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)

This slides

This slides

What's up doc?

Prologue . . . . . An inspiring initiative

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

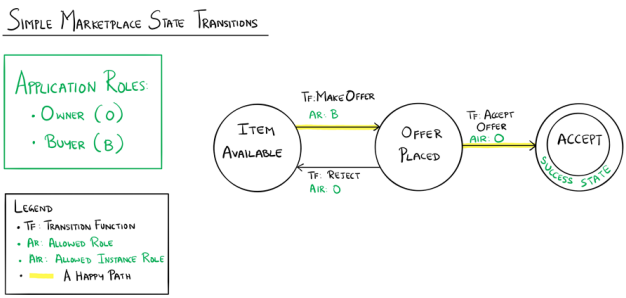
Epilogue . . . . . Work in progress

– Prologue –

[An inspiring initiative]

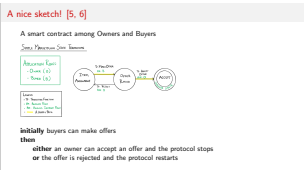
A nice sketch! [5, 6]

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?

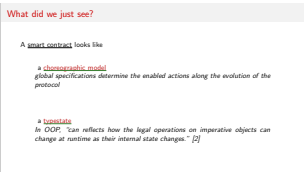
A smart contract looks like

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]



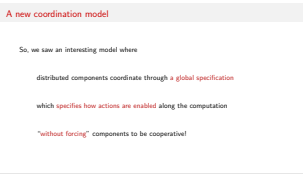
A new coordination model

So, we saw an interesting model where

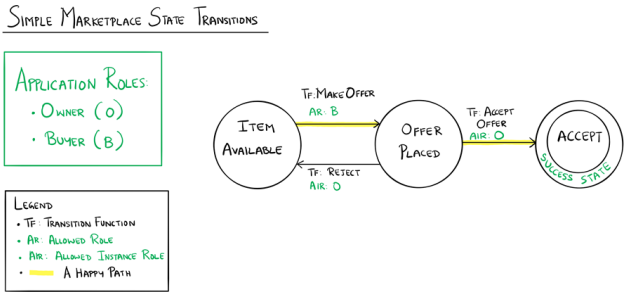
distributed components coordinate through a global specification

which specifies how actions are enabled along the computation

"without forcing" components to be cooperative!



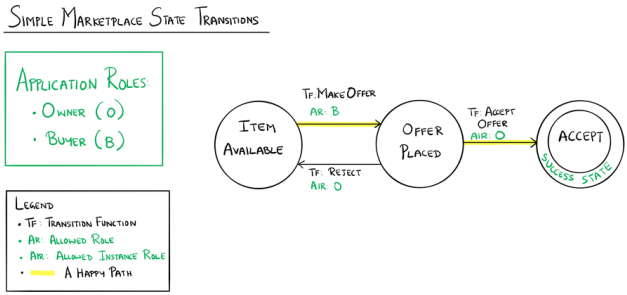
Let's look at our sketch again



Let's look at our sketch again

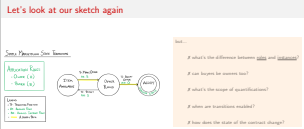
The diagram specifies a lot...

Let's look at our sketch again



but...

- ✗ what's the difference between 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?

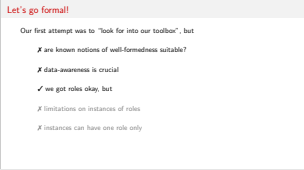


1. is the sketch giving semantics to roles and instances?
2. not forbidden...however what if we wanted to separate the roles?
3. from [6]: "The transitions between the Item Available and the Offer Placed states can continue until the owner is satisfied with the offer made." so, after a rejection, the 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

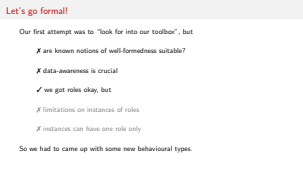


Let's go formal!

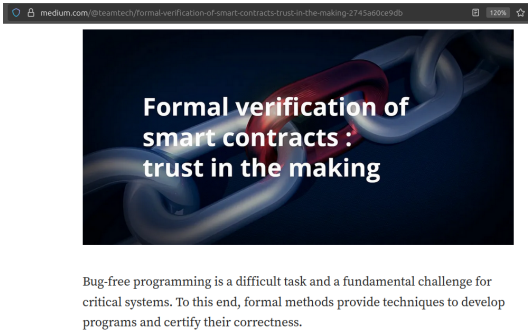
Our first attempt was to "look for into our toolbox", but

- ✗ 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

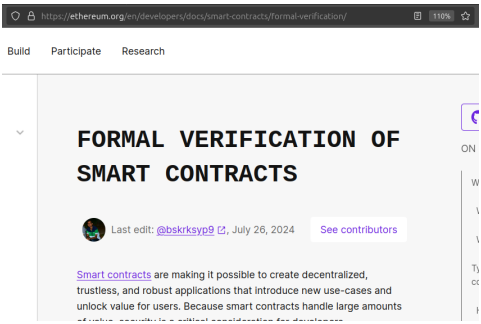
So we had to came up with some new behavioural types.



