ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
DIPARTIMENTO DI
INFORMATICA - SCIENZA E INGEGNERIA

SMART LEGAL CONTRACTS

– AN INTRODUCTION TO STIPULA –

COSIMO LANEVE

cosimo.laneve@unibo.it

4TH SCIENTIFIC SCHOOL ON BLOCKCHAIN & DISTRIBUTED LEDGER TECHNOLOGIES

CAGLIARI

CONTENTS

1. from legal contracts to codes: a brief intro
2. ***Stipula*** by examples
3. agreements, events, assets
4. the operational semantics of ***Stipula*** and the normative equivalence
5. the type inference
6. concluding remarks
7. demos

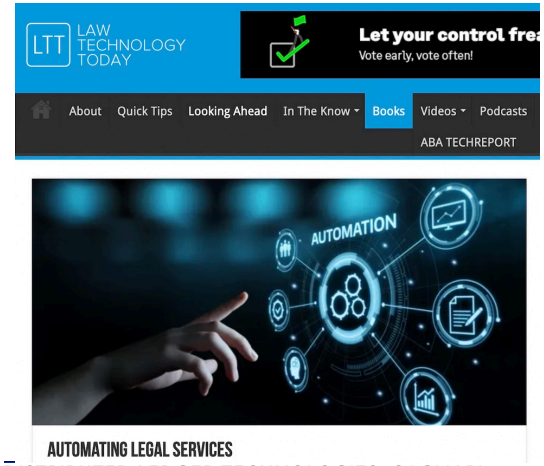
DIGITALIZING LAW

LAW IS BEING DEEPLY INFLUENCED BY THE DIGITAL REVOLUTION

PROS: a sensible digitalization of legal texts might

- enhance data organization and transparency of processes
- identify potential inconsistencies in regulations and reduce ambiguities
- speed-up automatic execution of procedures thus improving efficiency

tons of press releases:



OUR FOCUS

a specific subset of legal documents, the LEGAL CONTRACTS

LEGAL CONTRACTS ARE AGREEMENTS THAT GIVE RISE TO A LEGAL RELATIONSHIP THAT BINDS TWO OR MORE PARTIES

- they establish obligations, rights (such as rights to property), powers, prohibitions and liabilities between the parties,
- often subject to specific conditions and by taking advantage of escrows and securities

PRINCIPLE OF FREEDOM OF FORM: the agreements can be expressed by the parties using the language and medium they prefer

why not programming languages?

CONS: TRUST

TRUST IS A BOTTLENECK FOR DIGITALIZING LEGAL CONTRACTS

people and companies have always relied on the principle of trust between parties involved in legal contractas

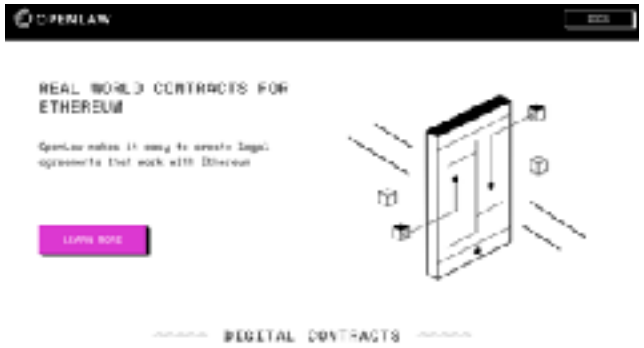
- trust is guaranteed by AUTHORITIES or COURTS
- trust is so fundamental that a number of INTERMEDIARY ROLES have been created
- trust slows down the overall efficiency of the processes and rise up costs

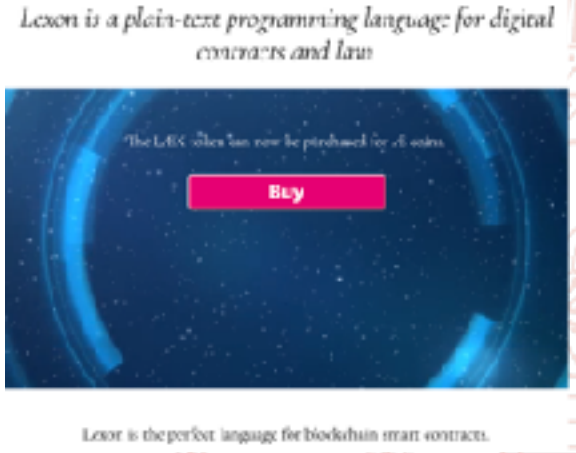
TRUST AND BLOCKCHAINS

BLOCKCHAINS ARE ALGORITHMIC ENABLERS OF TRUST

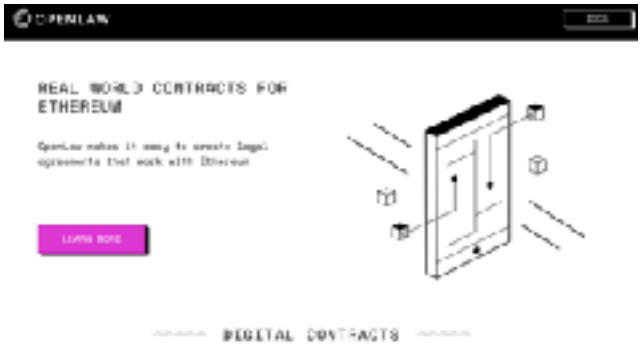
if a legal contract is transposed into a language that is adequate to a blockchain then efficiency might speed-up again!

