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

Work partly supported by the PRIN 2022 PNRR project DeLiCE (F53D23009130001)

What's up doc?

What's up doc?

What's up doc?

What's up doc?

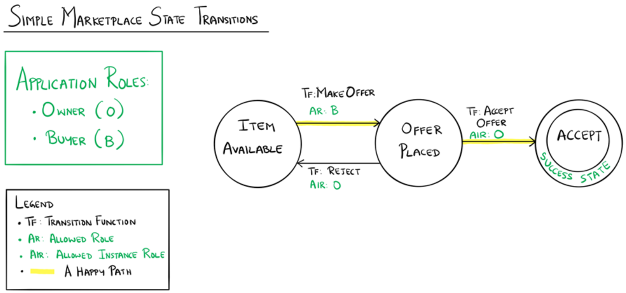
What's up doc?

– Prologue –

[An inspiring initiative]

A nice sketch! [6, 7]

A smart contract among Owners and Buyers



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." [3]

A new coordination model

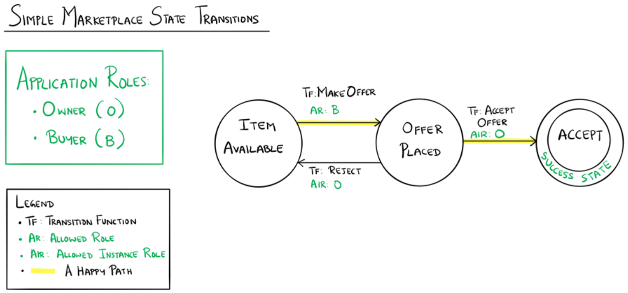
So, we saw an interesting model where

distributed components coordinate through a global specification

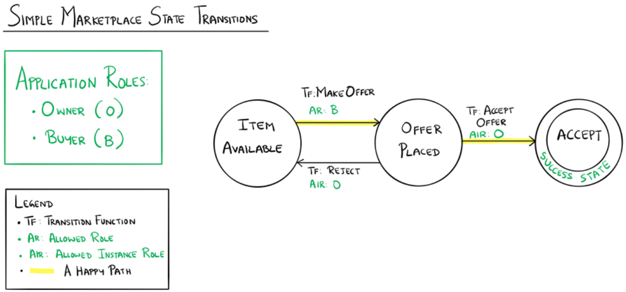
which specifies how actions are enabled along the computation

“without forcing” components to be cooperative!

Let's look at our sketch again



Let's look at our sketch again



but...

✗ what's the difference between roles and instances?

✗ 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

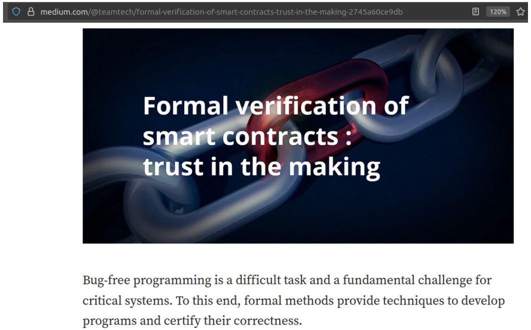
✓ we got roles okay, but

✗ limitations on instances of roles

✗ instances can have one role only

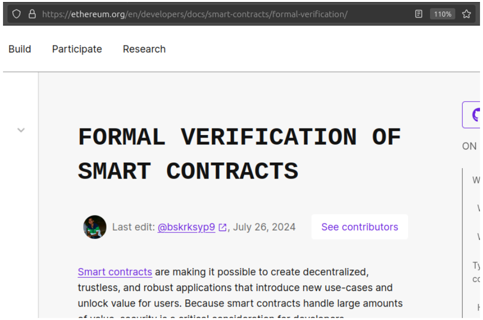
So we had to came up with some new behavioural types.

