Symboleo: a language for specifying legal contractual requirements and generating smart contracts

Daniel Amyot
WER 2022, Workshop on Requirements Engineering
Brazil (Online), August 23, 2022
damyot@uottawa.ca
On Laws and Contracts

• Laws and contracts provide requirements on how we ought to behave

• Laws are imposed by governments

• Contracts result from the mutual agreement of involved parties
  • Much more frequent and dynamic than laws
  • Smaller scope
  • Emphasis on time and deadlines
Contract Specification and Monitoring Lab @ uOttawa

Collaborators
- HEC Paris
  - David R. Amariles
  - Pablo M. Baquero
  - Rajaa El Hamdani
- IEMS Inc.
  - Ashkan R. Kian

New students
- Amal Anda (PostDoc)
- Sofana Alfuaid (PhD)
- Daniel Dias-Souza (MSc)
- Regan Meloche (MSc)

Alumni
- Sepehr Sharifi (MSc)
- Ali Roudak (BSc)
- Mustafa Bayirli (Co-Op)
- Sukhraj Bhogal (Co-Op)
- Jeremy Fang (Co-Op)
- Tristan Paul (Co-Op)
- Andrew Rizk (Co-Op)

https://sites.google.com/uottawa.ca/csmlab
Takeaways

- Interested in engineering **smart contracts as cyberphysical systems** that monitor, automate, and control the execution of legal contracts

- **Symboleo** is a formal specification language for contracts with:
  - A conceptual ontology
  - Event-based state machines and axiomatic semantics exploiting event calculus
  - Support for time and for sub-contracting and other dynamic modifications

- Verification of safety/liveness properties via a transformation to **nuXmv**
  - Useful for checking collections of contracts against properties, at **design time**

- Generation of smart contracts for **Hyperledger Fabric** (in JavaScript)
  - For the monitoring and partial automation of legal contracts, at **runtime**
On Legal and Smart Contracts

• *Legal contracts* specify the terms and conditions that apply to business transactions
  • Commonly expressed in natural language, often with ambiguous or incomplete text
  • Difficult/laborious to analyze, both when *designed* and when *executed/performed*
  • Problems compounded by the presence of many documents, e.g., subcontracts

• *Smart contracts* are programs intended to partially *automate* and *control* some aspects of the execution of legal contracts, and also *monitor* them for compliance with relevant terms and conditions
  • As such, they play the role of a person whose role is to ensure that a contract has been executed successfully and in compliance with its terms and conditions.
  • Do not necessarily imply blockchain-dependent “smart contracts” (name hijacked!)
Example of Contract Templates

Meat Purchase and Sale Agreement

Between Seller and Buyer

This agreement is entered into as of the date <effDate>, between <party1> as Seller with the address <retAdd>, and <party2> as Buyer with the address <delAdd>.

Terms and Conditions

1. Payment & Delivery
   1.1 Seller shall sell an amount of <qnt> meat with <qlt> quality (“goods”) to the Buyer.
   1.2 Title in the Goods shall not pass on to the Buyer until payment of the amount owed has been made in full.
   1.3 The Seller shall deliver the Order in one delivery within <delDueDateDays> days to the Buyer at its warehouse.
   1.4 The Buyer shall pay <amt> (“amount”) in <curr> (“currency”) to the Seller before <payDueDate>.
   1.5 In the event of late payment of the amount owed due, the Buyer shall pay interests equal to <intRate>% of the amount owed, and the Seller may suspend performance of all of its obligations under the agreement until payment of amounts due has been received in full.

2. Assignment
   2.1 The rights and obligations are not assignable by Buyer.

3. Termination
   3.1 Any delay in delivery of the goods will not entitle the Buyer to terminate the Contract unless such delay exceeds 10 Days.

4. Confidentiality
   4.1 Both Seller and Buyer must keep the contents of this contract confidential during the execution of the contract and six months after the termination of the contract.
Example of Contract Templates

- Transportation subcontract, with three instances:
  - Córdoba to Buenos Aires, by train
  - Buenos Aires port to Halifax port, by ship
  - Halifax port to Ottawa, by van

Agreement is entered into effect between <party1> as Shipper, and <party2> as Carrier.

O1 The Carrier agrees to transport the goods as stated in tender sheet (<qnt> of <qity> quality meat, in proper refrigerated conditions, from <pkAdd>, to <delAdd> on <delDueDate>).

O2 The Shipper should pay <amt> (“amount”) in <curr> (“currency”) to the Carrier for its services within 3 days after delivery of goods.

O3 The Shipper is additionally subjected to <intRate>% interest rate on the amount due if payment is breached.

- Penalties if delays, if meat not frozen all the time...
Example of a Smart Contract (Solidity)

```solidity
pragma solidity >=0.4.24 <0.7.0;

contract ReceiverPays {
    address owner = msg.sender;

    mapping(uint256 => bool) usedNonces;

    constructor() public payable {} 

    function claimPayment(uint256 amount, uint256 nonce, bytes memory signature) public {
        require((!usedNonces[nonce]));
        usedNonces[nonce] = true;

        // this recreates the message that was signed on the client
        bytes32 message = keccak256(abi.encodePacked(msg.sender, amount, nonce, this));

        require(recoverSigner(message, signature) == owner);

        msg.sender.transfer(amount);
    }

    /// destroy the contract and reclaim the leftover funds.
    function shutdown() public {
        require(msg.sender == owner);
        selfdestruct(msg.sender);
    }

    /// signature methods.
    function splitSignature(bytes memory sig) internal pure
```
Four challenges:
I. Specification
II. Analysis/Verification
III. NL Contracts → Specs
IV. Specs → Smart Contracts
Challenge I: Formal Specifications of Legal Contracts

• For each smart contract, we need a formal specification: an unambiguous, complete and consistent account of the legal contract’s terms and conditions; what the contract is supposed to achieve without the how

• The core problem is the development of a formal specification language for legal contracts

• This language needs to satisfy three key requirements:
  a) Be expressive enough to formalize legal contracts from different domains;
  b) Lead to specifications that can be analyzed formally to ensure they are consistent with contracting parties’ intentions
  c) Enable code generation for monitoring contract execution compliance
Our proposed language:

Symboleo

From the Greek word Συμβόλαιο, meaning contract and pronounced ‘simvoleo’
Meat Purchase Contract Template in Symboleo

Between Seller and Buyer
This agreement is entered into as of the date \(<effDate>\), between \(<party1>\) as Seller with the address \(<relAddr>\), and \(<party2>\) as Buyer with the address \(<delAddr>\).