...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/

...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]

Basic concepts and notation

Participants $p, p', \ldots$

Basic concepts and notation
Participants $p, p', \ldots$

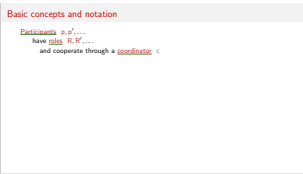
Basic concepts and notation

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

Basic concepts and notation
Participants $p, p', \ldots$
have roles $R, R', \ldots$

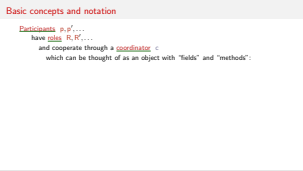
Basic concepts and notation

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



Basic concepts and notation

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



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

Basic concepts and notation

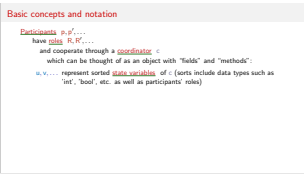
Participants $p, p', \ldots$

have roles $R, R', \ldots$

and cooperate through a coordinator c

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

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



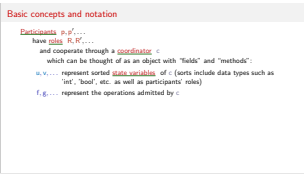
We assume that sorts can be inferred; **TRAC** instead requires to assign sorts explicitly

Basic concepts and notation

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

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

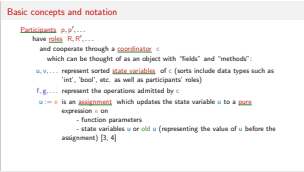
$f, g, \ldots$ represent the operations admitted by c



Basic concepts and notation

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

- $u, v, \ldots$ represent sorted state variables of c (sorts include data types such as 'int', 'bool', etc. as well as participants' roles)
- $f, g, \ldots$ represent the operations admitted by c
- $u := e$ is an assignment which updates the state variable u to a pure expression e on
  - function parameters
  - state variables u or old u (representing the value of u before the assignment) [3, 4]



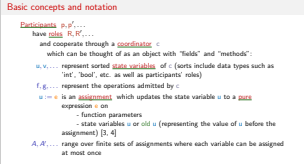
Expressions are standard but for state variables occurring in rhs e must have the old _ qualifier; this concept will be used in the definition of (progress for) well-formedness

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

- $u, v, \ldots$ represent sorted state variables of c (sorts include data types such as 'int', 'bool', etc. as well as participants' roles)
- $f, g, \ldots$ represent the operations admitted by c
- $u := e$ is an assignment which updates the state variable u to a pure expression e on
  - function parameters
  - state variables u or old u (representing the value of u before the assignment) [3, 4]
- $A, A', \ldots$ range over finite sets of assignments where each variable can be assigned at most once



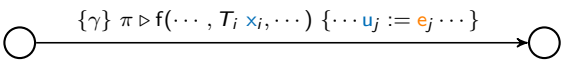
Data-Aware FSMs

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

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

Data-Aware FSMs

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

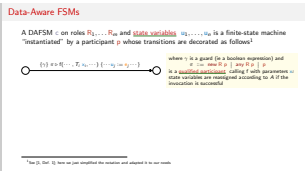


where γ is a guard (ie a boolean expression) and

$$\pi \ ::= \ \text{new R p} \ \mid \ \text{any R p} \ \mid \ \text{p}$$

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 on the formal parameters of the transition and state variables provided that it not a start transition
- has to be satisfied for the invocation to succeed: an invocation that makes the guard false is rejected

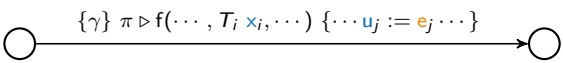
Invocations are rejected when the caller violates the qualitication

- new R p specifies that p must be a fresh participant with role R
- any R p qualifies p as an existing participant with role R
- refers to a participant in the scope of a binder

the variables occurring in the right-hand side of assignments in A are either state variables or parameters of the invocation

Data-Aware FSMs

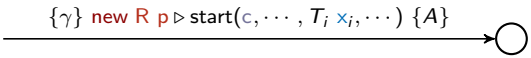
A DAFSM c on roles $R_1, \ldots R_m$ and state variables $u_1, \ldots, u_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated as follows[1]



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

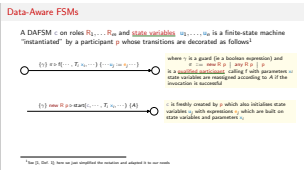
$$\pi \ ::= \ \texttt{new R p} \ \mid \ \texttt{any R p} \ \mid \ \texttt{p}$$

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



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

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

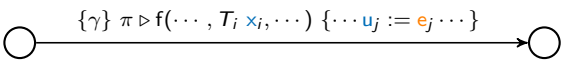


start is a "built-in" (and pleonastic) function name and there must be a unique start transition.

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

Data-Aware FSMs

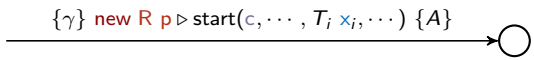
A DAFSM c on roles $R_1, \dots R_m$ and state variables $u_1, \dots, u_n$ is a finite-state machine "instantiated" by a participant p whose transitions are decorated as follows[1]



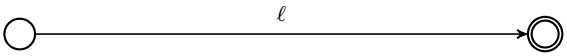
where γ is a guard (ie a boolean expression) and

$$\pi \ ::= \ \text{new R p} \ \mid \ \text{any R p} \ \mid \ \text{p}$$

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

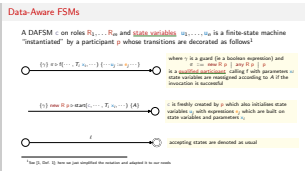


c is freshly created by p which also initialises state variables u_j with expressions e_j which are built on state variables and parameters x_i



accepting states are denoted as usual

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

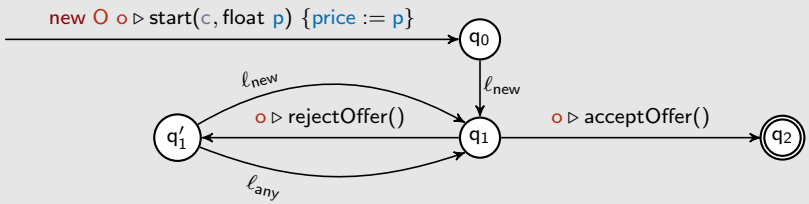


Exercise: modelling

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

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

Let $\ell_{\text{new}} = \{\text{newOffer} > 0\}$ new B b ▷ makeOffer(float newOffer) {offer := newOffer}
and $\ell_{\text{any}} = \{\text{newOffer} > 0\}$ any B b ▷ makeOffer(float newOffer) {offer := newOffer}



A new participant o acts as owner O for a coordinator c setting the state variable price to p in the initial state q_0 where the only enabled function is makeOffer(float offer). The first buyer b invoking this function with a strictly positive actual parameter newOffer moves the protocol to state q_1 and sets offer to newOffer.

