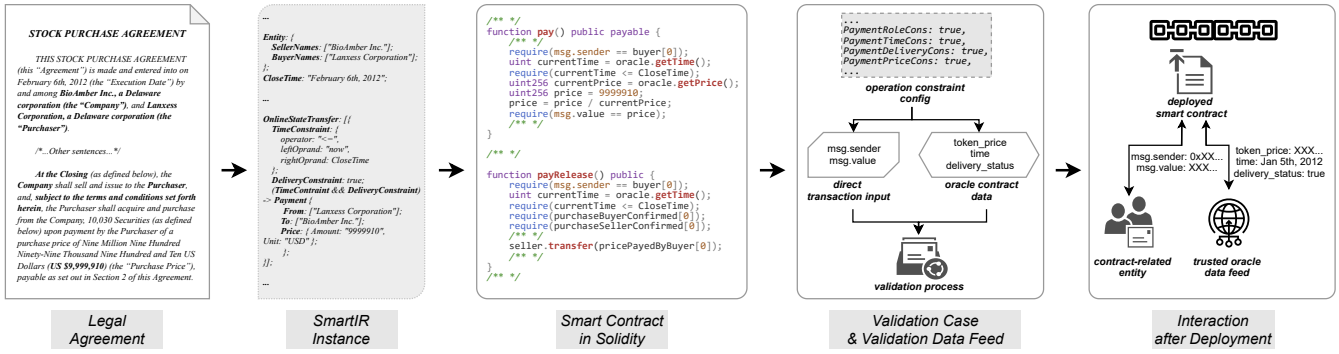

Figure 1: An end-to-end example of smart contract synthesis, validation and deployment using iSyn.

A APPENDIX

A.1 An End-to-End Example

We present an overview of iSyn using an end-to-end illustrative example in Figure 1.

Intent Definition. Financial agreements written by legal professionals are not originally meant to be translated into software code. Thus, the very first step in iSyn is to decide the proper programming model of legal agreements. We abstract four types of intents in our model to represent the core transaction logic in the legal agreements, as driven by our empirical study. In the example shown in Figure 1, we highlight a payment intent between two entities once certain conditions are satisfied.

SmartIR Construction. SmartIR is our critical design to bridge the gap between the financial agreements written in natural languages and software code written in turing-complete programming languages. On the one hand, the intents expressed in legal agreements are automatically extracted in structured SmartIR instances, underpinned by rigorous grammar rules. On the other hand, the formal specification of SmartIR instances provides a well-defined scope for program synthesis. In this example, the payment intent is translated into a SmartIR item called *OnlineStateTransfer* with two constraints representing the conditions defined in the legal agreement. Internally, the above procedure is accomplished using our synergistic pipeline that unifies multiple NLP techniques.

SmartIR Ambiguity Resolution with Knobs. To embrace ambiguities in legal agreements, for each SmartIR item, iSyn provides the top-5 candidates sorted based on NLP confidence scores. The top-1 candidate is by default presented in the concrete SmartIR instance generated by iSyn. All stakeholders may choose to manually inspect the SmartIR instance and alter it using our knobs to select other more appropriate candidates. Once the SmartIR instance is agreed, the subsequent process requires no human intervention. As shown in our experimental results, iSyn identifies the correct SmartIR items in the top-5 candidates with probability higher that 90%.

Smart Contract Synthesis. The final blockchain-executable smart contracts are synthesized based on the SmartIR instances, predefined smart contract templates, and optional user inputs. A template essentially defines the "backbone" of the smart contract in form of abstract syntax tree (AST). The synthesis process then modifies, populates, or trims certain nodes of the AST, based on the SmartIR instances, to eventually output the synthesized smart contract. We employ a template-driven contract synthesis for predictability of the final smart contract. The predictability facilitates the correctness check on the contract.

In this example, the SmartIR items are realized by two functions in Solidity, where the critical constraints are expressed via the require statements. The contract also relies on oracles to obtain authenticated offchain data, such as the delivery date of physical goods associated with the payment.

Smart Contract Validation. We build a generic validation framework to verify the correctness of iSyn-synthesized smart contracts and perform automated vulnerability checks on the synthesized contracts. This framework generates comprehensive validation cases to cover all possible execution branches determined by the operation constraints extracted from the SmartIR instances. In this example, the price value and the payment time are two critical operation constraints validated by the framework.

Post-deployment Interaction. The stakeholders of the legal agreements can interact with the post-deployed smart contracts as typical Ethereum smart contracts. In this example, the payer first deposits its payment into the smart contract by constructing a transaction to the pay interface. Once the payment conditions are satisfied, the money is released to the payee by calling the payRelease interface using the data provided by the trusted oracle.

A.2 The Other Details

We provide the detailed mapping between the SmartIR items and the elements in the contract template in Table 1. Additionally, we present the mapping from SmartIR items to operation constraints used to generate the validation cases in Table 2.

Finally, we attach our smart contract template in Code 2. The legal agreement document snippet, SmartIR and the iSyn-synthesized smart contract for the example in Figure 1 are shown in Document 1, Code 1 and Code 3, respectively. In addition, we present the extracted

operation constraints for generating validation cases for the example in Code 4, while the configuration of a validation case for it is shown in Code 5.