Terms and Conditions

1. Payment & Delivery
   1.1 Seller shall sell an amount of \(<qnt>\) meat with \(<qlt>\) quality (“goods”) to the Buyer.
   1.2 Title in the Goods shall not pass on to the Buyer until payment of the amount owed has been made in full.
   1.3 The Seller shall deliver the Order in one delivery within \(<delDueDateDays>\) days to the Buyer at its warehouse.
   1.4 The Buyer shall pay \(<amt>\) (“amount”) in \(<curr>\) (“currency”) to the Seller before \(<payDueDate>\).
   1.5 In the event of late payment of the amount owed due, the Buyer shall pay interest equal to \(<intRate>\%\) of the amount owed, and the Seller may suspend performance of all of its obligations under the agreement until payment of amounts due has been received in full.

2. Assignment
   2.1 The rights and obligations are not assignable by Buyer.

3. Termination
   3.1 Any delay in delivery of the goods will not entitle the Buyer to terminate the Contract unless such delay exceeds 10 Days.

4. Confidentiality
   4.1 Both Seller and Buyer must keep the contents of this contract confidential during the execution of the contract and six months after the termination of the contract.
Symboleo Ontology
State/Event View of Contracts

- Each instance of a contract, obligation, or power has a **state**

- **Events** enable transitioning between states. They come from:
  - Monitored execution, or
  - Other instances

- Instances can be suspended and can terminate successfully or not
Formal Axiomatic Semantics (38+ Axioms)

Do not read all of this! Bottom line:

• Formal underpinning in Event Calculus, with support for temporal aspects

• Enables many types of formal and automated analyses, including compliance monitoring

• Enables the handling of many advanced dynamic aspects (subcontracting, party substitution, assignment of rights), taking into account the specifics of a given jurisdiction

Table 2: Primitives

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e \text{ within } s$</td>
<td>situation $(s, T)$ appears or disappears</td>
</tr>
<tr>
<td>$\text{occurs}(s, T)$</td>
<td>situation $(s, T)$ occurs or stops occurring</td>
</tr>
<tr>
<td>$\text{initiates}(e, s)$</td>
<td>event $e$ begins at $(s, T)$</td>
</tr>
<tr>
<td>$\text{terminates}(e, s)$</td>
<td>event $e$ ceases or disappears at $(s, T)$</td>
</tr>
<tr>
<td>$\text{happens}(e, t)$</td>
<td>event $e$ happens at $(s, T)$</td>
</tr>
<tr>
<td>$\text{holdsAt}(s, t)$</td>
<td>situation $(s, T)$ holds at $(s, T)$</td>
</tr>
</tbody>
</table>

Axiom 1 (Create a conditional obligation): for all conditions $c$, if $o$ is triggered then a new obligation $o$ is created. Assumption: $o.\text{ant} = o$.

$$\text{Effect}(e) \land \neg(o.\text{ant} = \text{true})$$ (1)

Objects $(o), \text{create}(o)$

Obligation by a power: for any contract $c$, if the consequent of $c$ is true then $c$ is valid, and if $c$ is invalid then $p$ is exerted while $p$ is in the process of being exerced.

$$\text{within active}(o) \land \text{within InEffect}(c) \land \text{Effect}(e) \land \neg\text{happens}(\text{terminated}(o), \_).$$ (2)

Obligation by contract suspension: for all $c$, if $c$ is suspended while $o$ is valid.

$$\text{suspended}(e(c)) \land \neg\text{happens}(e, \_).$$ (3)

$$\text{initiates}(e, \text{Suspension}(o)) \land \text{terminates}(e, \text{InEffect}(o)).$$ (4)
Jurisdiction-Specific Axioms

• Take advantage of primitive execution-time operators for transferring or sharing **liable**, **rightHolder**, and **performer** relations

• Like macros!

**Party Substitution**

**Signature:** \( \text{substituteC}(c, r, p_{old}, p_{new}) \)

**Semantics:** A contractual party might decide to leave a contract execution and have a third-party replace her in the contract. A party \( p_{old} \) who has a role \( r \) in contract \( c \) can substitute herself with another party \( p_{new} \) and transfer all of the rights, responsibilities, and performance of all the active obligations/powers \( x \) to \( p_{new} \), given the consent of all original parties and of \( p_{new} \).

**Axiom 5.12:** Given the consent of \( p_{old}, p_{new}, \) and other parties of the contract \( c \) to \( \text{substituteC}(c, r, p_{old}, p_{new}) \), and given contract \( c \), obligation/power \( x \), and role \( r \), and the fact that \( \text{substitutedC}(c, r, p_{old}, p_{new}) \) is the event that occurs and initiates the substitution, then there exists a time \( t \) for which this holds:

\[
\forall x \in c.\text{legalPosition} : \text{happens}(\text{consented}(\text{substitutedC}(c, r, p_{old}, p_{new})), t) \\
\land \text{happens}(\text{substitutedC}(c, r, p_{old}, p_{new}), t) \\
\land \text{holdsAt}(\text{active}(c), t) \land \text{holdsAt}(\text{bind}(c,r,p_{old}), t) \rightarrow \\
\text{initiates}(\text{substitutedC}(c, r, p_{old}, p_{new}), \text{bind}(c,r,p_{new})) \\
\land \text{terminates}(\text{substitutedC}(c, r, p_{old}, p_{new}), \text{bind}(c,r,p_{old})) \\
\land \text{happens}(\text{transferredR}(c.x, p_{old}, p_{new}), t) \\
\land \text{happens}(\text{transferredL}(c.x, p_{old}, p_{new}), t) \\
\land \text{happens}(\text{transferredP}(c.x, p_{old}, p_{new}), t)
\]
Symboleo’s Syntax by Example (1/3)

