# A Choreographic View of Smart Contracts

Elvis Gerardin Konjoh Selabi
@GSSI & UniCam

Maurizio Murgia
@GSSI

António Ravara
@NOVA

Emilio Tuosto
@ GSSI

A tutorial @ FORTE 2025, Lille

# What's up doc?

# What's up doc?

# What's up doc?

# What's up doc?

– Prologue –

[ An inspiring initiative ]

# A nice sketch! [6, 7]

A smart contract among Owners and Buyers



Simple Marketplace State Transitions

Application Roles:
- Owner (O)
- Buyer (B)

Legend
- TF: Transition Function
- AR: Allowed Role
- AIR: Allowed Instance Role
- ▬ A Happy Path
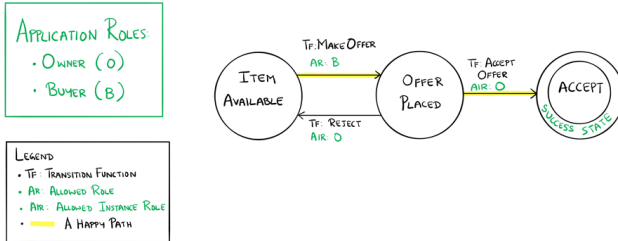
**initially** buyers can make offers
**then**
    **either** an owner can accept an offer and the protocol stops
    **or** the offer is rejected and the protocol restarts

# What did we just see?

A <u>smart contract</u> looks like

    a <u>choreographic model</u>
*global specifications determine the enabled actions along the evolution of the protocol*

    a <u>typestate</u>
*In OOP, "can reflects how the legal operations on imperative objects can change at runtime as their internal state changes." [3]*

# A new coordination model

So, we saw an interesting model where

distributed components coordinate through a global specification

which specifies how actions are enabled along the computation

"without forcing" components to be cooperative!
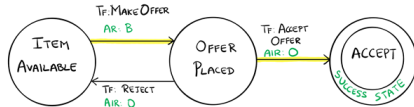
# Let's look at our sketch again

# Let's look at our sketch again



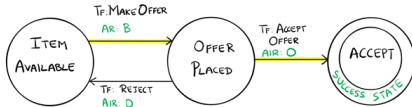Simple Marketplace State Transitions

Application Roles:
· Owner (O)
· Buyer (B)

Legend
· TF: Transition Function
· AR: Allowed Role
· AIR: Allowed Instance Role
· ▬▬ A Happy Path

but…

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

✗ can buyers be owners too?

✗ what's the scope of quantifications?

✗ when are transitions enabled?

✗ how does the state of the contract change?

# Let's go formal!

Our first attempt was to "look for into our toolbox", but

    ✗ are known notions of well-formedness suitable?

    ✗ data-awareness is crucial

    ✓ we got roles okay, but

    ✗ limitations on instances of roles

    ✗ instances can have one role only

## Let's go formal!

Our first attempt was to "look for into our toolbox", but

    ✗ are known notions of well-formedness suitable?

    ✗ data-awareness is crucial

    ✓ we got roles okay, but

    ✗ limitations on instances of roles

    ✗ instances can have one role only

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

# ...and by the way



Bug-free programming is a difficult task and a fundamental challenge for critical systems. To this end, formal methods provide techniques to develop programs and certify their correctness.

https://medium.com/@teamtech/formal-verification-of-smart-contracts-trust-in-the-making-2745a60ce9db



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

– Act I –

[ A coordination framework ]

# Basic concepts and notation

<u>Participants</u>  $p, p', \ldots$

# Basic concepts and notation

Participants  $p, p', \ldots$
  have roles  $R, R', \ldots$

# Basic concepts and notation

Participants  $p, p', \ldots$
  have roles  $R, R', \ldots$
    and cooperate through a coordinator  $c$

# Basic concepts and notation

Participants $p, p', \ldots$
    have roles $R, R', \ldots$
        and cooperate through a coordinator $c$
            which can be thought of as an object with "fields" and "methods":

# Basic concepts and notation

Participants  $p, p', \ldots$

have roles  $R, R', \ldots$

and cooperate through a coordinator  c

which can be thought of as an object with "fields" and "methods":

$u, v, \ldots$ represent sorted state variables  of c (sorts include data types such as 'int', 'bool', etc. as well as participants' roles)

# Basic concepts and notation

Participants $p, p', \ldots$
  have roles $R, R', \ldots$
    and cooperate through a coordinator $c$
      which can be thought of as an object with "fields" and "methods":

  $u, v, \ldots$ represent sorted state variables of $c$ (sorts include data types such as 'int', 'bool', etc. as well as participants' roles)

