

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?

Prologue An inspiring initiative

What's up doc?

Prologue An inspiring initiative

Act I A coordination framework

What's up doc?

Prologue An inspiring initiative

Act I A coordination framework

Act II Some tool support

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?

Prologue An inspiring initiative

Act I A coordination framework

Act II Some tool support

Act III A little exercise

Epilogue Work in progress

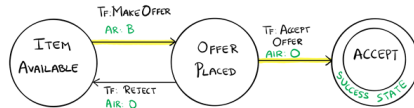
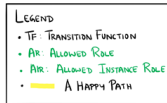
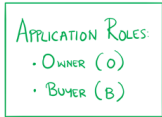
– Prologue –

[An inspiring initiative]

A nice sketch! [5, 6]

A smart contract among Owners and Buyers

SIMPLE MARKETPLACE STATE TRANSITIONS



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

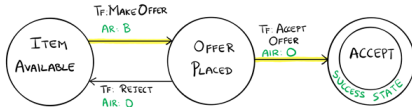
SIMPLE MARKETPLACE STATE TRANSITIONS

APPLICATION ROLES:

- OWNER (O)
- BUYER (B)

LEGEND

- TF: TRANSITION FUNCTION
- AR: ALLOWED ROLE
- AIR: ALLOWED INSTANCE ROLE
- — A HAPPY PATH



Let's look at our sketch again

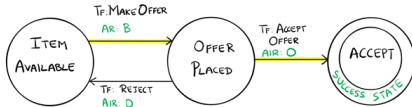
SIMPLE MARKETPLACE STATE TRANSITIONS

APPLICATION ROLES:

- OWNER (O)
- BUYER (B)

LEGEND

- TF: TRANSITION FUNCTION
- AR: ALLOWED ROLE
- AIR: ALLOWED INSTANCE ROLE
- — A HAPPY PATH



but...

✗ what's the difference between 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 come up with some new behavioural types.

...and by the way

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



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

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

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

Buidl Participate Research

FORMAL VERIFICATION OF SMART CONTRACTS



Last edit: @bskrksyp9, July 26, 2024

[See contributors](#)

[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 of value, security is a critical consideration for developers.

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

– Act I –

[A coordination framework]

Basic concepts and notation

Participants p, p', \dots

Basic concepts and notation

Participants p, p', \dots

have roles R, R', \dots

Basic concepts and notation

Participants p, p', \dots

have roles R, R', \dots

and cooperate through a coordinator c

Basic concepts and notation

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”:

Basic concepts and notation

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

f, g, \dots represent the operations admitted by c

Basic concepts and notation

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

f, g, \dots 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 $\text{old } u$ (representing the value of u before the assignment) [3, 4]

Basic concepts and notation

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

f, g, \dots 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 $\text{old } u$ (representing the value of u before the assignment) [3, 4]

A, A', \dots 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, \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 follows¹

¹See [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 follows¹

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

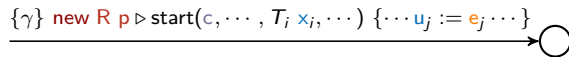


c is freshly created by p which also initialises state variables u_j with expressions e_j which are built on state variables and parameters x_i

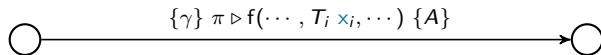
¹See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

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 follows¹



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

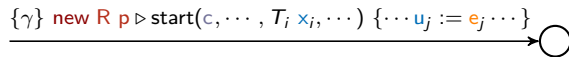


where γ is a guard (ie a boolean expression) and $\pi ::= \text{new } R \text{ } p \mid \text{any } R \text{ } p \mid p$ is a qualified participant calling f with parameters x_i ; state variables are reassigned according to A if the invocation is successful

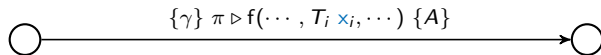
¹See [1, Def. 1]; here we just simplified the notation and adapted it to our needs

Data-Aware FSMs

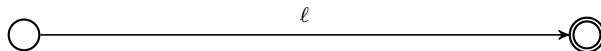
A DAFSM c on roles R_1, \dots, R_m and state variables u_1, \dots, u_n is a finite-state machine “instantiated” by a participant p whose transitions are decorated as follows¹



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



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



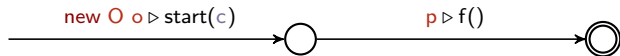
accepting states are denoted as usual

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

Exercise: modelling

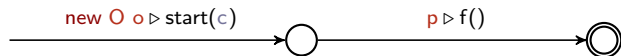
Give a DAFSM for the protocol on slide 7 resolving the ambiguities discussed there.

