Verification of Smart Contracts Based on Natural Language with Description Logic

LANMR 2024

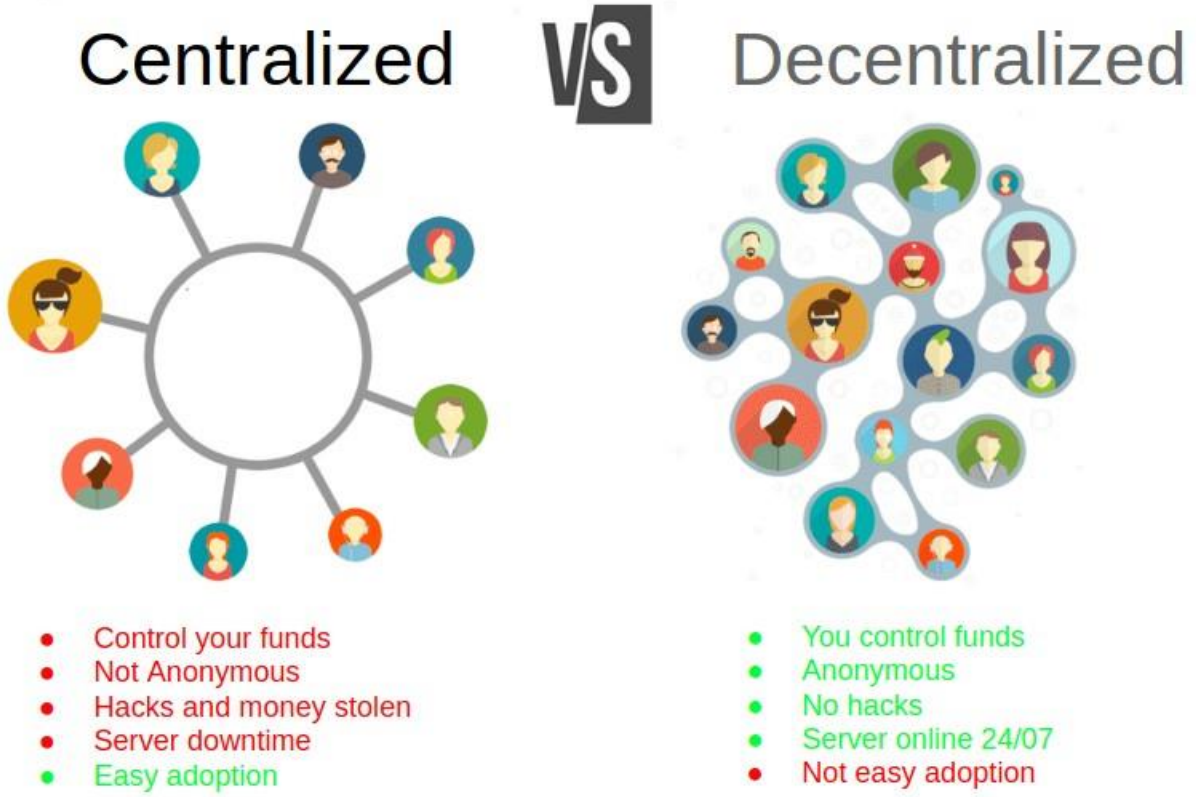
Gabriel Alejandro May Lozano

Ismael Everardo Bárcenas Patiño

- Blockchain technology has transformed the way we manage and record transactions by offering a decentralized and immutable system
- This technology ensures the integrity and transparency of operations by eliminating the need for intermediaries and recording each transaction in a way that cannot be altered.

INTRODUCTION

Difference with tradicional systems



Smart Contracts

- Computer programas that are executing with the terms of the agreement between the contracting parties written into its lines of code
- Features
  - **Self-Executing**
  - **Immutable**
  - **Transparent & Decentralized**
  - **Programmable**
  - **Interoperability**

Applications of SC

By 2muchcoffee.com

Challenges & Issues of Smart Contracts

- Code Vulnerabilities
- Immutability
- Complexity
- Legal and Regulatory Uncertainty
- Security Risks
- Interoperability with other platforms
- Gas Costs
- Testing and Validation

Related Work

- Different tools and perspectives have been implemented to achieve the design and construction of smart contracts that provide security and confidence in their security
- Formal methods have been fundamental in providing a rigorous framework for mathematical validation, ensuring the correctness and security of the code and its operation
- The integration of artificial intelligence (AI) has managed to automate the detection of vulnerabilities and errors, improving efficiency in the review