  $f, g, \ldots$ represent the operations admitted by $c$

# Basic concepts and notation

Participants p, p', …
  have roles R, R', …
    and cooperate through a coordinator c
      which can be thought of as an object with "fields" and "methods":

  u, v, … represent sorted state variables of c (sorts include data types such as 'int', 'bool', etc. as well as participants' roles)

  f, g, … represent the operations admitted by c

  u := e is an assignment which updates the state variable u to a pure expression e on
    - function parameters
    - state variables u or old u (representing the value of u before the assignment) [4, 5]

# Basic concepts and notation

Participants   p, p′, . . .
  have <u>roles</u>   R, R′, . . .
    and cooperate through a <u>coordinator</u>   c
      which can be thought of as an object with "fields" and "methods":
  u, v, . . . represent sorted <u>state variables</u>   of c (sorts include data types such as
        'int', 'bool', etc. as well as participants' roles)
  f, g, . . . represent the operations admitted by c
  u := e  is an <u>assignment</u>   which updates the state variable u to a <u>pure</u>
        expression e on
            - function parameters
            - state variables u or old u (representing the value of u before the
        assignment) [4, 5]
  A, A′, . . . range over finite sets of assignments where each variable can be assigned
        at most once

## Data-Aware FSMs

A DAFSM $c$ on roles $R_1, \ldots R_m$ and <u>state variables</u> $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant $p$ whose transitions are decorated as follows[1]

---

[1]See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

# Data-Aware FSMs

A DAFSM $c$ on roles $R_1, \ldots R_m$ and <u>state variables</u> $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant $p$ whose transitions are decorated as follows[1]

$$\{\gamma\} \; \mathsf{new} \; \mathsf{p} \colon \mathsf{R} \triangleright \mathsf{start}(\mathsf{c}, \cdots, T_i \, \mathsf{x}_i, \cdots) \; \{\cdots \mathsf{u}_j := \mathsf{e}_j \cdots\} \longrightarrow \!\!\! \circ$$

$c$ is freshly created by $p$ which also initialises state variables $u_j$ with expressions $e_j$ which are built on state variables and parameters $x_i$

---

[1]See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

# Data-Aware FSMs

A DAFSM $c$ on roles $R_1, \ldots R_m$ and <u>state variables</u> $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant $p$ whose transitions are decorated as follows[1]

$$\{\gamma\} \text{ new } p\colon R \rhd \text{start}(c, \cdots, T_i\ x_i, \cdots) \{\cdots u_j := e_j \cdots\}$$

$c$ is freshly created by $p$ which also initialises state variables $u_j$ with expressions $e_j$ which are built on state variables and parameters $x_i$
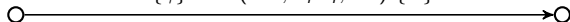
$$\{\gamma\} \ \pi \rhd \text{f}(\cdots, T_i\ x_i, \cdots) \{A\}$$
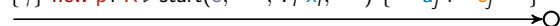
where $\gamma$ is a guard (ie a boolean expression) and
$$\pi ::= \text{new } p\colon R \ \mid \ \text{any } p\colon R \ \mid \ p$$
is a <u>qualified participant</u> calling f with parameters $x_i$
state variables are reassigned according to $A$ if the invocation is successful

---

[1]See [1, Def. 1]; here we just simplified the notation and adapted it to our needs
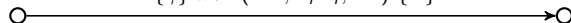
# Data-Aware FSMs

A DAFSM $c$ on roles $R_1, \ldots R_m$ and <u>state variables</u> $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant $p$ whose transitions are decorated as follows[1]

$$\xrightarrow{\{\gamma\} \; \text{new } p\colon R \rhd \text{start}(c, \cdots, T_i \; x_i, \cdots) \; \{\cdots u_j := e_j \cdots\}} \circ$$
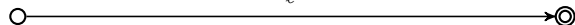
$c$ is freshly created by $p$ which also initialises state variables $u_j$ with expressions $e_j$ which are built on state variables and parameters $x_i$

$$\circ \xrightarrow{\{\gamma\} \; \pi \rhd f(\cdots, T_i \; x_i, \cdots) \; \{A\}} \circ$$

where $\gamma$ is a guard (ie a boolean expression) and
$$\pi \;::=\; \text{new } p\colon R \;\mid\; \text{any } p\colon R \;\mid\; p$$
is a <u>qualified participant</u> calling f with parameters $x_i$ state variables are reassigned according to $A$ if the invocation is successful

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

accepting states are denoted as usual

---

[1]See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

Give a DAFSM for the protocol on slide 7 resolving the ambiguities discussed there.