Not all DAFSMs “make sense”

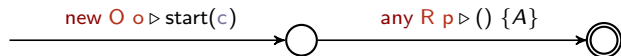


names' freeness

Not all DAFSMs “make sense”

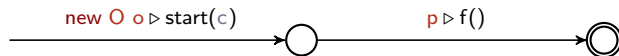


names' freeness

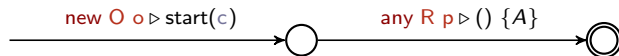


role emptiness

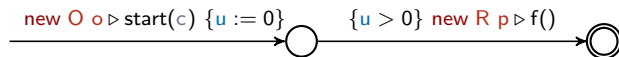
Not all DAFSMs “make sense”



names' freeness

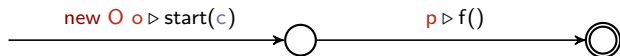


role emptiness

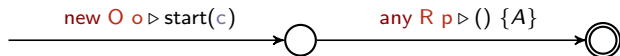


no progress

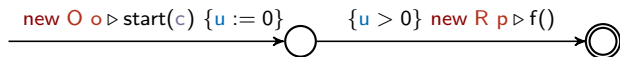
Not all DAFSMs “make sense”



names' freeness



role emptiness



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 (with scope local to the transition),
new $R\ p$, and any $R\ p$ (with scope along paths)

Closed DAFSMs

Binders: 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 $\bigcirc \xrightarrow{\{\gamma\} \pi \triangleright f(\dots, T_i x_i, \dots) \{A\}} \bigcirc$ if, for some role R ,

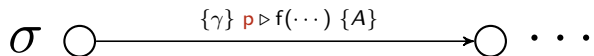
$\pi = \text{new } R \ p$ or $\pi = \text{any } R \ p$ or there is i s.t. $x_i = p$ and $T = R_i$

Closed DAFSMs

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



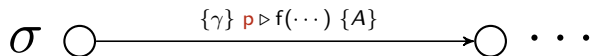
if p is bound in a transition of σ

Closed DAFSMs

Binders: 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 $\bigcirc \xrightarrow{\{\gamma\} \pi \triangleright f(\dots, T_i\ x_i, \dots) \{A\}} \bigcirc$ 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 $T = R_i$

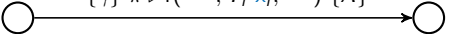
The occurrence of p is bound in a path



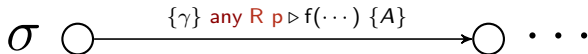
if p is bound in a transition of σ

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

Role emptiness

A transition  expands role R if $\pi = \text{new } R \text{ } p$ or there is i s.t. $x_i = p$ and $T_i = R$

Role R is expanded in a path

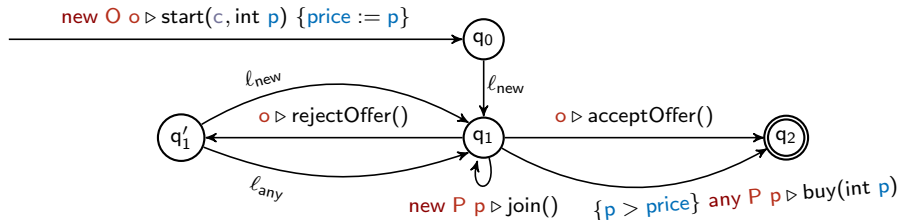


if a transition in σ expands R

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

Exercise: Role emptiness

Is the DAFSM below empty-role free?



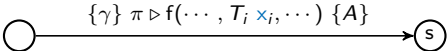
where

$\ell_{\text{new}} = \{ \text{newOffer} > 0 \} \text{ new } B \ b \triangleright \text{makeOffer}(\text{int } \text{newOffer}) \{ \text{offer} := \text{newOffer} \}$ and

$\ell_{\text{any}} = \{ \text{newOffer} > 0 \} \text{ any } B \ b \triangleright \text{makeOffer}(\text{int } \text{newOffer}) \{ \text{offer} := \text{newOffer} \}$

Progress

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

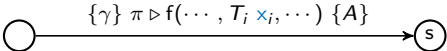
for each 

$\forall_U \mathbb{E}_X (\gamma\{\text{old } u_1, \dots, \text{old } u_n / u_1, \dots, u_n\} \wedge \gamma_A \implies \bigvee_{1 \leq j \leq m} \mathbb{E}_{\gamma_j} \gamma_j)$ is satisfiable

where

Progress

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

for each 

$\forall U \mathbb{E}_X (\gamma\{\text{old } u_1, \dots, \text{old } u_n / u_1, \dots, u_n\} \wedge \gamma_A \implies \bigvee_{1 \leq j \leq m} \mathbb{E}_{Y_j} \gamma_j)$ is satisfiable

where

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

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

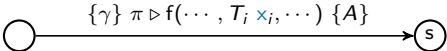
$$\gamma_A = \bigwedge_{u := e \in A} u = e \wedge \bigwedge_{u \notin A} u = \text{old } u$$

