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)

2025-05-22

A Choreographic View of Smart Contracts

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?

Prologue . . . . An inspiring initiative

What's up doc?

What's up doc?
Prologue . . . . An inspiring initiative
Act I . . . . A coordination framework

What's up doc?

What's up doc?

Prologue . . . . An inspiring initiative

Act I . . . . A coordination framework

Act II . . . . Some tool support

What's up doc?

What's up doc?

Prologue .... An inspiring initiative

Act I .... A coordination framework

Act II .... Some tool support

Act III .... A little exercise

What’s up doc?

What’s up doc?

Prologue . . . . An inspiring initiative

Act I . . . . A coordination framework

Act II . . . . Some tool support

Act III . . . . A little exercise

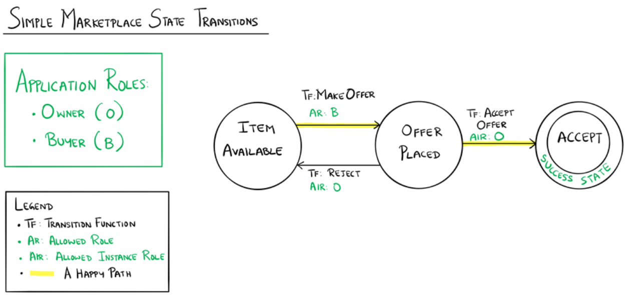
Epilogue . . . . Work in progress

– Prologue –

[An inspiring initiative]

– Prologue –
[An inspiring initiative]

A smart contract among Owners and Buyers

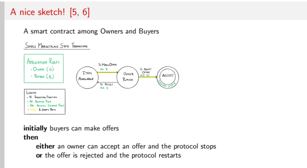


initially buyers can make offers
then
 either an owner can accept an offer and the protocol stops
 or the offer is rejected and the protocol restarts

A Choreographic View of Smart Contracts

2025-05-22

└A nice sketch! [5, 6]


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 how actions are enabled along the computation

"without forcing" components to be cooperative!

A new coordination model

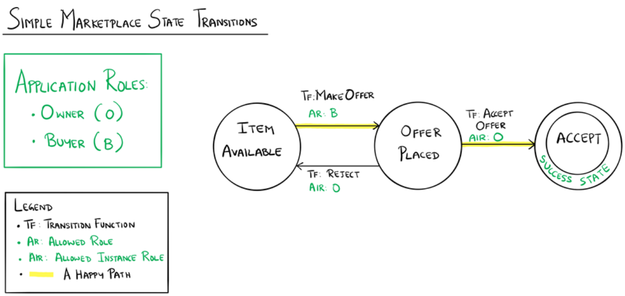
So, we saw an interesting model where

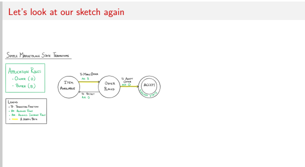
distributed components coordinate through a global specification

which specifies how actions are enabled along the computation

"without forcing" components to be cooperative!

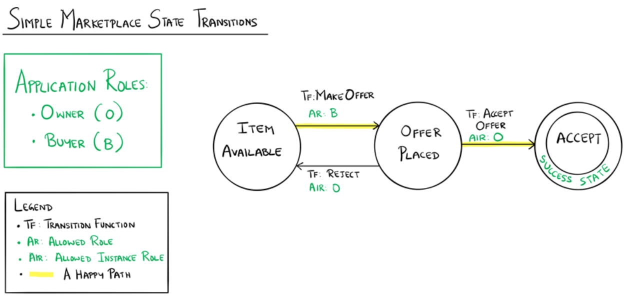
Let's look at our sketch again





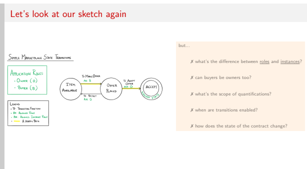
The diagram specifies a lot...

Let's look at our sketch again



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?



The diagram specifies a lot...

1. is the sketch giving semantics to roles and instances?
2. not forbidden...however what if we wanted to separate the roles?
3. 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?
4. ok
5. should the price of the item remain unchanged when the owner rejects offers?

