Resource-Aware Session Types for Digital Contracts

Ankush Das
Stephanie Balzer
Jan Hoffmann
Frank Pfenning
Ishani Santurkar
Carnegie Mellon University

Session 2, CSF 2021
The Nomos Language

Digital Contracts
The Nomos Language

Challenges:
1. contract protocols are violated
2. asset duplication/deletion
3. automatic inference of execution cost

Digital Contracts
The Nomos Language

Challenges:
1. contract protocols are violated
2. asset duplication/deletion
3. automatic inference of execution cost
The Nomos Language

Digital Contracts

Challenges:
1. contract protocols are violated
2. asset duplication/deletion
3. automatic inference of execution cost

Solutions:
1. session types enforce protocols
2. linear types track assets
3. resource-aware types infer execution cost

Resource-Aware Session Types
The Nomos Language

Digital Contracts

Challenges:
1. contract protocols are violated
2. asset duplication/deletion
3. automatic inference of execution cost

Solutions:
1. session types enforce protocols
2. linear types track assets
3. resource-aware types infer execution cost

Resource-Aware Session Types
What are Digital Contracts?

*Programs implementing a digital (legal or financial) transaction*
What are Digital Contracts?

Programs implementing a digital (legal or financial) transaction

Online Transactions
What are Digital Contracts?

Programs implementing a digital (legal or financial) transaction

Online Transactions

Wealth Management Applications
What are Digital Contracts?

**Programs implementing a digital (legal or financial) transaction**

- Online Transactions
- Wealth Management Applications
- recently popularized in the form of Blockchains and Cryptocurrencies
Example: Auction Contract

status: running
Example: Auction Contract

Bidder 1
Bid 1

Bidder 2
Bid 2

Bidder 3
Bid 3

status: running
Example: Auction Contract

Bidder 1

Bidder 2

Bidder 3

Bid 1

Bid 2

Bid 3

status: running
Example: Auction Contract

Bidder 1

Bidder 2

Bidder 3

status: ended

Bid 1

Bid 2

Bid 3
Example: Auction Contract

Bidder 1

Bidder 2

Bidder 3

Bid 1
Bid 2
Bid 3

status: ended
Example: Auction Contract

Bidder 1

Bidder 2

Bidder 3

Bid 1
Bid 2
Bid 3

status: ended
Example: Auction Contract

Bidder 1
Bid 1

Bidder 2
Bid 2

Bidder 3
Bid 3

status: ended
function bid() public payable {
    bid = msg.value;
    bidder = msg.sender;
    pendingReturns[bidder] = bid;
    if (bid > highestBid) {
        highestBidder = bidder;
        highestBid = bid;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    return true;
}
function bid() public payable {
    bid = msg.value;
    bidder = msg.sender;
    pendingReturns[bidder] = bid;
    if (bid > highestBid) {
        highestBidder = bidder;
        highestBid = bid;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    return true;
}
What happens if `collect` is called when `auction` is running?

```solidity
function bid() public payable {
    bid = msg.value;
    bidder = msg.sender;
    pendingReturns[bidder] = bid;
    if (bid > highestBid) {
        highestBidder = bidder;
        highestBid = bid;
    }
}

function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    return true;
}
```
What happens if `collect` is called when auction is running?

Add clause:
```
require (status == ended);
```

```
function bid() public payable {
    bid = msg.value;
    bidder = msg.sender;
    pendingReturns[bidder] = bid;
    if (bid > highestBid) {
        highestBidder = bidder;
        highestBid = bid;
    }
}
```

```
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    return true;
}
```
What happens if `collect` is called when auction is running?

Add clause:
```
require (status == ended);
```

What happens if `collect` is called twice by the same user?
Auction in Solidity

What happens if `collect` is called when auction is running?

Function `bid()`
```solidity
function bid() public payable {
    bid = msg.value;
    bidder = msg.sender;
    pendingReturns[bidder] = bid;
    if (bid > highestBid) {
        highestBidder = bidder;
        highestBid = bid;
    }
}
```

Add clause:
```solidity
require (status == ended);
```

What happens if `collect` is called twice by the same user?

Function `collect()`
```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    return true;
}
```

Set:
```solidity
pendingReturns[msg.sender] = 0;
```
Auction in Solidity

What happens if `collect` is called when auction is running?