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

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

Progress

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

for each 

$\forall_U \mathbb{E}_X (\gamma\{\text{old } u_1, \dots, \text{old } u_n / u_1, \dots, u_n\} \wedge \gamma_A \implies \bigvee_{1 \leq j \leq m} \mathbb{E}_{Y_j} \gamma_j)$ is satisfiable

and  is such that $\mathbb{E}_X \gamma$ is satisfiable

where

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

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

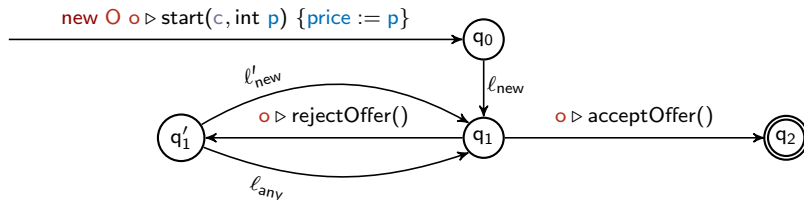
$$\gamma_A = \bigwedge_{u := e \in A} u = e \wedge \bigwedge_{u \notin A} u = \text{old } u$$

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

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

Exercise: Consistency

Is the DAFSM below consistent?



where

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

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

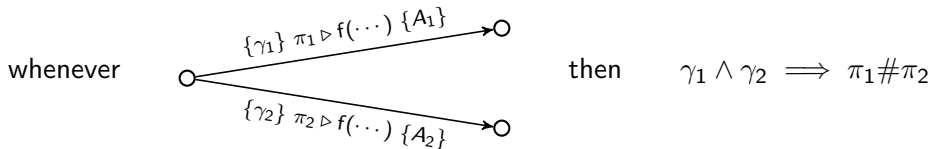
$\ell_{\text{any}} = \{\text{newOffer} \geq \text{price} * 1.05\} \ \text{any } B \ b \triangleright \text{makeOffer}(\text{int } \text{newOffer}) \ \{\text{offer} := \text{newOffer}\}$

Determinism

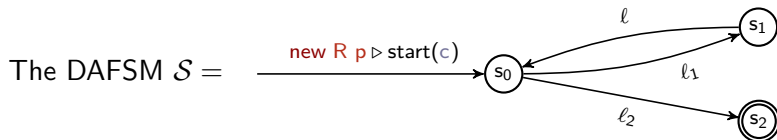
Let $\#$ be the least binary symmetric relation s.t.

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

A DAFSM is deterministic if



Exercise: Determinism



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,

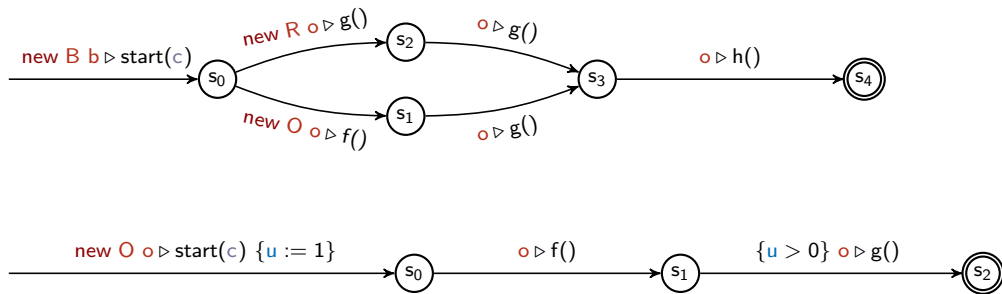
empty-role free,

consistent, and

deterministic

Exercise: Well-formedness

Which of the following DAFSM is well-formed?