• An Eclipse-based editor is available
  • https://github.com/Smart-Contract-Modelling-uOttawa/Symboleo-IDE

• First, the domain (vocabulary, data structures) is described, as an extension of the Symboleo ontology

```plaintext
Domain meatSaleD
    Seller isA Role with returnAddress: String;
    Buyer isA Role with warehouse: String;
    Currency isA Enumeration(‘CAD’, ‘USD’, ‘EUR’);
    PerishableGood isA Asset with quantity: Number, quality: MeatQuality;
    Meat isA PerishableGood;
    Delivered isA Event with item: Meat, deliveryAddress: String, delDueD: Date;
    Paid isA Event with amount: Number, currency: Currency, from: Role, to: Role, payDueD: Date;
    PaidLate isA Event with amount: Number, currency: Currency, from: Role, to: Role;
    Disclosed isA Event with contractID : String;
    // Axiom : happensBefore(Paid, Paid.payDueD) → isOwner(Meat, Buyer)
endDomain
```
Symboleo’s Syntax by Example (2/3)

• The contract’s signature defines the template parameters
  • Specific values are passed when instantiating the contract
• Local declarations as well as pre/post conditions can be defined

```
Contract meatSale(buyer: Buyer, seller: Seller, qty: Number, qlt: MeatQuality, amt: Number, curr: Currency, payDueDate: Date, delAdd: String, effDate: Date, delDueDateDays: Number, intRate: Number)

Declarations
    goods : Meat with quantity := qty, quality := qlt;
    delivered : Delivered with item := goods, deliveryAddress := delAdd, delDueD := effDate + delDueDateDays;
    paid : Paid with amount := amt, currency := curr, from := buyer, to := seller, payDueD := payDueDate;
    paidLate : PaidLate with amount := (1 + intRate/100) * amt, currency := curr, from := buyer, to := seller;
    disclosed : Disclosed with contract := self;

Preconditions
    occurs(isEqual(goods.ownership, seller), [_, self.start])

Postconditions
    occurs(isEqual(goods.ownership, buyer), [_, self.end]) and not occurs(isEqual(goods.ownership, seller), [self.end, _])
```
Symboleo’s Syntax by Example (3/3)

• **Obligations and Powers** are then defined
  
  • [trigger -> ] \( O(\text{debtor}, \text{creditor}, \text{antecedent}, \text{consequent}) \)
  
  • [trigger -> ] \( P(\text{creditor}, \text{debtor}, \text{antecedent}, \text{consequent}) \)

\[
\begin{align*}
\text{Obligations} \\
O_{\text{del}} : O(\text{seller, buyer, true, happensBefore}(\text{delivered, delivered.delDueD})); \\
O_{\text{pay}} : O(\text{buyer, seller, true, happensBefore}(\text{paid, paid.payDueD})); \\
O_{\text{pay}} : \text{violates}(O_{\text{pay}}.\text{instance}) \rightarrow O(\text{buyer, seller, true, happens}(\text{paidLate, _}));
\end{align*}
\]

\[
\begin{align*}
\text{SurvivingObls} \\
SO_{\text{sellDisclosure}} : O(\text{seller, buyer, true, not happens}(\text{disclosed(self), t}) \text{ and (t within activates(self) + 6 months)}); \\
SO_{\text{buyDisclosure}} : O(\text{buyer, seller, true, not happens}(\text{disclosed(self), t}) \text{ and (t within activates(self) + 6 months)});
\end{align*}
\]

\[
\begin{align*}
\text{Powers} \\
P_{\text{suspendDelivery}} : \text{violates}(O_{\text{pay}}.\text{instance}) \rightarrow \\
P(\text{seller, buyer, true, suspends}(O_{\text{del}}.\text{instance})); \\
P_{\text{resuDelivery}} : \text{happensWithin}(\text{paidLate, suspension}(O_{\text{del}}.\text{instance})) \rightarrow \\
P(\text{buyer, seller, true, resumes}(O_{\text{del}}.\text{instance})); \\
P_{\text{termContract}} : \text{not}(\text{happensBefore}(\text{delivered, delivered.delDueDate+10 days})) \rightarrow \\
P(\text{buyer, seller, true, terminates(self)});
\end{align*}
\]
Xtext-based IDE for Eclipse

- Syntax highlight
- Syntax checking
- Type checking
- Code completion
- Other well-formedness rules
- ...

```
grammar ca.uottawa.csmlab.symboleo.Symboleo with org.eclipse.xtext.common.Terminals

generate symboleo "http://www.uottawa.ca/cslab/symboleo/Symboleo"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Model:
  'Domain' (domainName=ID)
    (domainTypes=DomainType ':')+
  endDomain
  'Contract' contractName=ID '{' (parameters=Parameter ':')+
    (parameters=Parameter ':')'}
  '{' (declarations=Declaration ':')+
    (declarations=Declaration ':')'}
  '{' (preconditions=Precondition ':')+
    (preconditions=Precondition ':')'}
  '{' (postconditions=postcondition ':')+
    (postconditions=postcondition ':')'}
  '{' (obligations=Obligation ':')+
    (obligations=Obligation ':')'}
  '{' (survivingObligations=SurvivingObligation ':')+
    (survivingObligations=SurvivingObligation ':')'}
  '{' (pows=powers ':')+
    (pows=powers ':')'}
  '{' (constraints=Constraint ':')+
    (constraints=Constraint ':')'}
  endContract;

DomainType:
  Alias | RegularType | Enumeration;

Alias:
  name=ID 'isA' typeBaseType;

Enumeration:
  name=ID 'isAn' 'Enumeration' '{' (enumerationItems=EnumItem ':')+
    (enumerationItems=EnumItem ':')'};

EnumItem:
  name=ID;

// TODO not extend itself, prevent cycles

RegularType:
  name=ID ('isA' | 'isAn') ontologyType=OntologyType '{' (attributes=Attribute ':')+
    (attributes=Attribute ':')'}
  name=ID ('isA' | 'isAn') regularTypes=[RegularType] '{' (attributes=Attribute ':')+
    (attributes=Attribute ':')'};

Attribute:
  attributeModifier=AttributeModifier? name=ID ':' baseType=BaseType |
    attributeModifier=AttributeModifier? name=ID ':' domainType=[DomainType];

BaseType:
  name=('Number' | 'String' | 'Date' | 'Boolean');

OntologyType:
  name=('Asset' | 'Event' | 'Role' | 'Contract');
```
Declarations

seller: Seller with name := "Seller";
buyer : Buyer with name := 2 + 3;
Challenge II: Analysis of Specifications
Verification with Model Checking

