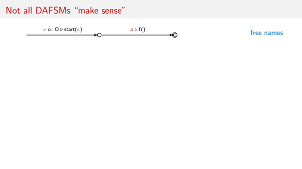
Give a DAFSM for the following contract protocol:

let them play with qualified participants

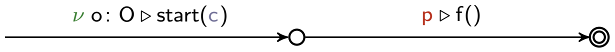
Not all DAFSMs "make sense"



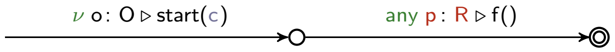
free names



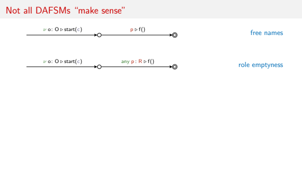
Not all DAFSMs “make sense”



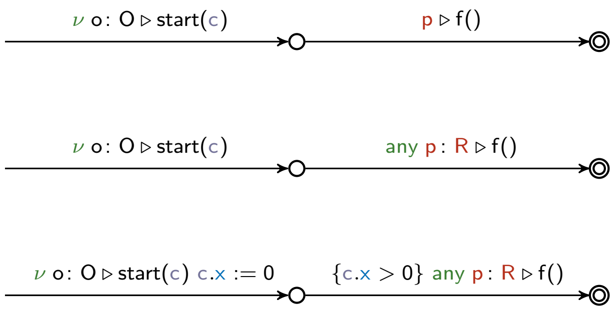
free names



role emptyness



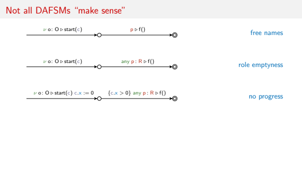
Not all DAFSMs "make sense"



free names

role emptyness

no progress



Not all DAFSMs “make sense”

$$\xrightarrow{\nu\ \mathsf{o} : \mathsf{O} \triangleright \mathsf{start}(\mathsf{c})} \bigcirc \xrightarrow{\mathsf{p} \triangleright \mathsf{f}()} \circledcirc$$

free names

$$\xrightarrow{\nu\ \mathsf{o} : \mathsf{O} \triangleright \mathsf{start}(\mathsf{c})} \bigcirc \xrightarrow{\mathsf{any}\ \mathsf{p} : \mathsf{R} \triangleright \mathsf{f}()} \circledcirc$$

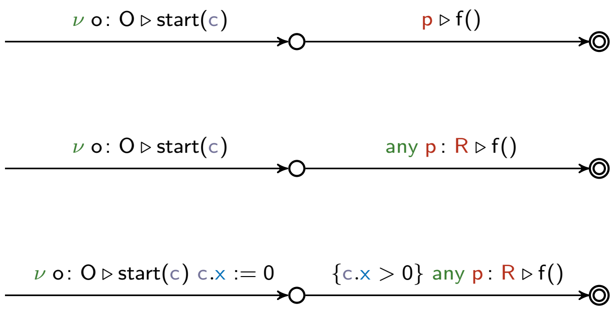
role emptyness

$$\xrightarrow{\nu\ \mathsf{o} : \mathsf{O} \triangleright \mathsf{start}(\mathsf{c})\ \mathsf{c.x} := 0} \bigcirc \xrightarrow{\{\mathsf{c.x} > 0\}\ \mathsf{any}\ \mathsf{p} : \mathsf{R} \triangleright \mathsf{f}()} \circledcirc$$

no progress

Not all DAFSMs “make sense”
free names
role emptyness
no progress

Not all DAFSMs “make sense”

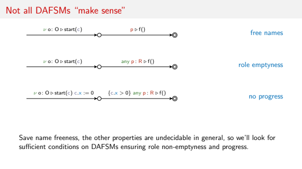


free names

role emptyness

no progress

Save name freeness, the other properties are undecidable in general, so we’ll look for sufficient conditions on DAFSMs ensuring role non-emptyness and progress.