– Act II –

[A tool]

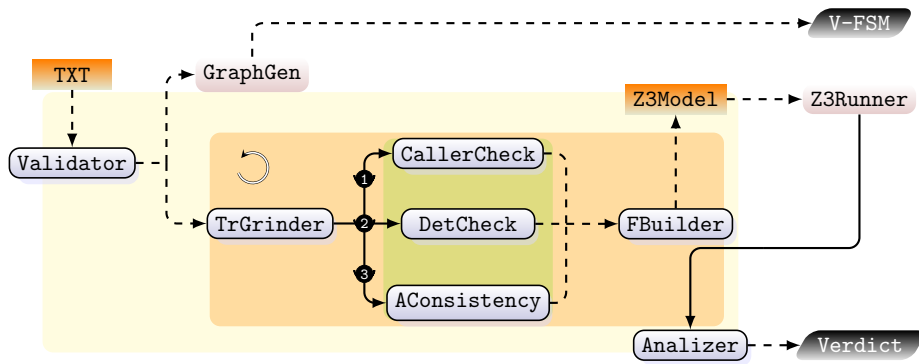
Verification

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

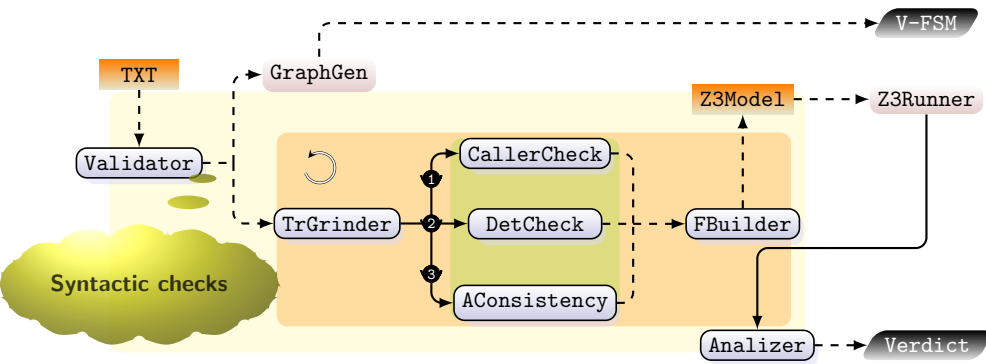
So we implemented a tool which

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

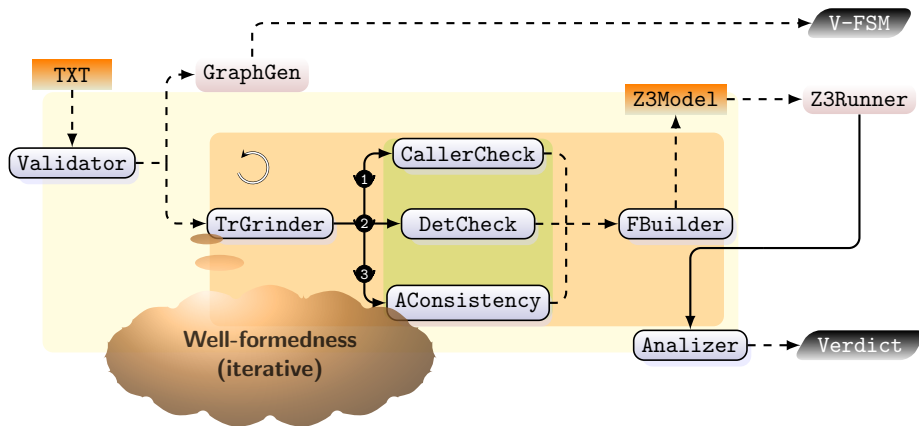
The architecture of TRAC



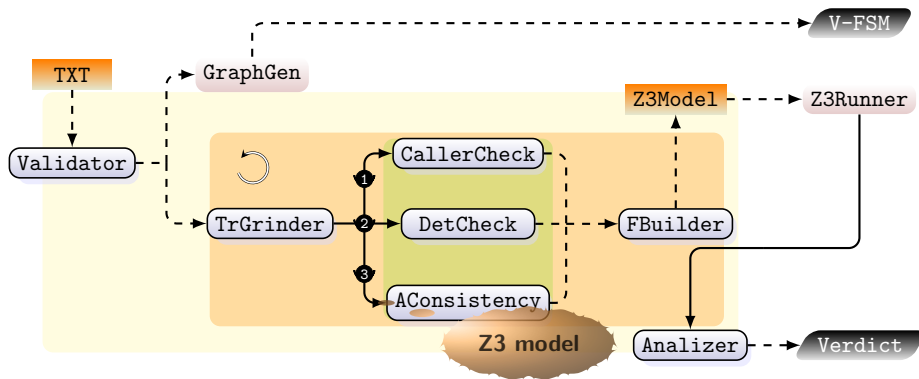
The architecture of TRAC



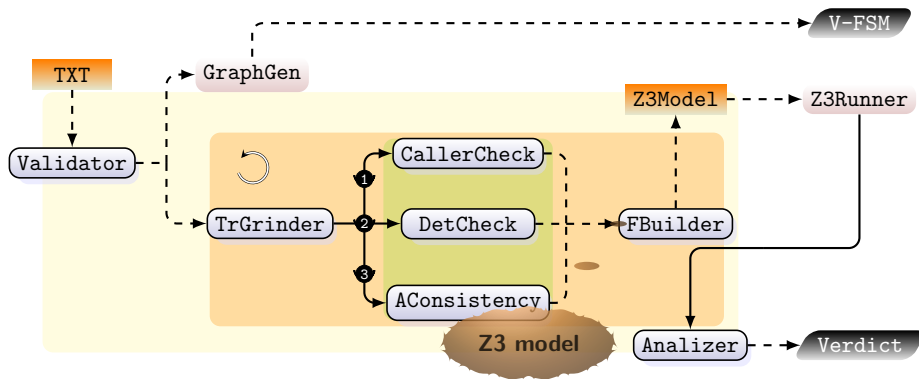
The architecture of TRAC



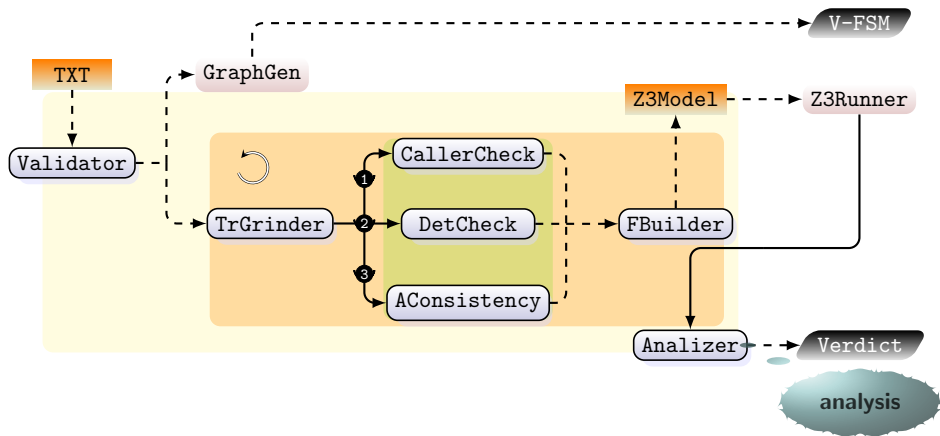
The architecture of TRAC



The architecture of TRAC



The architecture of TRAC



Getting TRAC

In the cloud: <https://trac-5sy1.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

Concrete syntax (I)

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

```
roles  $\langle str \rangle^+$   
dafsm  $\langle str \rangle(\langle pars \rangle)$  by  $\langle dcl \rangle$  {
```

```
// set the roles  
//  $\langle dcl \rangle$  declares the participant creating the contract
```

Concrete syntax (I)

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

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

Concrete syntax (I)

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

```
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
  :
   $\langle dcl \rangle := e$ ; // state variables (if any) with their initial assignment
  :
  if  $\gamma$  // initial guard (this clause can be omitted)
}
```

Concrete syntax (I)

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