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

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

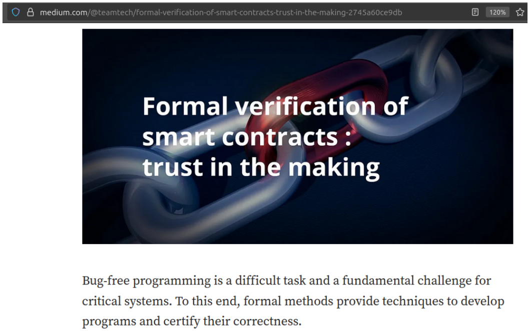
✗ limitations on instances of roles

✗ instances can have one role only

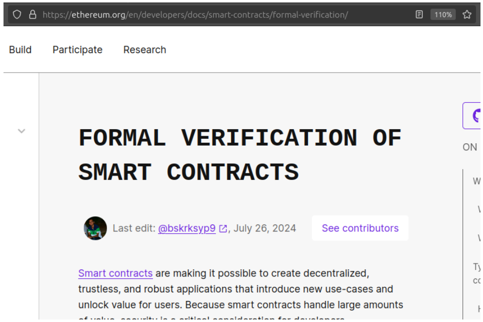
So we had to came up with some new behavioural types.

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.

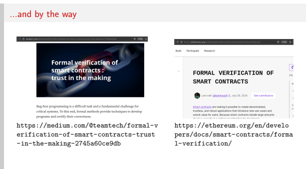
...and by the way



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]

– Act I –

[A coordination framework]

Basic concepts and notation

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

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 $R, R', \ldots$

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

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

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":

states of the coordinator determine which operations each roles is entitled to invoke

Basic concepts and notation

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

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

and cooperate through a coordinator c

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

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

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

We assume that sorts can be inferred; **TRAC** instead requires to assign sorts explicitly

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′, . . .
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

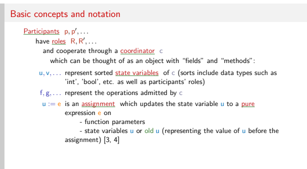
Basic concepts and notation

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) [3, 4]


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) [3, 4]

Expressions are standard but for state variables occurring in rhs e must have the old $\_$ qualifier; this concept will be used in the definition of (progress for) well-formedness

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., $x = x+1 \iff$ False.

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) [3, 4]
- $A, A', \ldots$ range over finite sets of assignments where each variable can be assigned at most once

Basic concepts and notation

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

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

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) [3, 4]

$A, A', \ldots$ range over finite sets of assignments where each variable can be assigned at most once

Data-Aware FSMs

A DAFSMs c on state variables $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated with specific labels as follows[1]

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

Data-Aware FSMs
A DAFSMs c on state variables $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated with specific labels as follows[1]
[1] See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

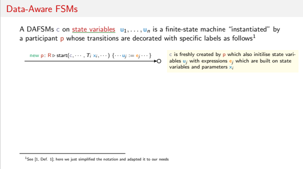
Data-Aware FSMs

A DAFSMs c on <u>state variables</u> $\mathsf{u}_1, \ldots, \mathsf{u}_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated with specific labels as follows[1]

$$\xrightarrow{\mathsf{new}\ \mathsf{p} \colon \mathsf{R} \triangleright \mathsf{start}(\mathsf{c}, \cdots, T_i\ \mathsf{x}_i, \cdots)\ \{\cdots \mathsf{u}_j := \mathsf{e}_j \cdots\}} \bigcirc$$

c is freshly created by p which also initilise 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



start is a "built-in" (and pleonastic) function name

each state variable is declared and initialises with type-consistent expressions on state variables and parameters x_i

Data-Aware FSMs

A DAFSMs c on state variables $\mathsf{u}_1, \ldots, \mathsf{u}_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated with specific labels as follows[1]

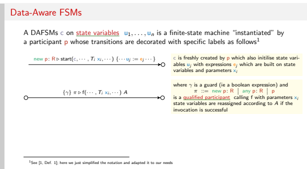
$$\xrightarrow{\mathsf{new}\ \mathsf{p} \colon \mathsf{R} \triangleright \mathsf{start}(\mathsf{c}, \cdots, T_i\ \mathsf{x}_i, \cdots)\ \{\cdots \mathsf{u}_j := \mathsf{e}_j \cdots\}} \circ$$

