### A Choreographic View of Smart Contracts

Elvis Gerardin Konjoh Selabi

Maurizio Murgia

António Ravara

Emilio Tuosto

© GSSI

A tutorial @ FORTE 2025, Lille

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

## This slides



Prologue . . . . . . . . An inspiring initiative

Prologue . . . . . . . . An inspiring initiative

Act I ..... A coordination framework

Prologue ...... An inspiring initiative

Act I ...... A coordination framework

Act II ..... Some tool support

Prologue ...... An inspiring initiative

Act I ...... A coordination framework

Act II ...... Some tool support

Act III ....... A little exercise

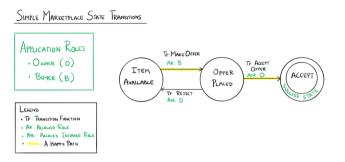
Prologue An in	spiring initiative
Act I A coordina	ation framework
Act II Son	me tool support
Act III	A little exercise
Epilogue V	Vork in progress

# - Prologue -

[ An inspiring initiative ]

## A nice sketch! [5, 6]

A smart contract among Owners and Buyers



initially buyers can make offers
then

**either** an owner can accept an offer and the protocol stops **or** the offer is rejected and the protocol restarts

## What did we just see?

#### A smart contract looks like

#### a choreographic model

global specifications determine the enabled actions along the evolution of the protocol

#### a typestate

In OOP, "can reflects how the legal operations on imperative objects can change at runtime as their internal state changes." [2]

#### A new coordination model

So, we saw an interesting model where

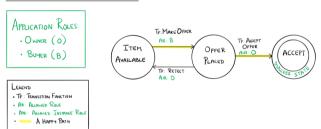
distributed components coordinate through a global specification

which specifies how actions are enabled along the computation

"without forcing" components to be cooperative!

## Let's look at our sketch again

### SIMPLE MARKETPLACE STATE TRANSITIONS

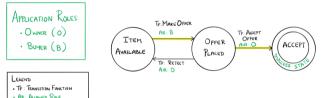


### Let's look at our sketch again

### SIMPLE MARKETPLACE STATE TRANSITIONS

ALLOWED INSTANCE ROLE

A HAPPY PATH



but...

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

X can buyers be owners too?

X what's the scope of quantifications?

X when are transitions enabled?

X how does the state of the contract change?

## Let's go formal!

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

**X** are known notions of well-formedness suitable?

X data-awareness is crucial

✓ we got roles okay, but

X limitations on instances of roles

X instances can have one role only

## Let's go formal!

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

- **X** are known notions of well-formedness suitable?
- X data-awareness is crucial
- ✓ we got roles okay, but
- X limitations on instances of roles
- X instances can have one role only

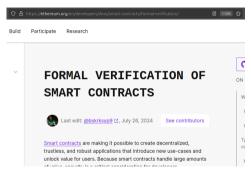
So we had to came up with some new behavioural types.

### ...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/develo pers/docs/smart-contracts/forma l-verification/

# - Act I -

[ A coordination framework ]

Participants  $p, p', \dots$ 

```
Participants p, p', ... have roles R, R', ...
```

```
Participants p, p', ...
have roles R, R', ...
and cooperate through a coordinator c
```

```
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":
```

```
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)
```

```
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
```

```
Participants p, p', \dots
    have roles R, R', \dots
      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]
```

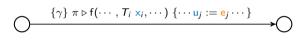
at most once

```
Participants p, p', \dots
    have roles R, R', \dots
      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]
   A.A'... range over finite sets of assignments where each variable can be assigned
```

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<sup>1</sup>

<sup>&</sup>lt;sup>1</sup>See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

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<sup>1</sup>



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

 $<sup>^1\</sup>mathrm{See}\ [1,\ \mathrm{Def.}\ 1];$  here we just simplified the notation and adapted it to our needs

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<sup>1</sup>

$$\bigcirc \qquad \{\gamma\} \ \pi \triangleright \mathsf{f}(\cdots, \mathcal{T}_i \times_i, \cdots) \ \{\cdots \mathsf{u}_j := \mathsf{e}_j \cdots \}$$

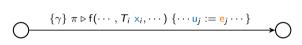
where  $\gamma$  is a guard (ie a boolean expression) and  $\pi ::= \text{new R p } \mid \text{any R p } \mid \text{p}$  is a <u>qualified participant</u> calling f with parameters  $x_i$  state variables are reassigned according to A if the invocation is successful

$$\{\gamma\}$$
 new R p  $\triangleright$  start $(c, \dots, T_i \times_i, \dots)$   $\{A\}$ 

c is freshly created by p which also initialises state variables  $\mathbf{u}_j$  with expressions  $\mathbf{e}_j$  which are built on state variables and parameters  $\mathbf{x}_i$ 