# Not all DAFSMs "make sense"



new o : O ▷ start(c)        p ▷ f()
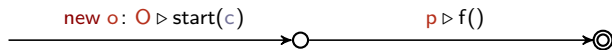
names' freeness

# Not all DAFSMs "make sense"
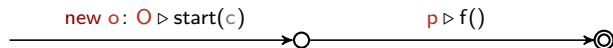


new o: $O \triangleright$ start(c)      $p \triangleright f()$      names' freeness

new o: $O \triangleright$ start(c)      any p: $R \triangleright () \{A\}$      role emptyness

# Not all DAFSMs "make sense"

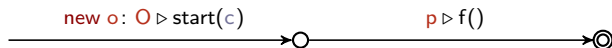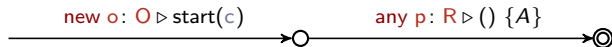$$\xrightarrow{\text{new o}: \mathsf{O} \triangleright \mathsf{start}(\mathsf{c})} \bigcirc \xrightarrow{\mathsf{p} \triangleright \mathsf{f}()} \circledcirc \qquad \text{names' freeness}$$

$$\xrightarrow{\text{new o}: \mathsf{O} \triangleright \mathsf{start}(\mathsf{c})} \bigcirc \xrightarrow{\text{any p}: \mathsf{R} \triangleright () \{A\}} \circledcirc \qquad \text{role emptyness}$$

$$\xrightarrow{\text{new o}: \mathsf{O} \triangleright \mathsf{start}(\mathsf{c}) \{\mathsf{u} := 0\}} \bigcirc \xrightarrow{\{\mathsf{u} > 0\} \text{ new p}: \mathsf{R} \triangleright \mathsf{f}()} \circledcirc \qquad \text{no progress}$$

# Not all DAFSMs "make sense"

new o: $O \triangleright \mathsf{start}(c)$      $p \triangleright f()$      names' freeness

new o: $O \triangleright \mathsf{start}(c)$      any $p: R \triangleright () \{A\}$      role emptyness

new o: $O \triangleright \mathsf{start}(c) \{u := 0\}$      $\{u > 0\}$ new $p: R \triangleright f()$      no progress
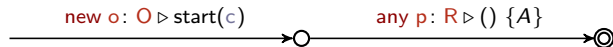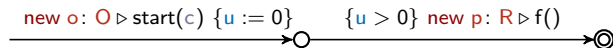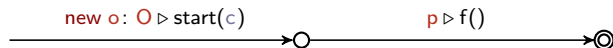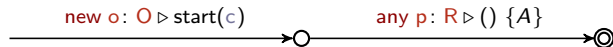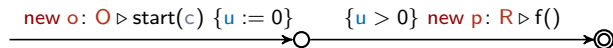
# Not all DAFSMs "make sense"
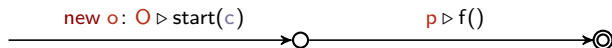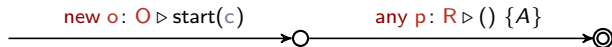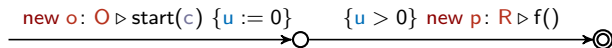


Save names' freeness, the other properties are undecidable in general, so we'll look for sufficient conditions to rule out nonsensical DAFSMs

# Closed DAFSMs

Binders:  parameter declarations in function calls, new $p$: R, and any $p$: R

# Closed DAFSMs

Binders:  parameter declarations in function calls, new p: R, and any p: R

p is <u>bound</u> in  ◯ —————$\{\gamma\}\ \pi \triangleright f(\cdots, T_i\ x_i, \cdots)\ \{A\}$————→◯  if, for some role R,

$\pi = $ new p: R   or   $\pi = $ any p: R   or   there is $i$ s.t. $x_i = p$ and $T_i = R$

# Closed DAFSMs

Binders: parameter declarations in function calls, new $p\colon R$, and any $p\colon R$

p is <u>bound</u> in $\bigcirc \xrightarrow{\{\gamma\}\ \pi \triangleright f(\cdots,\ T_i\ x_i,\cdots)\ \{A\}} \bigcirc$ if, for some role $R$,

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

The occurrence of p is <u>bound</u> in a path

$$\sigma \quad \bigcirc \xrightarrow{\{\gamma\}\ p \triangleright f(\cdots)\ \{A\}} \bigcirc \quad \cdots$$
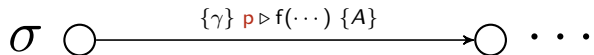
if p is bound in a transition of $\sigma$

# Closed DAFSMs

Binders:  parameter declarations in function calls, new $p\colon R$, and any $p\colon R$

p is <u>bound</u> in $\bigcirc \xrightarrow{\{\gamma\}\ \pi \rhd f(\cdots,\, T_i\, x_i, \cdots)\ \{A\}} \bigcirc$ if, for some role $R$,

$\pi = $ new $p\colon R$   or   $\pi = $ any $p\colon R$   or   there is $i$ s.t. $x_i = p$ and $T_i = R$