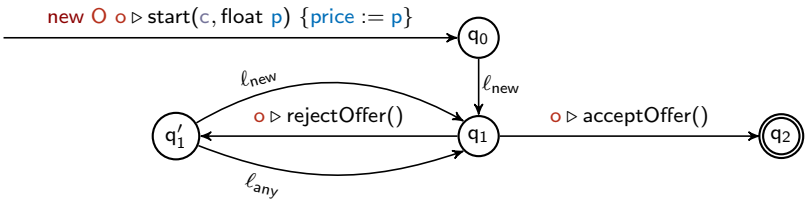
From q_1 the protocol progresses only if o accepts or rejects the offer. The protocol resp. reaches the accepting state q_2 or q_1' where either an existing buyer or a new one can make further offers.

Exercise: modelling

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

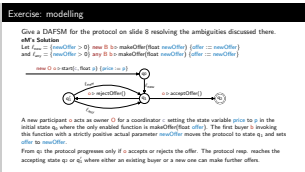
eM's Solution

Let $\ell_{\text{new}} = \{\text{newOffer} > 0\}$ new B b ▷ makeOffer(float newOffer) {offer := newOffer}
and $\ell_{\text{any}} = \{\text{newOffer} > 0\}$ any B b ▷ makeOffer(float newOffer) {offer := newOffer}

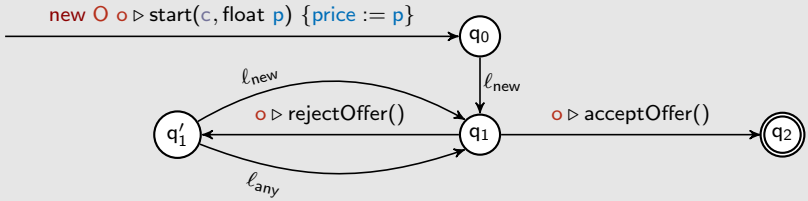


A new participant o acts as owner O for a coordinator c setting the state variable price to p in the initial state q_0 where the only enabled function is makeOffer(float offer). The first buyer b invoking this function with a strictly positive actual parameter newOffer moves the protocol to state q_1 and sets offer to newOffer.

From q_1 the protocol progresses only if o accepts or rejects the offer. The protocol resp. reaches the accepting state q_2 or q_1' where either an existing buyer or a new one can make further offers.



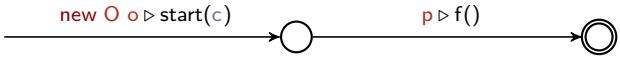
Let $\ell_{\text{new}} = \{\text{newOffer} > 0\}$ new B b ▷ makeOffer(float newOffer) {offer := newOffer}
and $\ell_{\text{any}} = \{\text{newOffer} > 0\}$ any B b ▷ makeOffer(float newOffer) {offer := newOffer}



A new participant o acts as owner O for a coordinator c setting the state variable price to p in the initial state q_0 where the only enabled function is makeOffer(float offer). The first buyer b invoking this function with a strictly positive actual parameter newOffer moves the protocol to state q_1 and sets offer to newOffer.

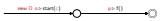
From q_1 the protocol progresses only if o accepts or rejects the offer. The protocol resp. reaches the accepting state q_2 or q_1' where either an existing buyer or a new one can make further offers.

Not all DAFSMs “make sense”



names' freeness

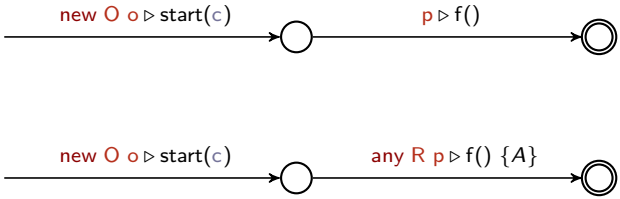
Not all DAFSMs “make sense”

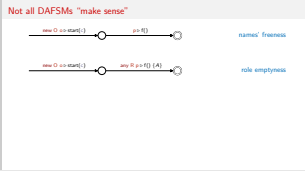


names' freeness

p is not qualified!

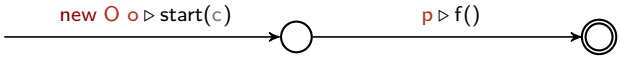
Not all DAFSMs “make sense”



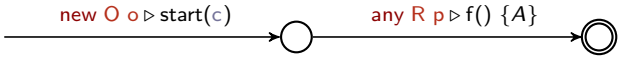


no participants in role R

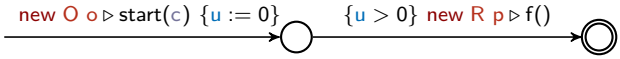
Not all DAFSMs “make sense”



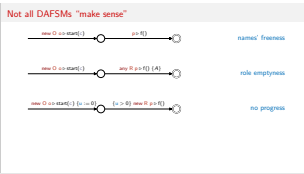
names' freeness



role emptyness

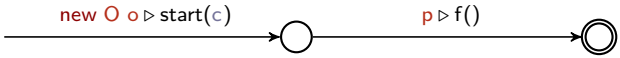


no progress

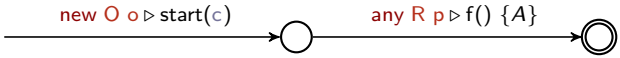


guard $u > 0$ unsatisfiable

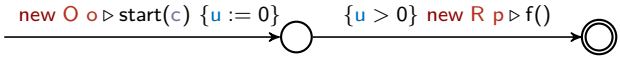
Not all DAFSMs “make sense”



names’ freeness

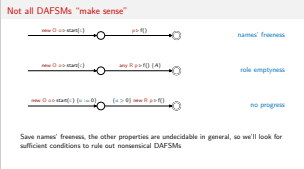


role emptyness



no progress

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



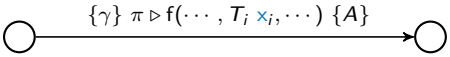
Closed DAFSMs

Binders: parameter declarations in function calls (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)

