Nomos: Resource-Aware Session Types for Programming Digital Contracts

Stephanie Balzer, Ankush Das, Jan Hoffmann, and Frank Pfenning

With some slides from Ankush.
Digital Contracts (or Smart Contracts)

**Smart contracts (Ethereum): programs stored on a blockchain**

- Carry out (financial) transactions between (untrusted) agents
- Cannot be modified but have state
  - Community needs to reach consensus on the result of execution
  - Users need to pay for the execution cost upfront
Digital Contracts (or Smart Contracts)

Smart contracts (Ethereum): programs stored on a blockchain

• Carry out (financial) transactions between (untrusted) agents

• Cannot be modified but have state

  • Community needs to reach consensus on the result of execution

  • Users need to pay for the execution cost upfront
Bugs in Digital Contracts are Expensive

- Bugs result in financial disasters (DAO, Parity Wallet, King of Ether, …)
- Bugs are difficult to fix because they alter the contract
Can Programming Languages Prevent Bugs?
Can Programming Languages Prevent Bugs? 

Yes!
Can Programming Languages Prevent Bugs?

Example: memory safety

• Most security vulnerabilities are based on memory safety issues (Microsoft: 70% over past 12 years in MS products)

• Why stick with unsafe languages? Legacy code, developers (training, social factors, …)
Can Programming Languages Prevent Bugs?

Example: memory safety

• Most security vulnerabilities are based on memory safety issues (Microsoft: 70% over past 12 years in MS products)

• Why stick with unsafe languages?
  Legacy code, developers (training, social factors, …)

Languages for Digital Contracts

• Great opportunity to start from a clean slate

• Correctness and readability of contracts are priorities
Can Programming Languages Prevent Bugs?

Example: memory safety

• Most security vulnerabilities are based on memory safety issues (Microsoft: 70% over past in the past 12 years in MS products)

• Why stick with unsafe languages?
  Legacy code, developers (training, social factors, …)

Languages for Digital Contracts

• Great opportunity to start from a clean slate

• Correctness and readability of contracts are priorities

Nomos

• Build on state-of-the-art: statically-typed, strict, functional language

• Address domain-specific issues
Domain Specific Bugs: Auction Contract

status: running
Domain Specific Bugs: Auction Contract

Bidder 1

Bid 1

Bidder 2

Bid 2

Bidder 3

Bid 3

status: running
Domain Specific Bugs: Auction Contract

Bidder 1

Bidder 2

Bidder 3

status: running

Bid 1
Bid 2
Bid 3
Domain Specific Bugs: Auction Contract

Bidder 1
Bidder 2
Bidder 3

Bid 1
Bid 2
Bid 3

status: ended
Domain Specific Bugs: Auction Contract

Bidder 1

Bidder 2

Bidder 3

Bid 1

Bid 2

Bid 3

status: ended
Domain Specific Bugs: Auction Contract

Bidder 1
Bid 1

Bidder 2
Bid 2

Bidder 3
Bid 3

status: ended
Auction Contract in Solidity

```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;
}
```
Auction in Solidity

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

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

```solidity
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?
Auction in Solidity

```solidity
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
add require (status == ended);
```
Auction in Solidity

What happens if collect is called when auction is running?

Protocol is not statically enforced!

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

add require (status == ended);
Auction in Solidity

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

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

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

What happens if collect is called twice?
Auction in Solidity

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

What happens if `collect` is called twice?

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

What happens if collect is called twice?

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

Linearity is not enforced!

set pendingReturns[msg.sender] = 0
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

Method ‘send’ potentially transfers control to other contract.
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

Method ‘send’ potentially transfers control to other contract.

‘send’ should be the last instruction.
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    msg.sender.send(amount);
    pendingReturns[msg.sender] = 0;
    return true;
}
```

Method ‘send’ potentially transfers control to other contract.

‘send’ should be the last instruction.

Re-entrancy attack
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    pendingReturns[msg.sender] = 0;
    msg.sender.send(amount);
    return true;
}
```
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    pendingReturns[msg.sender] = 0;
    msg.sender.send(amount);
    return true;
}
```
Auction in Solidity

```
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    pendingReturns[msg.sender] = 0;
    msg.sender.send(amount);
    return true;
}
```

Method ‘send’ potentially transfers control to other contract.
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    pendingReturns[msg.sender] = 0;
    msg.sender.send(amount);
    return true;
}
```

Method ‘send’ potentially transfers control to other contract.

Need to check return value of ‘send’.
Auction in Solidity