...and by the way


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

Formal verification of smart contracts : trust in the making

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

FORMAL VERIFICATION OF SMART CONTRACTS

Last edit: @bskrksyp9, July 26, 2024

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

https://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

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

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

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

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 $\mathsf{p}, \mathsf{p}', \ldots$
have roles $\mathsf{R}, \mathsf{R}', \ldots$
and cooperate through a coordinator c
which can be thought of as an object with “fields” and “methods”:

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

$\mathsf{f}, \mathsf{g}, \ldots$ represent the operations admitted by 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":

$u, v, \ldots$	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
$u := e$	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 $\mathsf{p}, \mathsf{p}', \ldots$
have roles $\mathsf{R}, \mathsf{R}', \ldots$
and cooperate through a coordinator c
which can be thought of as an object with "fields" and "methods":

$\mathsf{u}, \mathsf{v}, \ldots$	represent sorted state variables of c (sorts include data types such as 'int', 'bool', etc. as well as participants' roles)
$\mathsf{f}, \mathsf{g}, \ldots$	represent the operations admitted by c
$\mathsf{u} := \mathsf{e}$	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]
$A, A', \ldots$	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 state variables $u_1, \ldots, u_n$ 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 $R_1, \dots R_m$ and state variables $u_1, \dots, u_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated as follows[1]

$$\xrightarrow{\{\gamma\}\ \mathsf{new}\ R\ p \triangleright \mathsf{start}(c, \cdots, T_i\ x_i, \cdots)\ \{\cdots u_j := e_j \cdots\}} \circ$$

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

[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 $\mathsf{R}_1, \ldots \mathsf{R}_m$ and state variables $\mathsf{u}_1, \ldots, \mathsf{u}_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated as follows[1]

$$\begin{tikzcd} {} \arrow[rrr, "\{\gamma\}\ \mathsf{new}\ \mathsf{R}\ \mathsf{p} \triangleright \mathsf{start}(\mathsf{c}, \cdots, T_i\ \mathsf{x}_i, \cdots)\ \{\cdots \mathsf{u}_j := \mathsf{e}_j \cdots\}"] & & & \circ \end{tikzcd}$$

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

$$\begin{tikzcd} \circ \arrow[rrr, "\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}"] & & & \circ \end{tikzcd}$$

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

$$\pi \ ::= \ \mathsf{new}\ \mathsf{R}\ \mathsf{p} \ \big| \ \mathsf{any}\ \mathsf{R}\ \mathsf{p} \ \big| \ \mathsf{p}$$

is a qualified participant 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 $\mathsf{R}_1, \dots \mathsf{R}_m$ and state variables $\mathsf{u}_1, \dots, \mathsf{u}_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated as follows[1]

$$\begin{tikzcd} {} \arrow[r, "\{\gamma\}\ \mathsf{new}\ \mathsf{R}\ \mathsf{p} \triangleright \mathsf{start}(\mathsf{c}, \cdots, T_i\ \mathsf{x}_i, \cdots)\ \{\cdots \mathsf{u}_j := \mathsf{e}_j \cdots\}"] & \circ \end{tikzcd}$$

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

$$\begin{tikzcd} \circ \arrow[r, "\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}"] & \circ \end{tikzcd}$$

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

$$\pi \ ::= \ \mathsf{new}\ \mathsf{R}\ \mathsf{p} \ \big|\ \mathsf{any}\ \mathsf{R}\ \mathsf{p} \ \big|\ \mathsf{p}$$

is a qualified participant calling f with parameters x_i state variables are reassigned according to A if the invocation is successful

$$\begin{tikzcd} \circ \arrow[r, "\ell"] & \circledcirc \end{tikzcd}$$

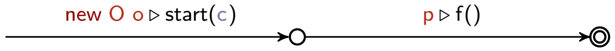
accepting states are denoted as usual

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

Exercise: modelling

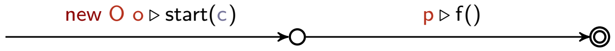
Give a DAFSM for the protocol on slide 7 resolving the ambiguities discussed there.

Not all DAFSMs “make sense”

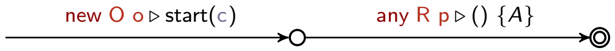


names’ freeness

Not all DAFSMs "make sense"



names' freeness



role emptyness

Not all DAFSMs “make sense”

$$\begin{tikzcd}[column sep=huge] {} \arrow[r, "\mathsf{new}\ \mathsf{O}\ \mathsf{o} \triangleright \mathsf{start}(\mathsf{c})"] & \bigcirc \arrow[r, "\mathsf{p} \triangleright \mathsf{f}()"] & \circledcirc \end{tikzcd}$$

names’ freeness

$$\begin{tikzcd}[column sep=huge] {} \arrow[r, "\mathsf{new}\ \mathsf{O}\ \mathsf{o} \triangleright \mathsf{start}(\mathsf{c})"] & \bigcirc \arrow[r, "\mathsf{any}\ \mathsf{R}\ \mathsf{p} \triangleright ()\ \{A\}"] & \circledcirc \end{tikzcd}$$