Table 1: The mapping between the SmartIR items and contract template elements.

SmartIR Item	*Template Element Name*	*Solidity Type*
⟨*EntityNamesDef*⟩	entityNames	string array(state)
⟨*CloseTimeDef*⟩	CloseTime	uint256(state)
⟨*EffectiveTimeDef*⟩	EffectiveTime	uint256(state)
⟨*ExpiryTimeDef*⟩	ExpireTime	uint256(state)
⟨*PaymentDef*⟩	pay & payRelease	function
⟨*SourceDef*⟩	require()	guard statement
⟨*DestinationDef*⟩	require()	guard statement
⟨*TimeConstraintDef*⟩ (OnlineStateTransfer) *	require()	guard statement
⟨*DeliveryConstraintDef*⟩ (OnlineStateTransfer)	payConfirm & require()	function & guard statement
⟨*PriceDef*⟩	price	uint256(local)
⟨*DeliveryConstraintDef*⟩ (OfflineDelivery)	uploadFileHash	function
⟨*DeliveryConstraintDef*⟩ (Termination)	terminateByDelivery & terminateConfirm	function
⟨*TimeConstraintDef*⟩ (Termination)	terminateByExpiry	function
⟨*OtherConstraintDef*⟩	terminateByOther	function

∗ Referring to the ⟨*TimeConstraintDef*⟩ belongs to **OnlineStateTransfer**. Same notations for others.

Table 2: The mapping between the SmartIR items and operation constraints for validation case.

SmartIR Item	*Operation onstraint*
⟨*EffectiveTimeDef*⟩	EffectiveTimeCons
⟨*SourceDef*⟩ (OnlineStateTransfer)	PaymentRoleCons
⟨*DestinationDef*⟩ (OnlineStateTransfer)	PaymentRoleCons
⟨*TimeConstraintDef*⟩ (OnlineStateTransfer)	PaymentTimeCons
⟨*DeliveryConstraintDef*⟩ (OnlineStateTransfer)	PaymentDeliveryCons
⟨*PriceDef*⟩ (OnlineStateTransfer)	PaymentPriceCons
⟨*DeliveryConstraintDef*⟩ (OfflineDelivery)	ValidFileSignUploaderCons
⟨*DeliveryConstraintDef*⟩ (Termination)	TerminationDeliveryCons
⟨*TimeConstraintDef*⟩ (Termination)	TerminationTimeCons
⟨*OtherConstraintDef*⟩ (Termination)	TerminationOtherCons

STOCK PURCHASE AGREEMENT

THIS STOCK PURCHASE AGREEMENT (this “Agreement”) is made and entered into on February 6th, 2012 (the “Execution Date”) by and among BioAmber Inc., a Delaware corporation (the “Company”), and Lanxess Corporation, a Delaware corporation (the “Purchaser”).

/*...Other sentences...*/

At the Closing (as defined below), the Company shall sell and issue to the Purchaser, and, subject to the terms and conditions set forth herein, the Purchaser shall acquire and purchase from the Company, 10,030 Securities (as defined below) upon payment by the Purchaser of a purchase price of Nine Million Nine Hundred Ninety-Nine Thousand Nine Hundred and Ten US Dollars (US$9,999,910) (the “Purchase Price”), payable as set out in Section 2 of this Agreement.

/*...Other sentences...*/

At the Closing, subject to the terms and conditions hereof, the Company shall deliver to the Purchaser the following:
(a) ... (b) ... (c)... (d)... (e)... (f)... (g)... (h)...
At the Closing, subject to the terms and conditions hereof, the Purchaser shall pay the Purchase Price by wire transfer of immediately available funds to an account designated in writing by the Company not less than two business days prior to the Closing and shall deliver to the Company the following:
(a) ... (b) ... (c) ... (d)...

/*...Other sentences...*/

This Agreement may be terminated by the Purchaser in the event that all of the conditions set forth in this Section 6 (other than this Section 6.7) do not occur on or before February 8th, 2012 and, upon such termination by the Purchaser, this Agreement shall become null and void, and there shall be no liability or obligation on the part of the Purchaser or its respective officers, directors, stockholders or affiliates.

/*...Other sentences...*/

Document 1: An example legal agreement.

```
ContractCategory: "StockPurchaseAgreement";

Entity: {
    SellerNames: ["BioAmber Inc."];
    BuyerNames: ["Lanxess Corporation"];
};
EffectiveTime: "February 6th, 2012";
CloseTime: "February 6th, 2012";
ExpiryTime: "February 8th, 2012";

OnlineStateTransfer: [{
    TimeConstraint: {
        operator: "<=",
        leftOprand: "now",
        rightOprand: CloseTime
    };
    DeliveryConstraint: true;
    (TimeContraint && DeliveryConstraint) -> Payment {
        From: ["Lanxess Corporation"];
        To: ["BioAmber Inc."];
        Price: { Amount: "9999910", Unit: "USD" };
    };
}];
OfflineDelivery: { ... };
Termination: { ... };
```