```solidity
function collect() public returns (bool) {
    require (msg.sender != highestBidder);
    require (status == ended);
    uint amount = pendingReturns[msg.sender];
    pendingReturns[msg.sender] = 0;
    msg.sender.send(amount);
    return true;
}
```

Method ‘send’ potentially transfers control to other contract.

Out-of-gas exception.

Need to check return value of ‘send’.
Bugs in Digital Contracts are Expensive

- Bugs result in financial disasters (DAO, Parity Wallet, King of Ether, …)
- Bugs are difficult to fix because they alter the contract

A $50 million hack just showed that the DAO was all too human

'$300m in cryptocurrency' accidentally lost forever due to bug

A coding error led to $30 million in Ethereum being stolen
Parity Wallet: Unintended Interaction

**Bug: Initialization function could be called by unauthorized user**

- Ownership of the wallet could be changed after initialization
- Funds can be extracted by owner
- Damage: ~$280 million

**Problem: Interaction protocol with contracts**

- Protocols of interaction not explicit in the language
- Protocols of interaction not enforced
The DAO: Reentrancy

**Bug: Function can be called again during its ongoing execution**

- A money transfer to a contract triggers a function call
- The called function can call the function that initiated the transfer

⇒ DAO: money gets transferred without updating the balance

**Problem 1: Unnecessary permissiveness in communication**

- Violation of function invariants

**Problem 2: Failure to keep track of assets**

- Incorrect book-keeping of stored funds (returned payment)
Domain-Specific Issues with Digital Contracts

1. **Resource consumption (gas cost)**
   - Participants have to agree on the result of a computation
     ➔ Denial of service attacks
     ➔ Would like to have static gas bounds
Domain-Specific Issues with Digital Contracts

1. **Resource consumption (gas cost)**
   - Participants have to agree on the result of a computation
     ➡ Denial of service attacks
     ➡ Would like to have static gas bounds

2. **Contract protocols and interfaces**
   - Contract protocols should be described and enforced
     ➡ Prevent issues like reentrancy bugs (DAO)
Domain-Specific Issues with Digital Contracts

1. Resource consumption (gas cost)
   - Participants have to agree on the result of a computation
     ➤ Denial of service attacks
     ➤ Would like to have static gas bounds

2. Contract protocols and interfaces
   - Contract protocols should be described and enforced
     ➤ Prevent issues like reentrancy bugs (DAO)

3. Keeping track of assets (crypto coins)
   - Assets should not be duplicated
   - Assets should not be lost
Nomos: A Type-Based Approach

A statically-typed, strict, functional language

- Functional fragment of ML

Additional features for domain-specific requirements

<table>
<thead>
<tr>
<th>Language feature</th>
<th>Expertise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gas bounds</td>
<td>Automatic amortized resource analysis</td>
</tr>
<tr>
<td>Tracking assets</td>
<td>Linear type system</td>
</tr>
<tr>
<td>Contract interfaces</td>
<td>Shared binary session types</td>
</tr>
</tbody>
</table>
Nomos: A Type-Based Approach

A statically-typed, strict, functional language

- Functional fragment of ML

Additional features for domain-specific requirements

<table>
<thead>
<tr>
<th>Language feature</th>
<th>Expertise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gas bounds</td>
<td>Automatic amortized resource analysis</td>
</tr>
<tr>
<td>Tracking assets</td>
<td>Linear type system</td>
</tr>
<tr>
<td>Contract interfaces</td>
<td>Shared binary session types</td>
</tr>
</tbody>
</table>

Based on a linear type system
1. Automatic amortized resource analysis (AARA)
Resource Bound Analysis

Given: A (functional) program P

Question: What is the (worst-case) resource consumption of P as a function of the size of its inputs?
Resource Bound Analysis

Given: A (functional) program P

Question: What is the (worst-case) resource consumption of P as a function of the size of its inputs?

Clock cycles, heap space, gas, ...
Resource Bound Analysis

Given: A (functional) program P

Question: What is the (worst-case) resource consumption of P as a function of the size of its inputs?

Clock cycles, heap space, gas, ...

Not only asymptotic bounds but concrete constant factors.
Resource Bound Analysis

Given: A (functional) program P

Question: What is the (worst-case) resource consumption of P as a function of the size of its inputs?

Clock cycles, heap space, gas, ...

Not only asymptotic bounds but concrete constant factors.
Given: A (functional) program P

Question: What is the (worst-case) resource consumption of P as a function of the size of its inputs?

Clock cycles, heap space, gas, ...

Not only asymptotic bounds but concrete constant factors.

Goal: produce proofs (easily checkable)
AARA: Use Potential Method

• Assign potential functions to data structures
  ➡ States are mapped to non-negative numbers

• Potential pays the resource consumption and the potential at the following program point

• Initial potential is an upper bound
AARA: Use Potential Method

- Assign potential functions to data structures
  - States are mapped to non-negative numbers
  \[ \Phi(state) \geq 0 \]

- Potential pays the resource consumption and the potential at the following program point
  \[ \Phi(before) \geq \Phi(after) + cost \]
  \[ \downarrow \text{telescoping} \]
  \[ \Phi(initial\ state) \geq \sum cost \]

Type systems for automatic analysis

- Fix a format of potential functions (basis like in linear algebra)
- Type rules introduce linear constraint on coefficients
AARA: Use Potential Method

- Assign potential functions to data structures
  - States are mapped to non-negative numbers

- Potential pays the resource consumption and the potential at the following program point

- Initial potential is an upper bound

Type systems for automatic analysis

- Fix a format of potential functions (basis like in linear algebra)

- Type rules introduce linear constraint on coefficients
Example: Append for Persistent Lists

append(x, y)

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)
Example: Append for Persistent Lists