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

$$\circ \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ \mathsf{x}_i, \cdots)\ A} \circ$$

where γ is a guard (ie a boolean expression) and
$$\pi \ ::= \ \mathsf{new}\ \mathsf{p} \colon \mathsf{R} \ \big|\ \mathsf{any}\ \mathsf{p} \colon \mathsf{R} \ \big|\ \mathsf{p}$$
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



γ predicates over state variables and formal parameters of its transition; guards have to be satisfied for the invocation to succeed: an invocation that makes the guard false is rejected

- $\mathsf{new}\ \mathsf{p} \colon \mathsf{R}$ specifies that p must be a fresh participant with role R
- $\mathsf{any}\ \mathsf{p} \colon \mathsf{R}$ qualifies p as an existing participant with role R
- p refers 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 A are either state variables or parameters of the invocation

Data-Aware FSMs

A DAFSMs c on <u>state variables</u> $u_1, \ldots, u_n$ is a finite-state machine “instantiated” by a participant p whose transitions are decorated with specific labels as follows[1]

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

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

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

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

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

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

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

accepting states are denoted as usual

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

2025-05-22

A Choreographic View of Smart Contracts

└─Data-Aware FSMs

Exercise: modelling

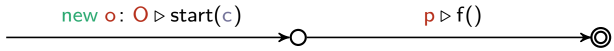
Give a DAFSM for the protocol on slide 7 resolving the ambiguities discussed there.

Exercise: modelling

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

let them play with qualified participants

Not all DAFSMs “make sense”

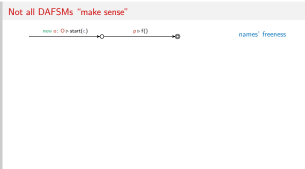


names’ freeness

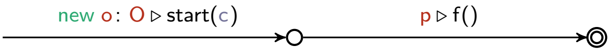
2025-05-22

A Choreographic View of Smart Contracts

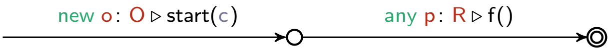
└─Not all DAFSMs “make sense”



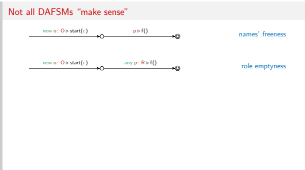
Not all DAFSMs “make sense”



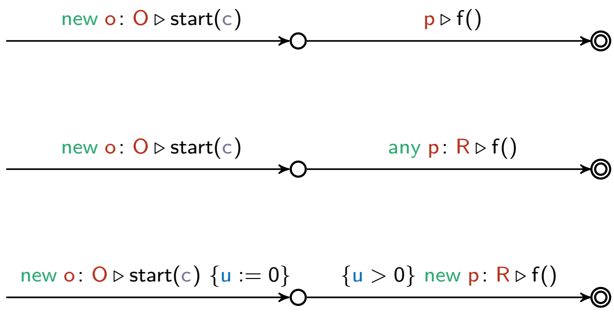
names’ freeness



role emptyness



Not all DAFSMs “make sense”



names’ freeness

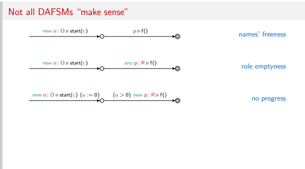
role emptyness

no progress

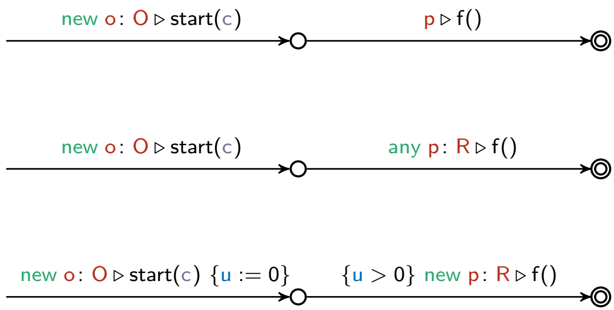
2025-05-22

A Choreographic View of Smart Contracts

└Not all DAFSMs “make sense”