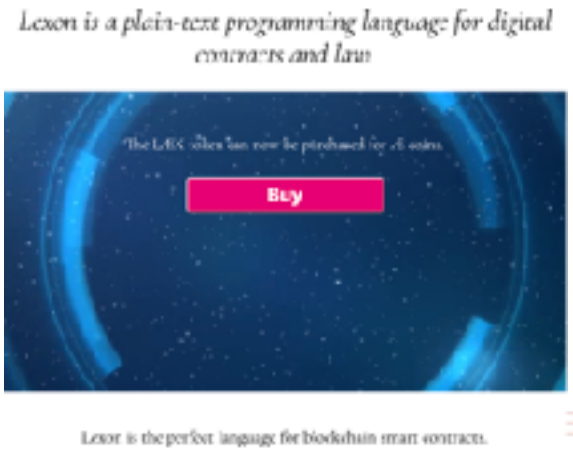
- several projects are being developed for porting legal contracts to blockchain systems: → SMART LEGAL CONTRACT





MAIN PROJECTS ON SMART LEGAL CONTRACTS





they provides an open, standardized format for **smart legal contracts**, consisting of natural language and a **`GUI`**

- contracts can then be interpreted by machines (not necessarily blockchains)
- sets of templates are provided that can be customized by setting template's parameters with appropriate values
- contracts are translated into **`Solidity`** and are automatically triggerered once the agreement is digitally signed by all parties
- parties' signatures are stored in the **`Ethereum`** blockchain

CONS OF MAIN PROJECTS

source legal contracts in ACCORD/LEXON/OPENLAW are written in some natural language

- that have no semantics
- it is not possible to reason on sources
- the coding in (blockchain) programming languages is not formalized
- reasoning on the low-level code is hard

SOURCE CODES SHOULD BE HIGH-LEVEL, CONCISE, DOMAIN-SPECIFIC AND WITH A PRECISE SEMANTICS, THUS SUPPORTING VERIFICATIONS THAT ARE ADEQUATE TO LAWYERS

CONS OF DIGITALIZING LEGAL PROCEDURES/1

transposing legal procedures into technical/programming rules is always problematic

- the **inherent ambiguity** of the legal system is necessary to ensure a proper application of the law on a case-by-case basis
- **regulation by code is always more specific and less flexible** than the legal provisions it claims to implement

DIGITALIZING LEGAL PROCEDURES GIVES SOFTWARE DEVELOPERS AND ENGINEERS THE POWER TO EMBED THEIR OWN INTERPRETATION OF THE LAW INTO THE TECHNICAL ARTEFACTS THAT THEY CREATE

CONS OF DIGITALIZING LEGAL PROCEDURES/2

legal contracts have an intrinsic OPEN NATURE and may depend on TIME:

- may depend on external data, *e.g.* a bet on a football match, insurance against a flight delay
- may depend on conditions that can be hardly digitalized, *e.g.* diligent storage and care in a rental, using a good only as intended, good faith
- may take into account events and time, *e.g.* something must occur within a deadline

THE LEGAL PROCEDURES AND THEIR SEMANTICS MUST TAKE INTO ACCOUNT PARTIES, SUCH AS AUTHORITIES OR ORACLES, AND TIME

OUR RESEARCH PROGRAM

define a programming language for legal contracts

- that is pivoted on few selected, concise and intelligible primitives, together with a precise formalisation of its syntax and semantics
- whose theory provides static analysis and verification tools
- whose design and definition is implementation agnostic, but it may be compiled to full-fledged programming languages and platforms
- that should be easy-to-use and to understand for legal practitioners and ordinary citizens

Stipula https://github.com/stipula-language

THE *STIPULA* GROUP: me + S. Crafa + A. Parenti + G. Sartor + A. Veschetti + students

Stipula BY EXAMPLE

the Deposit contract

1. Term.

This Agreement is between the Farm and the Client and remain in full force and effect till deadline from its starting date.

2. Storage.

The Farm shall store flour into deposit after the starting date.

3. Payment.

The flour will be paied Euro cost per kg by the Client in advance. Then the corresponding amount of flour is delivered to Client and the payed amount is sent to Farm.

4. Termination.

This Agreement shall terminate on the date specified in Article 1. At that date, all the flour stored in deposit is returned to Farm.

5. Disputes.

Every dispute arising from the relationship governed by the above general conditions will be managed by the Court the Farm company is based, which will decide compensations for Lender and Borrower.

Stipula BY EXAMPLE

SIMILAR TO A CLASS IN OO-PROGRAMMING

- with fields (and assets)
- with constructor = agreement
- with methods (where we also specify callers)

```
stipula Deposit {
    parties Client, Farm
    fields cost, deadline
    assets deposit

    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run {   deposit ⊸ Farm   } ⇛ @End
        } ⇛ @Run

    @Run Farm: send()[h]{
            h → Client      h ⊸ deposit
        } ⇛ @Run

    @Run Client: buy()[w] (w/cost_flour ≤ flour){
            (w/cost_flour)×flour ⊸ Client      w ⊸ Farm
        } ⇛ @Run
}
```

Stipula BY EXAMPLE

SIMILAR TO A CLASS IN OO-PROGRAMMING

```
stipula Deposit {
    parties Client, Farm
    fields cost, deadline
    assets deposit
```

- with constructor = agreement

```
    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run {   deposit —○ Farm   } ⇒ @End
        } ⇒ @Run
```

MEETING OF MINDS

the two parties express their consent

- to participate to the contract — technically a **multiparty synchronization**
- to set the values of the fields cost and deadline
- about the termination protocol

the contract starts its legal effects by entering in the state `@Run`

Stipula BY EXAMPLE

STIPULA HAS A STATE-BASED PROGRAMMING STYLE

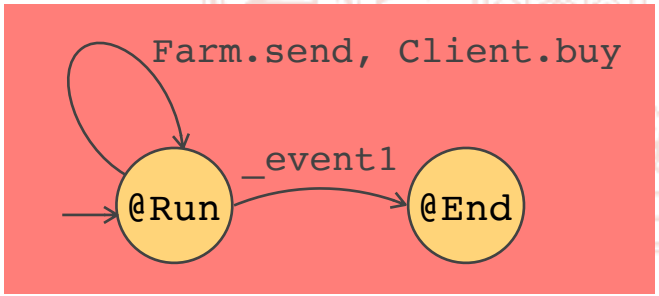
- widely used to specify **interaction protocols**
- encoding of **permissions** and **prohibitions**

```
stipula Deposit {
    parties Client, Farm
    fields cost, deadline
    assets deposit

    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run { deposit ⊸ Farm } ⇒ @End
        } ⇒ @Run

    @Run Farm: send()[h]{
            h → Client     h ⊸ deposit
        } ⇒ @Run

    @Run Client: buy()[w] (w/cost_flour ≤ flour){
            (w/cost_flour)×flour ⊸ Client     w ⊸ Farm
        } ⇒ @Run
}
```