$\langle dcl \rangle ::= \langle str \rangle \langle str \rangle$

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

$\langle asgs \rangle ::= \varepsilon \mid \langle asg \rangle (; \langle asg \rangle)^*$

$\langle asg \rangle ::= \langle str \rangle := e$

```
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
:
if  $\gamma$  // initial guard (this clause can be omitted)
}

:
[ $\langle str \rangle$ ]  $\langle lbl \rangle$  [ $\langle str \rangle$ ]; // the initial state defaults to the source state of the first transition
: // final states are strings with a trailing '+' sign
```

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

- `Not(e)`
- `And(e_1, \dots, e_n)`
- `Or(e_1, \dots, e_n)`
- `Imply(e_1, e_2)`

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 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

roles Onwer Constributor Agent

```
dafsm fundraising(float g) by Onwer o {  
  float goal :=g;  
  bool status :=False;  
  float balance :=0;  
  if g > 0  
}  
[q0]  
  {And(Not(status), _amount > 0, balance < goal)}  
  new Constributor u > deposit(float _amount)  
  {balance :=balance +_amount}  
[q0]  
  
[q0] {Not(status)} o > stop() {status :=True} [q0]  
  
[q0]  
  {And(Not(status), balance > = goal)}  
  o > sendInspection(Agent a1)  
[q1]  
  
[q1] a1 > inspect() {}[q2]  
  
[q2] new Agent a2 > inspect() {}[q3]
```

– 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] 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.