The occurrence of p is <u>bound</u> in a path

$$\sigma\ \bigcirc \xrightarrow{\{\gamma\}\ p \rhd f(\cdots)\ \{A\}} \bigcirc \ \cdots$$
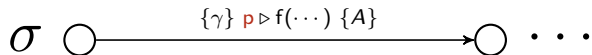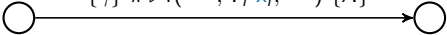
if p is bound in a transition of $\sigma$

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

# Role emptyness

A transition $\bigcirc \xrightarrow{\{\gamma\}\ \pi \triangleright f(\cdots, T_i\ x_i, \cdots)\ \{A\}} \bigcirc$ expands role R if
$\pi = \mathsf{new}\ \mathsf{p}\colon \mathsf{R}$ or there is $i$ s.t. $x_i = \mathsf{p}$ and $T_i = \mathsf{R}$

Role R is expanded in a path

$$\sigma\ \bigcirc \xrightarrow{\{\gamma\}\ \mathsf{any}\ \mathsf{p}\colon \mathsf{R} \triangleright f(\cdots)\ \{A\}} \bigcirc \ \cdots$$
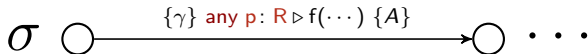
if a transition in $\sigma$ expands R

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

Is the DAFSM below empty-role free?
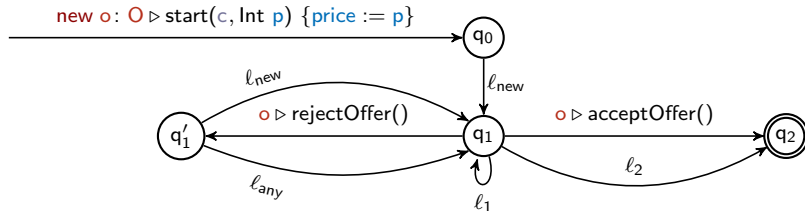


where
$\ell_{new} = \{newOffer > 0\}$ new $b \colon B \triangleright makeOffer(Int\ newOffer)\ \{offer := newOffer\}$,
$\ell_{any} = \{newOffer > 0\}$ any $b \colon B \triangleright makeOffer(Int\ newOffer)\ \{offer := newOffer\}$,
$\ell_1 =$ new $p \colon P \triangleright join()$
and $\ell_2 = \{p > price\}$ any $p \colon P \triangleright buy(Int\ p)$ .
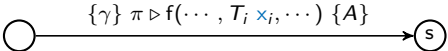
# Progress

A DAFSM with state variables $u_1, \ldots, u_n$ is <u>consistent</u> if

for each $\bigcirc \xrightarrow{\{\gamma\} \, \pi \triangleright f(\cdots, T_i \, x_i, \cdots) \, \{A\}} (s)$

$\mathbb{V}_U \, \overline{\mathbb{H}}_X \, (\gamma\{\text{old } u_1, \ldots, \text{old } u_n / u_1, \ldots, u_n\} \, \wedge \, \gamma_A \implies \bigvee_{1 \leq j \leq m} \overline{\mathbb{H}}_{Y_j} \, \gamma_j)$ is satisfiable

where

# Progress

A DAFSM with state variables $u_1, \ldots, u_n$ is <u>consistent</u>   if

for each

$$\bigcirc \xrightarrow{\{\gamma\}\ \pi \triangleright f(\cdots, T_i\ x_i, \cdots)\ \{A\}} \ \text{\small s}$$

$\forall_U\ \boxminus_X\ (\gamma\{\text{old } u_1, \ldots, \text{old } u_n/u_1, \ldots, u_n\}\ \wedge\ \gamma_A \implies \bigvee_{1 \le j \le m} \boxminus_{Y_j}\ \gamma_j)$   is satisfiable

where

$U = \{u_i, \text{old } u_i\}_{1 \le i \le n}$

$X = \{x \mid \exists i : x = x_i\}$

$\gamma_A = \bigwedge_{u := e \in A} u = e\ \wedge\ \bigwedge_{u \notin A} u = \text{old } u$

$Y_j = \{x \mid x \text{ is a parameter of the } j^{\text{th}} \text{ outgoing transition of s}\}$

$\gamma_j = \begin{cases} \text{the guard of the } j^{\text{th}} \text{ outgoing transitions of s} & \text{if s not accepting} \\ \text{True} & \text{otherwise} \end{cases}$

