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

2025-01-08

Automata for smart contracts...and more

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?

2025-01-08

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?

2025-01-08

└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

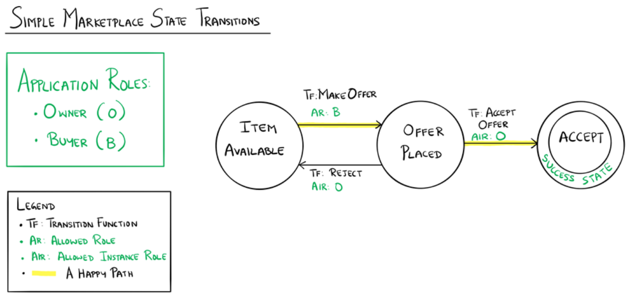
– Prologue –

[An inspiring initiative]

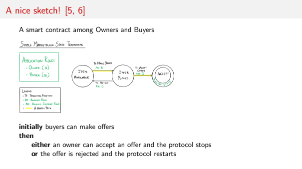
– Prologue –
[An inspiring initiative]

A nice sketch! [5, 6]

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]

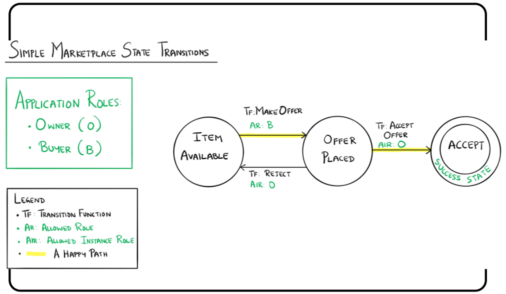
A new coordination model

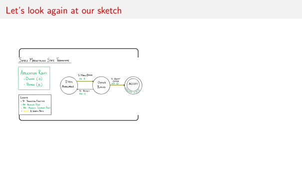
So, we saw an interesting model where

- distributed components coordinate through a global specification
- which specifies which actions enabled along the computation
- and it “does not force” components to be cooperative!

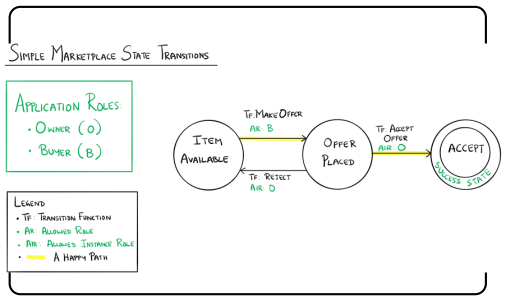
A new coordination model
So, we saw an interesting model where
distributed components coordinate through a global specification
which specifies which actions enabled along the computation
and it “does not force” components to be cooperative!

Let's look again at our sketch





Let’s look again at our sketch



but...

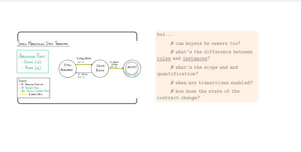
✗ can buyers be owners too?

✗ what’s the difference between roles and instances?

✗ what’s the scope and and quantification?

✗ when are transitions enabled?

✗ how does the state of the contract change?



ok

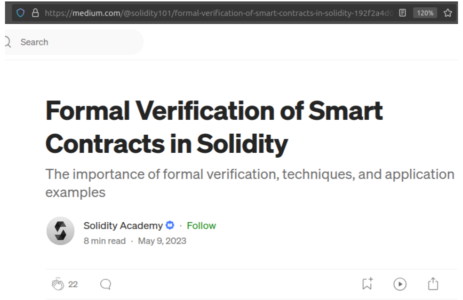
ok

from [6]: “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?