Code 1: IR specifications for the example legal agreement.

```
pragma solidity 0.5.16;
/** */
contract ContractTemplate {
    address payable[2] public entities;
    OracleContract internal oracle;
    string[2] public entityNames;

    uint256 public EffectiveTime;
    uint256 public CloseTime;
    uint256 public ExpiryTime;

    uint256[1][1] public pricePayed;

    bool[1][1] public payEntityConfirmed;
    bool[1][1] public terminateEntityConfirmed;

    mapping(string => bytes) fileSignMap;

    enum State { Created, Locked, Released, Inactive}

    State[1] public state;

    event Payed(uint payerIndex, uint receiverIndex);
    event Released(uint payerIndex, uint receiverIndex);
    event DeliveryTerminated(uint terminatorIndex);
    event ExpiryTerminated();
    event Closed();

    constructor() public payable {/*...*/}

    function pay(uint32 targetIndex) public payable {/*...*/}
    function payConfirm(uint32 targetIndex) public {/*...*/}
    function payRelease(uint32 targetIndex) public {/*...*/}

    function uploadFileSign(string memory fileName, bytes memory sign) public {
        /*...*/
    }

    function terminateConfirm(uint32 targetIndex) public {/*...*/}
    function terminateByDelivery(uint targetIndex) public {/*...*/}
    function terminateByExpiry() public {/*...*/}
    function terminateByOther() public {/*...*/}

    function close() public {/*...*/}

    /*...*/
}
```

Code 2: Smart contract template used by iSyn

```
pragma solidity 0.5.16;
/** */
contract StockPurchaseAgreement {
    address payable public seller;
    address payable[1] public buyer;
    OracleContract internal oracle;
    string public sellerName;
    string[1] public buyerName;
    uint256 public EffectiveTime;
    uint256 public CloseTime;
    uint256 public ExpiryTime;
    uint256[1] public pricePayedByBuyer;
    bool[1] public paySellerConfirmed;
    bool[1] public payBuyerConfirmed;
    mapping(string => bytes) fileSignMap;
    enum State { Created, Locked, Released, Inactive }
    State[1] public state;
  /* ... */
    constructor() public payable {
        EffectiveTime = 1328457600;
        CloseTime = 1328544000;
        ExpiryTime = 1328630400;
        sellerName = "BioAmber Inc.";
        seller = address(0);
        buyerName =["Lanxess Corporation"];
        buyer =[address(0)];
    }
    function pay_0() public payable {
        require(state[0] == State.Created || state[0] == State.Locked);
        uint currentTime = oracle.getTime();
        require(currentTime <= CloseTime);
        uint256 currentPrice = oracle.getPrice();
        uint256 price = 500000;
        price = price / currentPrice;
        require(msg.value == price);
        /** */
    }
    /* ... */
    function payRelease_0() public {
        require(msg.sender == buyer[0]);
        uint currentTime = oracle.getTime();
        require(currentTime <= CloseTime);
        require(purchaseBuyerConfirmed[0]);
        require(purchaseSellerConfirmed[0]);
        /** */
        seller.transfer(pricePayedByBuyer[0]);
        /** */
    }
    function uploadFileSign(string memory fileName, bytes memory sign) public {
        /** */
        require(validSender);
        fileSignMap[fileName] = sign;
    }
    function terminateByExpiry() public {
        uint currentTime = oracle.getTime();
        require(currentTime >= OutSideClosingDate);
        /** */
    }
    function terminateByOther() public {
        uint currentTime = oracle.getTime();
        require(currentTime <= CloseTime);
        bool conditionState = oracle.getConditionState();
        require(conditionState);
        /** */
    }
  /* ... */
}
```

Code 3: Code snippet of the synthesized smart contract for the example legal agreement

```
EffectiveTimeCons: true;
ExpiryTimeCons: true;
CloseTimeCons: true;
PaymentRoleCons: true;
PaymentTimeCons: true;
PaymentPriceCons: true;
PaymentDeliveryCons: true;
ValidFileSignUploaderCons: true;
OtherTerminationCons: true;
ExpiryTerminationCons: true;
DeliveryTerminationCons: false;
```

Code 4: Extracted operation constraints of the example legal agreement for constructing validation cases, where *true* means the corresponding operation constraint needs to be tested during validation.

```
Invoked function:
    pay_0
Direct transaction input:
    msg.sender: 0xXXXXX (the buyer's address),
    msg.value: XXXXX (paid amount in ETH)
Oracle input:
    token_price: XXXXX,
    time: Feb 10th, 2012,
    delivery_status: true
```

Code 5: Configuration of a validation case for the example legal agreement's synthesized smart contract, where all operation constraints but *PaymentTimeCons* are satisfied. It triggers an expected failure due to illegal payment time when invoking function `pay_0`.

```
...

Entity: {
    SellerNames: ["Entech Solar" → "Entech Solar, Inc."];
    BuyerNames: ["The Quercus Trust"];
};
...
```

Code 6: IR correction required by legal experts