role emptyness

$$\begin{tikzcd}[column sep=huge] {} \arrow[r, "\mathsf{new}\ \mathsf{O}\ \mathsf{o} \triangleright \mathsf{start}(\mathsf{c})\ \{\mathsf{u} := 0\}"] & \bigcirc \arrow[r, "\{\mathsf{u} > 0\}\ \mathsf{new}\ \mathsf{R}\ \mathsf{p} \triangleright \mathsf{f}()"] & \circledcirc \end{tikzcd}$$

no progress

Not all DAFSMs “make sense”

$$\begin{tikzcd}[column sep=huge] {} \arrow[r, "\mathsf{new}\ \mathsf{O}\ \mathsf{o} \triangleright \mathsf{start}(\mathsf{c})"] & \circ \arrow[r, "\mathsf{p} \triangleright \mathsf{f}()"] & \circledcirc \end{tikzcd}$$

names’ freeness

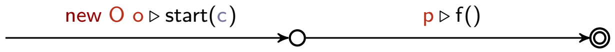
$$\begin{tikzcd}[column sep=huge] {} \arrow[r, "\mathsf{new}\ \mathsf{O}\ \mathsf{o} \triangleright \mathsf{start}(\mathsf{c})"] & \circ \arrow[r, "\mathsf{any}\ \mathsf{R}\ \mathsf{p} \triangleright ()\ \{A\}"] & \circledcirc \end{tikzcd}$$

role emptyness

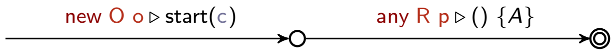
$$\begin{tikzcd}[column sep=huge] {} \arrow[r, "\mathsf{new}\ \mathsf{O}\ \mathsf{o} \triangleright \mathsf{start}(\mathsf{c})\ \{\mathsf{u} := 0\}"] & \circ \arrow[r, "\{\mathsf{u} > 0\}\ \mathsf{new}\ \mathsf{R}\ \mathsf{p} \triangleright \mathsf{f}()"] & \circledcirc \end{tikzcd}$$

no progress

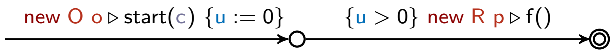
Not all DAFSMs “make sense”



names’ freeness



role emptyness



no progress

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 R p, and any R p

Closed DAFSMs

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

p is bound in

$$\begin{tikzcd} \bigcirc \arrow[r, "\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}"] & \bigcirc \end{tikzcd}$$

if, for some role R,

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

Closed DAFSMs

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

p is bound in $\bigcirc \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}} \bigcirc$ if, for some role R,

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

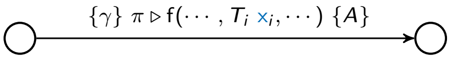
The occurrence of p is bound in a path

$$\sigma \quad \bigcirc \xrightarrow{\{\gamma\}\ \mathsf{p} \triangleright \mathsf{f}(\cdots)\ \{A\}} \bigcirc \cdots$$

if p is bound in a transition of σ

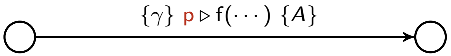
Closed DAFSMs

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

p is bound in 
$\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}$
if, for some role R,

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

The occurrence of p is bound in a path

σ 
$\{\gamma\}\ \mathsf{p} \triangleright \mathsf{f}(\cdots)\ \{A\}$
$\cdots$

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

Role emptyness

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

Role R is expanded in a path

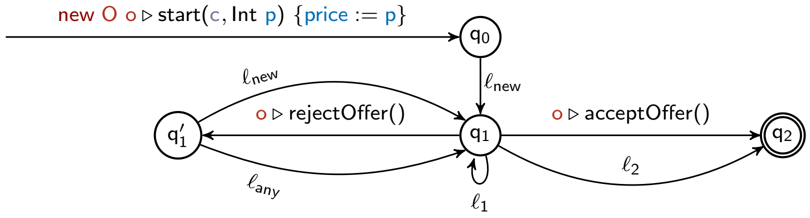
$$\sigma \quad \bigcirc \xrightarrow{\{\gamma\}\ \text{any R p} \triangleright \mathsf{f}(\cdots)\ \{A\}} \bigcirc \quad \cdots$$

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

Exercise: Role emptyness

Is the DAFSM below empty-role free?



where

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

$\ell_{any} = \{newOffer > 0\}$ any B b ▷ makeOffer(Int newOffer) {offer := newOffer},

$\ell_1 =$ new P p ▷ join()

and $\ell_2 = \{p > price\}$ any P p ▷ buy(Int p) .