Not all DAFSMs "make sense"



names' freeness

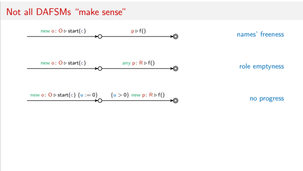
role emptyness

no progress

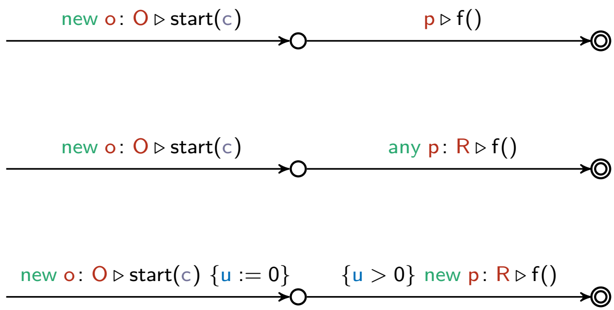
2025-05-22

A Choreographic View of Smart Contracts

└Not all DAFSMs "make sense"



Not all DAFSMs “make sense”



names’ freeness

role emptyness

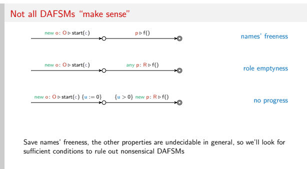
no progress

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

2025-05-22

A Choreographic View of Smart Contracts

└Not all DAFSMs “make sense”



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

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

p is bound in $\circ \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ x_i, \cdots)\ A} \circ$ 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: R, and any p: R

p is bound in $\circ \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots, T_i\ x_i, \cdots)\ A} \circ$ 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: R, and any p: R

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

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

The occurrence of p is bound in a path

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

if p is bound in a transition of σ

A Choreographic View of Smart Contracts

2025-05-22

└Closed DAFSMs

Closed DAFSMs

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

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

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

The occurrence of p is bound in a path

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

if p is bound in a transition of σ

Closed DAFSMs

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

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

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

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

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

Closed DAFSMs

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

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

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

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

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

Roles non-emptyness

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

Role R is expanded in a path

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

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

2025-05-22

A Choreographic View of Smart Contracts

└─Roles non-emptyness

Roles non-emptyness

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

Role R is expanded in a path

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

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

todo

Exercise: Role emptyness

todo

Progress

A DAFSM with state variables $u_1, \ldots, u_n$ is consistent if it is closed and the following implication holds for each transition $\circ \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots,\, T_i\ x_i, \cdots)\ A} \text{(s)}$

$$\mathbb{V}_U\, \mathbb{E}_X \left(\gamma\{\text{old } u_1, \ldots, \text{old } u_n / u_1, \ldots, u_n\} \;\wedge\; \gamma_A \implies \gamma_s\right)$$

where

Progress

A DAFSM with state variables $u_1, \ldots, u_n$ is consistent if it is closed and the following implication holds for each transition $\circ \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots,\, T_i\ x_i, \cdots)\ A} \text{(s)}$

$\mathbb{V}_U\, \mathbb{E}_X \left(\gamma\{\text{old } u_1, \ldots, \text{old } u_n / u_1, \ldots, u_n\} \wedge \gamma_A \implies \gamma_s\right)$

where

for a finite set of symbols Z, $\mathbb{V}_Z\left(\_\right)$ and $\mathbb{E}_Z\left(\_\right)$ are the universal and existential closures of a logical formula on the symbols in Z

Progress

A DAFSM with state variables $u_1, \ldots, u_n$ is consistent if it is closed and the following implication holds for each transition $\circ \xrightarrow{\{\gamma\}\ \pi \triangleright \mathsf{f}(\cdots,\ T_i\ x_i, \cdots)\ A} \text{(s)}$

$$\forall_U \exists_X \left(\gamma\{\text{old } u_1, \ldots, \text{old } u_n / u_1, \ldots, u_n\} \ \wedge\ \gamma_A \implies \gamma_s\right)$$

where

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

$$X = \{x \mid \exists i\ :\ x = x_i \text{ or } x \text{ is a parameter of an outgoing transition of s}\}$$

$$\gamma_s = \begin{cases} \text{the disjunction of guards of the outgoing transitions of s} & \text{is not accepting} \\ \text{True} & \text{otw} \end{cases}$$

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

for a finite set of symbols Z, $\forall_Z\,(\_)$ and $\exists_Z\,(\_)$ are the universal and existential closures of a logical formula on the symbols in Z

$u \notin A$
iff
for all $v := e \in A$, $u \neq v$ and old u does not occur in e

Exercise: Consistency

todo

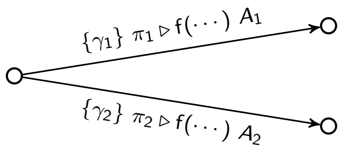
Exercise: Consistency
todo

Determinism

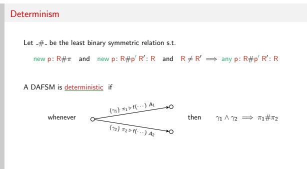
Let $\_\#\_$ be the least binary symmetric relation s.t.