- **Model checking** is an algorithmic procedure that enables the verification of (state-based) formal specifications against **properties**, usually expressed in **temporal logic**
  - Linear Temporal Logic (LTL)
  - Compositional Tree Logic (CTL)

- **Liveness** properties: “good things will eventually happen”
  - The meat sale contract eventually terminates.

- **Safety** properties: “something bad will not happen”
  - In case of late payment, the buyer cannot be penalized more than once.
  - Normal payment and late payment obligations cannot be active together.
From Symboleo to nuXmv

• The nuXmv model checker tool is the evolution of the NuSMV tool

Extending nuXmv with Timed Transition Systems and Timed Temporal Properties
A Cimatti, A Griggio, E Magnago, M Roveri, S Tonetta
International Conference on Computer Aided Verification, 376-386

• nuXmv has its own textual input format for specifications
• nuXmv supports LTL and CTL properties, provides some time-related constructs, and generates counter-examples in case of violations
• Symboleo’s semantics and axioms are state-based, and hence a good match for a mapping to nuXmv!
Overview of the SymboleoPC Property Checker
Primitive Axioms for the *Obligation* Concept in nuXmv

- Similar trusted modules exist for the other language concepts
- These trusted library modules are for the language itself, and are reusable for all Symboleo specifications
Contract-Specific Obligations/Powers

\begin{verbatim}
VAR
delivered : Event(Odel.state=inEffect & !(suspended_delivery._happened & !resumed_delivery._happened), del_due_days);
paid : Event(Opay.state=inEffect, pay_due_days);
paidLate : Event(Opay.state=violation, pay_late_due_days);
suspended_delivery : Event(PsusDel.state = inEffect, sus_del_due_days);
resumed_delivery : Event(PresumDel.state = inEffect, resume_del_due_days);
terminated_cnt : Event(PtermCnt.state = inEffect, term_cnt_due_days);

contr : Contract(TRUE, TRUE, PtermCnt_exerted, FALSE, FALSE, FALSE, FALSE,
Gsuc_terminated);
Odel : Obligation(FALSE, contr._obls_activated, PtermCnt_exerted,
(delivered._happened & delivered.performer = Odel.debtor._name &
Odel.debtor.is_performer), TRUE, Odel.violated, FALSE, FALSE,
PsusDel_exerted, FALSE, FALSE, PresumDel_exerted, FALSE, FALSE, TRUE);
Opay : Obligation(FALSE, contr._obls_activated, PtermCnt_exerted,
Opay_fulfilled, TRUE, Opay_violated, FALSE, FALSE, FALSE, FALSE,
FALSE, FALSE, FALSE, FALSE, TRUE);
OlatePay : Obligation(FALSE, contr._obls_activated, PtermCnt_exerted,
OlatePay_fulfilled, Opay_violated, paidLate.expired, FALSE, FALSE,
FALSE, FALSE, FALSE, FALSE, FALSE, TRUE);
PsusDel : Power(contr._obls_activated, Opay_violated, FALSE, FALSE, FALSE,
FALSE, FALSE, PsusDel_exerted, FALSE, FALSE, TRUE);
PresumDel: Power(contr._obls_activated, OlatePay_fulfilled, FALSE, FALSE,
FALSE, FALSE, PresumDel_exerted, FALSE, FALSE, TRUE);
PtermCnt : Power(contr._obls_activated, Odel_violated, FALSE, FALSE, FALSE,
FALSE, FALSE, PtermCnt_exerted, FALSE, FALSE, TRUE);
\end{verbatim}
Example of Successful CTL Property

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>desirable-safety</td>
<td>occurrence</td>
</tr>
</tbody>
</table>

**Description**

MeatSale is free of useless obligations or powers: all obligations and powers can be activated.

**Property**

```plaintext
CTLSPEC NAME CTL4_1 := EF(sales_cnt.PsusDel._active)
CTLSPEC NAME CTL4_2 := EF(sales_cnt.PresumDel._active)
CTLSPEC NAME CTL4_3 := EF(sales_cnt.PterCnt._active)
CTLSPEC NAME CTL4_4 := EF(sales_cnt.Odel._active)
CTLSPEC NAME CTL4_5 := EF(sales_cnt.Opay._active)
CTLSPEC NAME CTL4_6 := EF(sales_cnt.OlatePay._active)
```

Result: succeed
Example of Failing LTL Property

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>desirable-safety</td>
<td>precedence</td>
</tr>
</tbody>
</table>

**Description**
Delivery of goods always happens after payment.

**Property**

\[
\text{LTLSPEC NAME LTL3 := !}(\text{sales_cnt.delivered._happened \& sales_cnt.delivered.performer = sales_cnt.Odel_debtor._name \& sales_cnt.Odel_debtor._is_performer}) \ U (\text{sales_cnt.paid._happened \& sales_cnt.paid.performer = sales_cnt.Opay_debtor._name \& sales_cnt.Odel_debtor._is_performer})
\]

**Result:** failed

Explanation: The delivery obligation is independent of the payment obligation.
Performance and Scalability

Symboleo Performance Evaluation Infrastructure

Test Params (e.g. # power, depth # LTL, ...)