A DAFSM with state variables $u_1, \ldots, u_n$ is <u>consistent</u> if

for each $\bigcirc \xrightarrow{\{\gamma\} \, \pi \triangleright f(\cdots, T_i \, x_i, \cdots) \, \{A\}} \, (s)$
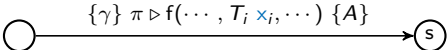
$\forall_U \, \exists\!\!\!\exists_X \, (\gamma\{\text{old } u_1, \ldots, \text{old } u_n / u_1, \ldots, u_n\} \, \wedge \, \gamma_A \implies \bigvee_{1 \leq j \leq m} \exists\!\!\!\exists_{Y_j} \, \gamma_j)$ is satisfiable and

there is $\xrightarrow{\{\gamma\} \, \pi \triangleright \text{start}(\cdots, T_i \, x_i, \cdots) \, \{A\}} \, (s)$ such that $\exists\!\!\!\exists_X \, \gamma$ is satisfiable

where

$U = \{u_i, \text{old } u_i\}_{1 \leq i \leq n}$

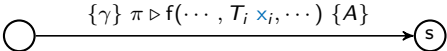$X = \{x \mid \exists i : x = x_i\}$

$\gamma_A = \bigwedge_{u := e \in A} u = e \; \wedge \bigwedge_{u \notin A} u = \text{old } u$

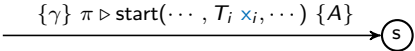$Y_j = \{x \mid x \text{ is a parameter of the } j^{\text{th}} \text{ outgoing transition of s}\}$

$\gamma_j = \begin{cases} \text{the guard of the } j^{\text{th}} \text{ outgoing transitions of s} & \text{if s not accepting} \\ \text{True} & \text{otherwise} \end{cases}$

Is the DAFSM below consistent?
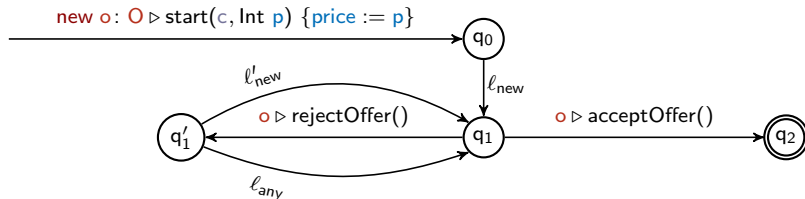


where

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

$\ell'_{\text{new}} = \{\text{newOffer} \geq price * 1.05\}$ new $b\colon B \triangleright$ makeOffer(Int newOffer) $\{\text{offer} := \text{newOffer}\}$, and
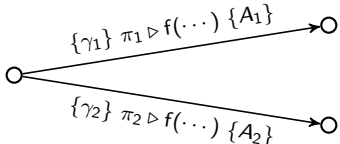
$\ell_{\text{any}} = \{\text{newOffer} \geq price * 1.05\}$ any $b\colon B \triangleright$ makeOffer(Int newOffer) $\{\text{offer} := \text{newOffer}\}$

# Determinism

Let $\_\#\_$ be the least binary symmetric relation s.t.

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

A DAFSM is <u>deterministic</u>  if



whenever $\quad\quad$ then $\quad\quad \gamma_1 \wedge \gamma_2 \implies \pi_1\#\pi_2$

where the diagram shows two transitions from a single state: $\{\gamma_1\}\ \pi_1 \triangleright f(\cdots)\ \{A_1\}$ and $\{\gamma_2\}\ \pi_2 \triangleright f(\cdots)\ \{A_2\}$

The DAFSM $\mathcal{S} = $ 
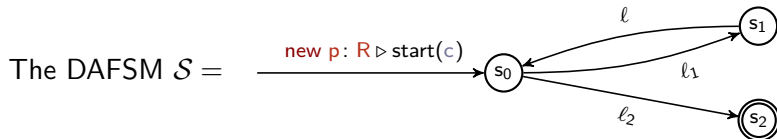
is deterministic or not, depending on the labels $\ell_1$ and $\ell_2$.

1. Is it the case that $\mathcal{S}$ is not deterministic whenever $\ell_1 = \ell_2$?

2. Find two labels $\ell_1$ and $\ell_2$ that make $\mathcal{S}$ deterministic

3. Find two labels $\ell_1 \neq \ell_2$ that make $\mathcal{S}$ non-deterministic

# Well-formedness

A DAFSM is <u>well-formed</u> when it is

closed,

empty-role free,

consistent, and

deterministic

Which of the following DAFSM is well-formed?
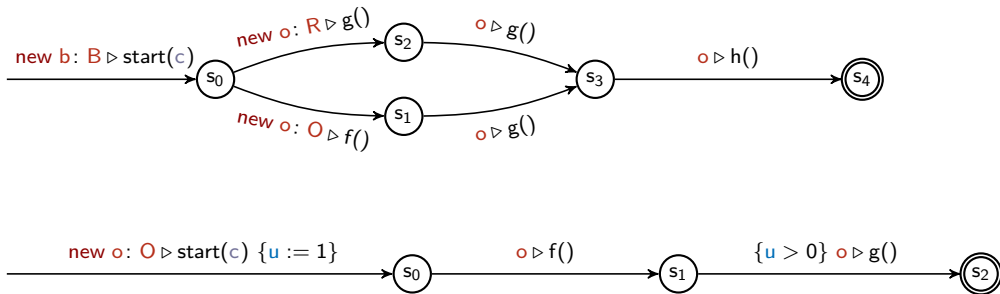
– Act II –