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

A DAFSM is deterministic if

whenever 

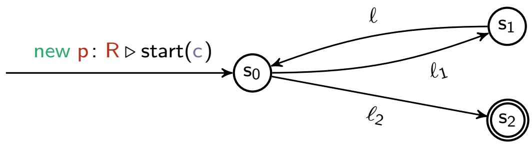
then $\gamma_1 \wedge \gamma_2 \implies \pi_1 \# \pi_2$



transitions from the same source state and calling the same function

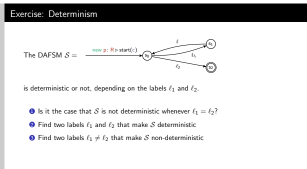
Exercise: Determinism

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 $\ell_1 = \ell_2 =$ new p: R $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new p: R ▷ f($\cdots$, T_i x_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 ▷ f(x : Int) and $\ell_2 = \{x \geq -1\}$ p ▷ f(x : Int) make $\mathcal{S}$ non-deterministic because the guards of ℓ_1 and of ℓ_2 are not disjoint therefore the next state is not determined by the caller

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

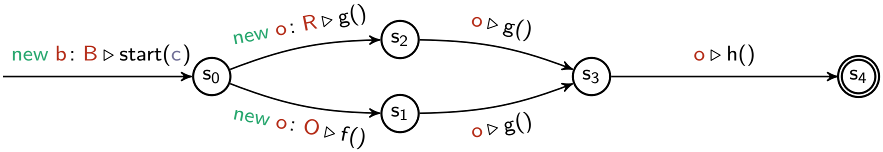
empty-role free

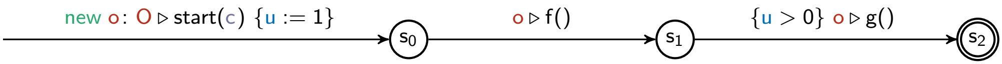
consistent, and

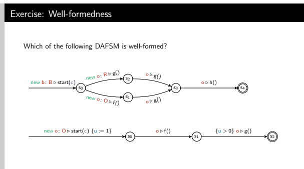
deterministic

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 consistency since True does not imply $u > 0$ hinting that the protocol could get stuck in state s_1. However, this never happens because u 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 (and boring)

So we implemented **TRAC**, which

✓ transforms DAFSMs in a DSL to specify DAFSMs

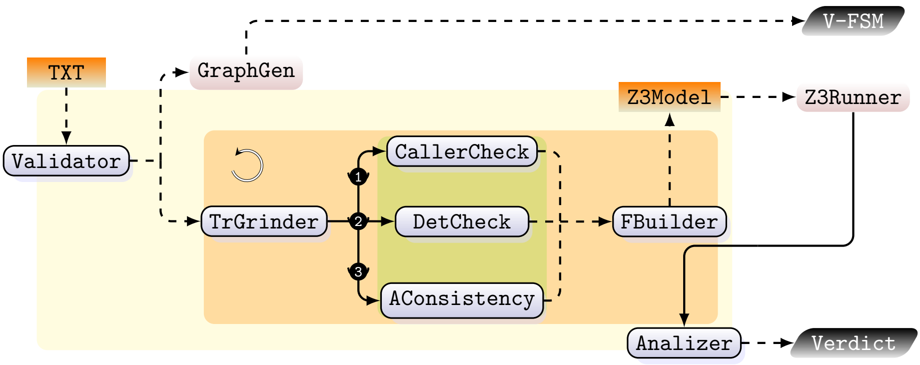
✓ verifies well-formedness condition relying on the SMT solver Z3