```plaintext
append(x, y)
```

Heap-space usage is $2n$ if

- $n$ is the length of list $x$
- One list element requires two heap cells (data and pointer)

Example evaluation:
Example: Append for Persistent Lists

\[ \text{append}(x, y) \]

Heap-space usage is \( 2n \) if
- \( n \) is the length of list \( x \)
- One list element requires two heap cells (data and pointer)

Example evaluation:

\[ x \rightarrow a \rightarrow b \rightarrow c \]
\[ y \rightarrow d \rightarrow e \]
Example: Append for Persistent Lists

\[ \text{append}(x, y) \]

Heap-space usage is \( 2n \) if

- \( n \) is the length of list \( x \)
- One list element requires two heap cells (data and pointer)

Example evaluation:

- \( x \rightarrow a \rightarrow b \rightarrow c \)
- \( y \rightarrow d \rightarrow c \rightarrow e \)
Example: Append for Persistent Lists

**append(x, y)**

Heap-space usage is $2n$ if

- $n$ is the length of list $x$
- One list element requires two heap cells (data and pointer)

**Example evaluation:**

$x \rightarrow a \rightarrow b \rightarrow c$

$y \rightarrow d \rightarrow c \rightarrow b \rightarrow e$
Example: Append for Persistent Lists

append(x, y)

Heap-space usage is $2n$ if

- $n$ is the length of list $x$
- One list element requires two heap cells (data and pointer)

Example evaluation:

```
x -> a -> b -> c
```
```
y -> d -> c -> b -> a
```
```
```
Example: Append for Persistent Lists

\[ \text{append}(x, y) \]

Heap-space usage is \( 2n \) if

- \( n \) is the length of list \( x \)
- One list element requires two heap cells (data and pointer)

Example evaluation:
Example: Append for Persistent Lists