 $<sup>^{1}\</sup>mathrm{See}$  [1, Def. 1]; here we just simplified the notation and adapted it to our needs

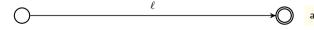
A DAFSM c on roles  $R_1, \dots R_m$  and state variables  $u_1, \dots, u_n$  is a finite-state machine "instantiated" by a participant p whose transitions are decorated as follows<sup>1</sup>



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

$$\{\gamma\} \text{ new R p} \triangleright \text{start}(c, \cdots, T_i \times_i, \cdots) \{A\}$$

c is freshly created by p which also initialises state variables  $\mathbf{u}_j$  with expressions  $\mathbf{e}_j$  which are built on state variables and parameters  $\mathbf{x}_i$ 

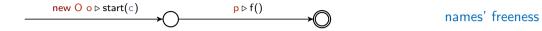


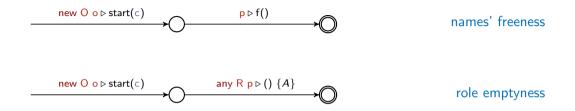
accepting states are denoted as usual

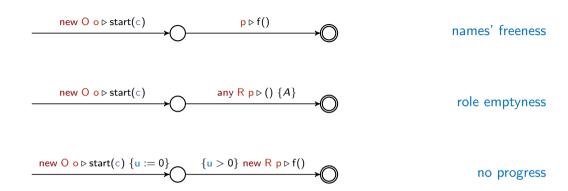
 $<sup>^{1}\</sup>mathrm{See}$  [1, Def. 1]; here we just simplified the notation and adapted it to our needs

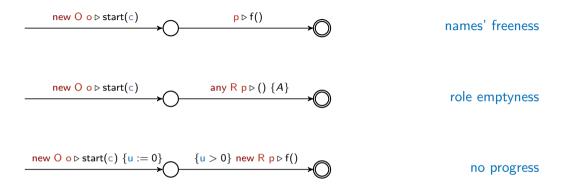
### Exercise: modelling

Give a DAFSM for the protocol on slide 8 resolving the ambiguities discussed there.









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

### Closed DAFSMs

<u>Binders:</u> parameter declarations in function calls (with scope local to the transition), new R p, and any R p (with scope along paths)

### Closed DAFSMs

<u>Binders:</u> parameter declarations in function calls (with scope local to the transition), new R p, and any R p (with scope along paths)

p is bound in 
$$\{\gamma\} \ \pi \triangleright \mathsf{f}(\cdots, T_i \times_i, \cdots) \ \{A\} \}$$
 if, for some role R,  $\pi = \mathsf{new} \ \mathsf{R} \ \mathsf{p}$  or  $\pi = \mathsf{any} \ \mathsf{R} \ \mathsf{p}$  or there is  $i \ \mathsf{s.t.} \ \times_i = \mathsf{p}$  and  $\mathsf{T} = \mathsf{R}_i$ 

#### Closed DAFSMs

<u>Binders:</u> parameter declarations in function calls (with scope local to the transition), new R p, and any R p (with scope along paths)

p is bound in 
$$\{\gamma\} \ \pi \triangleright f(\cdots, T_i \times_i, \cdots) \ \{A\} \}$$
 if, for some role R,  $\pi = \text{new R p}$  or  $\pi = \text{any R p}$  or there is  $i$  s.t.  $x_i = p$  and  $T = R_i$ 

The occurrence of p is bound in a path

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

if  ${\bf p}$  is bound in a transition of  $\sigma$ 

#### Closed DAFSMs

<u>Binders:</u> parameter declarations in function calls (with scope local to the transition), new R p, and any R p (with scope along paths)

p is bound in 
$$\bigcap$$
  $\{\gamma\} \ \pi \triangleright f(\cdots, T_i \times_i, \cdots) \ \{A\}$  if, for some role R,  $\pi = \text{new R p}$  or  $\pi = \text{any R p}$  or there is  $i \text{ s.t. } x_i = \text{p}$  and  $T = R_i$ 