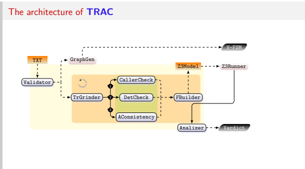
✓ it's efficient enough

✗ but cannot handle roles and inter-contract interactions

Verification
Checking well-formedness by hand is laborious and cumbersome (and boring)
So we implemented TRAC, which
✓ transforms DAFSMs in a DSL to specify DAFSMs
✓ verifies well-formedness condition relying on the SMT solver Z3
✓ it's efficient enough
✗ but cannot handle roles and inter-contract interactions

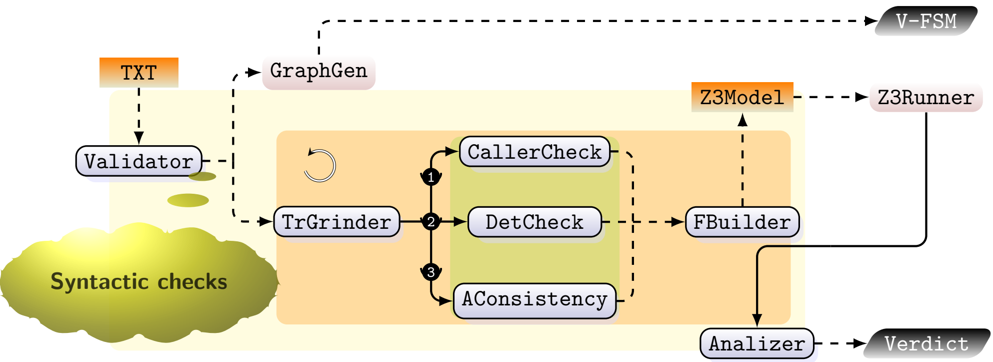
The architecture of **TRAC**

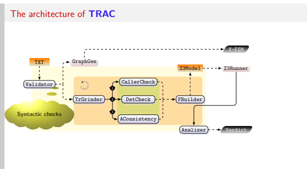




the architecture of **TRAC** is compartmentalised into two principal modules:
parsing and visualisation (yellow box) and
TRAC's core (orange box). The latter module implements well-formedness check (green box).
Solid arrows represent calls between components while dashed arrows data IO.

The architecture of **TRAC**

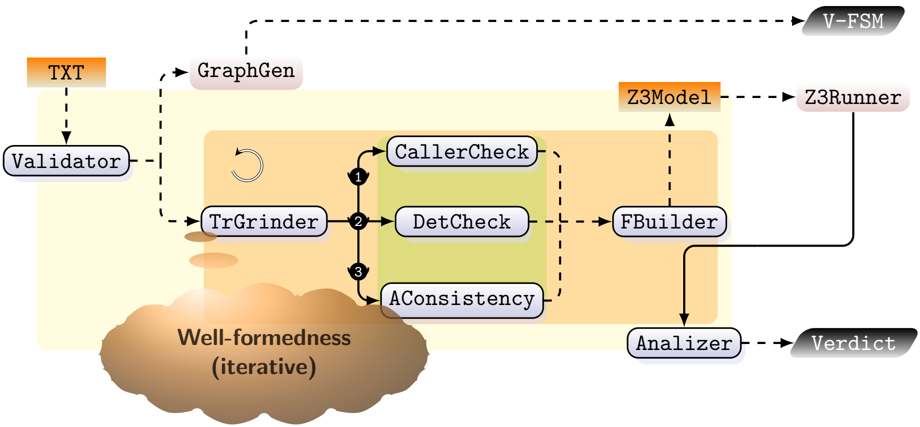


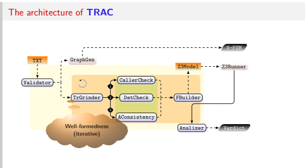


basic syntactic checks on a DSL representation of DAFSMs and transforming the input in a format that simplifies the analysis of the following phases:

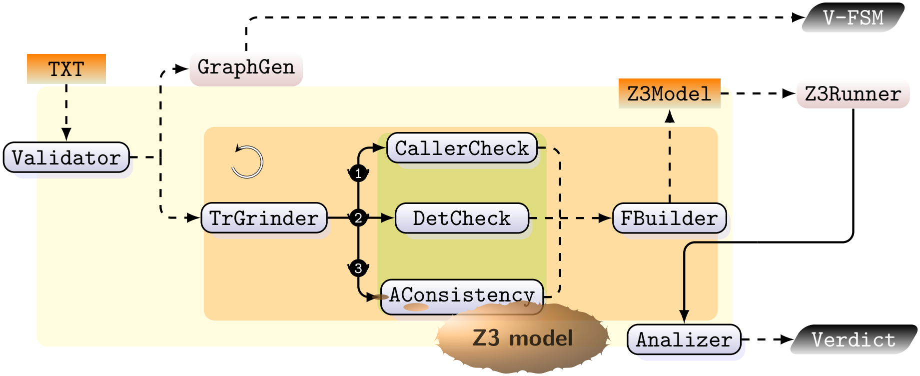
- passed to `GraphGen` for visual representation of DAFSMs (`V-FSM` output)
- passed to the `TrGrinder` component (orange box) for well-formedness checking.