Progress

A DAFSM with state variables $u_1, \ldots, u_n$ is consistent if

for each $$\begin{tikzcd} \bigcirc \arrow[r, "\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ x_i, \cdots)\ \{A\}"] & \textcircled{s} \end{tikzcd}$$

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

where

Progress

A DAFSM with state variables $\mathsf{u}_1, \ldots, \mathsf{u}_n$ is consistent if

for each

$$\begin{tikzcd} \bigcirc \arrow[r, "\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots,\, T_i\ \mathsf{x}_i, \cdots)\ \{A\}"] & \mathsf{s} \end{tikzcd}$$

$$\forall\!\!\!\forall_U\ \exists\!\!\!\exists_X\ \left(\gamma\{\mathsf{old}\ \mathsf{u}_1, \ldots, \mathsf{old}\ \mathsf{u}_n / \mathsf{u}_1, \ldots, \mathsf{u}_n\}\ \wedge\ \gamma_A\ \Longrightarrow\ \bigvee_{1 \le j \le m} \exists\!\!\!\exists_{Y_j}\ \gamma_j\right)$$ is satisfiable

where

$$U = \{\mathsf{u}_i, \mathsf{old}\ \mathsf{u}_i\}_{1 \le i \le n}$$

$$X = \{\mathsf{x} \mid \exists i\ :\ \mathsf{x} = \mathsf{x}_i\}$$

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

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

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

Progress

A DAFSM with state variables $\mathsf{u}_1, \ldots, \mathsf{u}_n$ is consistent if

for each $\begin{tikzcd} \bigcirc \arrow[rrr, "\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}"] & & & \textcircled{s} \end{tikzcd}$

$$\forall_U\, \exists_X\, (\gamma\{\mathsf{old}\ \mathsf{u}_1, \ldots, \mathsf{old}\ \mathsf{u}_n / \mathsf{u}_1, \ldots, \mathsf{u}_n\} \ \wedge\ \gamma_A \implies \bigvee_{1 \le j \le m} \exists_{Y_j}\, \gamma_j)$$ is satisfiable

and $\begin{tikzcd} {} \arrow[rrr, "\{\gamma\}\ \pi \triangleright \mathsf{start}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ \{A\}"] & & & \textcircled{s} \end{tikzcd}$ is such that $\exists_X\, \gamma$ is satisfiable

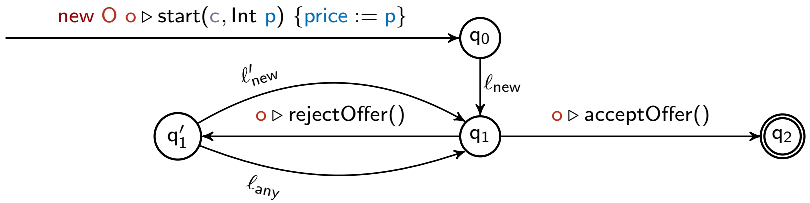
where

$$U = \{\mathsf{u}_i, \mathsf{old}\ \mathsf{u}_i\}_{1 \le i \le n}$$
$$X = \{\mathsf{x} \mid \exists i\ :\ \mathsf{x} = \mathsf{x}_i\}$$
$$\gamma_A = \bigwedge_{\mathsf{u} := \mathsf{e} \in A} \mathsf{u} = \mathsf{e}\ \wedge \bigwedge_{\mathsf{u} \notin A} \mathsf{u} = \mathsf{old}\ \mathsf{u}$$

$$Y_j = \{\mathsf{x} \mid \mathsf{x} \text{ is a parameter of the } j^{\text{th}} \text{ outgoing transition of s}\}$$
$$\gamma_j = \begin{cases} \text{the guard of the } j^{\text{th}} \text{ outgoing transitions of s} & \text{if s not accepting} \\ \text{True} & \text{otherwise} \end{cases}$$

Exercise: Consistency

Is the DAFSM below consistent?