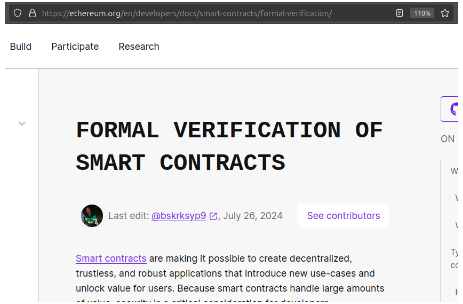
ok

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

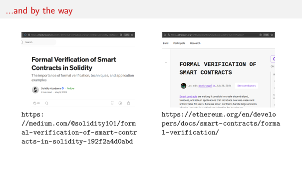
...and by the way



https://medium.com/@solidity101/formal-verification-of-smart-contracts-in-solidity-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
- ✗ are the known notions of well-formedness suitable?
- ✗ 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
✗ are the known notions of well-formedness suitable?
✗ 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
- ✗ are the known notions of well-formedness suitable?
- ✗ 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
✗ are the known notions of well-formedness suitable?
✗ 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 $\mathsf{p}, \mathsf{p}', \ldots$

[1] Plus the old qualifier for state variables as in [3, 4]

Basic concepts and notation

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

[1] Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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 $\mathsf{R}, \mathsf{R}', \ldots$

[1] Plus the old qualifier for state variables as in [3, 4]

Basic concepts and notation
Participants $\mathsf{p}, \mathsf{p}', \ldots$
have roles $\mathsf{R}, \mathsf{R}', \ldots$
[1] Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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

[1] Plus the old qualifier for state variables as in [3, 4]

Basic concepts and notation

Participants p, p′, . . .

have roles R, R′, . . .

cooperate through a coordinator c which is

[1] Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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":

[1]Plus the old qualifier for state variables as in [3, 4]

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":

[1]Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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.)

[1] Plus the old qualifier for state variables as in [3, 4]

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.)
[1] Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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 $\mathsf{p}, \mathsf{p}', \dots$

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

cooperate through a coordinator c which is

basically an object with fields and "methods":

- $\mathsf{c.x}, \mathsf{c.y}, \dots$ represent sorted state variables 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

[1] Plus the old qualifier for state variables as in [3, 4]

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
1Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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.)
- $\mathsf{c}.\mathsf{f}, \mathsf{c}.\mathsf{f}', \ldots$ which are the functions operation admitted by c

[1] Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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> $\mathsf{p}, \mathsf{p}', \ldots$

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

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

basically an object with fields and "methods":

- $\mathsf{c}.\mathsf{x}, \mathsf{c}.\mathsf{y}, \ldots$ represent sorted <u>state variables</u> 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

<u>Assignment</u> $\mathsf{c}.\mathsf{x} := \mathsf{e}$ where e is a standard syntax of <u>pure</u> expressions[1]; let $B, B', \ldots$ range over finite sets of assignments where each variable can be assigned at most once

[1]Plus the old qualifier for state variables as in [3, 4]

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 [4] which, as explained in [3] 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[2]

[2] 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[2]

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

Data-Aware FSMs

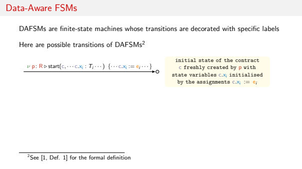
DAFSMs are finite-state machines whose transitions are decorated with specific labels

Here are possible transitions of DAFSMs[2]

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

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

[2] 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[2]

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

```
initial state of the contract
c freshly created by p with
state variables c.xi initialised
by the assignments c.xi := ei
```

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

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

[2] 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

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
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[2]

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

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

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

$$\circ \xrightarrow{\ell} \circledcirc$$

where g (ie a boolean
expression) is a guard and
$\pi ::= \nu\ \mathsf{p} \colon \mathsf{R} \mid \mathsf{any}\ \mathsf{p} \colon \mathsf{R} \mid \mathsf{p}$
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

Data-Aware FSMs
DAFSMs are finite-state machines whose transitions are decorated with specific labels
Here are possible transitions of DAFSMs[2]

Exercise: modelling

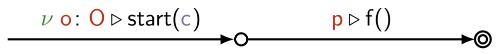
Give a DAFSM for the following contract protocol:

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

Not all DAFSMs “make sense”

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

free names

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

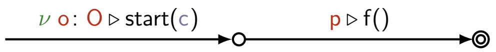
role emptyness

Not all DAFSMs “make sense”

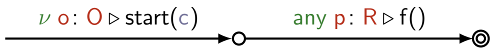
free names

role emptyness

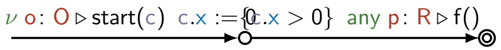
Not all DAFSMs "make sense"