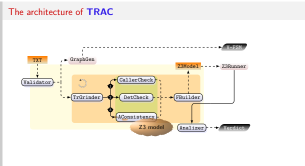
The architecture of **TRAC**



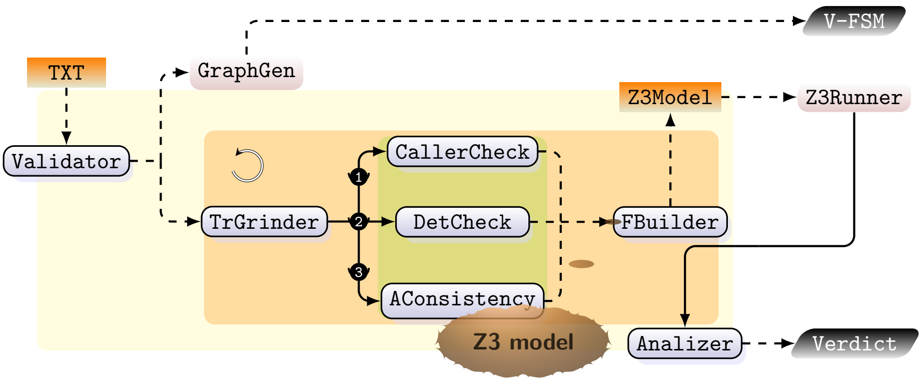


The architecture of **TRAC**





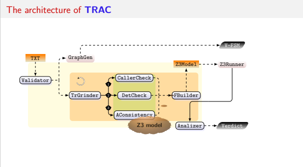
`AConsistency` (arrow ❸) to generate a Z3 formula which holds if, and only if, the transtion is consistent.



2025-05-22

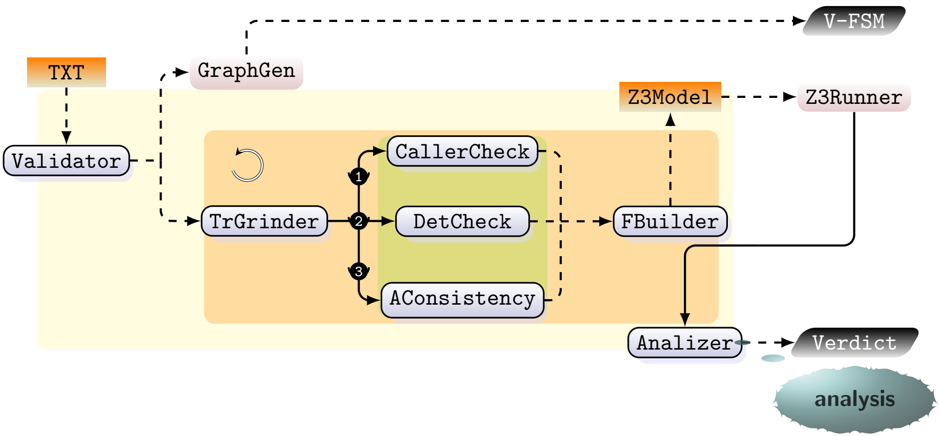
A Choreographic View of Smart Contracts

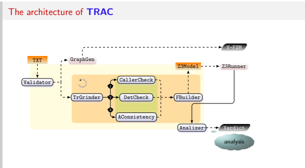
└The architecture of TRAC



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 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 networkx

Concrete syntax (I)

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

roles $\langle \texttt{str} \rangle^+$ . . . . role declaration
dafsm $\langle \texttt{str} \rangle$ ($\langle \texttt{pars} \rangle$) by $\langle \texttt{dcl} \rangle$ { $\qquad$ # $\langle \texttt{dcl} \rangle$ declares the participant creating the contract