where

ℓ_{new} = {newOffer > 0} new B b ▷ makeOffer(Int newOffer) {offer := newOffer},
ℓ'_{new} = {newOffer ≥ *price* ∗ 1.05} new B b ▷ makeOffer(Int newOffer) {offer := newOffer}, and
ℓ_{any} = {newOffer ≥ *price* ∗ 1.05} any B b ▷ makeOffer(Int newOffer) {offer := newOffer}

Determinism

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

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

A DAFSM is deterministic if

whenever

$$\begin{tikzcd} & \circ \\ \circ \arrow[ur, "\{\gamma_1\}\ \pi_1 \triangleright \mathsf{f}(\cdots)\ \{A_1\}"] \arrow[dr, "\{\gamma_2\}\ \pi_2 \triangleright \mathsf{f}(\cdots)\ \{A_2\}"'] & \\ & \circ \end{tikzcd}$$

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

Exercise: Determinism

The DAFSM $\mathcal{S} =$

$$\begin{tikzcd}
{} \arrow[r, "\mathsf{new}\ R\ p \triangleright \mathsf{start}(c)"] & s_0 \arrow[r, bend right, "\ell_1"'] \arrow[rd, "\ell_2"'] & s_1 \arrow[l, bend right, "\ell"'] \\
& & s_2
\end{tikzcd}$$

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

Well-formedness

A DAFSM is well-formed when it is

closed,

empty-role free,

consistent, and

deterministic

Exercise: Well-formedness

Which of the following DAFSM is well-formed?

$$\begin{tikzcd}
 & & s_2 \arrow[dr, "o \triangleright g()"] & & \\
{} \arrow[r, "\text{new } B\ b \triangleright \text{start}(c)"] & s_0 \arrow[ur, "\text{new } R\ o \triangleright g()"] \arrow[dr, "\text{new } O\ o \triangleright f()"'] & & s_3 \arrow[r, "o \triangleright h()"] & s_4 \\
 & & s_1 \arrow[ur, "o \triangleright g()"'] & &
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r, "\text{new } O\ o \triangleright \text{start}(c)\ \{u := 1\}"] & s_0 \arrow[r, "o \triangleright f()"] & s_1 \arrow[r, "\{u > 0\}\ o \triangleright g()"] & s_2
\end{tikzcd}$$

– Act II –

[A tool]