PREVIOUS WORKS OF SMART CONTRACT WITH FM & AI

FM	AI
Amani et al.: Formal verification of Solidity smart contracts	Momeni et al.: Machine learning model that detects patterns of security vulnerabilities in smart contracts
Abdellatif et al.: Verify SC with a statical model checking approach.	Shakya et al.: Features extraction in Solidity SC
Nam et al.: Verification of SC with ATL.	Tang et al. LightingCat model to identify malicious code

By industrywired.com

New approach

- The implementation of natural language processing models has resulted in promising tools to analyze and generate contracts, facilitating understanding and validation
- These advances represent the convergence of traditional techniques and emerging technologies to achieve more robust and accurate verification

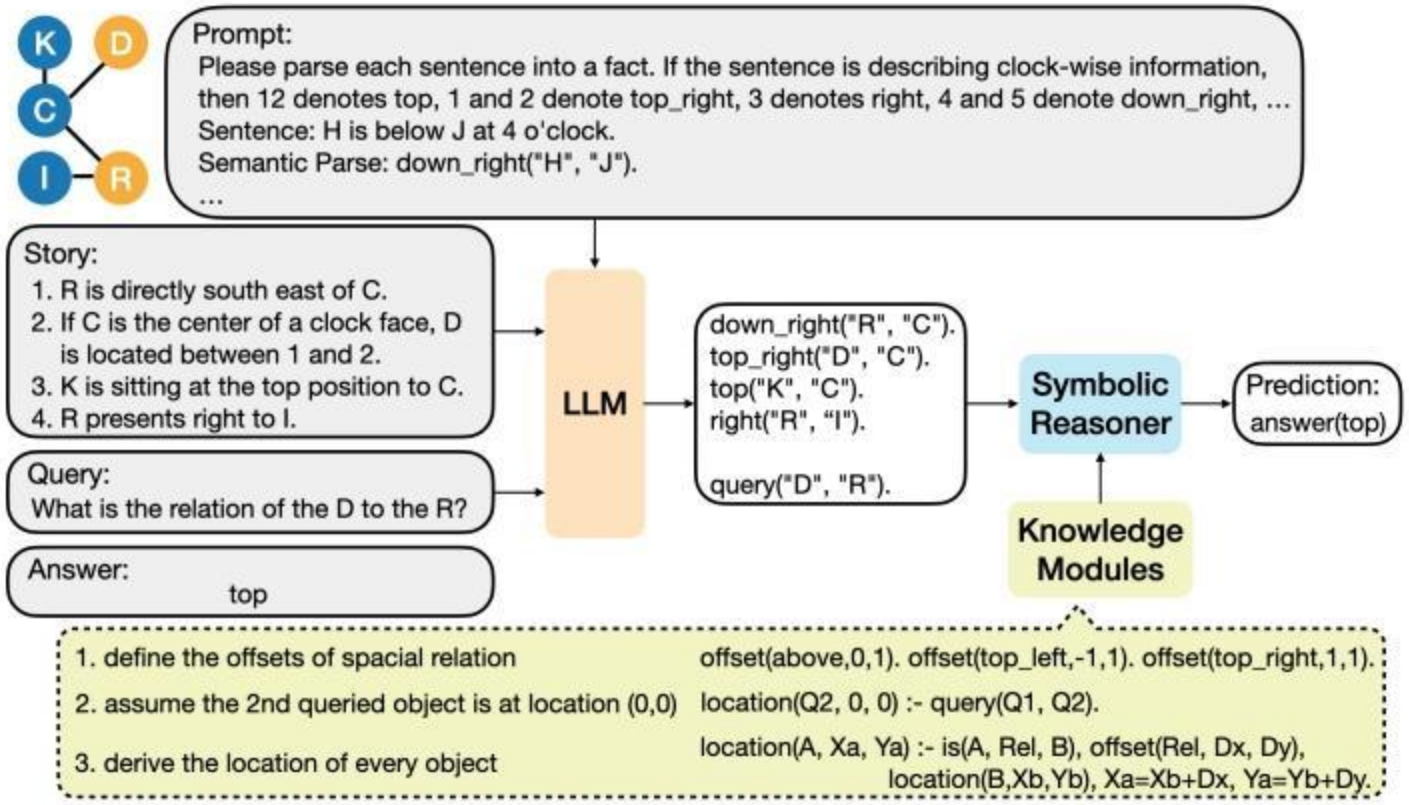
Natural Language Processing (NLP)

- NLP is a field of AI focused on the interaction between computers and human language, enabling machines to understand, interpret, and generate human language
- Nowadays, they have focused on the development of Large Language Models (LLM)
- Companies use it for several automated tasks, such as to:
  - Process, analyze, and archive large documents
  - Analyze customer feedback or call center recordings
  - Classify and extract text
  - Text generation
  - Chatbots and Virtual Assistants

Related Work with NLP

- Karanjai et al.: Works of generation of SC with ChatGPT and Google PaLM2
- Soud et al.: Automated prediction system and prioritization of vulnerabilities in smart contracts
- Nelaturu et al.: Framework of verification for Move SC

NLP reasoning problem



While large language models (LLMs) appear to be robust and general, their reasoning ability is not at a level to understanding natural language reasoning problems

However, Yang et al. observed that a large LLM can serve as a highly effective few-shot semantic parser.