```
append(x, y)
```

Heap-space usage is $2n$ if

- $n$ is the length of list $x$
- One list element requires two heap cells (data and pointer)

Example evaluation:

Heap usage: $2n = 2 \times 3 = 6$
Example: Composing Calls of Append

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

\[
f(x,y,z) =
\begin{align*}
& \text{let } t = \text{append}(x,y) \text{ in} \\
& \text{append}(t,z)
\end{align*}
\]
Example: Composing Calls of Append

\[ f(x,y,z) = \]
\[
\text{let } t = \text{append}(x,y) \text{ in } \text{append}(t,z) \]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

\[
f(x, y, z) = \begin{align*}
&\text{let } t = \text{append}(x, y) \text{ in} \\
&\text{append}(t, z)
\end{align*}
\]

Heap usage of \( f(x, y, z) \) is \( 2n + 2(n+m) \) if
\[\begin{align*}
\text{‣ } &n \text{ is the length of list } x \\
\text{‣ } &m \text{ is the length of list } y
\end{align*}\]

Initial potential: \( 4n + 2m = 4 \cdot 3 + 2 \cdot 2 = 16 \)
Example: Composing Calls of Append

\[
\begin{align*}
f(x,y,z) &= \\
&\text{let } t = \text{append}(x,y) \text{ in} \\
&\text{append}(t,z)
\end{align*}
\]

Heap usage of \(f(x,y,z)\) is \(2n + 2(n+m)\) if

- \(n\) is the length of list \(x\)
- \(m\) is the length of list \(y\)

Initial potential: \(4*n + 2*m = 4*3 + 2*2 = 16\)
Example: Composing Calls of Append

\[
\begin{align*}
\text{f(x,y,z)} &= \\
&= \text{let } t = \text{append}(x,y) \text{ in } \text{append}(t,z)
\end{align*}
\]

Heap usage of f(x,y,z) is \(2n + 2(n+m)\) if

- \(n\) is the length of list \(x\)
- \(m\) is the length of list \(y\)

Initial potential: \(4n + 2m = 4 \times 3 + 2 \times 2 = 16\)
Example: Composing Calls of Append

\[ f(x,y,z) = \]
\[ \text{let } t = \text{append}(x,y) \text{ in } \text{append}(t,z) \]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4\times3 + 2\times2 = 16 \)
Example: Composing Calls of Append

\[ f(x,y,z) = \]
\[ \text{let } t = \text{append}(x,y) \text{ in } \text{append}(t,z) \]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if
- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

f(x,y,z) = 
let t = append(x,y) in
append(t,z)

Heap usage of f(x,y,z) is \(2n + 2(n+m)\) if

- \(n\) is the length of list \(x\)
- \(m\) is the length of list \(y\)

Initial potential: \(4n + 2m = 4 \times 3 + 2 \times 2 = 16\)
Example: Composing Calls of Append

\[ f(x,y,z) = \]
\[ \text{let } t = \text{append}(x,y) \text{ in} \]
\[ \text{append}(t,z) \]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

\- \( n \) is the length of list \( x \)
\- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

\[ f(x,y,z) = \]
\[ \text{let } t = \text{append}(x,y) \text{ in} \]
\[ \text{append}(t,z) \]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

\[ f(x,y,z) = \]
\[ \text{let } t = \text{append}(x,y) \text{ in} \]
\[ \text{append}(t,z) \]

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

Heapp usage of $f(x,y,z)$ is $2n + 2(n+m)$ if

- $n$ is the length of list $x$
- $m$ is the length of list $y$

$$f(x,y,z) = \text{let } t = \text{append}(x,y) \text{ in } \text{append}(t,z)$$

Initial potential: $4n + 2m = 4*3 + 2*2 = 16$
Example: Composing Calls of Append

\[
f(x,y,z) = \begin{array}{l}
\text{let } t = \text{append}(x,y) \text{ in } \\
\text{append}(t,z)
\end{array}
\]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

\[
f(x,y,z) = \begin{align*}
\text{let } t &= \text{append}(x,y) \text{ in} \\
\text{append}(t,z)
\end{align*}
\]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if:

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

\[
f(x,y,z) = \begin{align*}
\text{let } t &= \text{append}(x,y) \text{ in} \\
\text{append}(t,z)
\end{align*}
\]

Heap usage of \( f(x,y,z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

\[
f(x, y, z) = \begin{align*}
    & \text{let } t = \text{append}(x, y) \text{ in} \\
    & \text{append}(t, z)
\end{align*}
\]

Heap usage of \( f(x, y, z) \) is \( 2n + 2(n+m) \) if

- \( n \) is the length of list \( x \)
- \( m \) is the length of list \( y \)

Initial potential: \( 4n + 2m = 4 \times 3 + 2 \times 2 = 16 \)
Example: Composing Calls of Append

\[ f(x, y, z) = \{ \text{let } t = \text{append}(x, y) \text{ in} \]
\[ \text{append}(t, z) \}

The most general type of append is specialized at call-sites:

\[ \text{append}: (L^q (\text{int}), L^p (\text{int})) \rightarrow L^r (\text{int}) \mid \phi \]

Linear constraints.
<table>
<thead>
<tr>
<th>Linear Potential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined <strong>resource metrics</strong> (i.e., by <code>tick(q)</code> in the code)</td>
</tr>
<tr>