[ A tool ]

# Verification

Checking well-formedness by hand is laborious and cumbersome (and boring)

So we implemented **TRAC**, which

✓ features a DSL to specify DAFSMs

✓ verifies well-formedness (relying on the SMT solver Z3)

✓ it's efficient enough

✗ but cannot handle roles and inter-contract interactions

# The architecture of **TRAC**
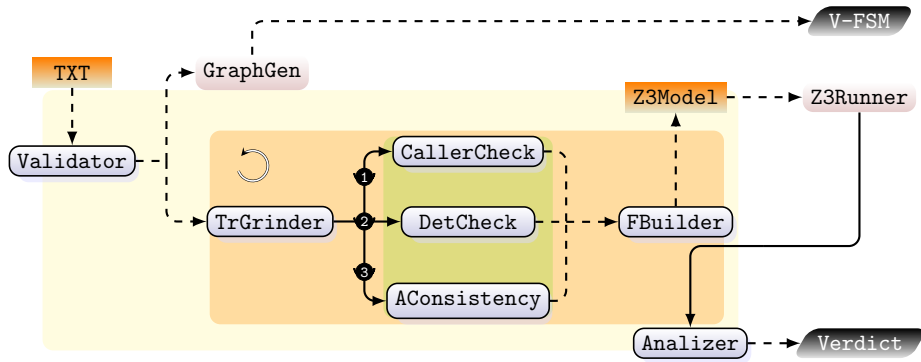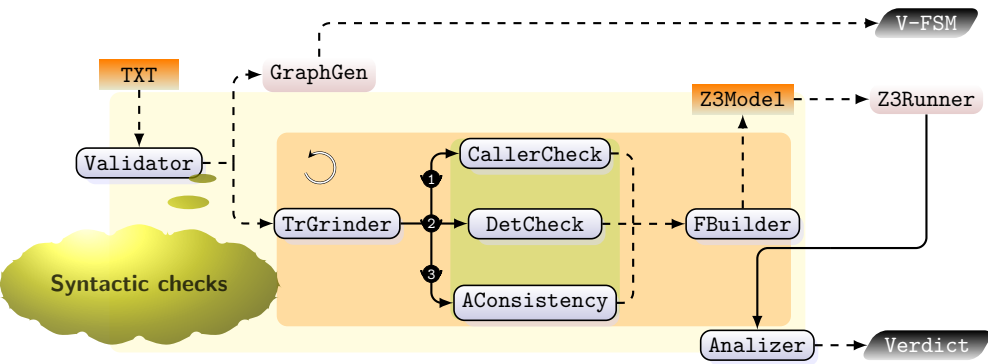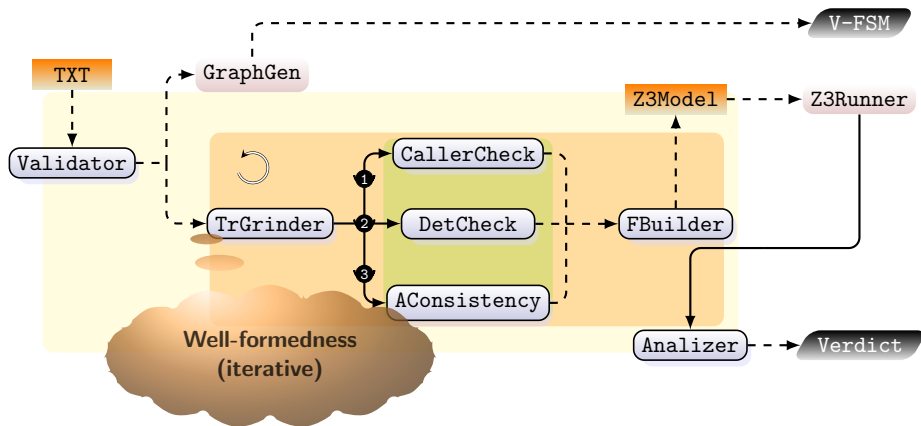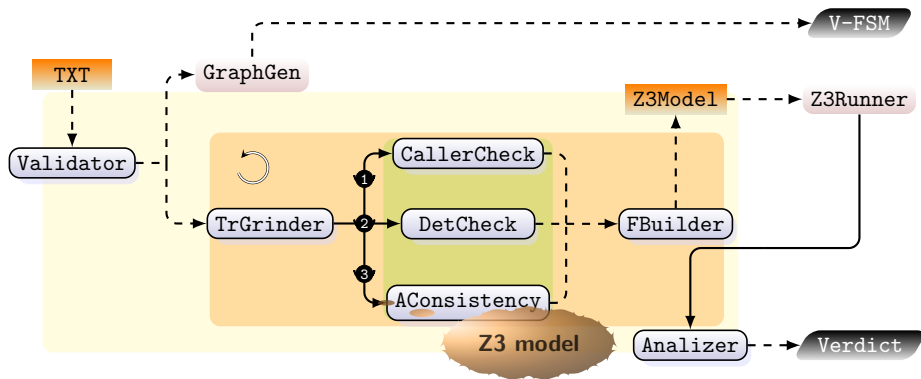
# The architecture of **TRAC**
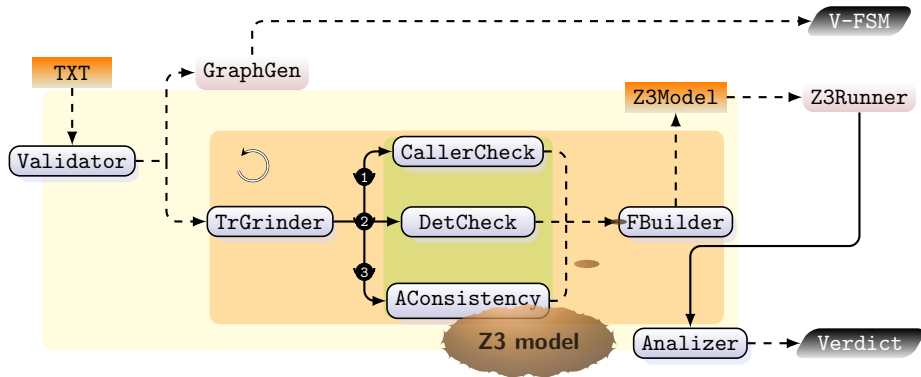
# The architecture of **TRAC**
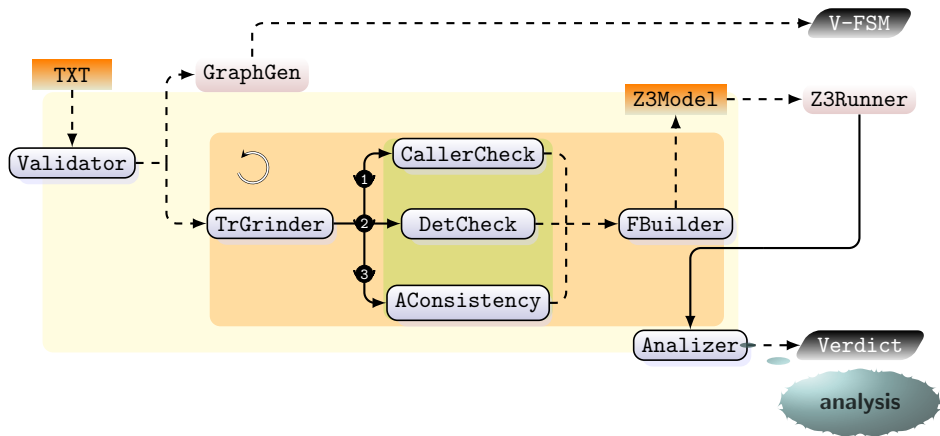
# Installation