They presents [LLM] +ASP

Smart contract feedback

Formal verification of smart contracts is a critical and essential process to ensure their integrity

Necessary to build smart contracts with specific formats

Analysis of the logic that defines their characterization

Our motivation

- Dávila et al. mentions that the success of applying Description Logics (DL) in different domains, notably in the Semantic Web, motivates proposing DLs as a formality to verify Smart Contracts
- Some contributions in this context are:
  - Van Der Straeten et al. UML models consistency with DL
  - Chaabani et al. Proof procedure for the description logic ALC
  - Vieira et al. Use of Large Language Models to Generate Capability Ontologie

OBJETIVES

- General:

  Develop a verification system of smart contract by implementing large language models and description logics

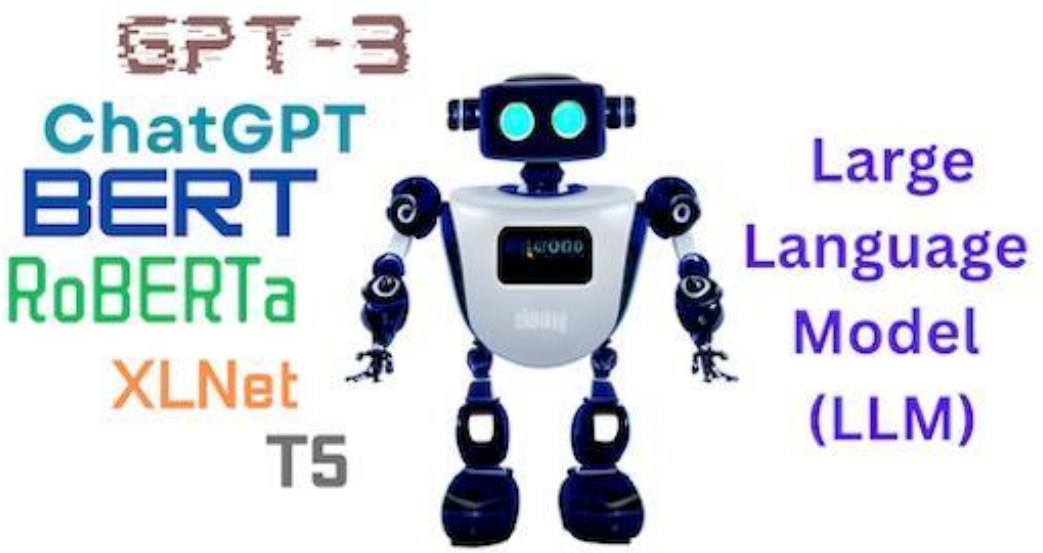
- Specifics:

- Develop a natural language-based interface to specify smart contracts in terms of descriptive logicsDevelop