Verification

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

So we implemented **TRAC**, which

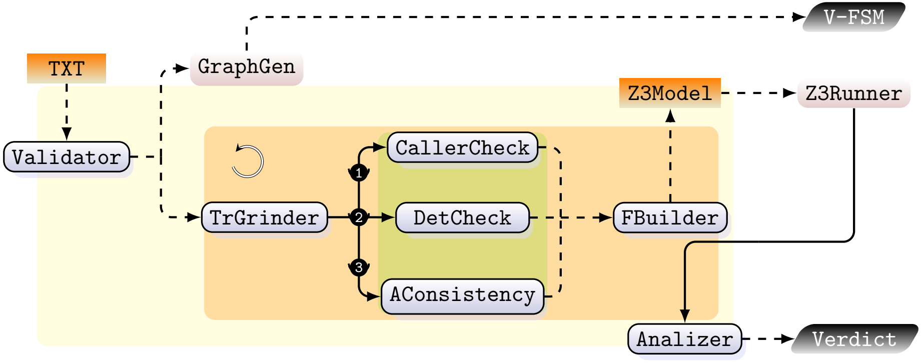
✓ features a DSL to specify DAFSMs

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

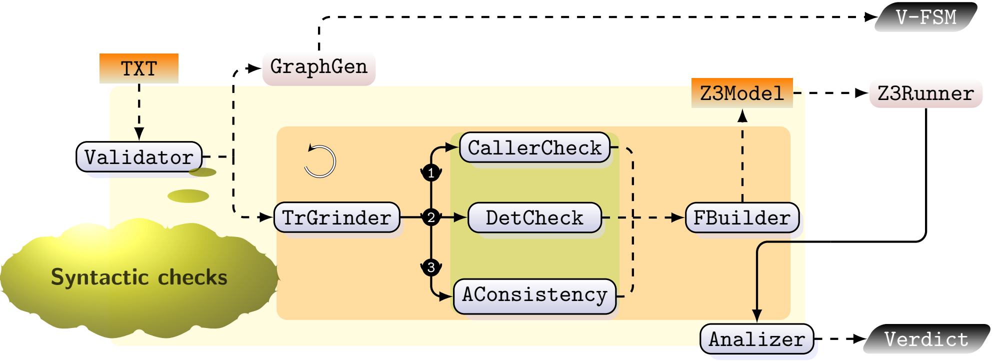
✓ it's efficient enough

✗ but cannot handle roles and inter-contract interactions

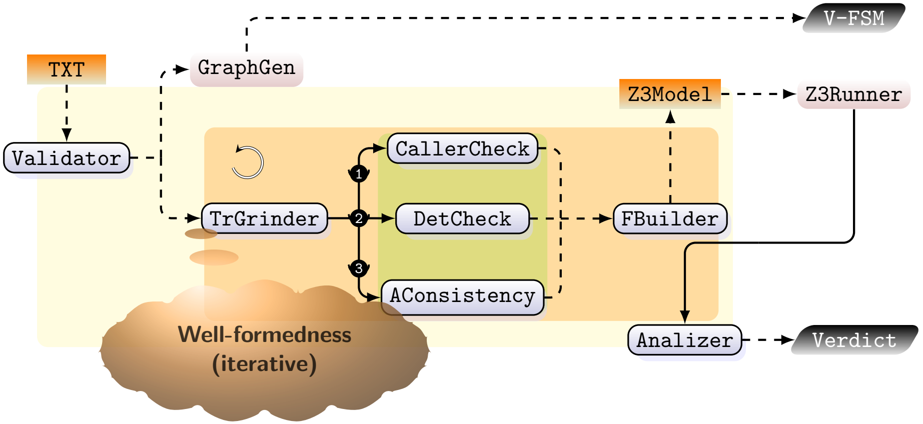
The architecture of **TRAC**



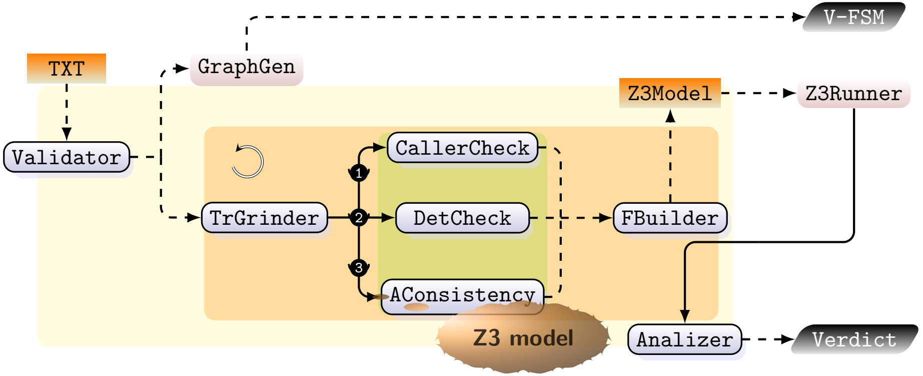
The architecture of **TRAC**



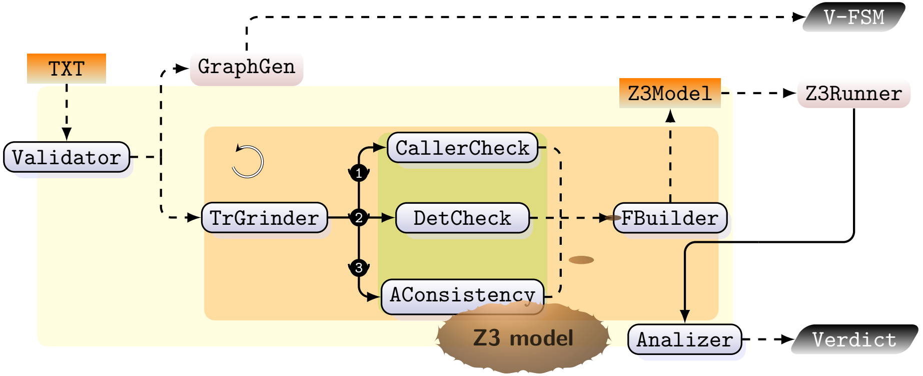
The architecture of **TRAC**



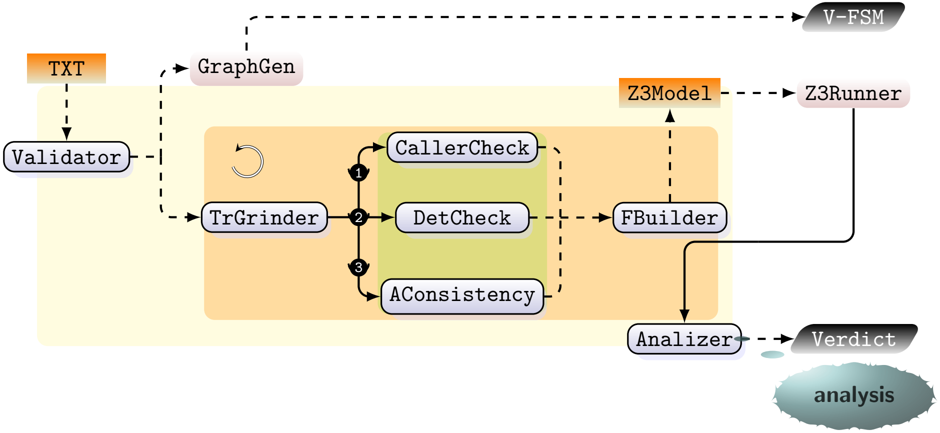
The architecture of **TRAC**



The architecture of **TRAC**



The architecture of **TRAC**



Installation

Dependencies: GraphViz and Python 3.6 or later

Detailed instructions in the `README.md` file at
`https://github.com/loctet/TRAC/tree/TRAC_v1/`

Concrete syntax (I)

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

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)

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

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

$\vdots$

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

Concrete syntax (I)

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

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

Concrete syntax (I)

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

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

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

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

⋮
[⟨str⟩] ⟨lbl⟩ [⟨str⟩] ;                               // the initial state defaults to the source state of the first transition
⋮                                                     // final states are strings with a trailing '+' sign