Random Test Generator

nuXmv encoding of Symbole Contracts

nuXmv Model Checker

Verification result

Statistics Extraction

Library of nuXmv Modules for Symbole Concepts

LTL and CTL properties

Obligation Number

Time To Compute Reachable States (seconds)

power#1, power#2, power#4, power#8, power#16

University of Ottawa
# Comparison with Formal Contractual Languages

<table>
<thead>
<tr>
<th>Time support</th>
<th>Legal concepts</th>
<th>Events/Values</th>
<th>Imperative/Declarative</th>
<th>Contract reparations</th>
<th>Contract parameterization</th>
<th>Compliance monitoring</th>
<th>Subcontracting</th>
<th>Executable analysis</th>
<th>Automated verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
<td>C9</td>
<td>C10</td>
</tr>
<tr>
<td>t</td>
<td>Deon</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>UFO-L</td>
<td>E, V</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>Co</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>Deon</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>Co</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>Deon</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>H</td>
<td>E</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>Deon</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>N</td>
<td>Deon</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>O, Pe</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B</td>
<td>Deon</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>t</td>
<td>H</td>
<td>E</td>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

| Symboleo | UFO-L | E, V | D | ✓ | ✓ | ✓ | ✓ | ✓ |

## Summary

- **Time support**: Various models support different forms of time awareness.
- **Legal concepts**: Different models support a range of legal concepts, including declarative and imperative approaches.
- **Events/Values**: Models vary in their support for events and value definitions.
- **Imperative/Declarative**: Models differ in their support for imperative and declarative contract formulations.
- **Contract reparations**: Models vary in their support for contract reparations.
- **Contract parameterization**: Models differ in their support for contract parameterization.
- **Compliance monitoring**: Models vary in their support for compliance monitoring.
- **Subcontracting**: Models vary in their support for subcontracting.
- **Executable analysis**: Models differ in their support for executable analysis.
- **Automated verification**: Models vary in their support for automated verification.
<table>
<thead>
<tr>
<th>Paper</th>
<th>Goal</th>
<th>Technique</th>
<th>Tools</th>
<th>L</th>
<th>Language</th>
<th>SLC</th>
<th>LP</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>[73]</td>
<td>V</td>
<td>Model Checking</td>
<td>NuXMV</td>
<td>S</td>
<td>CL</td>
<td>✓</td>
<td>×</td>
<td>Conformance checking</td>
</tr>
<tr>
<td>[9]</td>
<td>V</td>
<td>Model Checking</td>
<td>Spin</td>
<td>S</td>
<td>State Machine</td>
<td>✓</td>
<td>×</td>
<td>Reachability</td>
</tr>
<tr>
<td>[13]</td>
<td>V</td>
<td>Automation Search</td>
<td>RECALL</td>
<td>S</td>
<td>RCL</td>
<td>✓</td>
<td>✓</td>
<td>Conflict</td>
</tr>
<tr>
<td>[30]</td>
<td>V</td>
<td>Model Checking</td>
<td>CLAN</td>
<td>S</td>
<td>CL</td>
<td>✓</td>
<td>✓</td>
<td>Conflict/Reachability</td>
</tr>
<tr>
<td>[8]</td>
<td>V</td>
<td>Automation Search</td>
<td>Unknown</td>
<td>S</td>
<td>State Machine</td>
<td>✓</td>
<td>✓</td>
<td>Conflict</td>
</tr>
<tr>
<td>[63]</td>
<td>V</td>
<td>Model Checking</td>
<td>NuXMV</td>
<td>S</td>
<td>State Machine</td>
<td>✓</td>
<td>×</td>
<td>Reachability</td>
</tr>
<tr>
<td>[6, 32]</td>
<td>V</td>
<td>Model Checking</td>
<td>Corral/ETHBMC</td>
<td>P</td>
<td>Solidity</td>
<td>×</td>
<td>×</td>
<td>Compliance with functional requirements</td>
</tr>
<tr>
<td>[45, 72, 88]</td>
<td>V</td>
<td>Model Checking</td>
<td>SMT-solver/NuXMV</td>
<td>P</td>
<td>Solidity</td>
<td>×</td>
<td>×</td>
<td>Compliance with functional requirements</td>
</tr>
<tr>
<td>[3]</td>
<td>V</td>
<td>Model Checking</td>
<td>NuXMV</td>
<td>P</td>
<td>State Machine</td>
<td>×</td>
<td>×</td>
<td>Compliance with functional requirements</td>
</tr>
<tr>
<td>[62]</td>
<td>V</td>
<td>Model Checking/Colored Petri Net</td>
<td>ASK-CTL/State Space</td>
<td>P</td>
<td>Colored Petri Net</td>
<td>×</td>
<td>×</td>
<td>Compliance with functional requirements</td>
</tr>
<tr>
<td>[18, 19]</td>
<td>CC</td>
<td>Monitoring</td>
<td>REFC/SCIFF</td>
<td>S</td>
<td>Event calculus</td>
<td>✓</td>
<td>×</td>
<td>Conformance checking</td>
</tr>
<tr>
<td>Thesis</td>
<td>V/CC</td>
<td>Model Checking/Monitoring</td>
<td>SYMBOLEOPC &amp; NuXMV/ SYMBOLEOCC</td>
<td>S</td>
<td>Symboloo</td>
<td>✓</td>
<td>✓</td>
<td>All four approaches</td>
</tr>
</tbody>
</table>

**Goal:** Verification (V) or conformance checking (CC)  
**L:** Specification level (S) or programming level (P)  
**SLC:** Support legal contracts  
**LP:** Legal positions
Challenge III: From NL to specifications
Challenge III: From NL to specifications

• The process of generating a formal specification for a given legal contract expressed in Natural Language (NL) is both time-consuming and error-prone.

• Accordingly, a third major objective of the project is to develop tools that support the translation from NL to Symboleo specifications.

• We expect that such tools won’t be able to fully automate the translation process because legal contract text is often ambiguous and incomplete, and resolving such issues always requires human intervention. Rather, we aim for tools that (a) Reduce manual effort, (b) Improve overall quality of specifications
  • Ontology population, combination and instantiation of preexisting clauses, etc.