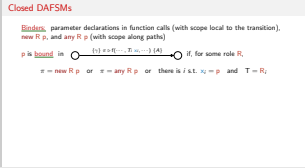
Closed DAFSMs

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 

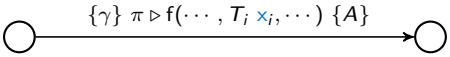
if, for some role R,

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



Closed DAFSMs

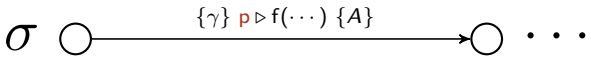
Binders: parameter declarations in function calls (with scope local to the transition), new R p, and any R p (with scope along paths)

p is bound in 

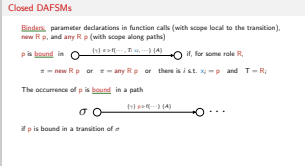
if, for some role R,

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

The occurrence of p is bound in a path

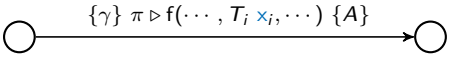


if p is bound in a transition of σ



Closed DAFSMs

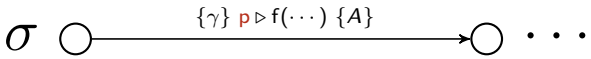
Binders: parameter declarations in function calls (with scope local to the transition), new R p, and any R p (with scope along paths)

p is bound in 

if, for some role R,

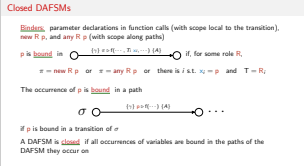
π = new R p or π = any R p or there is i s.t. x_i = p and T = R_i

The occurrence of p is bound in a path

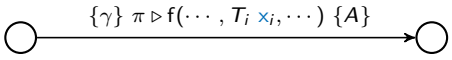


if p is bound in a transition of σ

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

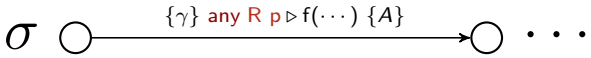


Role emptyness

A transition 

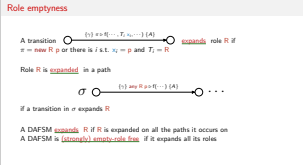
expands role R if
π = new R p or there is i s.t. x_i = p and T_i = R

Role R is expanded in a path



if a transition in σ expands R

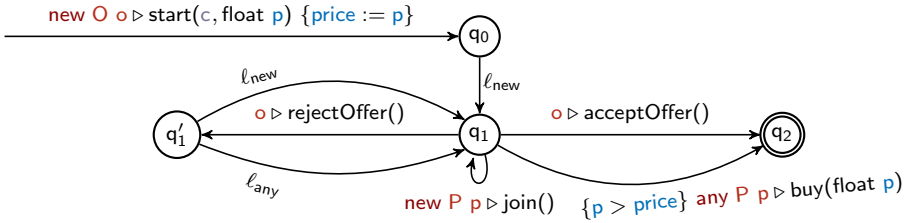
A DAFSM expands R if R is expanded on all the paths it occurs on
A DAFSM is (strongly) empty-role free if it expands all its roles



'expands' means register a new participant with that role in the protocol (the participant might already be registered with a different role)

Exercise: Role emptyness

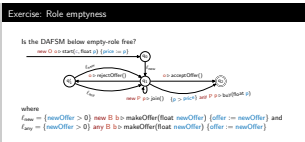
Is the DAFSM below empty-role free?



where

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

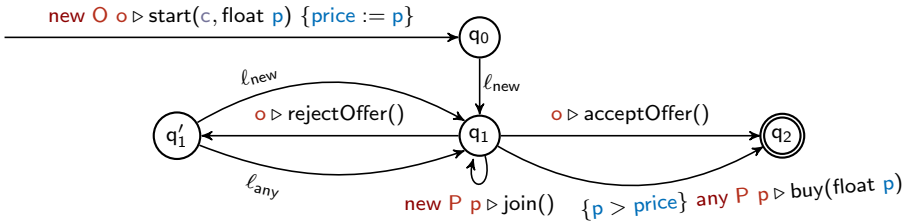
$\ell_{\text{any}} = \{\text{newOffer} > 0\}$ any B b ▷ makeOffer(float newOffer) {offer := newOffer}



No, because of the paths to q_2 excluding the self-loop.

Exercise: Role emptyness

Is the DAFSM below empty-role free?



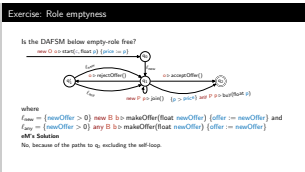
where

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

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

eM's Solution

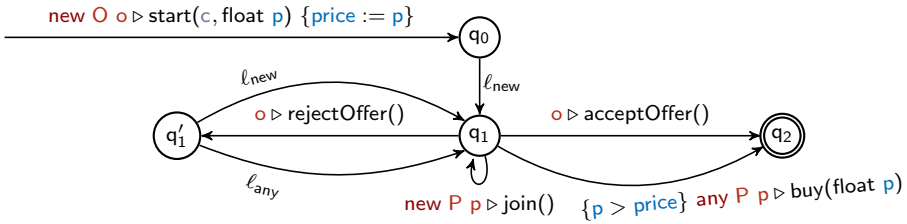
No, because of the paths to q_2 excluding the self-loop.



No, because of the paths to q_2 excluding the self-loop.

Exercise: Role emptyness

Is the DAFSM below empty-role free?



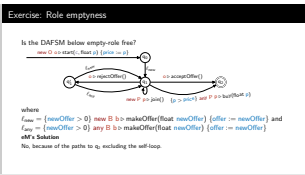
where

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

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

eM's Solution

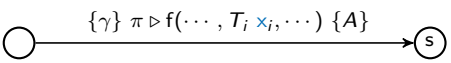
No, because of the paths to q_2 excluding the self-loop.