- Implement a verification module of smart contract based on descriptive logics

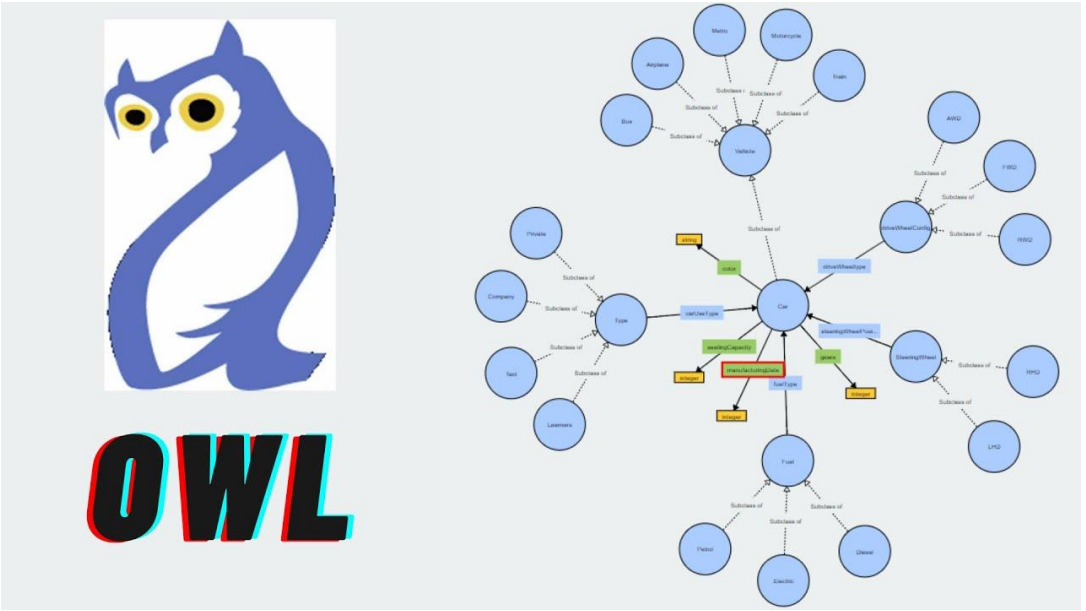
Tools: LLM's

- NLP models that are trained on vast amounts of text data from diverse sources to learn language patterns, context, and knowledge
- Implement a LLM would allow:
  - Parsing descriptions of a smart contract in natural language into a formal format interpretable by logical systems
  - Identifying the rules and conditions described in the input provided through an interactive user interface.



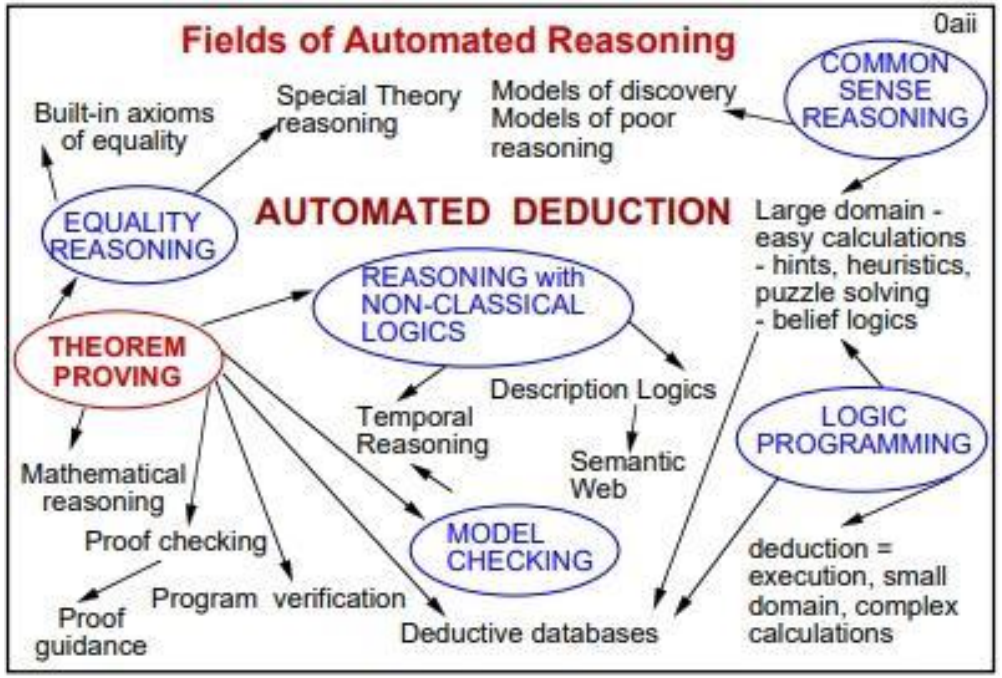
Tools: Description Logics (DL)

- Description Logic is a type of formal logical language that allows to represent and reason about knowledge representations
- The implementation of DLs aims to:
  - Model the properties of the smart contract by representing them in ontologies (OWL)
  - Provide a format for the application of deduction techniques for their subsequent verification



Tools: Automated reasoner

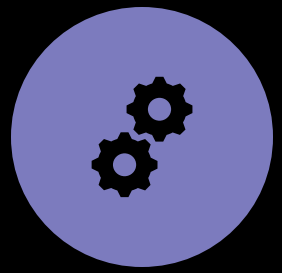
- Automated reasoners are software systems designed to make logical inferences and deduce new conclusions from prior knowledge.
- Some examples of reasoners are:
  - FaCT++
  - HermiT
  - Pellet
  - Others



APPROACH OF OUR PROPOSAL

USTED ENTERS THE DESCRIPTION OF THE SMART CONTRACT TO BE VERIFIED THROUGH THE NATURAL LANGUAGE-BASED INTERFACE.