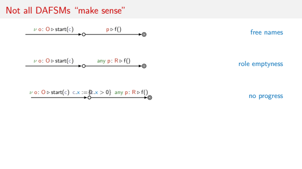
free names



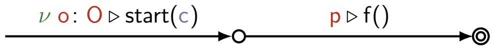
role emptyness



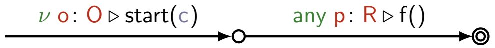
no progress



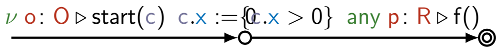
Not all DAFSMs "make sense"



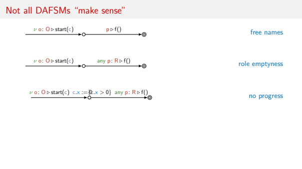
free names



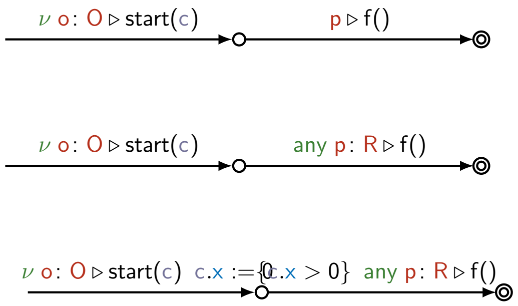
role emptyness



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.

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 declarations in function call, ν p: R, and any p: R

Closed DAFSMs

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

Closed DAFSMs

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

p is <u>bound</u> in $\circ \xrightarrow{\{\mathrm{g}\}\ \ \pi \triangleright \mathrm{f}(\cdots \mathrm{y}_i : T_i \cdots)\ \ B} \circ$ if, for some role R,

$\pi = \nu$ p: R or $\pi =$ any p: R or there is i s.t. $\mathrm{y}_i = \mathrm{p}$ and $T_i = \mathrm{R}$

Closed DAFSMs

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

p is bound in $\circ \xrightarrow{\{\mathrm{g}\}\ \ \pi \triangleright \mathrm{f}(\cdots \mathrm{y}_i : T_i \cdots)\ \ B} \circ$ if, for some role R,

$\pi = \nu$ p: R or $\pi =$ any p: R or there is i s.t. $\mathrm{y}_i = \mathrm{p}$ and $T_i = \mathrm{R}$

Closed DAFSMs

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

p is <u>bound</u> in $\circ \xrightarrow{\{g\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ$ if, for some role R,

$$\pi = \nu\ \mathsf{p} : \mathsf{R} \quad \text{or} \quad \pi = \text{any } \mathsf{p} : \mathsf{R} \quad \text{or} \quad \text{there is } i \text{ s.t. } \mathsf{y}_i = \mathsf{p} \text{ and } T_i = \mathsf{R}$$

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

Closed DAFSMs

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

p is bound in $\circ \xrightarrow{\{g\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ$ if, for some role R,

$\pi = \nu$ p: R or $\pi =$ any p: R or there is i s.t. $\mathsf{y}_i = \mathsf{p}$ and $T_i = \mathsf{R}$

The occurrence of p is bound in a path σ $\circ \xrightarrow{\{g\}\ \mathsf{p} \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ$ σ' if p is bound in a transition of σ

Closed DAFSMs

Binders: parameter declarations in function call, $\nu\ \mathsf{p} \colon \mathsf{R}$, and $\mathsf{any}\ \mathsf{p} \colon \mathsf{R}$

p is bound in $\circ \xrightarrow{\{g\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ$ if, for some role R,

$$\pi = \nu\ \mathsf{p} \colon \mathsf{R} \quad \text{or} \quad \pi = \mathsf{any}\ \mathsf{p} \colon \mathsf{R} \quad \text{or} \quad \text{there is } i \text{ s.t. } \mathsf{y}_i = \mathsf{p} \text{ and } T_i = \mathsf{R}$$

The occurrence of p is bound in a path $\sigma \circ \xrightarrow{\{g\}\ \mathsf{p} \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ\ \sigma'$ 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