Concrete syntax (I)

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

roles $\langle \texttt{str} \rangle^+$ . . . . role declaration
dafsm $\langle \texttt{str} \rangle$ ($\langle \texttt{pars} \rangle$) by $\langle \texttt{dcl} \rangle$ { # $\langle \texttt{dcl} \rangle$ declares the participant creating the contract

Concrete syntax (I)

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

roles $\langle \texttt{str} \rangle^+$ . . . . role declaration
dafsm $\langle \texttt{str} \rangle$ ($\langle \texttt{pars} \rangle$) by $\langle \texttt{dcl} \rangle$ { $\qquad$ # $\langle \texttt{dcl} \rangle$ declares the participant creating the contract

$\quad\vdots$

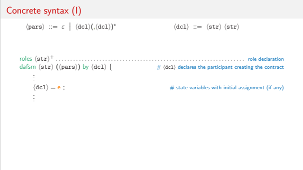
$\quad\langle \texttt{dcl} \rangle$ = e ; $\qquad$ # state variables with initial assignment (if any)

$\quad\vdots$

A Choreographic View of Smart Contracts

2025-05-22

└Concrete syntax (I)



Concrete syntax (I)

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

roles $\langle \mathtt{str} \rangle^+$ .... role declaration
dafsm $\langle \mathtt{str} \rangle$ ($\langle \mathtt{pars} \rangle$) by $\langle \mathtt{dcl} \rangle$ { # $\langle \mathtt{dcl} \rangle$ declares the participant creating the contract
$\vdots$
$\langle \mathtt{dcl} \rangle = e$; # state variables with initial assignment (if any)
$\vdots$
if γ # initial guard (this clause can be omitted)
}

recall that e and γ are SMT-Lib2 syntax for expressions and boolean expressions respectively

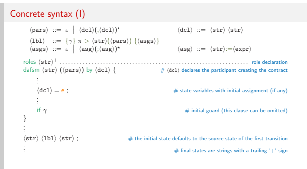
Concrete syntax (I)

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

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

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

roles $\langle\texttt{str}\rangle^+$ .... role declaration
dafsm $\langle\texttt{str}\rangle$ ($\langle\texttt{pars}\rangle$) by $\langle\texttt{dcl}\rangle$ { # $\langle\texttt{dcl}\rangle$ declares the participant creating the contract
  ⋮
  $\langle\texttt{dcl}\rangle$ = e ; # state variables with initial assignment (if any)
  ⋮
  if γ # initial guard (this clause can be omitted)
}
⋮
$\langle\texttt{str}\rangle$ $\langle\texttt{lbl}\rangle$ $\langle\texttt{str}\rangle$; # the initial state defaults to the source state of the first transition
⋮ # final states are strings with a trailing '+' sign



recall that e and γ are SMT-Lib2 syntax for expressions and boolean expressions respectively

Concrete syntax (I)

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

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

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

roles $\langle\texttt{str}\rangle^+$ . . . . role declaration

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

$\vdots$

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

$\vdots$

if γ # initial guard (this clause can be omitted)

}

$\vdots$

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

$\vdots$ # final states are strings with a trailing '+' sign

recall that e and γ are SMT-Lib2 syntax for expressions and boolean expressions respectively

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`

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

roles Owner Conterparty {}
dafsm basicProvenance(Owner o) by cp : Conterparty {}
q0 cp > c.TransferResponsibility(Conterparty cp) {} q1
q1 any *cp*: *Conterparty* > c.TransferResponsibility(Conterparty cp) {} q1
q1 o > c.Complete() {} q2+

Concrete syntax (II)

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

```
⟨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⟩+ )
```

(borrowed from [?])

https://smt-lib.org/papers/smt-lib-reference-v2.6-r2021-05-12.pdf
http://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf

Exercise: TRAC syntax (II)

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

Exercise: TRAC syntax (II)
Edit a .trac file for the DAFSM on slide 13.

– Act III –

[A little exercise]

– Act III –
[A little exercise]

A Choreographic View of Smart Contracts

2025-05-22

https://github.com/blockchain-unica/rosetta-smart-contracts/tree/main/contracts/vesting

– Epilogue –

[Work in progress]

– Epilogue –

[Work in progress]

Work in progress

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 *LNCS*, 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 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] 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.