• For this challenge, we propose to investigate different combinations of NLP and Machine Learning techniques that have been successful in similar contexts.
Extraction of Symboleo Concepts from NL Contracts
Challenge IV: From specifications to code
Challenge IV: From Specifications to SC Code

• Smart contracts include in their architecture both sensors for sensing the happening of events, e.g., the temperature in a van carrying meat from a supplier to a buyer has risen above -15C, as well as a blockchain platform for storing contracts and execution data, e.g., on delivery, payment...

• The fourth challenge of the project concerns generating code from a specification and other inputs, such as the physical network of sensors, what data are to be stored on a blockchain and what on a conventional data store to ensure reasonable performance, what privacy protection mechanisms are to be used, etc.

• First attempt: automatic transformation to Hyperledger Fabric in JavaScript
  • https://github.com/Smart-Contract-Modelling-uOttawa/Symboleo-JS-Core
Code Generation with the *Symboleo2SC* Tool

- **SYMBOLEO grammar in Xtext**
- **SYMBOLEO ontology (EMF)**
- **SYMBOLEO ontology (Umple)**
- IDE generation
- **SYMBOLEO specification in IDE (Eclipse)**
- **Generation**
- **SYMBOLEO2SC code generator (Xtend)**
- **Generation**
- **Smart contract (JS)**
- **Generation**
- **SYMBOLEOJS ontology and state machine library (JS)**
- **Usage**
- **Legal contract**
- **Manual conversion to SYMBOLEO**
Ontology and state machines in **Umple**

- **Ontology (Java)**
- **Ontology (JavaScript)**
- **Ontology (Java/EMF)**
- **Ontology (SMV)**
- **State Machines (Java)**
- **State Machines (JavaScript)**
- **State Machines (SMV)**
- **State Machines (nuXmv)**
Ontology Concepts as a JavaScript Library

```javascript
import { LegalPosition } from "./LegalPosition.js"
import { Event } from "./Event.js"
import { Events } from "./Events.js"
import { InternalEvent, InternalEventSource, InternalEventType } from "./InternalEvents.js"

export class Obligation extends LegalPosition {

    constructor(name, creditor, debtor, contract) {
        super(name, creditor, debtor, contract)
        this.setActiveState(ObligationActiveState.Null)
        this.setState(ObligationState.Create)
        this._events = {}
    }

    isViolated() {
        return this.state === ObligationState.Violation
    }

    isInEffect() {
        return this.state === ObligationState.Active &&
        this.activeState === ObligationActiveState.InEffect
    }

    suspended() {
        let wasEventProcessed = false

        let aStatusActive = this.activeState
        switch (aStatusActive) {
            case ObligationActiveState.InEffect:
                this.exitStatusActive()
                this.setActiveState(ObligationActiveState.Suspension)
                wasEventProcessed = true
                this._events.Suspended = new Event()
                this._events.Suspended.happen()
                Events.emitEvent(this.contract, new
                        InternalEvent(InternalEventSource.obligation,
                        InternalEventType.obligation.Suspended, this))
                break
            default:
                // Other states do respond to this event
        }

        return wasEventProcessed
    }
}
```
def void parse(Model model) {
    parameters.addAll(model.parameters)

    for (domainType : model.domainTypes) {
        if (domainType instanceof RegularType) {
            var RegularType base = getBaseType(domainType)
            if (base != null) {
                switch base.ontologyType.name {
                    case 'Asset': assets.add(domainType as RegularType)
                    case 'Event': events.add(domainType as RegularType)
                    case 'Role': roles.add(domainType as RegularType)
                }
            } else if (domainType instanceof Enumeration) {
                enumerations.add(domainType as Enumeration)
            }
        }
    }

    «FOR obligation : survivingObligationTriggerEvents.keySet»
    createSurvivingObligation(_obligation.name_, contract) {
        if (generatePropositionString(obligation.trigger)) {
            contract.survivingObligations._obligation.name_.addObligation('«obligation.name»,
                «IF »survivingObligationAntecedentEvents.containsKey(obligation)»
            if (generatePropositionString(obligation.antecedent)) {
                contract.survivingObligations._obligation.name_.triggeredUnconditional()
                «IF »survivingObligationFullfimentEvents.containsKey(obligation)»
            if (generatePropositionString(obligation.consequent)) {
                contract.survivingObligations._obligation.name_.fulfilled()
            } else {
                contract.survivingObligations._obligation.name_.violated()
            }
            «ENDIF»
        }
        «ENDIF»
    }
}
const { Asset } = require("\SYMB-js-core");

class PerishableGood extends Asset {
    constructor(_name, quantity, quality) {
        super()
        this._name = _name
        this.quantity = quantity
        this.quality = quality
    }
}

module.exports.PerishableGood = PerishableGood

const { PerishableGood } = require("./PerishableGood.js");

class Meat extends PerishableGood {
    constructor(_name, quantity, quality) {
        super(_name, quantity, quality)
    }
}

module.exports.Meat = Meat
class MeatSale extends SymboleoContract {
    constructor(buyer, seller, qnt, qlt, amt, curr, payDueDate, delAdd, effDate,
                delDueDateDays, interestRate) {
        super("MeatSale")
    
    "this._name = "MeatSale"
    this.buyer = buyer
    this.seller = seller

    
    this.delivered = new Delivered("delivered")
    this.delivered.item = this.goods
    this.delivered.deliveryAddress = this.delAdd
    this.delivered.delDueDate = Utils.addTime(this.effDate, this.delDueDateDays,
                                            "days")

    
    this.obligations.delivery = new Obligation('delivery', this.buyer, this.
                                              seller, this)
    this.obligations.payment = new Obligation('payment', this.seller, this.buyer,
                                             this)
}
}
module.exports.MeatSale = MeatSale
async trigger_delivered(ctx, args) {
  const inputs = JSON.parse(args);
  const contractId = inputs.contractId;
  const event = inputs.event;
  const contractState = await ctx.stub.getState(contractId)
  if (contractState == null) {
    return {successful: false}
  }
  const contract = deserialize(contractState.toString())
  this.initialize(contract)
  if (contract.isInEffect()) {
    contract.delivered.happen(event)
    Events.emitEvent(contract, new InternalEvent(InternalEventSource,
      delivered))
    await ctx.stub.putState(contractId, Buffer.from(serialze(contract)))
    return {successful: true}
  } else {
    return {successful: false}
  }
}
Code Generated for the Contract (Event Listeners)

```javascript
const EventListeners = {
    createPower_suspendDelivery(contract) {
        if (Predicates.happens(contract.obligations.payment._events.Violated)) {
            contract.powers.suspendDelivery = new Power('suspendDelivery', contract.seller, contract.buyer, contract)
            if (true) {
                contract.powers.suspendDelivery.triggeredUnconditional()
            }
        }
    }
}

function getEventMap(contract) {
    return [
        [[new InternalEvent(InternalEventSource.obligation, InternalEventType.obligation.Violated, contract.obligations.payment), ], EventListeners.createPower_suspendDelivery]
    ]
}
```
Code Generation: About 14:1 Ratio (JS:Symboleo)
Deployment Testing