Closed DAFSMs

Binders: parameter declarations in function call, $\nu\ \mathsf{p} \colon \mathsf{R}$, and $\mathsf{any}\ \mathsf{p} \colon \mathsf{R}$

p is bound in $\circ \xrightarrow{\{g\}\ \pi \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ$ if, for some role R,

$\pi = \nu\ \mathsf{p} \colon \mathsf{R}$ or $\pi = \mathsf{any}\ \mathsf{p} \colon \mathsf{R}$ or there is i s.t. $\mathsf{y}_i = \mathsf{p}$ and $T_i = \mathsf{R}$

The occurrence of p is bound in a path $\sigma \circ \xrightarrow{\{g\}\ \mathsf{p} \triangleright \mathsf{f}(\cdots \mathsf{y}_i : T_i \cdots)\ B} \circ\ \sigma'$ 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 $\circ \xrightarrow{\{g\}\ \pi \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ 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 σ $\circ \xrightarrow{\{g\}\ \text{any p: R} \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ σ' 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

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 σ $\circ \xrightarrow{\{g\}\ \text{any p: R} \triangleright f(\cdots y_i : T_i \cdots)\ B} \circ$ σ' 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 $X = \{\mathrm{c.x_1}, \ldots, \mathrm{c.x_n}\}$ is <u>consistent</u> if it is closed and the following implication holds for each transition $\circ \xrightarrow{\{\mathrm{g}\}\ \pi \triangleright \mathrm{f}(\cdots \mathrm{y}_i : T_i \cdots)\ B} \mathrm{(s)}$

$$\forall(\mathrm{c.x}, \mathrm{old\ c.x})_{\mathrm{c.x} \in X}\ \exists(y)_{y \in Y}\ :\ (\mathrm{g}\{\mathrm{old\ c.x}/\mathrm{c.x}\}_{\mathrm{c.x} \in X}\ \wedge\ \mathrm{g}_B) \implies \mathrm{g_s} \qquad \text{where}$$

$$Y = \{y \mid \exists i\ :\ y = y_i \text{ or } y \text{ is a parameter of an outgoing transition of s}\}$$

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

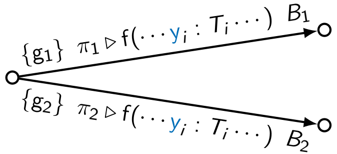
$$\mathrm{g}_B = \bigwedge_{(\mathrm{c.x} := \mathrm{e}) \in B} \mathrm{c.x} = \mathrm{e}\ \wedge \bigwedge_{\mathrm{c.x} \notin B} \mathrm{c.x} = \mathrm{old\ c.x}$$

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

Determinism

A DAFSM is deterministic if

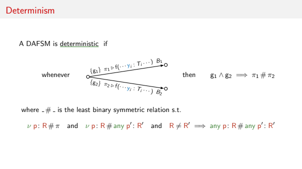
whenever



then $g_1 \wedge g_2 \implies \pi_1 \# \pi_2$

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

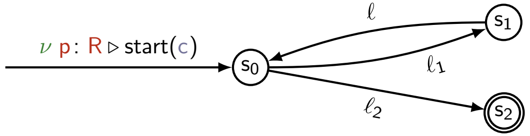
$\nu\, p\colon R \# \pi$ and $\nu\, p\colon R \# \text{any } p'\colon R'$ and $R \neq R' \implies \text{any } p\colon R \# \text{any } p'\colon R'$



transitions from the same source state and calling the same function

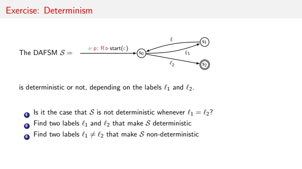
Exercise: Determinism

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 whenever $\ell_1 = \ell_2$?
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. $\ell_1 = \ell_2 =$ ν p: R $\triangleright$ 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. $\ell_1 = \{x \leq 0\}$ p $\triangleright$ f(x : Int) and $\ell_2 = \{x \geq -1\}$ p $\triangleright$ 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