Add clause:

```
require (status == ended);
```

What happens if `collect` is called twice by the same user?

Set:

```
pendingReturns[msg.sender] = 0;
```

How much fee should the user pay? No support for static and automatic cost prediction!
**The Nomos Language**

**Challenges:**
1. *contract protocols are violated*
2. *asset duplication/deletion*
3. *automatic inference of execution cost*

**Solutions:**
1. *session types enforce protocols*
2. *linear types track assets*
3. *resource-aware types infer execution cost*
The Nomos Language

Digital Contracts

Challenges:
1. contract protocols are violated
2. asset duplication/deletion
3. automatic inference of execution cost

Solutions:
1. session types enforce protocols
2. linear types track assets
3. resource-aware types infer execution cost

Resource-Aware Session Types
What are Session Types?

*Type system for implementing concurrent message-passing processes*
What are Session Types?

Type system for implementing concurrent message-passing processes
What are Session Types?

Type system for implementing concurrent message-passing processes

Bidder Process

assign session type to channels

Auction Process

Resource-Aware Session Types for Digital Contracts

Das, Balzer, Hoffmann, Pfenning, Santurkar | CSF 2021
What are Session Types?

Type system for implementing concurrent message-passing processes

Key Idea: Nomos uses session types to express contract protocols
Auction as a Session Type
\[
\text{type} \ \text{auction} = \oplus \{ \text{running} : \&\{\text{bid} : \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \&\{\text{collect} : \oplus \{\text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Auction as a Session Type

\[
\text{type} \quad \text{auction} = \bigoplus \{ \text{running} : \&\{ \text{bid} : \text{money} \rightarrow \text{auction} \}, \quad \\
\text{ended} : \&\{ \text{collect} : \bigoplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Auction as a Session Type

\[
type \; auction = \bigoplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \bigoplus \{ \text{won} : \text{monalisa} \otimes \text{auction} , \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Auction as a Session Type

\[\text{type auction} = \oplus \{\text{running} : \&\{\text{bid} : \text{money} \rightarrow \text{auction}\},
\text{ended} : \&\{\text{collect} : \oplus \{\text{won} : \text{monalisa} \otimes \text{auction},
\text{lost} : \text{money} \otimes \text{auction}\}\}\}\]
Auction as a Session Type

\[
\text{type } \text{auction} = \bigoplus \{\text{running} : \&\{\text{bid} : \text{money} \rightarrow \text{auction}\}, \\
\text{ended} : \&\{\text{collect} : \bigoplus \{\text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction}\}\}\}
\]
Auction as a Session Type

\[
\text{type auction} = \bigoplus \{ \text{running} : \&\{ \text{bid} : \text{money} \to \text{auction} \}, \\
\text{ended} : \&\{ \text{collect} : \bigoplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Auction as a Session Type

\[ \text{type auction} = \oplus \{ \text{running} : \&\{ \text{bid} : \text{money} \rightarrow \text{auction} \}, \]
\[ \text{ended} : \&\{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \]
\[ \text{lost} : \text{money} \otimes \text{auction} \} \} \]
Auction as a Session Type

\[
\text{type } \text{auction} = \oplus \{\text{running : } &\{\text{bid : money } \rightarrow\text{ auction}\}, \\
&\text{ended : } &\{\text{collect : } \oplus \{\text{won : monalisa } \otimes\text{ auction}, \\
&\text{lost : money } \otimes\text{ auction}\}\}\}
\]
Auction as a Session Type

\[
\text{type auction} = \bigoplus \{ \text{running} : \&\{ \text{bid} : \text{money} \to \text{auction} \}, \\
\text{ended} : \&\{ \text{collect} : \bigoplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Auction as a Session Type

\[
\text{type auction} = \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Auction as a Session Type

\[
\text{type auction} = \bigoplus \{\text{running} : \&\{\text{bid} : \text{money} \rightarrow \text{auction}\}, \\
\quad \text{ended} : \&\{\text{collect} : \bigoplus \{\text{won} : \text{monalisa} \otimes \text{auction}, \\
\quad \quad \text{lost} : \text{money} \otimes \text{auction}\}\}\}\}
\]
Auction as a Session Type

\[ \text{type auction} = \bigoplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \text{auction} \}, \]
\[ \text{ended} : \& \{ \text{collect} : \bigoplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \]
\[ \text{lost} : \text{money} \otimes \text{auction} \} \} \]
Auction as a Session Type

\[ \text{type auction} = \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \to \text{auction} \}, \]
\[ \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \]
\[ \text{lost} : \text{money} \otimes \text{auction} \} \} \} \]
Auction as a Session Type

\[ \text{type auction} = \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \} \]
Asset Preservation
Asset Preservation

Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)
Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)

```
type auction = ⊕{running : &{bid : money → auction},
               ended : &{collect : ⊕{won : monalisa ⊗ auction,
                              lost : money ⊗ auction}}}
```
Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)

$\text{type} \ \text{auction} = \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \multimap \text{auction} \},$

$\text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction},$

$\text{lost} : \text{money} \otimes \text{auction} \} \} \}$

Nomos statically guarantees:

- money and monalisa are neither duplicated nor discarded, only transferred between processes
- no additional machinery needed!!
Get Asset Preservation for Free!

Key Idea: Session types in Nomos are derived from linear logic (which is the logic of assets!)

\[\text{type auction} = \oplus\{\text{running} : \&\{\text{bid} : \text{money} \rightarrow \text{auction}\},\]
\[\quad \text{ended} : \&\{\text{collect} : \oplus\{\text{won} : \text{monalisa} \otimes \text{auction},\]
\[\quad \quad \quad \quad \text{lost} : \text{money} \otimes \text{auction}\}\}\}

Nomos statically guarantees:

- money and monalisa are neither duplicated nor discarded, only transferred between processes
- no additional machinery needed!!
Resource-Aware Auction

cost of bidding = 2, cost of collecting = 3
Resource-Aware Auction

cost of bidding = 2, cost of collecting = 3

Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost
Resource-Aware Auction

cost of bidding = 2, cost of collecting = 3

**Key Idea:** Nomos enhances session types with potential (or fee) that pays for execution cost

\[ \text{type auction} = \text{\texttt{\small\textlangle 3 \rangle + \{running : \&\{bid : \text{money} \rightarrow \text{\textlangle 1 \rangle auction}\}, ended : \&\{collect : \oplus\{won : \text{monalisa} \otimes \text{\textlangle 0 \rangle auction}, lost : \text{money} \otimes \text{\textlangle 0 \rangle auction}\}\}}} \]
Resource-Aware Auction

cost of bidding = 2, cost of collecting = 3

Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost

\[ \text{type } \text{auction} = \circ^3 \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \triangleright^1 \text{auction} \}, \text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \triangleright^0 \text{auction}, \text{lost} : \text{money} \otimes \triangleright^0 \text{auction} \} \} \}

user pays 3 units as fees at start of execution
Resource-Aware Auction

cost of bidding = 2, cost of collecting = 3

**Key Idea:** Nomos enhances session types with potential (or fee) that pays for execution cost

\[
\text{type } \text{auction} = \delta^3 \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \triangleright^1 \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \triangleright^0 \text{auction}, \\
\text{lost} : \text{money} \otimes \triangleright^0 \text{auction} \} \} \}
\]

*user pays 3 units as fees at start of execution*

*return leftover fee back to user in the end*
Resource-Aware Auction

cost of bidding = 2, cost of collecting = 3

Key Idea: Nomos enhances session types with potential (or fee) that pays for execution cost

\[
\text{type auction} = \otimes^3 \oplus \{ \text{running} : \& \{ \text{bid} : \text{money} \rightarrow \triangleright^1 \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \oplus \{ \text{won} : \text{monalisa} \otimes \triangleright^0 \text{auction}, \\
\text{lost} : \text{money} \otimes \triangleright^0 \text{auction} \} \} \} 
\]

user pays 3 units as fees at start of execution

return leftover fee back to user in the end

Potential annotations are automatically inferred using LP solver
Conclusion: Features of Nomos

- session types enforce contract protocols
- linear types track assets, prevent duplication/deletion
- resource-aware types infer execution cost
- re-entrancy attacks eliminated by construction
- type safety theorem guarantees all properties
- extensive evaluation on standard digital contracts