<td>Naturally <strong>compositional</strong>: tracks size changes, types are specifications</td>
</tr>
<tr>
<td>Bound inference by reduction to efficient <strong>LP solving</strong></td>
</tr>
<tr>
<td><strong>Type derivations prove bounds</strong> with respect to the cost semantics</td>
</tr>
</tbody>
</table>

**Polynomial Potential Functions**
### Linear Potential Functions

<table>
<thead>
<tr>
<th>Feature</th>
<th>✔️</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined <strong>resource metrics</strong> (i.e., by tick(q) in the code)</td>
<td></td>
</tr>
<tr>
<td>Naturally <strong>compositional</strong>: tracks size changes, types are specifications</td>
<td></td>
</tr>
<tr>
<td>Bound inference by reduction to efficient <strong>LP solving</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Type derivations prove bounds</strong> with respect to the cost semantics</td>
<td></td>
</tr>
</tbody>
</table>

**Strong soundness theorem.**
<table>
<thead>
<tr>
<th>User-defined <strong>resource metrics</strong> (i.e., by tick(q) in the code)</th>
<th>Linear Potential Functions</th>
<th>Multivariate Polynomial Potential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naturally <strong>compositional</strong>: tracks size changes, types are specifications</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bound inference by reduction to efficient <strong>LP solving</strong></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Type derivations prove bounds</strong> with respect to the cost semantics</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Strong soundness theorem.**

**Polynomial Potential Functions**
### Polynomial Potential Functions

User-defined **resource metrics** (i.e., by tick(q) in the code)  

<table>
<thead>
<tr>
<th>Linear Potential Functions</th>
<th>Multivariate Polynomial Potential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
</tr>
</tbody>
</table>

Naturally **compositional**: tracks size changes, types are specifications  

<table>
<thead>
<tr>
<th>Linear Potential Functions</th>
<th>Multivariate Polynomial Potential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
</tr>
</tbody>
</table>

Bound inference by reduction to efficient **LP solving**  

<table>
<thead>
<tr>
<th>Linear Potential Functions</th>
<th>Multivariate Polynomial Potential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
</tr>
</tbody>
</table>

**Type derivations prove bounds** with respect to the cost semantics  

<table>
<thead>
<tr>
<th>Linear Potential Functions</th>
<th>Multivariate Polynomial Potential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
<td><img src="https://example.com/yes-icon.png" alt="Yes" /></td>
</tr>
</tbody>
</table>

**Strong soundness theorem.**

**For example m*n^2.**
Implementations: RaML and Absynth

Resource Aware ML (RaML)

- Based on Inria’s OCaml compiler
- Polymorphic and higher-order functions
- User-defined data types
- Side effects (arrays and references)

Absynth

- Based on control-flow graph IR
- Different front ends
- Bounds are integer expressions
- Supports probabilistic programs

http://raml.co
<table>
<thead>
<tr>
<th>Function</th>
<th>Computed Bound</th>
<th>Actual Behavior</th>
<th>Analysis Runtime</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting A-nodes (asort)</td>
<td>$11+22kn +13k^2nv+13m +15n$</td>
<td>$O(k^2n+m)$</td>
<td>0.14 s</td>
<td>5656</td>
</tr>
<tr>
<td>Quick sort (lists of lists)</td>
<td>$3 -7.5nm +7.5n^2 +19.5m +16.5m^2$</td>
<td>$O(nm^2)$</td>
<td>0.27 s</td>
<td>8712</td>
</tr>
<tr>
<td>Merge sort (list.ml)</td>
<td>$43 + 30.5n + 8.5n^2$</td>
<td>$O(n \log n)$</td>
<td>0.11 s</td>
<td>3066</td>
</tr>
<tr>
<td>Split and sort</td>
<td>$11 + 47n + 29n^2$</td>
<td>$O(n^2)$</td>
<td>0.69 s</td>
<td>3793</td>
</tr>
<tr>
<td>Longest common subsequence</td>
<td>$23 + 10n + 52nm + 25m$</td>
<td>$O(nm)$</td>
<td>0.16 s</td>
<td>901</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>$3 + 2nm +18m + 22mxy +16my$</td>
<td>$O(mxy)$</td>
<td>1.11 s</td>
<td>3901</td>
</tr>
<tr>
<td>Evaluator for boolean expressions (tutorial)</td>
<td>$10+11n+16m+16mx+16my+20x+20y$</td>
<td>$O(mx+my)$</td>
<td>0.33 s</td>
<td>1864</td>
</tr>
<tr>
<td>Dijkstra’s shortest-path algorithm</td>
<td>$46 + 33n +111n^2$</td>
<td>$O(n^2)$</td>
<td>0.11 s</td>
<td>2808</td>
</tr>
<tr>
<td>Echelon form</td>
<td>$8 + 43m^2n + 59m + 63m^2$</td>
<td>$O(nm^2)$</td>
<td>1.81 s</td>
<td>8838</td>
</tr>
<tr>
<td>Binary multiplication (CompCert)</td>
<td>$2+17kr+10ks+25k +8l+2+7r+8$</td>
<td>$O(kr+ks)$</td>
<td>14.04 s</td>
<td>89,507</td>
</tr>
<tr>
<td>Square root (CompCert)</td>
<td>$13+66m+16mn +4m^2 +59n +4n^2$</td>