Well-formedness

A DAFSM is well-formed when it is

empty-role free

consistent, and

deterministic

Well-formedness

A DAFSM is well-formed when it is

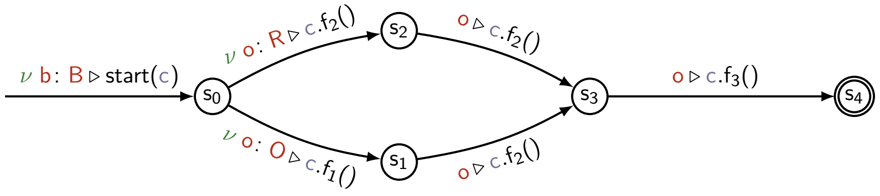
empty-role free

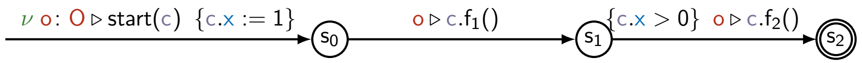
consistent, and

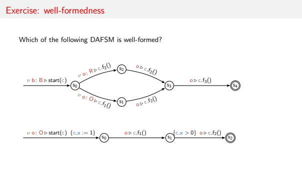
deterministic

Exercise: well-formedness

Which of the following DAFSM is well-formed?







yes: o is defined on paths it occurs on and the DAFSM is deterministic.

no: the transition from s_0 violates **??** since True does not imply $c.x > 0$ hinting that the protocol could get stuck in state s_1. However, this never happens because $c.x$ is initially set to 1 and never changed, hence the transition from s_1 would be enabled when the protocol lands in s_1.

– Act II –

[A tool]

– Act II –
[A tool]

Verification

Checking well-formedness by hand is laborious and cumbersome

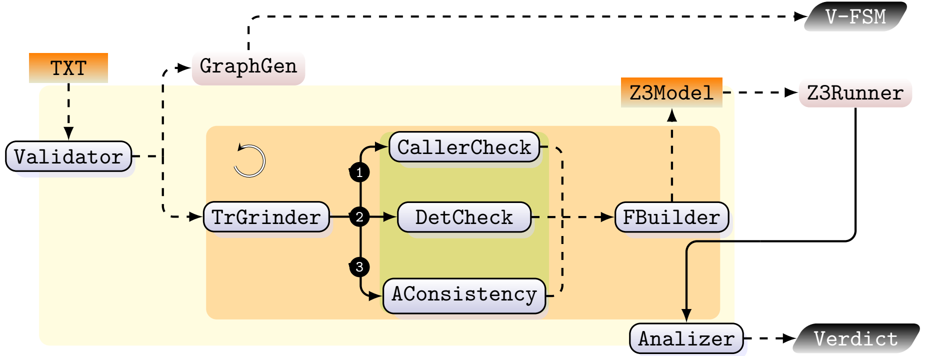
So we implemented **TRAC**, which
- transforms DAFSMs in a DSL to specify DAFSMs
- verifies well-formedness condition relying on the SMT solver Z3

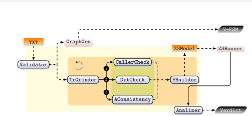
Verification

Checking well-formedness by hand is laborious and cumbersome

So we implemented **TRAC**, which
transforms DAFSMs in a DSL to specify DAFSMs
verifies well-formedness condition relying on the SMT solver Z3

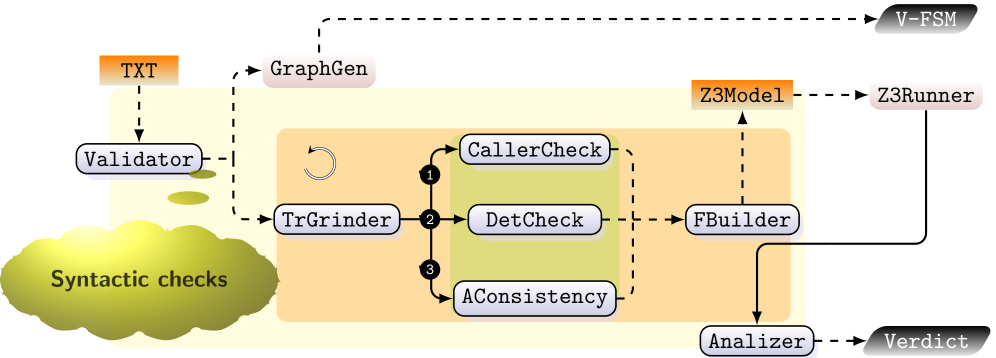
The architecture of **TRAC**





the architecture of **TRAC** is compartmentalised into two principal modules:
parsing and visualisation (yellow box) and
TRAC's core (orange box). The latter module implements well-formedness check (green box).
Solid arrows represent calls between components while dashed arrows data IO.