No, because of the paths to q_2 excluding the self-loop. A possible fix is to bind p in a parameter of the start transition.

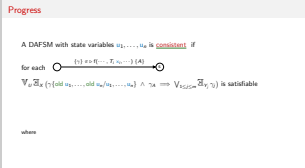
Progress

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

for each 

$\mathbb{\forall}_U \mathbb{\exists}_X \left(\gamma\{\text{old } u_1, \ldots, \text{old } u_n / u_1, \ldots, u_n\} \wedge \gamma_A \implies \bigvee_{1 \le j \le m} \mathbb{\exists}_{Y_j} \gamma_j\right)$ is satisfiable

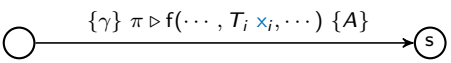
where



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

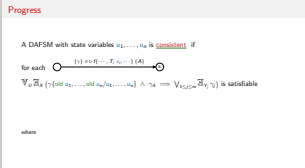
Progress

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

for each 

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

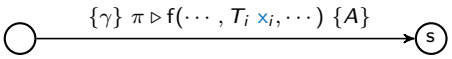
where



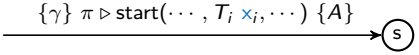
the guard of the transition and the equalities induced by the assignment
imply
the guard of one of the outgoing transition from the target state s

Progress

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

for each 

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

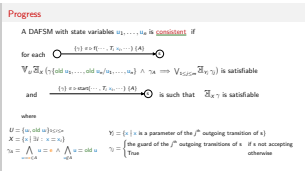
and 

is such that $\exists_X \gamma$ is satisfiable

where

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

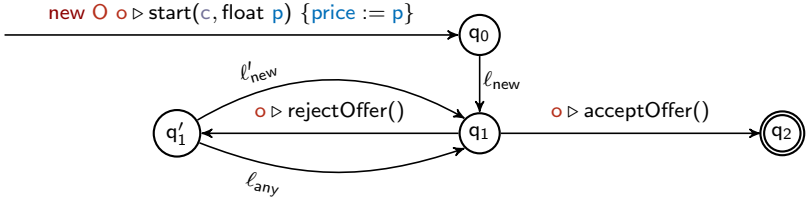
$Y_j = \{x \mid x \text{ is a parameter of the } j^{\text{th}} \text{ outgoing transition of s}\}$
$\gamma_j = \begin{cases} \text{the guard of the } j^{\text{th}} \text{ outgoing transitions of s} & \text{if s not accepting} \\ \text{True} & \text{otherwise} \end{cases}$



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

Exercise: Consistency

Is the DAFSM below consistent?

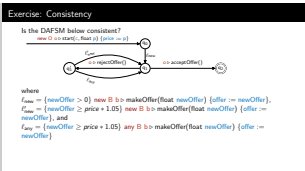


where

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

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

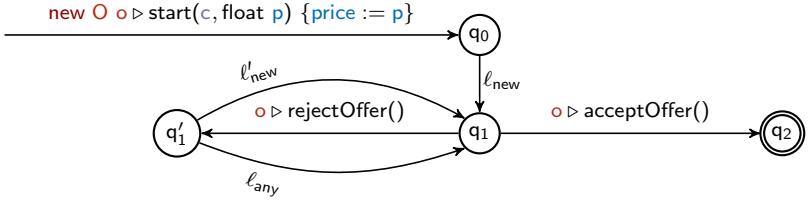
$\ell_{\text{any}} = \{\text{newOffer} \geq \textit{price} * 1.05\}$ any B b ▷ makeOffer(float newOffer) {offer := newOffer}



The DAFSM is consistent.

Exercise: Consistency

Is the DAFSM below consistent?



where

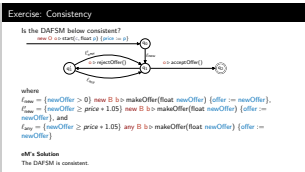
$\ell_{\text{new}} = \{\text{newOffer} > 0\}$ new B b ▷ makeOffer(float newOffer) {offer := newOffer},

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

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

eM's Solution

The DAFSM is consistent.



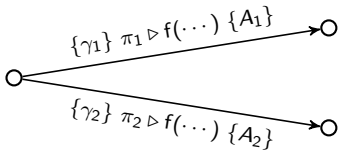
The DAFSM is consistent.

Determinism

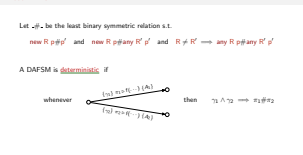
Let $\_\#\_$ be the least binary symmetric relation s.t.

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

A DAFSM is deterministic if

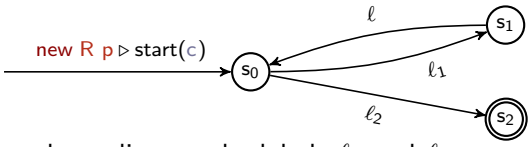
whenever 

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



transitions from the same source state and calling the same function must be told apart by participant instances

Exercise: Determinism

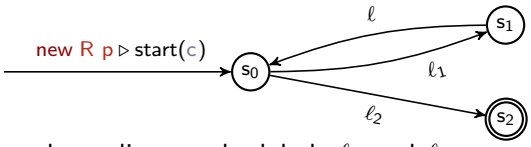
The DAFSM $\mathcal{S} =$ 

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

Exercise: Determinism

Exercise: Determinism

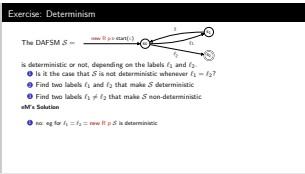
The DAFSM $\mathcal{S} =$ 

is deterministic or not, depending on the labels ℓ_1 and ℓ_2.