<td>$O(n^2)$</td>
<td>18.25 s</td>
<td>135,529</td>
</tr>
</tbody>
</table>
Quick Sort for Integers

Evaluation-step bound vs.
measured behavior

12x^2 + 14x + 3
measured worst-case cost
Longest Common Subsequence

Evaluation-step bound vs. measured behavior

measured worst-case steps
39xy + 6y + 21x + 19
Longest Common Subsequence

Evaluation-step bound vs. measured behavior

First automatically derived bound for LCS.
Automatic Amortized Resource Analysis (AARA)

Type system for deriving symbolic resource bounds

- **Compositional:** Integrated with type systems or program logics
- **Expressive:** Bounds are multivariate resource polynomials
- **Reliable:** Formal soundness proof wrt. cost semantics
- **Verifiable:** Produces easily-checkable certificates
- **Automatic:** No user interaction required

Applicable in practice

- **Implemented:** Resource Aware ML and Absynth
- **Effective:** Works for many typical programs
- **Efficient:** Inference via linear programming
Automatic Amortized Resource Analysis (AARA)

Type system for deriving symbolic resource bounds

- **Compositional**: Integrated with type systems or program logics
- **Expressive**: Bounds are multivariate resource polynomials
- **Reliable**: Formal soundness proof wrt. cost semantics
- **Verifiable**: Produces easily-checkable certificates
- **Automatic**: No user interaction required

Applicable in practice

- **Implemented**: Resource Aware ML and Absynth
- **Effective**: Works for many typical programs
- **Efficient**: Inference via linear programming

Type checking in linear time!
2. Shared (resource-aware) binary session types
Binary Session Types

• Implement message-passing concurrent programs
• Communication via typed bidirectional channels
• Curry-Howard correspondence with intuitionistic linear logic
• Client and provider have dual types

*Example type:*

\[
\text{queue}_A = &\{\text{ins} : A \multimap \text{queue}_A, \\
\text{del} : \oplus\{\text{none} : 1, \\
\text{some} : A \otimes \text{queue}_A\}\}
\]
Binary Session Types

- Implement message-passing concurrent programs
- Communication via typed bidirectional channels
- Curry-Howard correspondence with intuitionistic linear logic
- Client and provider have dual types

**Example type:**
\[
\text{queue}_A = \& \{ \text{ins} : A \rightarrow \text{queue}_A, \\
\text{del} : \oplus \{ \text{none} : 1, \\
\text{some} : A \otimes \text{queue}_A \} \}
\]
Binary Session Types

- Implement message-passing concurrent programs
- Communication via typed bidirectional channels
- Curry-Howard correspondence with intuitionistic linear logic
- Client and provider have dual types

**Example type:**

\[
\text{queue}_A = \& \{ \text{ins} : A \rightarrow \text{queue}_A, \\
\text{del} : \oplus \{ \text{none} : 1, \\
\text{some} : A \otimes \text{queue}_A \} \}
\]
Binary Session Types

- Implement message-passing concurrent programs
- Communication via typed bidirectional channels
- Curry-Howard correspondence with intuitionistic linear logic
- Client and provider have dual types

**Example type:**

\[
\text{queue}_A = \& \{ \text{ins} : A \to \text{queue}_A, \\
\text{del} : \oplus \{ \text{none} : 1, \\
\text{some} : A \otimes \text{queue}_A \} \}
\]

Type soundness (progress and preservation) implies deadlock freedom
Example: Queue

\[ \text{queue}_A = \& \{ \text{ins} : A \rightarrow \text{queue}_A, \]
\[ \text{del} : \oplus \{ \text{none} : 1, \]
\[ \text{some} : A \otimes \text{queue}_A \} \} \]

\begin{align*}
(x : A) \ (t : \text{queue}_A) & \vdash \text{elem} :: (s : \text{queue}_A) \\
\text{s} \leftarrow \text{elem} \leftarrow x \ t = \\
\text{case} \ s \ (\text{ins} \Rightarrow y \leftarrow \text{recv} \ s ; \\
& t.\text{ins} ; \\
& \text{send} \ t \ y ; \\
& s \leftarrow \text{elem} \leftarrow x \ t \\
| \text{del} \Rightarrow s.\text{some} ; \\
& \text{send} \ s \ x ; \\
& s \leftarrow t) \end{align*}
Example: Queue

\[(x : A) (t : \text{queue}_A) \vdash \text{elem} :: (s : \text{queue}_A)\]

\[
s \leftarrow \text{elem} \leftarrow x t =
\]

\[
\begin{cases}
\text{case } s (\text{ins } \Rightarrow y \leftarrow \text{recv } s ; \\
\quad t.\text{ins} ; \\
\quad \text{send } t y ; \\
\quad s \leftarrow \text{elem} \leftarrow x t \\
\mid \text{del } \Rightarrow s.\text{some} ; \\
\quad \text{send } s x ; \\
\quad s \leftarrow t
\end{cases}
\]

\[
\text{queue}_A = \&\{\text{ins} : A \rightarrow \text{queue}_A, \\
\text{del} : \oplus\{\text{none} : 1, \\
\text{some} : A \otimes \text{queue}_A\}\}
\]

recv ‘ins’ and y
Example: Queue

\[(x : A) (t : \text{queue}_A) \vdash \text{elem} :: (s : \text{queue}_A)\]

\[
\begin{align*}