THE DESCRIPTION IS TAKEN AS INPUT TO THE LLM TO GENERATE SMART CONTRACTS IN TERMS OF DESCRIPTION LOGICS (OWL)

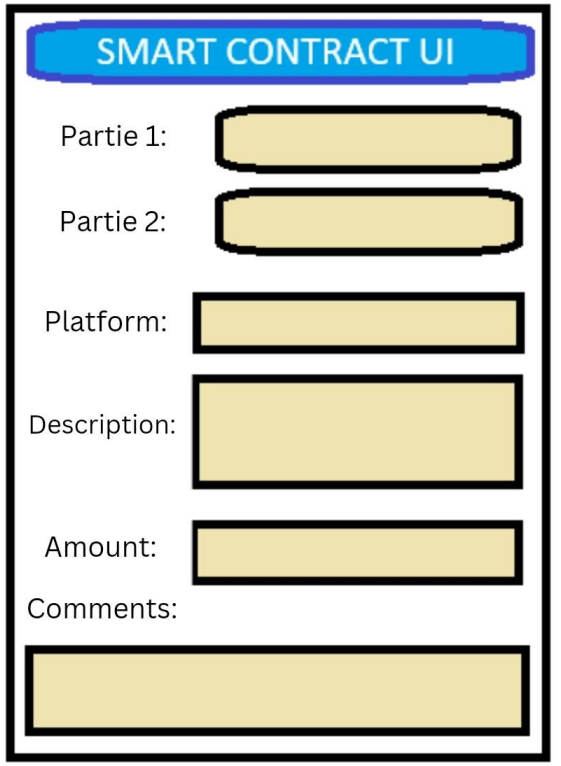


THE GENERATED SMART CONTRACT IS TAKEN AS INPUT TO AN AUTOMATED REASONER TO PERFORM VERIFICATION.

THE REASONER PROVIDES AS OUTPUT THE RESULTS OF THE CONSISTENCY AND FUNCTIONALITY OF THE DESCRIBED SMART CONTRACT

In-Context Learning Prompt

"A smart contract is designed to facilitate the buying and selling of an item. Users can list an item for sale by sending item details and a price to the contract. Buyers can purchase the item by sending the appropriate amount to the contract. When a buyer sends the correct amount, the contract transfers ownership of the item from the seller to the buyer and transfers the funds to the seller. Users can check the status of the item, including its price and the current seller."

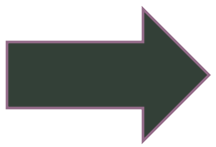


(Work in progress)