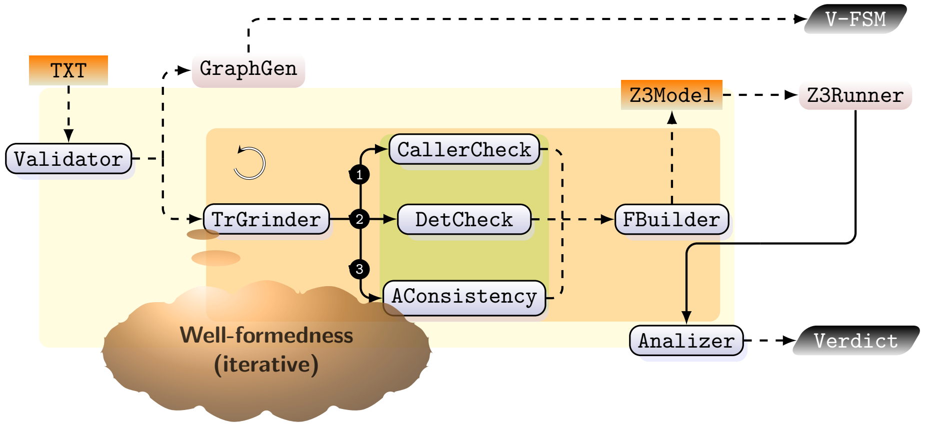
The architecture of **TRAC**



basic syntactic checks on a DSL representation of DAFSMs and transforming the input in a format that simplifies the analysis of the following phases:

- passed to `GraphGen` for visual representation of DAFSMs (`V-FSM` output)
- passed to the `TrGrinder` component (orange box) for well-formedness checking.