    s & \leftarrow \text{elem} \leftarrow x t = \\
    \text{case } s \left(\text{ins} \Rightarrow y & \leftarrow \text{recv } s ; \\
    t.\text{ins} ; \\
    \text{send } t y ; \\
    s & \leftarrow \text{elem} \leftarrow x t \\
    | \text{del} \Rightarrow s.\text{some} ; \\
    \text{send } s x ; \\
    s & \leftarrow t
\end{align*}
\]

\[
\text{queue}_A = \&\{\text{ins} : A \rightarrow \text{queue}_A, \\
    \text{del} : \oplus\{\text{none} : 1, \\
    \text{some} : A \otimes \text{queue}_A\}\}
\]
Example: Queue

\[ (x : A) \ (t : \text{queue}_A) \vdash \text{elem} :: (s : \text{queue}_A) \]

\[
s \leftarrow \text{elem} \leftarrow x \ t =
\]

\[
\begin{cases}
\text{case } s \ (\text{ins } \Rightarrow y \leftarrow \text{recv } s \ ; \\
\quad t.\text{ins} ; \\
\quad \text{send } t \ y \ ; \\
\quad s \leftarrow \text{elem} \leftarrow x \ t \\
\mid \text{del } \Rightarrow s.\text{some} \ ; \\
\quad \text{send } s \ x \ ; \\
\quad s \leftarrow t)
\end{cases}
\]

\[ \text{queue}_A = \&\{\text{ins} : A \rightarrow \text{queue}_A, \]
\[ \quad \text{del} : \oplus\{\text{none} : 1, \]
\[ \quad \text{some} : A \otimes \text{queue}_A\} \} \]
Example: Queue

\[
\begin{align*}
\text{queue}_A &= \&\{ \text{ins} : A \to \text{queue}_A, \\
& \quad \text{del} : \oplus\{ \text{none} : 1, \\
& \quad \text{some} : A \otimes \text{queue}_A \} \}\}
\end{align*}
\]

\[
(x : A) \ (t : \text{queue}_A) \vdash \text{elem} :: (s : \text{queue}_A) \\
\begin{align*}
& \quad s \leftarrow \text{elem} \leftarrow x \ t = \\
& \quad \text{case } s \ (\text{ins } \Rightarrow y \leftarrow \text{recv } s ; \\
& \quad \quad t.\text{ins} ; \\
& \quad \quad \text{send } t \ y ; \\
& \quad \quad s \leftarrow \text{elem} \leftarrow x \ t \\
& \quad | \text{del } \Rightarrow s.\text{some} ; \\
& \quad \quad \text{send } s \ x ; \\
& \quad \quad s \leftarrow t)
\end{align*}
\]
Example: Queue

\[
\text{queue}_A = \&\{\text{ins} : A \rightarrow \text{queue}_A, \text{del} : \oplus\{\text{none} : 1, \text{some} : A \otimes \text{queue}_A\}\}
\]

\[
(x : A) \ (t : \text{queue}_A) \vdash \text{elem} :: (s : \text{queue}_A)
\]

\[
s \leftarrow \text{elem} \leftarrow x \ t =
\]

\[
\begin{align*}
\text{case } s & (\text{ins} \Rightarrow y \leftarrow \text{recv } s \ ; \\
& t.\text{ins} \ ; \\
& \text{send } t \ y \ ; \\
& s \leftarrow \text{elem} \leftarrow x \ t \\
\mid \text{del} & \Rightarrow s.\text{some} \ ; \\
& \text{send } s \ x \ ; \\
& s \leftarrow t
\end{align*}
\]

- recv ‘ins’ and y
- send ‘ins’ and y
- recurse
- send ‘some’, x
- terminate
Example: Queue

queue_A = \&\{\text{ins} : A \rightarrow queue_A, \\
\text{del} : \oplus\{\text{none} : 1, \\
\text{some} : A \otimes queue_A\}\}

\[ s \leftarrow elem \leftarrow x \leftarrow t = \]

\text{case } s \ (\text{ins } \Rightarrow y \leftarrow \text{recv } s ; \\
t.\text{ins} ; \\
\text{send } t \ y ; \\
s \leftarrow elem \leftarrow x \ t \]

| \text{del } \Rightarrow s.\text{some} ; \\
\text{send } s \ x ; \\
s \leftarrow t \)

recv 'ins' and y
send 'ins' and y
recurse
send 'some', x
terminate
Example: Auction

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

\[
\text{auction} = \bigoplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \text{auction} \}, \\
\quad \text{ended} : \& \{ \text{collect} : \text{id} \supset \bigoplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\quad \text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Example: Auction

\[
\text{auction} = \oplus \{ \text{running} : \land \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \land \{ \text{collect} : \text{id} \supset \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Example: Auction

\[
\text{auction} = \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \text{id} \supset \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Example: Auction

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

- sends status of auction
- offers choice of bidding
- receive id and money
- recurse
Example: Auction

\[
auction = \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \text{auction} \},
\text{ended} : \& \{ \text{collect} : \text{id} \supset \oplus \{ \text{won} : \text{monalisa} \times \text{auction},
\text{lost} : \text{money} \times \text{auction} \} \} \}
\]

- sends status of auction
- offers choice of bidding
- receive id and money
- recurse
- offers choice to collect
Example: Auction

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

\[
\text{auction} = \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \text{id} \supset \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Example: Auction

\[