Stipula BY EXAMPLE

STIPULA HAS EVENTS

- to encode **obligations**

by scheduling a **future statement** that automatically execute a corresponding action at a given time

```
stipula Deposit {
    parties Client, Farm
    fields cost, deadline
    assets deposit

    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run {  deposit —∘ Farm  } ⇒ @End
        } ⇒ @Run
```

- `now + deadline` is a **time-expression**
- the `deposit` is returned to `Farm` if the contract is in the `@Run` state

Stipula BY EXAMPLE

STIPULA ADHERE TO RESOURCE-AWARE PROGRAMMING

```
stipula Deposit {
    parties Client, Farm
    fields cost, deadline
    assets deposit

    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run {  deposit —∘ Farm  } ⇒ @End
        } ⇒ @Run

    @Run Farm: send()[h]{
            h → Client    h —∘ deposit
        } ⇒ @Run

    @Run Client: buy()[w] (w ≤ deposit×cost){
            (w/(deposit×cost))×deposit —∘ Client    w —∘ Farm
        } ⇒ @Run
}
```

value operations: `h → Client`

asset operations: `h —∘ deposit`

`h` is an asset formal parameter

square brackets!

- `h → Client` means that the value of `h` is sent to `Client`; `h` is not modified
- `h —∘ deposit` means that the value of `h` is moved to `deposit`; the value of `deposit` is **augmented** by the value of `h` and `h` is **emptied**

RESOURCE-AWARE PROGRAMMING IN *Stipula*

- assets are **linear resources** like (crypto) **currencies** or **tokens**
- useful for **payments**, **escrows** and **securities**
- assets cannot be **forged**, nor **double-spent**
- assets **should not be locked** in contracts (**liquidity**)

```
stipula Deposit {
    parties Client, Farm
    fields cost, deadline
    assets deposit

    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run {  deposit ⊸ Farm  } ⇒ @End
        } ⇒ @Run

    @Run Farm: send()[h]{
            h → Client     h ⊸ deposit
        } ⇒ @Run

    @Run Client: buy()[w] (w/cost_flour ≤ flour){
            (w/cost_flour)×flour ⊸ Client     w ⊸ Farm
        } ⇒ @Run
}
```

when `e = h`

- `e × h ⊸ h'` is shortened into `h ⊸ h'`

A PARADIGMATIC MOVE OPERATION

$e \times h \multimap h'$ is defined provided $0 \leq e \leq h$ and it means

- `h' = h' + e`
- `h = h - e`

Stipula, LEGALLY

standard **legal patterns** corresponds to precise ***Stipula* patterns**

meeting of the minds	agreement primitive
permissions, prohibitions	state-based programming
obligations	event primitive
transfer of currency or other assets	asset-aware (linear) programming
openness to external conditions or data	Intermediary pattern
judicial enforcement and exceptional behaviours	Authority pattern

Stipula, FORMALLY

Stipula has a precise **mathematical theory**

clear semantics

- the **syntax** and **operational semantics** are formally defined
- the execution prevents **unsafe asset operations**, e.g. attemting to drain too much value from an asset or to forge new assets

observational equivalence

- an equational theory based on **bisimulation** is developed that identifies contracts that differ for hidden elements

type inference

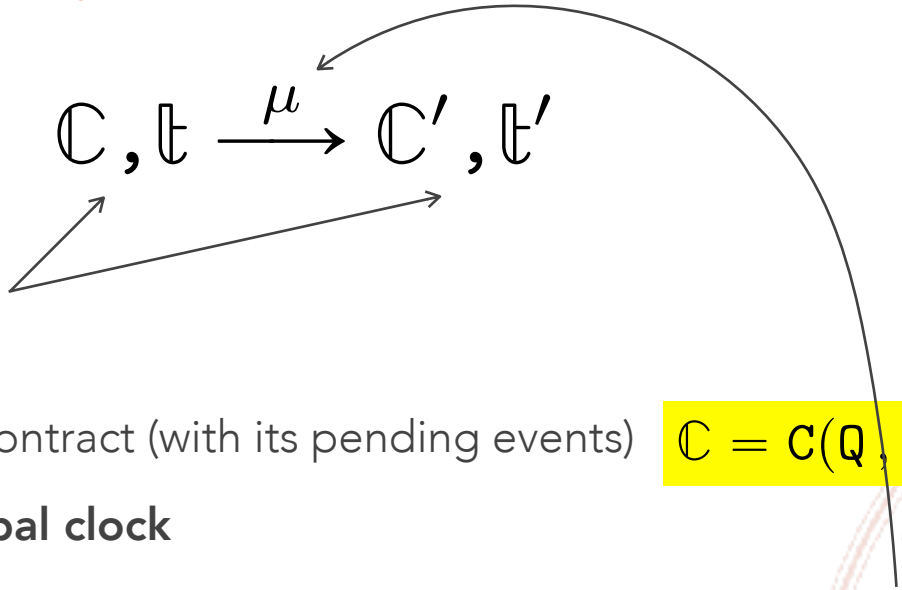
- *Stipula* is untyped, but **an algorithm for deriving types** of fields, assets and functions has been defined so that basic programming errors are prevented

liquidity analyzer

- a technique has been developed that statically verifies if a **contract do not freeze any asset forever** (liquid)