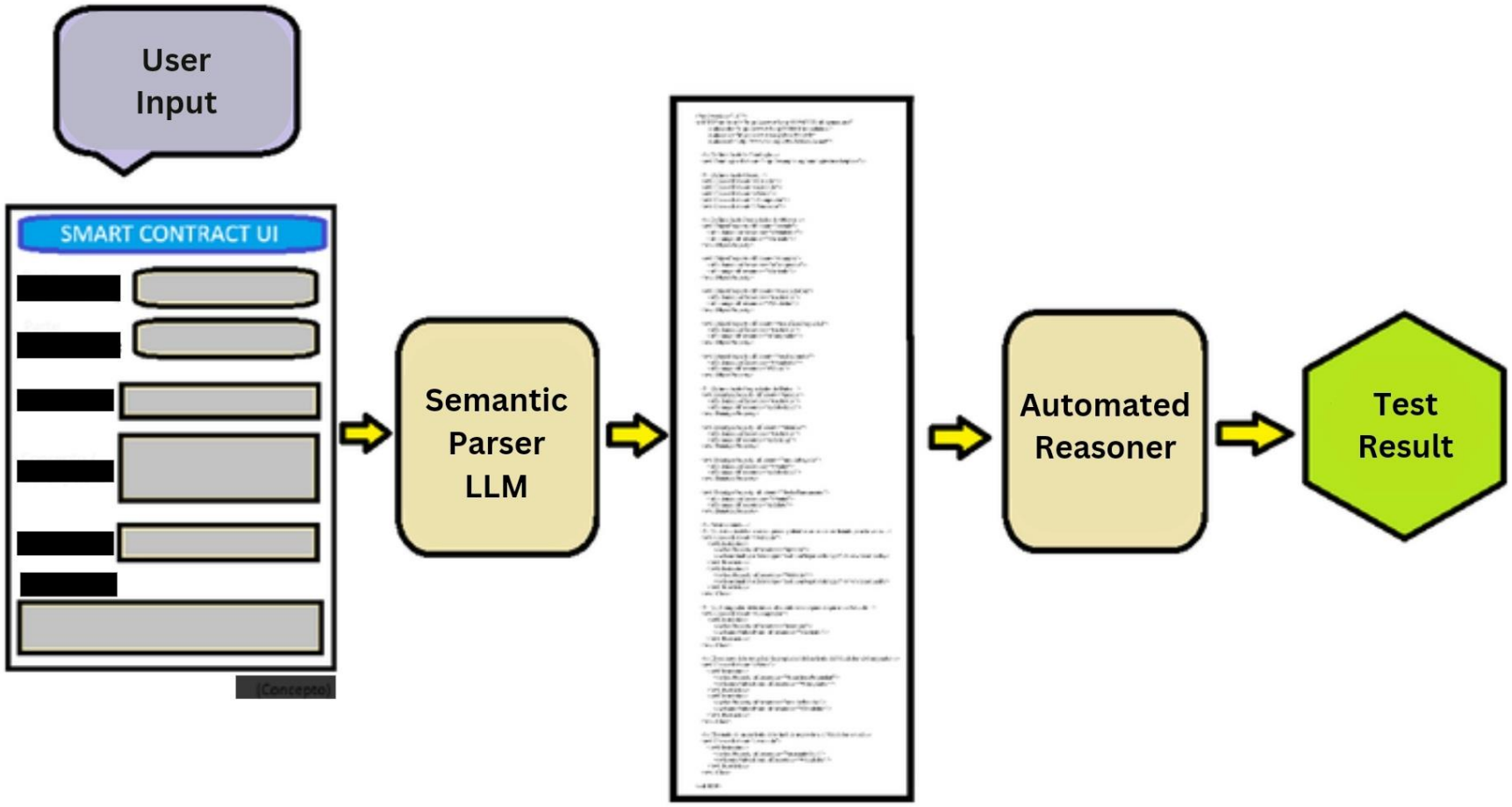
USER INPUT

LOGIC LANGUAGE OF SC

1. **Classes:**
   - `User`
   - `Item`
   - `Sale`
   - `Buyer`
   - `Seller`
2. **Object Properties:**
   - `sell` (from `Seller` to `Item`)
   - `buy` (from `Buyer` to `Item`)
   - `associatedWith` (from `Item` to `Seller`)
   - `transfersOwnership` (from `Item` to `Buyer`)
   - `receivesFunds` (from `Seller` to `Sale`
3. **Data Properties:**
   - `price` (from `Item` to a numeric value)
   - `details` (from `Item` to a descriptive text)
   - `amountPaid` (from `Sale` to a numeric value)
   - `transactionDate` (from `Sale` to a date
4. **Restrictions:**
   - An `Item` must have a `price` and `details` before it is listed for sale.
   - A `Buyer` must send the exact amount to purchase an `Item`.
   - The contract must transfer ownership of the `Item` from the `Seller` to the `Buyer` and transfer funds from the `Buyer` to the `Seller`.
   - The status of an `Item` must include its `price` and the current `Seller`.



```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

    <!-- Definición de la Ontología -->
    <owl:Ontology rdf:about="http://example.org/ontologies/marketplace"/>

    <!-- Definición de Clases -->
    <owl:Class rdf:about="#Usuario"/>
    <owl:Class rdf:about="#Artículo"/>
    <owl:Class rdf:about="#Venta"/>
    <owl:Class rdf:about="#Comprador"/>
    <owl:Class rdf:about="#Vendedor"/>

    <!-- Definición de Propiedades de Objetos -->
    <owl:ObjectProperty rdf:about="#vende">
        <rdfs:domain rdf:resource="#Vendedor"/>
        <rdfs:range rdf:resource="#Artículo"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#compra">
        <rdfs:domain rdf:resource="#Comprador"/>
        <rdfs:range rdf:resource="#Artículo"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#asociadoCon">
        <rdfs:domain rdf:resource="#Artículo"/>
        <rdfs:range rdf:resource="#Vendedor"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#transfierePropiedad">
        <rdfs:domain rdf:resource="#Artículo"/>
        <rdfs:range rdf:resource="#Comprador"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#recibeFondos">
        <rdfs:domain rdf:resource="#Vendedor"/>
        <rdfs:range rdf:resource="#Venta"/>
    </owl:ObjectProperty>

    <!-- Definición de Propiedades de Datos -->
    <owl:DatatypeProperty rdf:about="#precio">
        <rdfs:domain rdf:resource="#Artículo"/>
        <rdfs:range rdf:resource="xsd:decimal"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#detalles">
        <rdfs:domain rdf:resource="#Artículo"/>
        <rdfs:range rdf:resource="xsd:string"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#montoPagado">
        <rdfs:domain rdf:resource="#Venta"/>
        <rdfs:range rdf:resource="xsd:decimal"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#fechaTransacción">
        <rdfs:domain rdf:resource="#Venta"/>
        <rdfs:range rdf:resource="xsd:date"/>
    </owl:DatatypeProperty>

    <!-- Restricciones -->
    <!-- Un Artículo debe tener un precio y detalles antes de ser listado para la venta -->
    <owl:Class rdf:about="#Artículo">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#precio"/>
            <owl:cardinality rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#detalles"/>
            <owl:cardinality rdf:datatype="xsd:nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
    </owl:Class>

    <!-- Un Comprador debe enviar el monto exacto para adquirir un Artículo -->
    <owl:Class rdf:about="#Comprador">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#compra"/>
            <owl:someValuesFrom rdf:resource="#Artículo"/>
        </owl:Restriction>
    </owl:Class>

    <!-- El contrato debe transferir la propiedad del Artículo del Vendedor al Comprador -->
    <owl:Class rdf:about="#Venta">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#transfierePropiedad"/>
            <owl:someValuesFrom rdf:resource="#Comprador"/>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#recibeFondos"/>
            <owl:someValuesFrom rdf:resource="#Vendedor"/>
        </owl:Restriction>
    </owl:Class>

    <!-- El estado de un Artículo debe incluir su precio y el Vendedor actual -->
    <owl:Class rdf:about="#Artículo">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#asociadoCon"/>
            <owl:someValuesFrom rdf:resource="#Vendedor"/>
        </owl:Restriction>
    </owl:Class>

</rdf:RDF>
```