```

Exercise: **TRAC** usage (I)

Edit a `.trac` file for the contract specified at
`https://github.com/Azure-Samples/blockchain/blob/master/blockchain-workbench/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:

⟨*spec_constant*⟩	::=	⟨*numeral*⟩ | ⟨*decimal*⟩ | ⟨*hexadecimal*⟩ | ⟨*binary*⟩ | ⟨*string*⟩
⟨*s_expr*⟩	::=	⟨*spec_constant*⟩ | ⟨*symbol*⟩ | ⟨*reserved*⟩ | ⟨*keyword*⟩
	|	(⟨*s_expr*⟩*)
⟨*qual_identifier*⟩	::=	⟨*identifier*⟩ | (`as` ⟨*identifier*⟩ ⟨*sort*⟩)
⟨*var_binding*⟩	::=	(⟨*symbol*⟩ ⟨*term*⟩)
⟨*sorted_var*⟩	::=	(⟨*symbol*⟩ ⟨*sort*⟩)
⟨*pattern*⟩	::=	⟨*symbol*⟩ | (⟨*symbol*⟩ ⟨*symbol*⟩+)
⟨*match_case*⟩	::=	(⟨*pattern*⟩ ⟨*term*⟩)
⟨*term*⟩	::=	⟨*spec_constant*⟩
	|	⟨*qual_identifier*⟩
	|	(⟨*qual_identifier*⟩ ⟨*term*⟩+)
	|	(`let` (⟨*var_binding*⟩+) ⟨*term*⟩)
	|	(`lambda` (⟨*sorted_var*⟩+) ⟨*term*⟩)
	|	(`forall` (⟨*sorted_var*⟩+) ⟨*term*⟩)
	|	(`exists` (⟨*sorted_var*⟩+) ⟨*term*⟩)
	|	(`match` ⟨*term*⟩ (⟨*match_case*⟩+))
	|	(`!` ⟨*term*⟩ ⟨*attribute*⟩+)

HIC SUNT LEONES

probably not needed

(borrowed from [2])

Exercise: **TRAC** syntax (II)

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

– Act III –

[A little exercise]

A non-trivial contract

Use **TRAC** to specify a well-formed DAFSM for a contract that helps to raise funds.

The instantiating participant plays the role of the owner of the contract.

Once instantiated with a goal (the amount of money to raise), the contract handles a fundraising campaign whereby contributors can deposit funds if the campaign is active.

Once the goal is met, the owner triggers an inspection phase done by two agents.

At the end of the inspection one of the agents closes the campaign so that the owner can finally withdraw the funds.

eM’s solution

– 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.