Stipula OPERATIONAL SEMANTICS

a labelled transition system



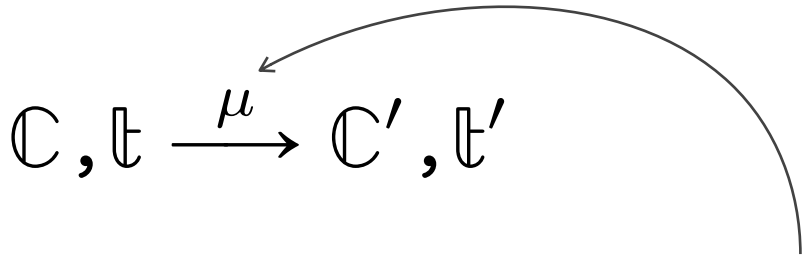
runtime configurations

- $\mathbb{C}$ is the **state** of the contract (with its pending events) $\mathbb{C} = \mathtt{C}(\mathtt{Q}, \ell, S\,W \Rightarrow @\mathtt{Q}', \Psi)$
- $\mathbb{t}$ is the **system's global clock**

labels highlight that the execution requires the **interaction with the external environment**

- *c.f.* the **open** nature of contract's behaviour
- $\mu \quad ::= \quad \_ \quad | \quad (\overline{A}, \overline{A_1} : \overline{v_1}, \cdots, \overline{A_n} : \overline{v_n}) \quad | \quad A : \mathtt{f}(\overline{u})[\overline{v}] \quad | \quad v \rightarrow A \quad | \quad v \multimap A$

Stipula SEMANTICS/LABELS



$$\mathbb{C}, \mathbb{t} \xrightarrow{\mu} \mathbb{C}', \mathbb{t}'$$

$$\mu \quad ::= \quad \_ \quad | \quad (\overline{A},\ \overline{A_1} : \overline{v_1}, \cdots, \overline{A_n} : \overline{v_n}) \quad | \quad A : \mathtt{f}(\overline{u})[\overline{v}] \quad | \quad v \rightarrow A \quad | \quad v \multimap A$$

- $\_$ is an **internal** transition of the contract, *e.g.* update of a field
- $(\overline{A},\ \overline{A_1} : \overline{v_1}, \cdots, \overline{A_n} : \overline{v_n})$ is the **agreement label**
  - $\overline{A}$ defines who is taking the legal responsability for each contract role (who are the **actual parties**)
  - $\overline{A_i} : \overline{v_i}$ defines what are the terms of the contracts, *i.e.* the **agreed initial values of the contract's fields**
- $A : \mathtt{f}(\overline{u})[\overline{v}]$ is the **possibility** at time $\mathbb{t}$ to invoke the function `f` by the party *A*
- $u \rightarrow A$ and $v \multimap A$ are the sending of the value *u* and of the asset *v* to the party *A* (at time $\mathbb{t}$)

Stipula SEMANTICS/THE AGREEMENT

[Agree]

$$\texttt{assets}\ \overline{\texttt{h}}\ \in\ \texttt{C} \qquad \texttt{agreement}(\overline{\texttt{A}})\{\ \overline{\texttt{A}_1 : \texttt{x}_1}\ \cdots\ \overline{\texttt{A}_n : \texttt{x}_n}\ W\ \} \Rightarrow \texttt{@Q}\ \in\ \texttt{C} \qquad [\![W\{^{\mathbb{t}}/_{\texttt{now}}\}]\!]_{\varnothing} = \Psi$$

$$\texttt{C}(\_, \varnothing, \_, \_)\ ,\ \mathbb{t} \xrightarrow{(\overline{A}, \overline{A_i : v_i}^{\,i \in 1..n})} \texttt{C}(\texttt{Q}, [\overline{\texttt{A}} \mapsto \overline{A}, \overline{\texttt{x}_i} \mapsto \overline{v_i}^{\,i \in 1..n}, \overline{\texttt{h}} \mapsto \overline{0}], \_, \Psi)\ ,\ \mathbb{t}$$

- $\texttt{C}(\_, \varnothing, \_, \_)$ is the initial runtime configuration of the contract C is
- the contract transits into the state Q
- and into a memory
  - that defines the **actual parties**
  - defines the **values of the contract's fields**
  - sets to 0 the values of the assets
- and with events whose time guards are computed

Stipula SEMANTICS/FUNCTIONS AND EVENTS

[FUNCTION]

$$\frac{\texttt{@Q A : f}(\overline{\texttt{y}})\,[\overline{\texttt{k}}]\ (E)\,\{\,S\ W\,\}\Rightarrow\texttt{@Q}' \in \texttt{C} \qquad \Psi\ ,\ \mathbb{t} \nrightarrow \qquad \ell(\texttt{A}) = A \quad \ell' = \ell[\overline{\texttt{y}} \mapsto \overline{u}, \overline{\texttt{k}} \mapsto \overline{v}] \quad [\![E]\!]_{\ell'} = \texttt{true}}{\texttt{C}(\texttt{Q}\,,\,\ell\,,\,\_\,,\,\Psi)\ ,\ \mathbb{t} \xrightarrow{A:\texttt{f}(\overline{u})[\overline{v}]} \texttt{C}(\texttt{Q}\,,\,\ell'\,,\,S\ W \Rightarrow \texttt{@Q}'\,,\,\Psi)\ ,\ \mathbb{t}}$$

no event is possible at $\mathbb{t}$

these are standard actual parameters

these are **resource** actual parameters

- they must be all positive!

an event is possible at $\mathbb{t}$

[EVENT-MATCH]

$$\frac{\Psi = \mathbb{t} \gg \texttt{@Q}\ \{\ S\ \} \Rightarrow \texttt{@Q}'\ \mid \Psi'}{\texttt{C}(\texttt{Q}\,,\,\ell\,,\,\_\,,\,\Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\,\ell\,,\,S \Rightarrow \texttt{@Q}'\,,\,\Psi')\ ,\ \mathbb{t}}$$

there is **no statement to execute**

BEWARE:

Stipula is NONDETERMINISTIC!