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 networkx
```

# Concrete syntax (I)

$$\langle pars \rangle \ ::= \ \varepsilon \ \mid \ \langle dcl \rangle(,\langle dcl \rangle)^\star \qquad\qquad \langle dcl \rangle \ ::= \ \langle str \rangle \ \langle str \rangle$$

roles $\langle str \rangle^+$                                          # set the roles
dafsm $\langle str \rangle(\langle pars \rangle)$ by $\langle dcl \rangle \{$      # $\langle dcl \rangle$ declares the participant creating the contract

# Concrete syntax (I)

$\langle pars \rangle ::= \varepsilon \mid \langle dcl \rangle (,\langle dcl \rangle)^{\star}$
$\qquad\qquad\qquad\qquad\qquad$ $\langle dcl \rangle ::= \langle str \rangle \langle str \rangle$

roles $\langle str \rangle^{+}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ # set the roles
dafsm $\langle str \rangle(\langle pars \rangle)$ by $\langle dcl \rangle${ $\qquad$ # $\langle dcl \rangle$ declares the participant creating the contract
  $\vdots$
  $\langle dcl \rangle$ = e ; $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ # state variables with initial assignment (if any)

# Concrete syntax (I)

$$\langle pars \rangle \ ::= \ \varepsilon \ \mid \ \langle dcl \rangle (,\langle dcl \rangle)^\star \qquad\qquad \langle dcl \rangle \ ::= \ \langle str \rangle \ \langle str \rangle$$