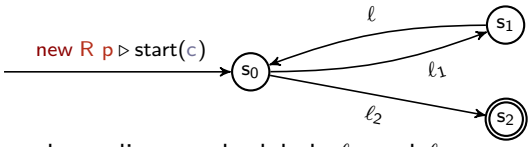
1. Is it the case that $\mathcal{S}$ is not deterministic whenever $\ell_1 = \ell_2$?
2. Find two labels ℓ_1 and ℓ_2 that make $\mathcal{S}$ deterministic
3. Find two labels $\ell_1 \neq \ell_2$ that make $\mathcal{S}$ non-deterministic

eM's Solution

1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic



Exercise: Determinism

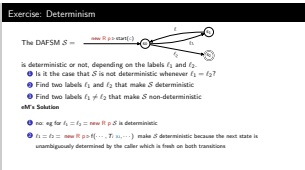
The DAFSM $\mathcal{S} =$ 

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

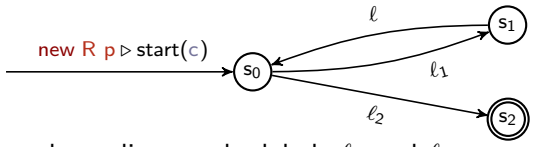
eM's Solution

1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new R p ▷ 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



1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new R p ▷ 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

Exercise: Determinism

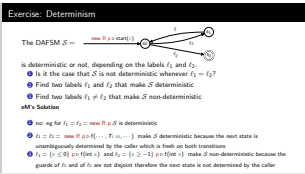
The DAFSM $\mathcal{S} =$ 

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

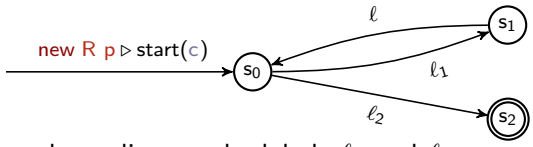
eM's Solution

1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new R p ▷ 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(int x) and $\ell_2 = \{x \geq -1\}$ p ▷ f(int x) 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



1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new R p ▷ 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(int x) and $\ell_2 = \{x \geq -1\}$ p ▷ f(int x) 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

Exercise: Determinism

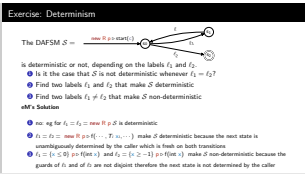
The DAFSM $\mathcal{S} =$ 

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

eM's Solution

1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new R p ▷ 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(int x) and $\ell_2 = \{x \geq -1\}$ p ▷ f(int x) 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



1. no: eg for $\ell_1 = \ell_2 =$ new R p $\mathcal{S}$ is deterministic
2. $\ell_1 = \ell_2 =$ new R p ▷ 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(int x) and $\ell_2 = \{x \geq -1\}$ p ▷ f(int x) 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

closed,

empty-role free,

consistent, and

deterministic

Well-formedness

A DAFSM is well-formed when it is

closed,

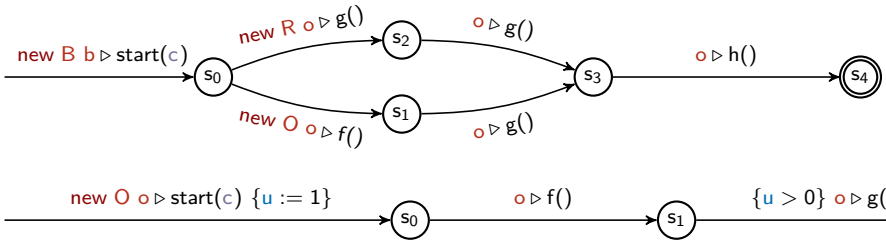
empty-role free,

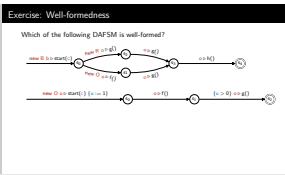
consistent, and

deterministic

Exercise: Well-formedness

Which of the following DAFSM is well-formed?



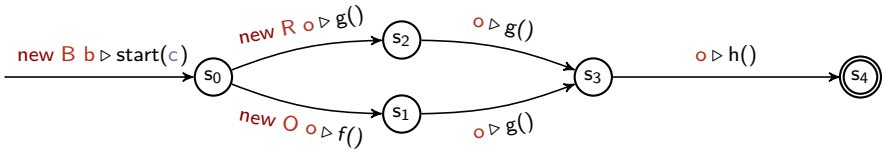


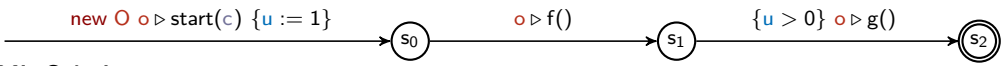
The first is well-formed: o is defined on paths it occurs on and the DAFSM is deterministic.

The second is not well-formed: 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. Note however that this cannot happen 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.

Exercise: Well-formedness

Which of the following DAFSM is well-formed?

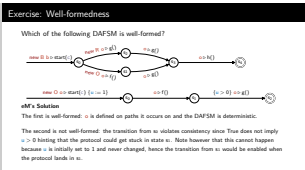




eM's Solution

The first is well-formed: o is defined on paths it occurs on and the DAFSM is deterministic.

The second is not well-formed: 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. Note however that this cannot happen 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.



The first is well-formed: o is defined on paths it occurs on and the DAFSM is deterministic.