Stipula SEMANTICS/FIELDS AND ASSETS

this is the standard **evaluation function** for expressions

[VALUE-SEND]

$$\frac{[\![E]\!]_\ell = v \quad \ell(\mathtt{A}) = A}{\mathtt{C}(\mathtt{Q}, \ell, E \to \mathtt{A}\ \Sigma, \Psi)\ ,\ \mathbb{t} \xrightarrow{v \to A} \mathtt{C}(\mathtt{Q}, \ell, \Sigma, \Psi)\ ,\ \mathbb{t}}$$

the label indicates that we are **sending** a value to a party

this is the **evaluation function for assets** — let $\ell(\mathtt{h}) = 2$

- $[\![E]\!]_\ell^{\mathtt{a}} = 1$ then $[\![h - a]\!]_\ell^{\mathtt{a}}$ is defined: the result is 1!
- $[\![E]\!]_\ell^{\mathtt{a}} = 3$ then $[\![h - a]\!]_\ell^{\mathtt{a}}$ is not defined: the result must be nonnegative!

[ASSET-SEND]

$$\frac{[\![E]\!]^{\mathtt{a}}_\ell = a \quad \ell(\mathtt{A}) = A \quad [\![\mathtt{h} - a]\!]^{\mathtt{a}}_\ell = a'}{\mathtt{C}(\mathtt{Q}, \ell, E \times \mathtt{h} \multimap \mathtt{A}\ \Sigma, \Psi)\ ,\ \mathbb{t} \xrightarrow{a \multimap A} \mathtt{C}(\mathtt{Q}, \ell[\mathtt{h} \mapsto a'], \Sigma, \Psi)\ ,\ \mathbb{t}}$$

the label highlights that we are **moving an asset**

the result must be positive!

Stipula SEMANTICS/THE TICK RULE

QUESTION: when the time elapses in ***Stipula***?

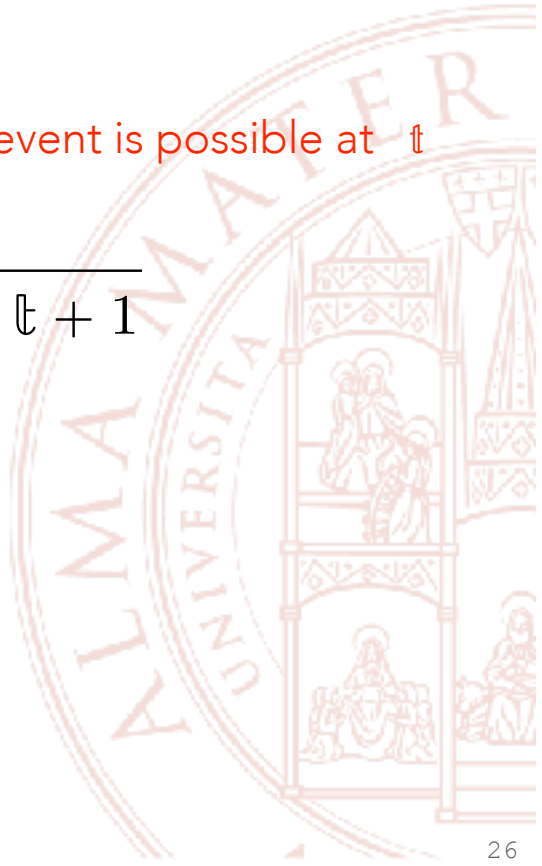
when there is an **empty statement to execute** and **no event can be triggered**

— the contract is idle —

$$[\textsc{Tick}]\quad \frac{\Psi \ , \ \mathbb{t} \nrightarrow}{\mathtt{C}(\mathtt{Q}\,,\,\ell\,,\,\_\,,\,\Psi)\ ,\ \mathbb{t} \longrightarrow \mathtt{C}(\mathtt{Q}\,,\,\ell\,,\,\_\,,\,\Psi)\ ,\ \mathbb{t}+1}$$

no event is possible at $\mathbb{t}$

from $\mathbb{t}$ to $\mathbb{t}+1$



Stipula SEMANTICS/THE SETS OF TRANSITIONS

[Agree]

$$\frac{\texttt{assets}\ \overline{\texttt{h}}\ \in\ \texttt{C} \qquad \texttt{agreement}(\overline{\texttt{A}})\{\ \overline{\texttt{A}_1 : \texttt{x}_1}\ \cdots\ \overline{\texttt{A}_n : \texttt{x}_n}\ W\ \} \Rightarrow \texttt{@Q}\ \in\ \texttt{C} \qquad [\![W\{^{\mathbb{t}}/_{\texttt{now}}\}]\!]_{\varnothing} = \Psi}{\texttt{C}(\_\,,\varnothing\,,\_\,,\_)\ ,\ \mathbb{t} \xrightarrow{(\overline{A},\ \overline{A_i : \overline{v_i}}^{\,i\in 1..n})} \texttt{C}(\texttt{Q}\,,\ [\overline{\texttt{A}} \mapsto \overline{A}, \overline{\texttt{x}_i}\ \mapsto\ \overline{v_i}^{\,i\in 1..n}, \overline{\texttt{h}}\ \mapsto\ \overline{0}]\,,\ \_\,,\ \Psi)\ ,\ \mathbb{t}}$$

[Function]

$$\frac{\texttt{@Q A} : \texttt{f}(\overline{\texttt{y}})\,[\overline{\texttt{k}}]\ (E)\,\{\,S\,W\,\} \Rightarrow \texttt{@Q}' \in \texttt{C} \qquad \Psi\ ,\ \mathbb{t} \nrightarrow \qquad \ell(\texttt{A}) = A \quad \ell' = \ell[\overline{\texttt{y}} \mapsto \overline{u}, \overline{\texttt{k}} \mapsto \overline{v}] \quad [\![E]\!]_{\ell'} = \texttt{true}}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ \_\,,\ \Psi)\ ,\ \mathbb{t} \xrightarrow{A:\texttt{f}(\overline{u})[\overline{v}]} \texttt{C}(\texttt{Q}\,,\ \ell'\,,\ S\,W \Rightarrow \texttt{@Q}'\,,\ \Psi)\ ,\ \mathbb{t}}$$