Meat Sale chain code tests

Test init transaction.
- should return error on init.
- should activate contract with the correct state for powers and obligations.

Test transactions for triggering Events.
- should change state of paid.
- should change state of delivered.
- should change state of paidlate.

Scenario: payment and delivery are fulfilled.
- should successfully terminate contract if payment and delivery are fulfilled.

Scenario: payment is violated.
- should violate payment.
- should trigger laterpayment and suspenddelivered if payment is violated.
- should suspend delivery if suspenddelivery is exerted.
- should trigger resumesdelivered and fulfill latepayment if paidlate is triggered.
- should resume delivery if resumesdelivery is exerted.
- should successfully terminate contract if delivered is triggered.
- should unsuccessfully terminate contract if latepayment is violated.

Scenario: delivery is violated.
- should violate delivery.
- should trigger terminateContract if delivery is violated.
- should terminateContract if terminateContract is exerted.

16 passing (53ms)
## Comparison with Related Work

<table>
<thead>
<tr>
<th>Work</th>
<th>Input</th>
<th>Output</th>
<th>Verification</th>
<th>Based on</th>
<th>Textual/Graphical</th>
<th>Technologies</th>
<th>on-chain Enforcement</th>
<th>Use case</th>
<th>IDE</th>
<th>Complete output</th>
<th>Expressiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>[20]</td>
<td>UML sequence diagrams</td>
<td>Hyperledger Fabric Go</td>
<td>No</td>
<td>MDA Platform Specific Model (PSM)</td>
<td>G</td>
<td>Eclipse Acceleo, Obee Designer, UML</td>
<td>Yes</td>
<td>General purpose smart contracts</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[17]</td>
<td>iContractML</td>
<td>Azure blockchain, Hyperledger Composer, Solidity, DAML</td>
<td>No</td>
<td>Abstraction of common concepts</td>
<td>G</td>
<td>Eclipse Acceleo, Obee Designer</td>
<td>Yes</td>
<td>General purpose smart contracts</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>[50]</td>
<td>Contract Modeling Language (DSL)</td>
<td>Solidity</td>
<td>No</td>
<td>Legal ontologies</td>
<td>T</td>
<td>Xtext</td>
<td>Yes</td>
<td>Legal contracts, General purpose smart contracts</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[26]</td>
<td>BPMN</td>
<td>Solidity</td>
<td>No</td>
<td>Process models (BPMN)</td>
<td>G</td>
<td>Caterpillar</td>
<td>No</td>
<td>Collaborative processes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[29]</td>
<td>Transition systems, NL templates</td>
<td>Solidity</td>
<td>Yes</td>
<td>Transition system model</td>
<td>G</td>
<td>WebGME, FSolidM</td>
<td>Yes</td>
<td>General purpose smart contracts</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[27]</td>
<td>custom visual DSL</td>
<td>Solidity</td>
<td>No</td>
<td>Abstraction of common concepts</td>
<td>G</td>
<td>Blockly, NLP, Python, RNN</td>
<td>Yes</td>
<td>General purpose smart contracts</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>[28]</td>
<td>Finite state machine</td>
<td>Solidity</td>
<td>No</td>
<td>Finite state machines</td>
<td>G</td>
<td>WebGME</td>
<td>Yes</td>
<td>General purpose smart contracts</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[8]</td>
<td>Business Collaboration Rules Language</td>
<td>Hyperledger Fabric</td>
<td>No</td>
<td>Rule based</td>
<td>T</td>
<td>JavaScript, nanoRETE</td>
<td>Yes</td>
<td>Collaborative business processes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[62]</td>
<td>TA-SPECS (DSL)</td>
<td>Solidity</td>
<td>No</td>
<td>Abstraction of common concepts</td>
<td>T</td>
<td>SPECS, Xtext</td>
<td>Yes</td>
<td>Asset management</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>[46]</td>
<td>BPMN</td>
<td>Solidity</td>
<td>No</td>
<td>Process models (BPMN)</td>
<td>G</td>
<td>bpmn-js</td>
<td>Yes</td>
<td>Collaborative business processes with asset management</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[42]</td>
<td>DasContract DSL</td>
<td>Solidity</td>
<td>No</td>
<td>Process models (BPMN)</td>
<td>G</td>
<td>C#</td>
<td>Yes</td>
<td>General purpose smart contracts</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>[13]</td>
<td>ADICO (DSL)</td>
<td>Solidity</td>
<td>No</td>
<td>Grammar of Institutions (ADICO)</td>
<td>T</td>
<td>Scala</td>
<td>Yes</td>
<td>Legal contracts</td>
<td>No</td>
<td>Partial</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Observations

• Legal contracts are complex artifacts with respect to the legal concepts they are founded on (obligation, power), and they are typically more complex than what is usually found in business processes.

• These concepts are deontic in nature, i.e., expressed in terms of ‘must’ and ‘can’, and are usually formalized in terms of modal logics which have proven difficult to analyze/reason with.