```
roles ⟨str⟩⁺                                                            # set the roles
dafsm ⟨str⟩(⟨pars⟩) by ⟨dcl⟩{          # ⟨dcl⟩ declares the participant creating the contract
  ⋮
  ⟨dcl⟩ = e ;                              # state variables with initial assignment (if any)
  ⋮
  if γ                                     # initial guard (this clause can be omitted)
}
```

# Concrete syntax (I)

$\langle pars \rangle ::= \varepsilon \mid \langle dcl \rangle (, \langle dcl \rangle)^\star$      $\langle dcl \rangle ::= \langle str \rangle \langle str \rangle$

$\langle lbl \rangle ::= \{\gamma\} \pi > \langle str \rangle (\langle pars \rangle) \{\langle asgs \rangle\}$

$\langle asgs \rangle ::= \varepsilon \mid \langle asg \rangle (; \langle asg \rangle)^\star$      $\langle asg \rangle ::= \langle str \rangle := \langle expr \rangle$

roles $\langle str \rangle^+$                                                          # set the roles

dafsm $\langle str \rangle (\langle pars \rangle)$ by $\langle dcl \rangle$ {              # $\langle dcl \rangle$ declares the participant creating the contract

 ⋮

 $\langle dcl \rangle =$ e ;                                       # state variables with initial assignment (if any)

 ⋮

 if $\gamma$                                                          # initial guard (this clause can be omitted)

 }

⋮

$\langle str \rangle \langle lbl \rangle \langle str \rangle$ ;                                     # the initial state defaults to the source state of the first transition

⋮                                                                      # final states are strings with a trailing ' + ' sign

Edit a .trac file for the contract specified at
https:
//github.com/Azure-Samples/blockchain/blob/master/blockchain-workben
ch/application-and-smart-contract-samples/basic-provenance/readme.md

# Concrete syntax (II)

The syntax of expressions (and hence of guards) follows the SMT-lib standard:

$\langle spec\_constant \rangle$ ::= $\langle numeral \rangle$ | $\langle decimal \rangle$ | $\langle hexadecimal \rangle$ | $\langle binary \rangle$ | $\langle string \rangle$

$\langle s\_expr \rangle$ ::= $\langle spec\_constant \rangle$ | $\langle symbol \rangle$ | $\langle reserved \rangle$ | $\langle keyword \rangle$
　　　　　 | ( $\langle s\_expr \rangle^*$ )

$\langle qual\_identifier \rangle$ ::= $\langle identifier \rangle$ | ( as $\langle identifier \rangle$ $\langle sort \rangle$ )

$\langle var\_binding \rangle$ ::= ( $\langle symbol \rangle$ $\langle term \rangle$ )

$\langle sorted\_var \rangle$ ::= ( $\langle symbol \rangle$ $\langle sort \rangle$ )

$\langle pattern \rangle$ ::= $\langle symbol \rangle$ | ( $\langle symbol \rangle$ $\langle symbol \rangle^+$ )

$\langle match\_case \rangle$ ::= ( $\langle pattern \rangle$ $\langle term \rangle$ )

$\langle term \rangle$ ::= $\langle spec\_constant \rangle$
　　　　 | $\langle qual\_identifier \rangle$
　　　　 | ( $\langle qual\_identifier \rangle$ $\langle term \rangle^+$ )
　　　　 | ( let ( $\langle var\_binding \rangle^+$ ) $\langle term \rangle$ )
　　　　 | ( lambda ( $\langle sorted\_var \rangle^+$ ) $\langle term \rangle$ )
　　　　 | ( forall ( $\langle sorted\_var \rangle^+$ ) $\langle term \rangle$ )
　　　　 | ( exists ( $\langle sorted\_var \rangle^+$ ) $\langle term \rangle$ )
　　　　 | ( match $\langle term \rangle$ ( $\langle match\_case \rangle^+$ ) )
　　　　 | ( ! $\langle term \rangle$ $\langle attribute \rangle^+$ )

(borrowed from [2])

HIC SUNT LEONES

probably not needed

Edit a .trac file for the DAFSM on slide 13.

– Act III –

[ A little exercise ]

– Epilogue –

[ Work in progress ]

# Work in progress

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 *LNCS*, pages 239–257. Springer, 2024.

[2] C. Barrett, P. Fontaine, and C. Tinelli. *The SMT-LIB Standard*, version 2.7 edition, 2025.

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

# References II

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

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

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

[7] 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.