[State-Change]

$$\frac{[\![W\{^{\mathbb{t}}/_{\texttt{now}}\}]\!]_{\ell} = \Psi'}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ \_\,W \Rightarrow \texttt{@Q}'\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}'\,,\ \ell\,,\ \_\,,\ \Psi' \mid \Psi)\ ,\ \mathbb{t}}$$

[Event-Match]

$$\frac{\Psi = \mathbb{t} \gg \texttt{@Q}\ \{\ S\ \} \Rightarrow \texttt{@Q}' \mid \Psi'}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ \_\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\ \ell\,,\ S \Rightarrow \texttt{@Q}'\,,\ \Psi')\ ,\ \mathbb{t}}$$

[Tick]

$$\frac{\Psi\ ,\ \mathbb{t} \nrightarrow}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ \_\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\ \ell\,,\ \_\,,\ \Psi)\ ,\ \mathbb{t}+1}$$

[Value-Send]

$$\frac{[\![E]\!]_{\ell} = v \quad \ell(\texttt{A}) = A}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ E \to \texttt{A}\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t} \xrightarrow{v \to A} \texttt{C}(\texttt{Q}\,,\ \ell\,,\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t}}$$

[Asset-Send]

$$\frac{[\![E]\!]^{\texttt{a}}_{\ell} = a \quad \ell(\texttt{A}) = A \quad [\![\texttt{h} - a]\!]^{\texttt{a}}_{\ell} = a'}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ E \times \texttt{h} \multimap \texttt{A}\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t} \xrightarrow{a \multimap A} \texttt{C}(\texttt{Q}\,,\ \ell[\texttt{h} \mapsto a']\,,\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t}}$$

[Field-Update]

$$\frac{[\![E]\!]_{\ell} = v}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ E \to \texttt{x}\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\ \ell[\texttt{x} \mapsto v]\,,\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t}}$$

[Asset-Update]

$$\frac{[\![E]\!]^{\texttt{a}}_{\ell} = a \quad [\![\texttt{h} - a]\!]^{\texttt{a}}_{\ell} = a' \quad [\![\texttt{h}' + a]\!]^{\texttt{a}}_{\ell} = a'' \quad \ell' = \ell[\texttt{h} \mapsto a', \texttt{h}' \mapsto a'']}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ E \times \texttt{h} \multimap \texttt{h}'\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\ \ell'\,,\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t}}$$

[Cond-true]

$$\frac{[\![E]\!]_{\ell} = \texttt{true}}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ (\texttt{if}\ (E)\,\{\,S\,\}\,\texttt{else}\,\{\,S'\,\}\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\ \ell\,,\ S\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t}}$$

[Cond-false]

$$\frac{[\![E]\!]_{\ell} = \texttt{false}}{\texttt{C}(\texttt{Q}\,,\ \ell\,,\ \texttt{if}\ (E)\,\{\,S\,\}\,\texttt{else}\,\{\,S'\,\}\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t} \longrightarrow \texttt{C}(\texttt{Q}\,,\ \ell\,,\ S'\ \Sigma\,,\ \Psi)\ ,\ \mathbb{t}}$$

THE NORMATIVE EQUIVALENCE

AIM: identify two contracts that are **legally equivalent**

— no-one using them can set the two contracts apart —

this captures the observation of permissions and prohibitions

this equivalence identifies two agreement labels that are equal up-to permuitations of lists

Definition (Normative Equivalence). *A symmetric relation* $\mathcal{R}$ *is a bisimulation between two configurations at time* $\mathbb{t}$*, written* $\mathbb{C}_1, \mathbb{t} \;\; \mathcal{R} \;\; \mathbb{C}_2, \mathbb{t}$*, whenever*

1. *if* $\mathbb{C}_1, \mathbb{t} \stackrel{\alpha}{\Longrightarrow} \mathbb{C}'_1, \mathbb{t}$ *then* $\mathbb{C}_2, \mathbb{t} \stackrel{\alpha'}{\Longrightarrow} \mathbb{C}'_2, \mathbb{t}$ *for some* α' *such that* $\alpha \sim \alpha'$ *and* $\mathbb{C}'_1, \mathbb{t} \;\; \mathcal{R} \;\; \mathbb{C}'_2, \mathbb{t}$;
2. *if* $\mathbb{C}_1, \mathbb{t} \stackrel{\mu_1}{\Longrightarrow} \cdots \stackrel{\mu_n}{\Longrightarrow} \mathbb{C}'_1, \mathbb{t} \longrightarrow \mathbb{C}'_1, \mathbb{t}+1$ *then there exist* $\mu'_1 \cdots \mu'_n$ *that is a permutation of* $\mu_1 \cdots \mu_n$ *such that* $\mathbb{C}_2, \mathbb{t} \stackrel{\mu'_1}{\Longrightarrow} \cdots \stackrel{\mu'_n}{\Longrightarrow} \mathbb{C}'_2, \mathbb{t} \longrightarrow \mathbb{C}'_2, \mathbb{t}+1$ *and* $\mathbb{C}'_1, \mathbb{t}+1 \;\; \mathcal{R} \;\; \mathbb{C}'_2, \mathbb{t}+1$.

this is an instance of the TICK-RULE

Let $\simeq$ *be the largest bisimulation, called* normative equivalence.

this abstract away the ordering of observations within the same clock

THE NORMATIVE EQUIVALENCE/THEOREMS

Theorem 1 (Internal refactoring). *Let* $\mathtt{C}$ *and* $\mathtt{C}'$ *be two contracts that are equal up-to a bijective renaming of states. Then* $\mathtt{C} \simeq \mathtt{C}'$. *Similarly, for bijective renaming of assets, fields and contract names.*

internal names are irrelevant

Theorem 2 (Time shift).