Closed DAFSMs

Binders : parameter declaration in function call, ν p : R, and any p : R

Closed DAFSMs
Binders : parameter declaration in function call, ν p : R, and any p : R

Closed DAFSMs

Binders : parameter declaration in function call, ν p : R, and any p : R

p is bound in $\bigcirc \xrightarrow{\{\mathrm{g}\}\ \pi \triangleright \mathrm{f}(\cdots \mathrm{y}_i : T_i \cdots)\ B} \bigcirc$ if, for some role R, $\pi = \nu$ p : R
or $\pi =$ any p : R or there is i s.t. $\mathrm{y}_i =$ p and $T_i =$ R

2025-01-06

Automata for smart contracts...and more

└Closed DAFSMs

Closed DAFSMs

Binders : parameter declaration in function call, ν p : R, and any p : R

p is bound in $\bigcirc \xrightarrow{\{\mathrm{g}\}\ \pi \triangleright \mathrm{f}(\cdots \mathrm{y}_i : T_i \cdots)\ B} \bigcirc$ if, for some role R, $\pi = \nu$ p : R
or $\pi =$ any p : R or there is i s.t. $\mathrm{y}_i =$ p and $T_i =$ R

Closed DAFSMs

<u>Binders</u> : parameter declaration in function call, ν p : R, and any p : R

p is <u>bound</u> in $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ if, for some role R, $\pi = \nu$ p : R
or $\pi =$ any p : R or there is i s.t. $y_i = p$ and $T_i = R$

The occurrence of p is <u>bound</u> in a path $\sigma \circ \xrightarrow{\{g\}\ p \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ \sigma'$ if p is bound in a transition of σ

Closed DAFSMs
<u>Binders</u> : parameter declaration in function call, ν p : R, and any p : R
p is <u>bound</u> in $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ if, for some role R, $\pi = \nu$ p : R
or $\pi =$ any p : R or there is i s.t. $y_i = p$ and $T_i = R$
The occurrence of p is <u>bound</u> in a path $\sigma \circ \xrightarrow{\{g\}\ p \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ \sigma'$ if p is bound in a transition of σ

Closed DAFSMs

<u>Binders</u> : parameter declaration in function call, ν p: R, and any p: R

p is <u>bound</u> in $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ if, for some role R, $\pi = \nu$ p: R
or $\pi =$ any p: R or there is i s.t. $y_i =$ p and $T_i =$ R

The occurrence of p is <u>bound</u> in a path σ $\circ \xrightarrow{\{g\}\ p \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ σ' if p is bound in a transition of σ

A DAFSM is <u>closed</u> if all occurrences of participant variables are bound in the paths of the DAFSM they occur on

Closed DAFSMs
Binders : parameter declaration in function call, ν p: R, and any p: R
p is bound in $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ if, for some role R, $\pi = \nu$ p: R
or $\pi =$ any p: R or there is i s.t. $y_i =$ p and $T_i =$ R
The occurrence of p is bound in a path σ $\circ \xrightarrow{\{g\}\ p \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ σ' if p is bound in a transition of σ
A DAFSM is closed if all occurrences of participant variables are bound in the paths of the DAFSM they occur on

Roles non-emptyness

A transition $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ <u>expands</u> role R if $\pi = \nu\ p : R$ or there is i s.t. $y_i = p$ and $T_i = R$

Role R is <u>expanded</u> in a path $\sigma \xrightarrow{\{g\}\ \text{any}\ p : R \triangleright f(\cdots y_i : T_i \cdots)\ B} \sigma'$ if a transition in σ expands R

A DAFSM <u>expands</u> R if all its paths expand R and is <u>(strongly) empty-role free</u> if it expands all its roles

Roles non-emptyness

A transition $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ expands role R if $\pi = \nu\ p : R$ or there is i s.t. $y_i = p$ and $T_i = R$

Role R is expanded in a path $\sigma \xrightarrow{\{g\}\ \text{any}\ p : R \triangleright f(\cdots y_i : T_i \cdots)\ B} \sigma'$ if a transition in σ expands R

A DAFSM expands R if all its paths expand R and is (strongly) empty-role free if it expands all its roles