The second is not well-formed: 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. Note however that this cannot happen 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 –

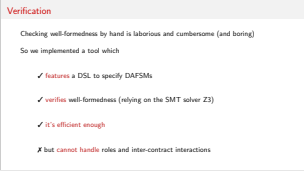
[A tool]

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

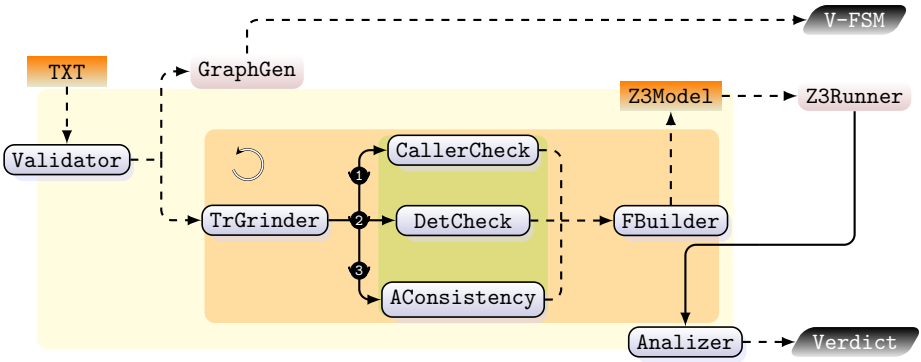


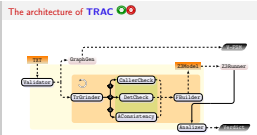
A first prototype got the best artefact price at coordination 2024

There it was used mainly to measure performances of well-formedness checking

Improvements have been made for this tutorial to enhance usability (Thank Elvis)

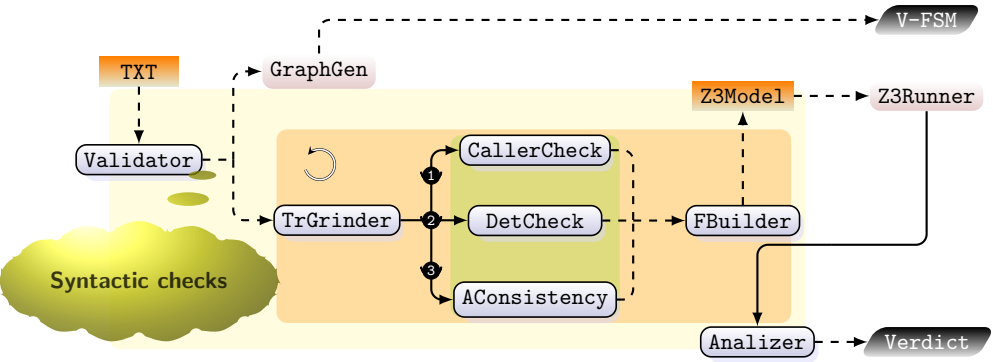
The architecture of TRAC

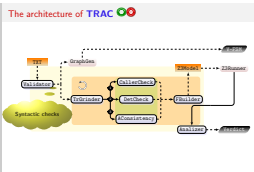




The architecture is compartmentalised into two principal modules: parsing and visualisation (yellow box) and **TRAC**'s core (orange box). The latter module implements well-formedness check (green box). Solid arrows = calls between components ; dashed arrows = data IO.

The architecture of TRAC

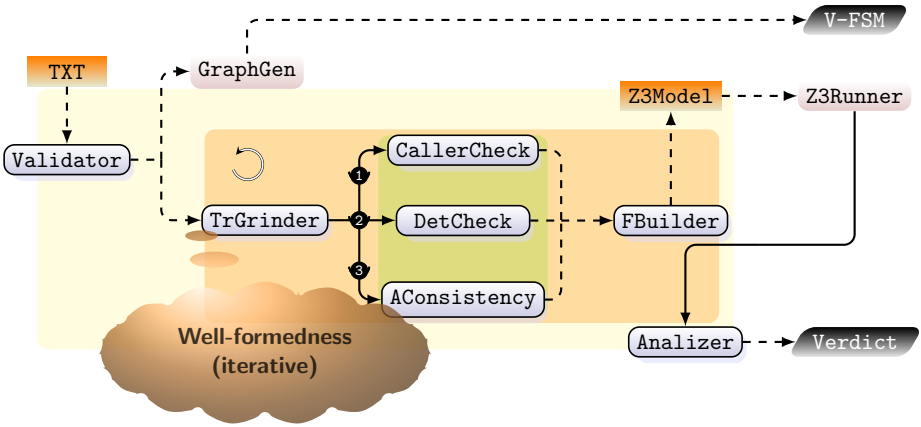


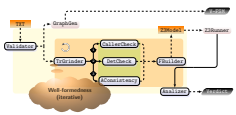


basic syntactic checks on a DSL representation of DAFSMs and transforming the input in a format that simplifies the analysis of the following phases:

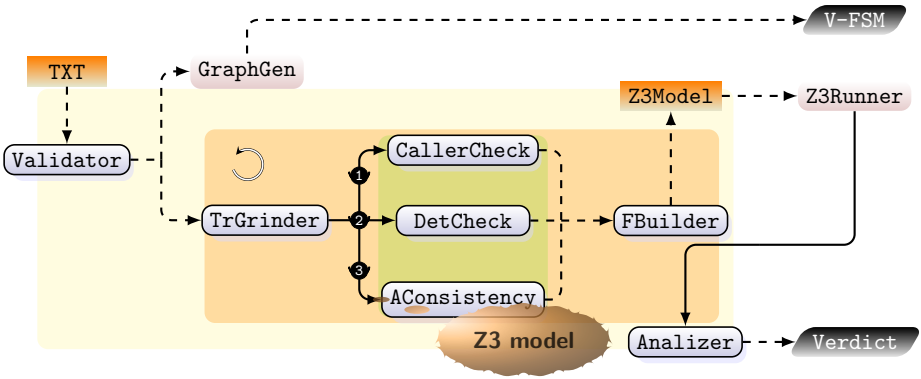
- passed to `GraphGen` for visual representation of DAFSMs (`V-FSM` output)
- passed to the `TrGrinder` component (orange box) for well-formedness checking.