Pipeline of our proposal

Example 1: Auction

- A smart contract is designed that manages an auction in which users can make bids to acquire an item. The bidding process works as follows:
  - Making a Bid
  - Manager Previous Bids
  - Highest Bidder Query
- The contract ensures that there is always a single highest bidder and a single highest bid at any given time. Additionally, it handles refunding of previous bids correctly to prevent funds from being lost.

LOGIC FORMAT OF AUCTION

- Classes:

  User

  Bid

  Item

  Auction

  Bidder

- Object Properties:

  makesBid (User to Bid)

  hasBid (Bid to Item)

  isGreaterThan (Bid to Bid)

  returnsFunds (Bid to User)

  queryBidder (User to Bidder)

- Data Properties:

  amount (Bid to a numeric value)

  status (Bid to a validity state, e.g. valid or invalid)

  date (Bid to a date)

- Restrictions:

  A Bid must have an amount that is strictly greater than the current highest bid (isGreaterThan).

  A User cannot make a bid if they are already the highest bidder (isBidder)

  When a User makes a valid bid and becomes the new highest bidder, the contract must return the funds from the previous bid to the User who was the previous highest bidder (returnsFunds).

  Any User may query who the highest bidder is at any time (queryBidder)

  The Auction must ensure that there is always a single highest bidder and a single highest offer at any given time (isBidder and isHigherThan).

EXAMPLE 2: LIFE INSURANCE

A smart contract is designed that manages life insurance and handles the associated payments and validations as follows:

- Purchasing Insurance
- Insurance Claims
- Claim Validation
- Insurance Status Check

The contract ensures that only valid payments are made and ensures that funds are handled correctly to cover claims without leaving room for error

FORMAL FORMAT OF INSURANCE LOGIC

- Classes:

  User

  Policy

  Insurance

  Claim

  Beneficiary

  Insured

- Object Properties:

  buyPolicy (User to Policy)

  isInsured (User to Insured)

  makesClaim (Insured to Claim)

  designatesBeneficiary (Insured to Beneficiary)

  pays (Insurance to Beneficiary)

  verifiesClaim (Claim to Insurance)

  queriesStatus (User to Insurance)

- Data Properties:

  paymentAmount (Policy to a numeric value)

  status (Policy to a status of validity)

  insuredAmount (Insurance to a numeric value)

  purchaseDate (Policy to a date)

  claimDate (Claim to a date)

  claimAmount (Claim to a numeric value)

- Restrictions:

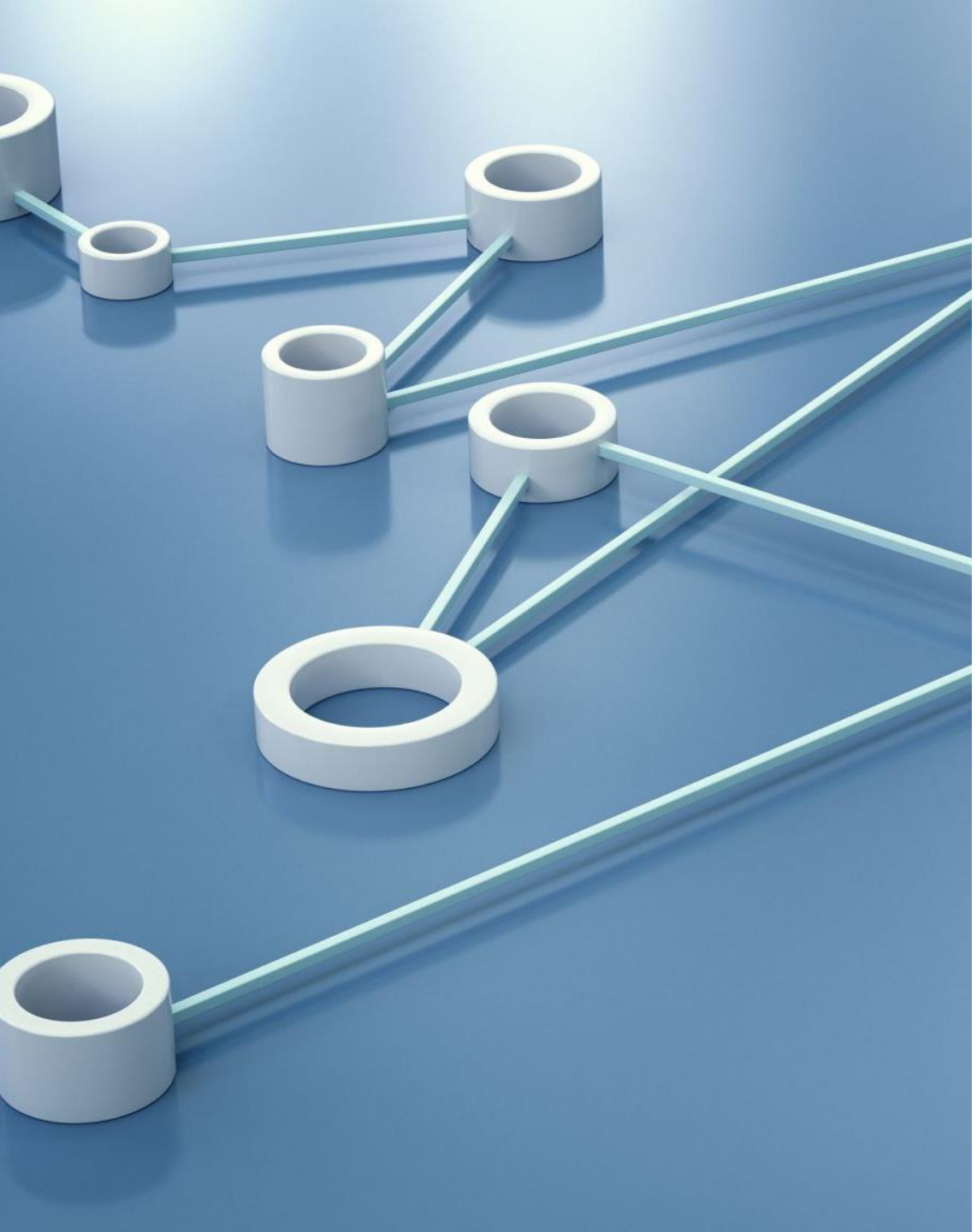
  A User You must send a fixed amount of funds to purchase an Insurance Policy, which registers the User as an Insured (buyPolicy and beInsured).

  The contract must maintain an adequate balance for the payment of future claims (InsuredAmount).

  Only an active Insured can make a Claim, and the Claim must be externally verified to be valid (makeClaim and verifyClaim).

  The contract must pay the insured amount to the designated Beneficiary only if the Claim is valid and the Insured was active at the time of the Claim (pays).

  Any User can check the status of their Policy, including whether they are insured and the total insured amount (checkStatus).



Vision

- System for transforming natural language inputs into logical language
- Formal verification of Smart Contracts with Description Logics
- Provides access to Smart Contract design for less experienced users

Info

- Presenter: BCS Gabriel Alejandro May Lozano
- Tutor: PhD. Ismael Everardo Barcenas Patiño
- Postgraduate Program in Computer Science and Engineering

UNAM POSGRADO

CONAHCYT
CONSEJO NACIONAL DE HUMANIDADES CIENCIAS Y TECNOLOGÍAS

Thank you! ☺