The occurrence of p is bound in a path

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

if  ${\bf p}$  is bound in a transition of  $\sigma$ 

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

### Role emptyness

A transition 
$$\bigcap$$
  $\{\gamma\}$   $\pi \triangleright t(\cdots, T_i \times_i, \cdots) \{A\}$   $\longrightarrow$  expands role R if  $\pi = \text{new R p}$  or there is  $i \text{ s.t. } \times_i = \text{p}$  and  $T_i = \text{R}$ 

Role R is expanded in a path

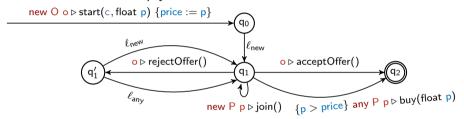
$$\sigma \cap (\gamma) \stackrel{\{\gamma\} \text{ any R } p \triangleright f(\cdots) \{A\}}{\longrightarrow} \cdots$$

if a transition in  $\sigma$  expands R

A DAFSM <u>expands</u> R if R is expanded on all the paths it occurs on A DAFSM is <u>(strongly) empty-role free</u> if it expands all its roles

#### Exercise: Role emptyness

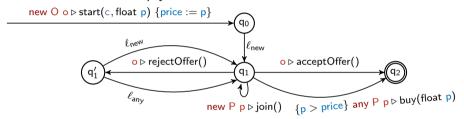
Is the DAFSM below empty-role free?



```
 \ell_{\mathsf{new}} = \{\mathsf{newOffer} > 0\} \ \mathsf{new} \ \mathsf{B} \ \mathsf{b} \rhd \mathsf{makeOffer}(\mathsf{float} \ \mathsf{newOffer}) \ \{\mathsf{offer} := \mathsf{newOffer}\} \ \mathsf{and} \ \ell_{\mathsf{any}} = \{\mathsf{newOffer} > 0\} \ \mathsf{any} \ \mathsf{B} \ \mathsf{b} \rhd \mathsf{makeOffer}(\mathsf{float} \ \mathsf{newOffer}) \ \{\mathsf{offer} := \mathsf{newOffer}\}
```

#### Exercise: Role emptyness

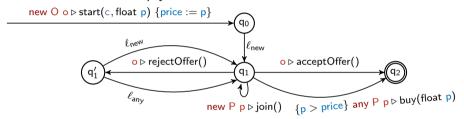
Is the DAFSM below empty-role free?



```
 \ell_{\mathsf{new}} = \{\mathsf{newOffer} > 0\} \ \mathsf{new} \ \mathsf{B} \ \mathsf{b} \rhd \mathsf{makeOffer}(\mathsf{float} \ \mathsf{newOffer}) \ \{\mathsf{offer} := \mathsf{newOffer}\} \ \mathsf{and} \ \ell_{\mathsf{any}} = \{\mathsf{newOffer} > 0\} \ \mathsf{any} \ \mathsf{B} \ \mathsf{b} \rhd \mathsf{makeOffer}(\mathsf{float} \ \mathsf{newOffer}) \ \{\mathsf{offer} := \mathsf{newOffer}\}
```

#### Exercise: Role emptyness

Is the DAFSM below empty-role free?



```
 \ell_{\mathsf{new}} = \{\mathsf{newOffer} > 0\} \ \mathsf{new} \ \mathsf{B} \ \mathsf{b} \rhd \mathsf{makeOffer}(\mathsf{float} \ \mathsf{newOffer}) \ \{\mathsf{offer} := \mathsf{newOffer}\} \ \mathsf{and} \ \ell_{\mathsf{any}} = \{\mathsf{newOffer} > 0\} \ \mathsf{any} \ \mathsf{B} \ \mathsf{b} \rhd \mathsf{makeOffer}(\mathsf{float} \ \mathsf{newOffer}) \ \{\mathsf{offer} := \mathsf{newOffer}\}
```

### **Progress**

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

$$\mathbb{V}_U \, \underline{\mathbb{I}}_X \left( \gamma \{ \mathsf{old} \, \, \mathsf{u}_1, \ldots, \mathsf{old} \, \, \mathsf{u}_n / \mathsf{u}_1, \ldots, \mathsf{u}_n \} \, \, \wedge \, \, \gamma_A \, \Longrightarrow \, \bigvee_{1 \leq j \leq m} \underline{\mathbb{I}}_{Y_j} \, \gamma_j \right) \, \, \mathsf{is} \, \, \mathsf{satisfiable}$$