Progress

A DAFSM with state variables $X = \{\mathsf{c}.\mathsf{x}_1, \ldots, \mathsf{c}.\mathsf{x}_\mathsf{n}\}$ is consistent if it is closed and the following implication holds for each transition $\circ \xrightarrow{\{\mathsf{g}\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \textcircled{s}$

$$\forall(\mathsf{c}.\mathsf{x}, \mathsf{old}\ \mathsf{c}.\mathsf{x})_{\mathsf{c}.\mathsf{x} \in X}\ \exists \ldots, \mathsf{y}_i, \ldots\ :\ (\mathsf{g}\{\mathsf{old}\ \mathsf{c}.\mathsf{x}/\mathsf{c}.\mathsf{x}\}_{\mathsf{c}.\mathsf{x} \in X} \wedge \mathsf{g}_B) \implies \mathsf{g}_\mathsf{s} \qquad \text{where}$$

$$\mathsf{g}_\mathsf{s} = \begin{cases} \text{True} & \text{if s is accepting} \\ \text{the disjunction of guards of the outgoing transitions of s} & \text{otw} \end{cases}$$

$$\mathsf{g}_B = \bigwedge_{(\mathsf{c}.\mathsf{x} := \mathsf{e}) \in B} \mathsf{c}.\mathsf{x} = \mathsf{e} \ \wedge \bigwedge_{\mathsf{c}.\mathsf{x} \notin B} \mathsf{c}.\mathsf{x} = \mathsf{old}\ \mathsf{c}.\mathsf{x}$$

$$\text{with} \quad \mathsf{c}.\mathsf{x} \notin B \iff (\mathsf{c}.\mathsf{y} := \mathsf{e}) \in B \implies \mathsf{x} \neq \mathsf{y} \quad \text{and} \quad \mathsf{old}\ \mathsf{c}.\mathsf{x} \text{does not occur in } \mathsf{e}$$

Progress
A DAFSM with state variables $X = \{\mathsf{c}.\mathsf{x}_1, \ldots, \mathsf{c}.\mathsf{x}_\mathsf{n}\}$ is consistent if it is closed and the following implication holds for each transition $\circ \xrightarrow{\{\mathsf{g}\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \textcircled{s}$
$\forall(\mathsf{c}.\mathsf{x}, \mathsf{old}\ \mathsf{c}.\mathsf{x})_{\mathsf{c}.\mathsf{x} \in X}\ \exists \ldots, \mathsf{y}_i, \ldots\ :\ (\mathsf{g}\{\mathsf{old}\ \mathsf{c}.\mathsf{x}/\mathsf{c}.\mathsf{x}\}_{\mathsf{c}.\mathsf{x} \in X} \wedge \mathsf{g}_B) \implies \mathsf{g}_\mathsf{s}$ where
$\mathsf{g}_\mathsf{s} = \begin{cases} \text{True} & \text{if s is accepting} \\ \text{the disjunction of guards of the outgoing transitions of s} & \text{otw} \end{cases}$
$\mathsf{g}_B = \bigwedge_{(\mathsf{c}.\mathsf{x} := \mathsf{e}) \in B} \mathsf{c}.\mathsf{x} = \mathsf{e} \ \wedge \bigwedge_{\mathsf{c}.\mathsf{x} \notin B} \mathsf{c}.\mathsf{x} = \mathsf{old}\ \mathsf{c}.\mathsf{x}$
with $\mathsf{c}.\mathsf{x} \notin B \iff (\mathsf{c}.\mathsf{y} := \mathsf{e}) \in B \implies \mathsf{x} \neq \mathsf{y}$ and $\mathsf{old}\ \mathsf{c}.\mathsf{x}$does not occur in e

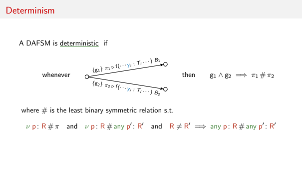
Determinism

A DAFSM is deterministic if

whenever
$$\begin{tikzcd} & \circ \\ \circ \arrow[ur, "\{g_1\}\ \pi_1 \triangleright f(\cdots y_i : T_i \cdots)\ B_1"] \arrow[dr, "\{g_2\}\ \pi_2 \triangleright f(\cdots y_i : T_i \cdots)\ B_2"'] & \\ & \circ \end{tikzcd}$$
then $g_1 \wedge g_2 \implies \pi_1 \# \pi_2$

where $\#$ is the least binary symmetric relation s.t.