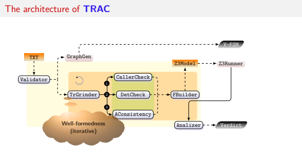
The architecture of **TRAC**



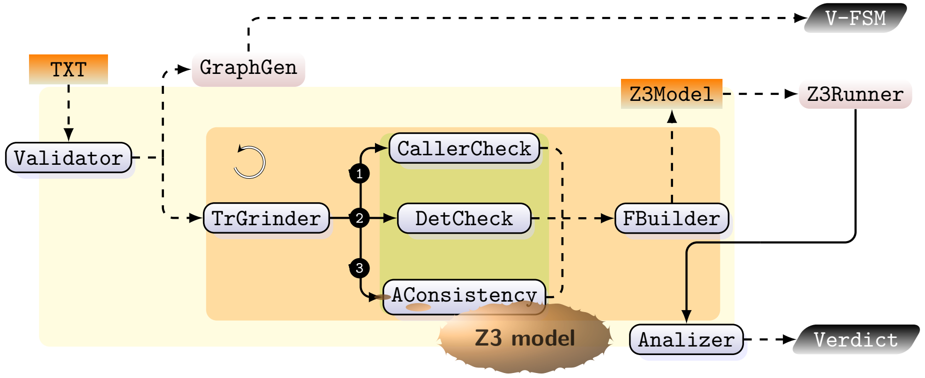
Automata for smart contracts...and more

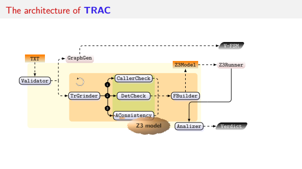
2025-01-08

└─The architecture of **TRAC**



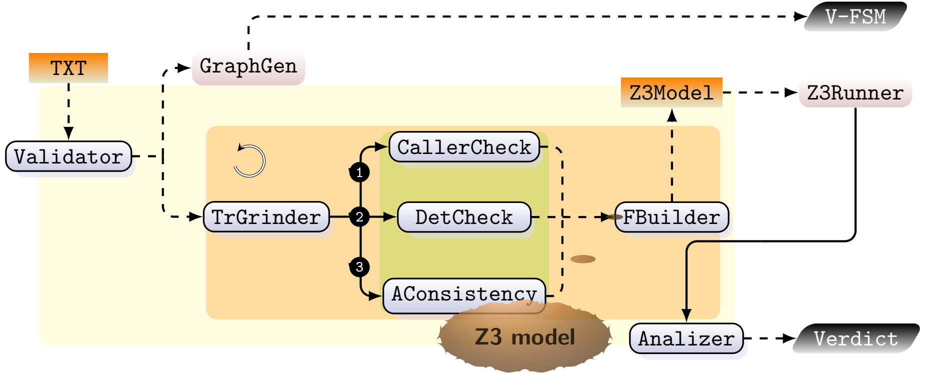
The architecture of **TRAC**

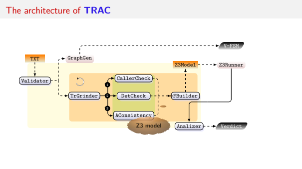




AConsistency (arrow 3) to generate a Z3 formula which holds if, and only if, the transtion is consistent.

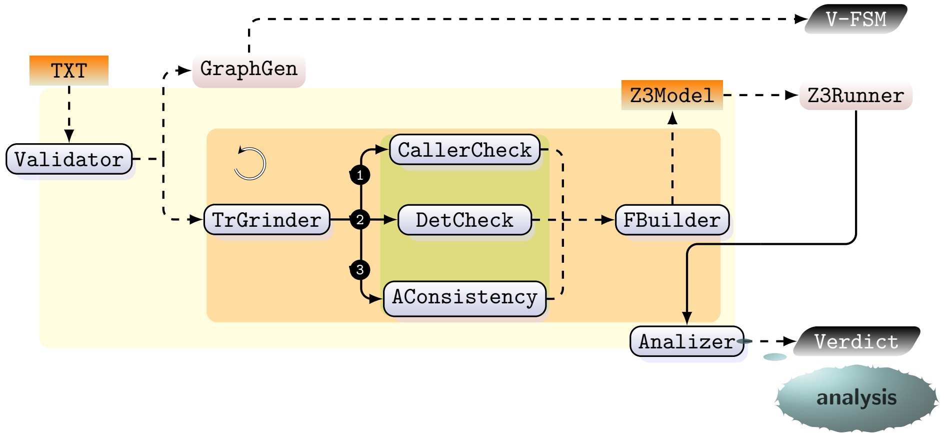
The architecture of **TRAC**

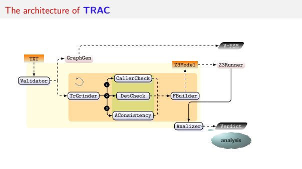




computes the z3 f.la equivalent to the conjunction of the outputs which is then passed to a Z3 engine to check its satisfiability

The architecture of **TRAC**





Finally, the `Analizer` component that diagnoses the output of Z3 and produces a `Verdict` which reports (if any) the violations of well-formedness of the DAFSM in input.

Installation

Detailed instructions at `https://github.com/loctet/TRAC`

Dependencies: Java RE (to render DAFSM graphically) & Python 3.6 or later

```
pip install z3-solver matplotlib numpy plotly pandas networkx
```


Installation

Detailed instructions at https://github.com/loctet/TRAC

Dependencies: Java RE (to render DAFSM graphically) & Python 3.6 or later

pip install z3-solver matplotlib numpy plotly pandas networkx

forse servono solo solo z3-solver e networkx

https://doi.org/10.5281/zenodo.10996456

– Act III –

[A little exercise]

2025-01-08

Automata for smart contracts...and more

– Act III –

[A little exercise]

– Epilogue –

[Work in progress]

Thank you

Thank you

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] B. Meyer. *Introduction to the Theory of Programming Languages*.
Prentice-Hall, 1990.

2025-01-08

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] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice-Hall, 1990.

References II

[4] B. Meyer. *Eiffel: The Language*.
Prentice-Hall, 1991.

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

[6] 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] B. Meyer. *Eiffel: The Language*.
Prentice-Hall, 1991.

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

[6] 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.