### **Progress**

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

$$\mathbb{V}_U \, \underline{\mathbb{I}}_X \left( \gamma \{ \mathsf{old} \, \, \mathsf{u}_1, \ldots, \mathsf{old} \, \, \mathsf{u}_n / \mathsf{u}_1, \ldots, \mathsf{u}_n \} \, \, \wedge \, \, \gamma_A \, \Longrightarrow \, \bigvee_{1 \leq j \leq m} \underline{\mathbb{I}}_{Y_j} \, \gamma_j \right) \, \, \mathsf{is} \, \, \mathsf{satisfiable}$$

### **Progress**

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

for each 
$$\bigcirc \qquad \qquad \{\gamma\} \ \pi \triangleright f(\cdots, T_i \times_i, \cdots) \ \{A\}$$

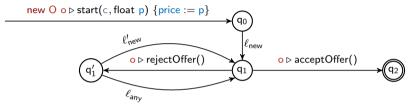
$$\mathbb{V}_U \, \mathbb{I}_X \left( \gamma \{ \text{old } \mathbf{u}_1, \dots, \text{old } \mathbf{u}_n / \mathbf{u}_1, \dots, \mathbf{u}_n \} \ \land \ \gamma_A \implies \bigvee_{1 \leq j \leq m} \mathbb{I}_{Y_j} \, \gamma_j \right) \text{ is satisfiable}$$

and 
$$\frac{\{\gamma\} \ \pi \triangleright \mathsf{start}(\cdots, T_i \times_i, \cdots) \ \{A\}}{}$$
 is such that  $\mathbb{H}_X \gamma$  is satisfiable

$$\begin{aligned} &U = \{ \mathbf{u}_i, \mathsf{old} \ \mathbf{u}_i \}_{1 \leq i \leq n} \\ &X = \{ \mathsf{x} \mid \exists i : \mathsf{x} = \mathsf{x}_i \} \\ &\gamma_A = \bigwedge_{\mathsf{u} := \mathsf{e} \in A} \mathsf{u} = \mathsf{e} \ \land \ \bigwedge_{\mathsf{u} \not\in A} \mathsf{u} = \mathsf{old} \ \mathsf{u} \end{aligned} \qquad \begin{aligned} &Y_j = \{ \mathsf{x} \mid \mathsf{x} \ \mathsf{is} \ \mathsf{a} \ \mathsf{parameter} \ \mathsf{of} \ \mathsf{the} \ j^{\mathsf{th}} \ \mathsf{outgoing} \ \mathsf{transitions} \ \mathsf{of} \ \mathsf{s} \end{aligned} \qquad \mathsf{if} \ \mathsf{s} \ \mathsf{not} \ \mathsf{accepting} \\ &\mathsf{True} \end{aligned}$$

### Exercise: Consistency

Is the DAFSM below consistent?



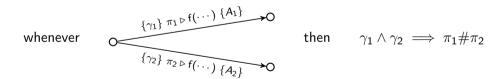
```
\begin{array}{l} \ell_{new} = \{ newOffer > 0 \} \text{ new B b} \triangleright makeOffer(float newOffer}) \; \{ offer := newOffer \}, \\ \ell'_{new} = \{ newOffer \geq price * 1.05 \} \text{ new B b} \triangleright makeOffer(float newOffer}) \; \{ offer := newOffer \}, \\ and \\ \ell_{any} = \{ newOffer \geq price * 1.05 \} \; \text{any B b} \triangleright makeOffer(float newOffer}) \; \{ offer := newOffer \} \end{array}
```

#### **Determinism**

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

new R p#p' and new R p#any R' p' and R 
$$\neq$$
 R'  $\Longrightarrow$  any R p#any R' p'

A DAFSM is deterministic if



#### Exercise: Determinism



is deterministic or not, depending on the labels  $\ell_1$  and  $\ell_2$ .

- **1** Is it the case that 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 well-formed when it is

closed,

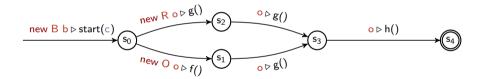
empty-role free,

consistent, and

deterministic

#### Exercise: Well-formedness

Which of the following DAFSM is well-formed?