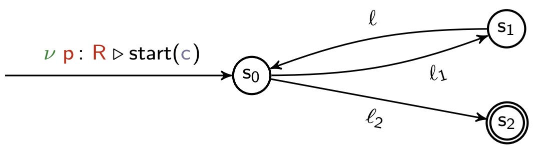
$\nu\ p : R \# \pi$ and $\nu\ p : R \# \text{any } p' : R'$ and $R \neq R' \implies \text{any } p : R \# \text{any } p' : R'$



transitions from the same source state and calling the same function

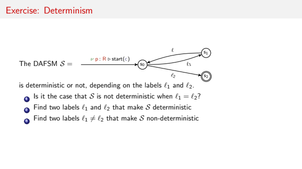
Exercise: Determinism

The DAFSM $\mathcal{S} =$



is deterministic or not, depending on the labels ℓ_1 and ℓ_2.

1. Is it the case that $\mathcal{S}$ is not deterministic when $\ell_1 = \ell_2$?
2. Find two labels ℓ_1 and ℓ_2 that make $\mathcal{S}$ deterministic
3. Find two labels $\ell_1 \neq \ell_2$ that make $\mathcal{S}$ non-deterministic



1. no: eg for $\ell_1 = \ell_2 = \nu$ p : R $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 = \nu$ p : R ▷ f($\cdots$ y_i : T_i $\cdots$) make $\mathcal{S}$ deterministic because the next state is unambiguously determined by the caller which is fresh on both transitions
3. $\ell_1 = \{x \leq 10\}$ p ▷ f(x : Int) and $\ell_2 = \{x \geq 10\}$ p ▷ f(x : Int) make $\mathcal{S}$ non-deterministic because the guards of ℓ_1 and of ℓ_2 are not disjoint therefore the next state is not determined by the caller of c.g. because guard $x \leq 10$ leading to s_1 and guard $x > 10$ leading to s_2 are disjoint; therefore the next state is determined by the value of the parameter x, and every value enables at most one transition.

Also, taking ℓ_1 as in the latter case and $\ell_2 = \{x \geq 10\}$ o ▷ c.g(x : Int) would make $\mathcal{S}$ non-deterministic

Well-formedness

A DAFSM is well-formed when

- it is empty-role free
- it is consistent, and
- it is deterministic

Well-formedness
A DAFSM is well-formed when
it is empty-role free
it is consistent, and
it is deterministic

– Act II –

[A tool]

– Act II –
[A tool]

– Act III –

[A little exercise]

– Act III –
[A little exercise]

– Epilogue –

[Work in progress]

– Epilogue –

[Work in progress]

Thank you

2025-01-06

Automata for smart contracts...and more

References I

[1] J. Afonso, E. Konjoh Selabi, M. Murgia, A. Ravara, and E. Tuosto. TRAC: A tool for data-aware coordination - (with an application to smart contracts). In I. Castellani and F. Tiezzi, editors, *Coordination Models and Languages - 26th IFIP WG 6.1 International Conference, COORDINATION 2024, Held as Part of the 19th International Federated Conference on Distributed Computing Techniques, DisCoTec 2024, Groningen, The Netherlands, June 17-21, 2024, Proceedings*, volume 14676 of *Lecture Notes in Computer Science*, pages 239–257. Springer, 2024.

[2] R. Garcia, E. Tanter, R. Wolff, and J. Aldrich. Foundations of typestate-oriented programming. *ACM Trans. Program. Lang. Syst.*, 36(4), Oct. 2014.

[3] Microsoft. The blockchain workbench. https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench, 2019.

2025-01-06

Automata for smart contracts...and more

└References

References I

[1] J. Afonso, E. Konjoh Selabi, M. Murgia, A. Ravara, and E. Tuosto. TRAC: A tool for data-aware coordination - (with an application to smart contracts). In I. Castellani and F. Tiezzi, editors, *Coordination Models and Languages - 26th IFIP WG 6.1 International Conference, COORDINATION 2024, Held as Part of the 19th International Federated Conference on Distributed Computing Techniques, DisCoTec 2024, Groningen, The Netherlands, June 17-21, 2024, Proceedings*, volume 14676 of *Lecture Notes in Computer Science*, pages 239–257. Springer, 2024.

[2] R. Garcia, E. Tanter, R. Wolff, and J. Aldrich. Foundations of typestate-oriented programming. *ACM Trans. Program. Lang. Syst.*, 36(4), Oct. 2014.

[3] Microsoft. The blockchain workbench. https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench, 2019.

References II

[4] Microsoft. Simple marketplace sample application for azure blockchain workbench. https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/simple-marketplace, 2019.

References II

[4] Microsoft. Simple marketplace sample application for azure blockchain workbench. https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/simple-marketplace, 2019.