1. *If* $\mathbb{C}, \mathbb{t} \simeq \mathbb{C}', \mathbb{t}$ *and* $\mathbb{t} \leqslant \mathbb{t}'$, *then* $\mathbb{C}, \mathbb{t}' \simeq \mathbb{C}', \mathbb{t}'$.
2. *If* $\mathbb{t} < \mathbb{t}'$ *then* $\mathtt{C}(\mathtt{Q}, \ell, \Sigma, \Psi \mid \mathbb{t} \gg \mathtt{@Q}\ \{\ \mathtt{S}\ \} \Rightarrow \mathtt{@Q}'), \mathbb{t}' \simeq \mathtt{C}(\mathtt{Q}, \ell, \Sigma, \Psi), \mathbb{t}'$.

garbage-collection of events that cannot be triggered anymore because the time for their scheduling is already elapsed

THE NORMATIVE EQUIVALENCE/THEOREMS

the order of operation is irrelevant, provided they do not interfere

Theorem 3. *The following non-interference laws hold in* Stipula *(whenever they are applicable, we assume* $\mathtt{x}, \mathtt{h}, \mathtt{h}' \notin fv(E')$ *and* $\mathtt{x}', \mathtt{h}'', \mathtt{h}''' \notin fv(E)$*:*

$$
\begin{array}{rcl}
E \to \mathtt{A} \;\; E' \to \mathtt{A}' & \simeq & E' \to \mathtt{A}' \;\; E \to \mathtt{A} \\
E \to \mathtt{x} \;\; E' \to \mathtt{A} & \simeq & E' \to \mathtt{A} \;\; E \to \mathtt{x} \\
E \to \mathtt{x} \;\; E' \to \mathtt{x}' & \simeq & E' \to \mathtt{x}' \;\; E \to \mathtt{x} \\
E \times \mathtt{h} \multimap \mathtt{A} \;\; E' \to \mathtt{A}' & \simeq & E' \to \mathtt{A}' \;\; E \times \mathtt{h} \multimap \mathtt{A} \\
E \times \mathtt{h} \multimap \mathtt{A} \;\; E' \to \mathtt{x}' & \simeq & E' \to \mathtt{x}' \;\; E \times \mathtt{h} \multimap \mathtt{A} \\
E \times \mathtt{h} \multimap \mathtt{h}' \;\; E' \to \mathtt{A} & \simeq & E' \to \mathtt{A} \;\; E \times \mathtt{h} \multimap \mathtt{h}' \\
E \times \mathtt{h} \multimap \mathtt{h}' \;\; E' \to \mathtt{x}' & \simeq & E' \to \mathtt{x}' \;\; E \times \mathtt{h} \multimap \mathtt{h}' \\
E \times \mathtt{h} \multimap \mathtt{A} \;\; E' \times \mathtt{h}'' \multimap \mathtt{A}' & \simeq & E' \times \mathtt{h}'' \multimap \mathtt{A}' \;\; E \times \mathtt{h} \multimap \mathtt{A} \\
E \times \mathtt{h} \multimap \mathtt{A} \;\; E' \times \mathtt{h}'' \multimap \mathtt{h}''' & \simeq & E' \times \mathtt{h}'' \multimap \mathtt{h}''' \;\; E \times \mathtt{h} \multimap \mathtt{A} \\
E \times \mathtt{h} \multimap \mathtt{h}' \;\; E' \times \mathtt{h}'' \multimap \mathtt{h}''' & \simeq & E' \times \mathtt{h}'' \multimap \mathtt{h}''' \;\; E \times \mathtt{h} \multimap \mathtt{h}'
\end{array}
$$

TYPE INFERENCE

derive types of assets, fields and functions

- and demonstrate that the type system is **sound** — well-typed contracts do not evolve to unsound states
- well typed configurations **just stuck on** an attempt of doing an unsafe asset operation, an access to an uninitialized field or a division by 0

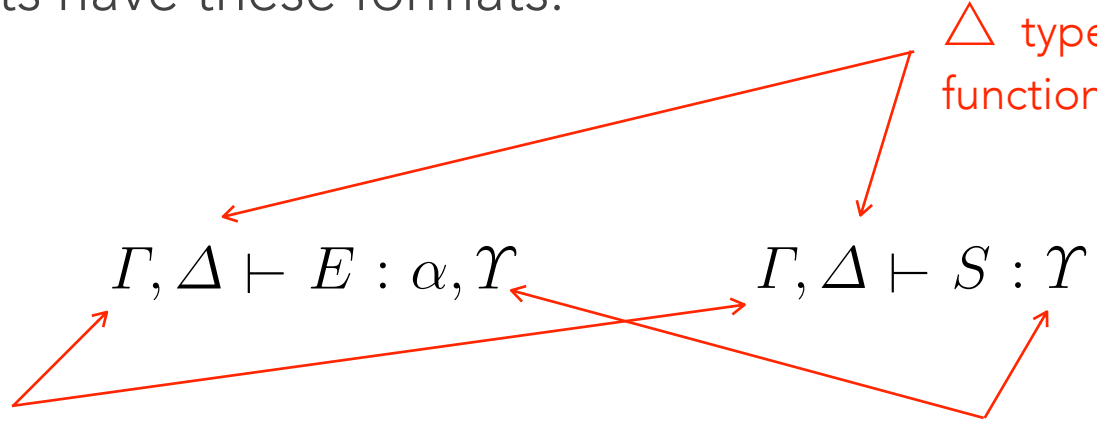
Stipula types: $T ::= \texttt{real} \mid \texttt{bool} \mid \texttt{string} \mid \texttt{time} \mid \texttt{asset}$

the algorithm for deriving types:

- use **type variables** for typing unknown snippets
- operations add **constraints** to type variables
- collect constraints by **parsing the code**
- the typing is **the best solution** of the set of constraints

TYPE INFERENCE/JUDGMENTS

judgments have these formats:



Δ types assets and asset functions' parameters

Γ types fields and non-asset functions' parameters

Υ is the set of constraints on type variables

α is the type of E

TYPE INFERENCE/RULES

basic rules:

[T-SEND]

$$\frac{\Gamma, \Delta \vdash E : \alpha, \Upsilon}{\Gamma, \Delta \vdash E \to \mathtt{A} : \Upsilon}$$

no constraint is added to Υ

[T-ASEND]

$$\frac{\Gamma, \Delta \vdash E : \alpha, \Upsilon}{\Gamma, \Delta \vdash E \times \mathtt{h} \multimap \mathtt{A} : \Upsilon \wedge \alpha = \mathtt{real} \wedge \Delta(\mathtt{h}) = \mathtt{asset}}$$

constrain α and $\Delta(\mathtt{h})$

[T-UPDATE]

$$\frac{\Gamma, \Delta \vdash E : \alpha, \Upsilon \quad \alpha \neq \mathtt{asset}}{\Gamma, \Delta \vdash E \to \mathtt{x} : \Upsilon \wedge (\Gamma(\mathtt{x}) = \alpha)}$$

the types of $\mathtt{x}$ and α must be the same

TYPE INFERENCE/RULES

other rules:

[T-COND]

$$\frac{\Gamma, \Delta \vdash E : \alpha, \Upsilon \quad \Gamma, \Delta \vdash S : \Upsilon' \quad \Gamma, \Delta \vdash S' : \Upsilon'' \quad \Upsilon''' = (\alpha = \texttt{bool}) \wedge \Upsilon \wedge \Upsilon' \wedge \Upsilon''}{\Gamma, \Delta \vdash \texttt{if}\,(E)\,\{\,S\,\}\,\texttt{else}\,\{\,S'\,\} : \Upsilon'''}$$

[T-EVENT]

$$\frac{\Gamma, \Delta \vdash E : \alpha, \Upsilon \qquad \Gamma, \Delta \vdash S : \Upsilon' \qquad \Gamma, \Delta \vdash W : \Upsilon'' \qquad \Upsilon'' = (\alpha = \texttt{time}) \wedge \Upsilon \wedge \Upsilon' \wedge \Upsilon''}{\Gamma, \Delta \vdash E \gg \texttt{@Q}\,\{\,S\,\} \Rightarrow \texttt{@Q}'\ W : \Upsilon'''}$$

Theorem (Subject reduction) *Let* $\Gamma, \Delta \vdash \mathbb{C}, \mathbb{t}$ *and* $\mathbb{C}, \mathbb{t} \xrightarrow{\mu} \mathbb{C}', \mathbb{t}'$. *Then* $\Gamma, \Delta \vdash \mathbb{C}'\ ,\ \mathbb{t}'$.

Corollary (Safety) *Let* $\texttt{stipula C}\,\{\,\texttt{assets}\ \overline{\texttt{h}}\ \ \texttt{fields}\ \overline{\texttt{x}}\ \texttt{G}\ \texttt{F}_1 \cdots \texttt{F}_n\,\}$ *be typed and* $\texttt{C}(\_\,,\,\varnothing\,,\,\_\,,\,\_)\ ,\ \mathbb{t} \xRightarrow{\overline{\mu}} \texttt{C}(\texttt{Q}\,,\,\ell\,,\,\Sigma\,,\,\Psi)\ ,\ \mathbb{t}'$ *with labels* $\mu \in \overline{\mu}$ *being type-correct. Then for every* $\texttt{h} \in dom(\ell)$, $\ell(\texttt{h}) \geqslant 0$.

LEFT OVER: DISPUTES

use a **JUDICIAL ENFORCEMENT PATTERN**

- extend parties with an `Authority`
- add functions that manage litigations

```
stipula Deposit {
    parties Client, Farm, Authority
    fields cost, deadline
    assets deposit

    agreement(Farm, Client){
            Farm, Client: cost, deadline
            now + deadline ≫ @Run { deposit ⊸ Farm } ⇛ @End
        } ⇛ @Run

    @Run Farm: send()[h]{
            h ⇒ Client     h ⊸ deposit
        } ⇛ @Run

    @Run Client: buy()[w] (w ≦ deposit×cost){
            (w/(deposit×cost))×deposit ⊸ Client     w ⊸ Farm
        } ⇛ @Run
}
    @Run Authority: stop(x){
            x → ~     0.5×deposit ⊸ Client     deposit ⊸ Farm
        } ⇒ @End
}
```

DISPUTES/COMMENTS

anyone can invoke the authority at any time by moving to an ad-hoc state, *e.g.* `@Dispute`

- the `Authority` communicate the decision, *e.g.* by sending a string `x` and splitting the `deposit` between the litigants
- a **controlled amount of intermediation**

ADD-ONS

additional papers on

- **liquidity property**

  `http://cs.unibo.it/~laneve/papers/Stipula_LiquidityFULL.pdf`

- **amendments**

  `http://cs.unibo.it/~laneve/papers/HO_Stipula_llncs.pdf`

the prototype at `https://github.com/stipula-language` contains

- *Stipula* and its extension with **amendments**
- a `GUI` to edit *Stipula* programs
- the type inference system

CONCLUSIONS

the assimilation of legal contracts and software contracts raises both legal and technological issues

the coauthors of ***Stipula*** are from the Departments of Legal Studies and Computer Science

AN INTERDISCIPLINARY ASSESSMENTS IS NECESSARY

- to understand the usability of software contracts
- to pinpoint partial or erroneous interpretations of the law embedded in technical artefacts
- to study the actual extent of the legal protection provided by the *software normativity*
- to identify an efficient and robust implementation of (legal) primitives
- to provide a legally correct management of identities, agreements, time in obligations and assets

CONCLUSIONS

THE LESSON THAT WE LEARNED

- the nature of legal contracts is **intrinsically open** and an automatic execution must also **consider interactions with the external context**
- the law admits **escape mechanisms** (interventions of *authorities*, *force-majeure*, *hardships*) that the software cannot disregard — in particular ad-hoc primitives/patterns must be identified
- a legal binding between what is true on-line and off-line is always necessary

A FORMAL MATHEMATICAL APPROACH ALLOWS US TO UNDERSTAND THE SUBTLETIES AND TO REASON CORRECTLY

CONCLUSIONS

Stipula is being tested by coding real legal contracts

- we are currently studying contradictory contracts that are encoded in ***Stipula***
- the aim is to design algorithms that spot contradictions at static time

THANK YOU