• In our work, we have opted for a state-transition semantics for legal concepts that fits better the monitoring function of smart contracts and better supports analysis/reasoning.
28th IEEE Requirements Engineering Conference (RE’20)

Symboleo: Towards a Specification Language for Legal Contracts

Sepehr Sharifi, Alireza Parvizimosaeed, Daniel Amyot, Luigi Logippo, John Mylopoulos
School of EECS, University of Ottawa, Ottawa, Canada
{sshar190, aparv007, damyat, logippo, jmylopou}@uottawa.ca

RE’20, Doctoral Symposium

Towards the Specification and Verification of Legal Contracts*

Alireza Parvizimosaeed
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
aparv007@uottawa.ca, https://scholar.google.com/citations?user=m18pXpkAAAAJ

39th International Conference on Conceptual Modelling (ER’20)

Subcontracting, Assignment, and Substitution for Legal Contracts in Symboleo *

Alireza Parvizimosaeed¹, Sepehr Sharifi¹, Daniel Amyot¹, Luigi Logippo¹-², and John Mylopoulos¹

¹ School of EECS, University of Ottawa, Ottawa, Canada
{aparv007, sshar190, damyat, logippo, jmylopou}@uottawa.ca
² Université du Québec en Outaouais, Gatineau, Canada

2020 IEEE PES Transactive Energy Systems Conference (TESC’20)

Compliance Checking for Transactive Energy Contracts using Smart Contracts

Alireza Parvizimosaeed
Masoud Bashiri, Ph.D.
Director of AI & Software
I-EMS Group Ltd.
Masoud.bashiri@iemsgroup.ca

Ashkan R. Kian, Ph.D., P.Eng.,
IEEE Senior Member
CEO, I-EMS Group Ltd.
arkian@iemsgroup.ca

Daniel Amyot, Ph.D., P.Eng., IEEE SM
School of EECS
University of Ottawa
damyat@uottawa.ca

John Mylopoulos, Ph.D., FRSC
School of EECS
University of Ottawa
im@cs.toronto.edu


Smart Contracts: from Formal Specification to Blockchain Code

by

Seyed Sepehr Sharifi
**Model-Checking Legal Contracts with SymboleoPC**

<table>
<thead>
<tr>
<th>Alireza Parvizimosaed</th>
<th>Aidin Rasti</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Ottawa</td>
<td>University of Ottawa</td>
</tr>
<tr>
<td>Ottawa, Canada</td>
<td>Ottawa, Canada</td>
</tr>
<tr>
<td><a href="mailto:aparv007@uottawa.ca">aparv007@uottawa.ca</a></td>
<td><a href="mailto:aidin.rasti@uottawa.ca">aidin.rasti@uottawa.ca</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Daniel Amyot</th>
<th>Luigi Logrippe</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Ottawa</td>
<td>Université du Québec en</td>
</tr>
<tr>
<td>Ottawa, Canada</td>
<td>Outaouais</td>
</tr>
<tr>
<td><a href="mailto:damyot@uottawa.ca">damyot@uottawa.ca</a></td>
<td>Gatineau, Canada</td>
</tr>
<tr>
<td></td>
<td><a href="mailto:luigi.logrippe@uqo.ca">luigi.logrippe@uqo.ca</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Marco Roveri</th>
<th>John Mylopoulos</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Trento</td>
<td>University of Ottawa</td>
</tr>
<tr>
<td>Trento, Italy</td>
<td>Ottawa, Canada</td>
</tr>
<tr>
<td><a href="mailto:marco.roveri@unitn.it">marco.roveri@unitn.it</a></td>
<td><a href="mailto:jmylopou@uottawa.ca">jmylopou@uottawa.ca</a></td>
</tr>
</tbody>
</table>

---

**Symboleo2SC: From Legal Contract Specifications to Smart Contracts**

<table>
<thead>
<tr>
<th>Aidin Rasti</th>
<th>Daniel Amyot</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:aidin.rasti@uottawa.ca">aidin.rasti@uottawa.ca</a></td>
<td><a href="mailto:damyot@uottawa.ca">damyot@uottawa.ca</a></td>
</tr>
<tr>
<td>School of EECS, University of Ottawa, Ontario, Canada</td>
<td>School of EECS, University of Ottawa, Ontario, Canada</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Marco Roveri</th>
<th>Luigi Logrippe</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:marco.roveri@unitn.it">marco.roveri@unitn.it</a></td>
<td><a href="mailto:logrippe@uottawa.ca">logrippe@uottawa.ca</a></td>
</tr>
<tr>
<td>Dept. of Information Engineering and Computer Science, University of Trento, Trento, Italy</td>
<td>Dept. d’info. et ingénierie, UQO, Gatineau, Quebec, Canada</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alireza Parvizimosaed</th>
<th>Amal Ahmed Anda</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:aparv007@uottawa.ca">aparv007@uottawa.ca</a></td>
<td>aamal@<a href="mailto:27@gmail.com">27@gmail.com</a></td>
</tr>
</tbody>
</table>

| John Mylopoulos            |                                  |
|----------------------------|                                  |
| jmylopou@uottawa.ca        |                                  |

---


**Symboleo: Specification and Verification of Legal Contracts**

by

Alireza Parvizimosaed

---

**M.Sc. Thesis, uOttawa (to submitted this week!)**

**From Symboleo to Smart Contracts – A Code Generator**

by

Aedin Rasti

---

**Journal paper under review...**

**Specification and Analysis of Legal Contracts with Symboleo**

Alireza Parvizimosaed · Sepehr Sharifi · Daniel Amyot · Luigi Logrippe · Marco Roveri · Aidin Rasti · Ali Roudak · John Mylopoulos
Conclusions and Future Work

• Interested in engineering **smart contracts as cyberphysical systems** that monitor, automate and control the execution of legal contracts

• **Symboleo** is a formal specification language for contracts with:
  • Conceptual ontology
  • Event-based state machines and axiomatic semantics exploiting event calculus
  • Support for time and for sub-contracting and other dynamic modifications

• Verification of safety/liveness properties via a transformation to **nuXmv**
  • Useful for checking collections of contracts against properties, at design time

• Generation of smart contracts for **Hyperledger Fabric** (in JavaScript)
  • For the monitoring and partial automation of legal contracts, at runtime

• **Upcoming**: NL contracts to Symboleo, quality requirements (security, distribution, robustness,...), during code generation, and more!