– Act II –

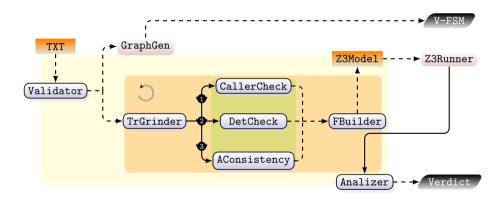
[ A tool ]

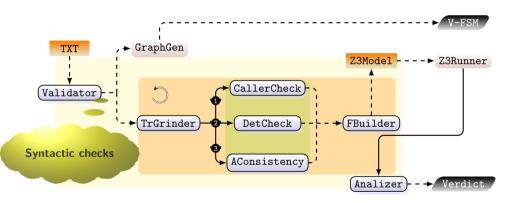
#### Verification

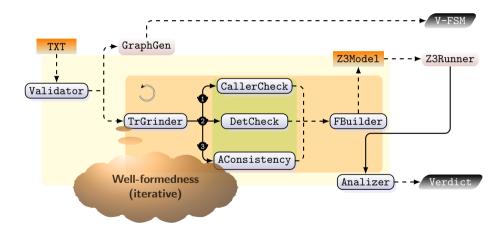
Checking well-formedness by hand is laborious and cumbersome (and boring)

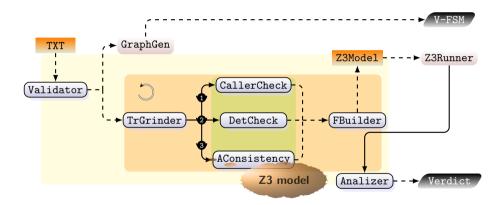
So we implemented a tool which

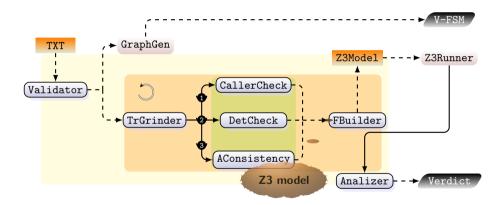
- ✓ features a DSL to specify DAFSMs
- ✓ verifies well-formedness (relying on the SMT solver Z3)
- ✓ it's efficient enough
- X but cannot handle roles and inter-contract interactions

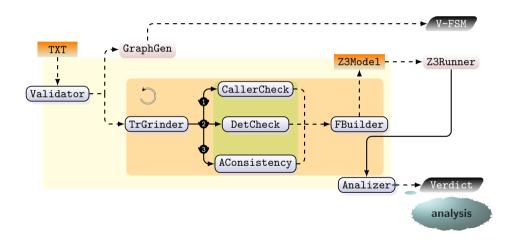












### Getting **TRAC**

```
In the cloud: https://trac-5syl.onrender.com/
or
In your hands: https://github.com/loctet/TRAC/tree/TRAC_v1/
    Dependencies: GraphViz and Python 3.7 or later
    Installation instructions in the README, md.
```

```
\langle pars \rangle ::= \varepsilon \mid \langle dcl \rangle (,\langle dcl \rangle)^* \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
```

```
\langle pars \rangle ::= \varepsilon \mid \langle dcl \rangle (,\langle dcl \rangle)^* \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 // state variables (if any) with their initial assignment
```

```
\langle pars \rangle ::= \varepsilon \mid \langle dcl \rangle (,\langle dcl \rangle)^* \qquad \langle dcl \rangle ::= \langle str \rangle \langle str \rangle
```

```
\langle pars \rangle ::= \varepsilon \mid \langle dcl \rangle (,\langle dcl \rangle)^*
                                                                                                 \langle dcl \rangle ::= \langle str \rangle \langle str \rangle
    \langle IbI \rangle ::= \{ \gamma \} \ \pi > \langle str \rangle (\langle pars \rangle) \{ \langle asgs \rangle \}
     \langle asgs \rangle ::= \varepsilon \mid \langle asg \rangle (;\langle asg \rangle)^*
                                                                                                 \langle asg \rangle ::= \langle str \rangle := e
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 (if any) with their initial assignment
                                                                                           // 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
```

### Exercise: TRAC usage (I)

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

The syntax of non-logical expressions is as in python while logical expressions are of the form

- Not(e)
- And( $e_1, \ldots, e_n$ )
- $Or(e_1, \ldots, e_n)$
- Imply(e<sub>1</sub>, e<sub>2</sub>)

See https://ericpony.github.io/z3py-tutorial/guide-examples.htm for examples

# Exercise: TRAC syntax (II)

Edit a .trac file for the DAFSM on slide 14.

# - 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.

- Epilogue -

[ Work in progress ]

### Work in progress

#### DAFSMs

Refine well-formedness Extend the model Projections & code generation

#### **TRAC**

Other verification approaches (model checking)
Integration with other tools
Keep improving usability

# 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 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.