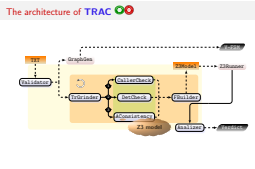
The architecture of TRAC





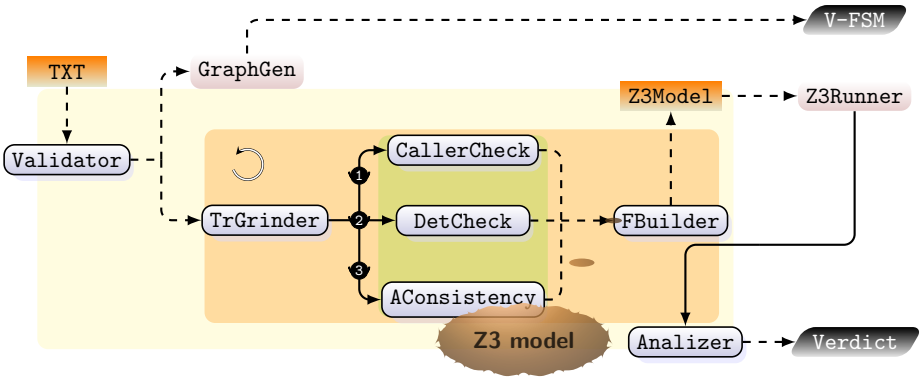
The architecture of TRAC

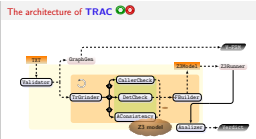




`AConsistency` (arrow ❸) to generate a Z3 formula which holds if, and only if, the transtion is consistent.

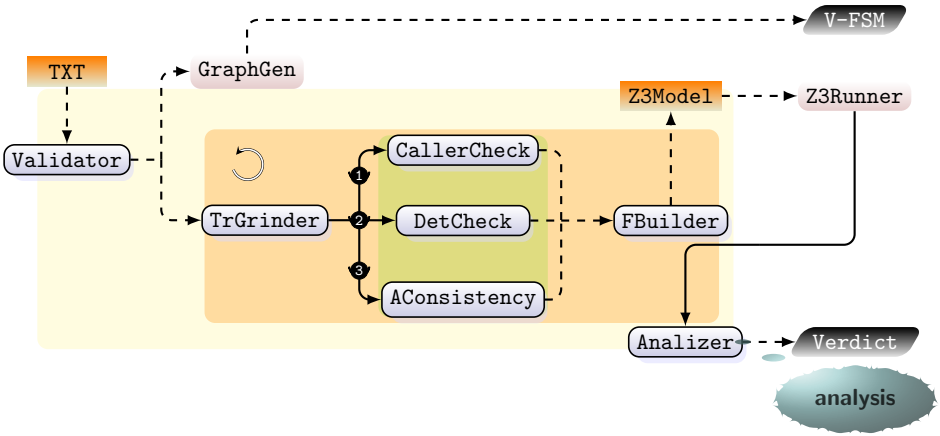
The architecture of TRAC

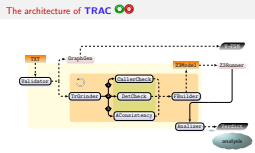




computes the z3 f.la equivalent to the conjunction of the outputs which is then passed to a Z3 engine to check its satisfiability

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.

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`

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)

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

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

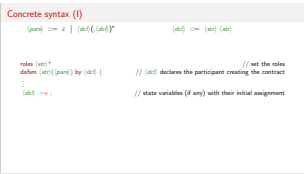
Assignemnts and guard of the instantiation part are optional

guards option are omitted when they are tautologies

Concrete syntax (I)

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

roles $\langle str \rangle^+$ // set the roles
dafsm $\langle str \rangle (\langle pars \rangle)$ by $\langle dcl \rangle$ { // $\langle dcl \rangle$ declares the participant creating the contract
⋮
$\langle dcl \rangle$:=e ; // state variables (if any) with their initial assignment



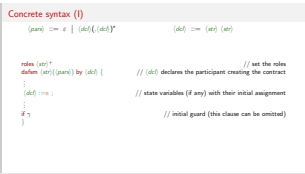
Assignemnts and guard of the instantiation part are optional

guards option are omitted when they are tautologies

Concrete syntax (I)

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

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



Assignemnts and guard of the instantiation part are optional

guards option are omitted when they are tautologies

Concrete syntax (I)

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

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

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

⋮

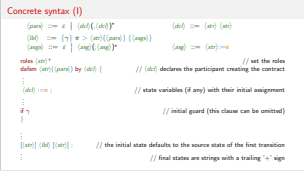
if γ // initial guard (this clause can be omitted)

}

⋮

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

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



Assignemnts and guard of the instantiation part are optional

guards option are omitted when they are tautologies

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 Conterparty cp {}
q0 cp > TransferResponsibility(Conterparty cp) {} q1
q1 any Conterparty cp > TransferResponsibility(Conterparty cp) {} q1
q1 o > Complete() {} q2+
```

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`

eM's Solution

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

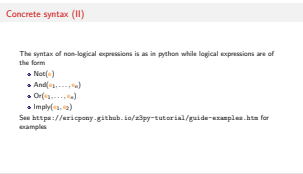
roles Owner Conterparty
dafsm basicProvenance(Owner o) by Conterparty cp {}
q0 cp > TransferResponsibility(Conterparty cp) {} q1
q1 any *Conterparty cp* > TransferResponsibility(Conterparty cp) {} q1
q1 o > Complete() {} q2+

Concrete syntax (II)

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

- Not(e)
- And($e_1, \ldots, e_n$)
- Or($e_1, \ldots, e_n$)
- Imply(e_1, e_2)

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



https://smt-lib.org/papers/smt-lib-reference-v2.6-r2021-05-12.pdf
http://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf
"A <keyword> is a token of the form

Exercise: TRAC syntax (II)

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

Exercise: TRAC syntax (II)
Edit a .trac file for the DAFSM on slide 14.

TODO

– 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.

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]

[q3]
    any Agent a > close() {status :=True}
[q4+]

[q4]
    {balance > =_amount}
    o > withdraw(float _amount)
    {balance :=balance −_amount}
[q4+]
```

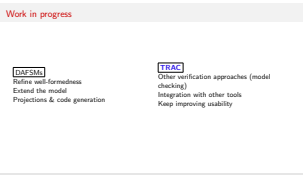

– Epilogue –

[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



WF: consistency can be relax...but would that be feasible?
how do we overcome the limiatations describe before?
can we project on local models and check them? or generate code as done with global types?

Elvis is working on that right now in Denmark
Can we use tools such as ...
Make TRAC more usable (e.g., improve error messages)

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.