auction = \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \text{id} \supset \oplus \{ \text{won} : \text{monalisa} \otimes \text{auction}, \\
\text{lost} : \text{money} \otimes \text{auction} \} \} \}
\]
Type Rules

\[
\frac{(k \in L) \quad \Omega \vdash P :: (x : A_k)}{\Omega \vdash (x.k ; P) :: (x : \bigoplus_{\ell \in L} A_{\ell})} \quad \bigoplus R
\]

\[
\frac{(\forall \ell \in L) \quad \Omega, x:A_{\ell} \vdash Q_{\ell} :: (z : C)}{\Omega, x:\bigoplus_{\ell \in L} A_{\ell} \vdash \text{case } x (\ell \Rightarrow Q_{\ell})_{\ell \in L} :: (z : C)} \quad \bigoplus L
\]

\[
\frac{(\forall \ell \in L) \quad \Omega \vdash P_{\ell} :: (x : A_{\ell})}{\Omega \vdash \text{case } x (\ell \Rightarrow P_{\ell})_{\ell \in L} :: (x : \bigland_{\ell \in L} A_{\ell})} \quad \land R
\]

\[
\frac{\Omega, x:A_k \vdash Q :: (z : C)}{\Omega, x:\bigland_{\ell \in L} A_{\ell} \vdash (x.k ; Q) :: (z : C)} \quad \land L
\]
Resource-Aware Session Types

- Each process stores potential in functional data
- Potential can be transferred via messages
- Potential is used to pay for performed work
Resource-Aware Session Types

• Each *process stores potential* in functional data
• Potential can be *transferred via messages*
• Potential is used to *pay for performed work*

Potential transfer only at the type level, not at runtime.
Resource-Aware Session Types

- Each **process stores potential** in functional data.
- Potential can be **transferred via messages**.
- Potential is used to **pay for performed work**.

Potential transfer only at the type level, not at runtime.

User-defined cost metric.
Resource-Aware Session Types

- Each **process stores potential** in functional data
- Potential can be **transferred via messages**
- Potential is used to **pay for performed work**
- Message potential is a function of (functional) payload

\[
A, B, C ::= \tau \uparrow A \\
\tau \land A \quad \text{input value of type } \tau \text{ and continue as } A \\
\text{L}^2(\text{int}) \quad \text{output value of type } \tau \text{ and continue as } A
\]
Resource-Aware Session Types

- Each **process stores potential** in functional data
- Potential can be **transferred via messages**
- Potential is used to **pay for performed work**

- Message potential is a function of (functional) payload

\[
A, B, C ::= \tau \supset A \quad \tau \land A
\]

- Syntactic sugar (no payload)

\[
A ::= \ldots \mid \triangleright^r A \mid \triangleleft^r A
\]

- Only in intermediate language:

\[
\text{get } x_m \{r\} ; P \\
\text{pay } x_m \{r\} ; P
\]
Resource-Aware Session Types

- Each **process stores potential** in functional data
- Potential can be **transferred via messages**
- Potential is used to **pay for performed work**

- Message potential is a function of (functional) payload

\[ A, B, C ::= \tau \triangleright A \quad \tau \wedge A \]

- Syntactic sugar (no payload)

\[ A ::= \ldots | \triangleright A | \triangleleft A \]

- Only in intermediate language:

\[ \text{get } x_m \{ r \} ; P \quad \text{pay } x_m \{ r \} ; P \]
Example: Type of an Auction Contract

\[
\text{auction} = \uparrow_{\mathcal{L}}^\mathcal{S} \oplus \{ \text{running} : \&\{ \text{bid} : \text{id} \supset \text{money} \rightarrow \uparrow^1 \downarrow_{\mathcal{L}}^\mathcal{S} \text{auction}, \\
\text{cancel} : \uparrow^8 \downarrow_{\mathcal{L}}^\mathcal{S} \text{auction} \}, \\
\text{ended} : \&\{ \text{collect} : \text{id} \supset \\
\oplus\{ \text{won} : \text{lot} \otimes \uparrow^3 \downarrow_{\mathcal{L}}^\mathcal{S} \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow_{\mathcal{L}}^\mathcal{S} \text{auction} \}, \\
\text{cancel} : \uparrow^8 \downarrow_{\mathcal{L}}^\mathcal{S} \text{auction} \} \}
\]
Sharing: Need to acquire contract before use.

\[
auction = \uparrow^S_L \downarrow^{11}_L \oplus \{running : \&\{bid : id \supset money \rightarrow \uparrow^1 \downarrow^S_L auction,
         cancel : \uparrow^8 \downarrow^S_L auction\},
\]
\[
   ended : \&\{collect : id \supset
         \oplus\{won : lot \otimes \uparrow^3 \downarrow^S_L auction,
               lost : money\otimes \downarrow^S_L auction\},
         cancel : \uparrow^8 \downarrow^S_L auction\}\}
\]

Example: Type of an Auction Contract
Sharing: Need to acquire contract before use.

\[ \text{auction} = \uparrow^S_L \downarrow^{11} \oplus \{ \text{running} : \&\{ \text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow^1_L \uparrow^S_L \text{auction}, \text{cancel} : \downarrow^8_L \uparrow^S_L \text{auction} \}, \text{ended} : \&\{ \text{collect} : \text{id} \supset \oplus \{ \text{won} : \text{lot} \otimes \downarrow^3_L \uparrow^S_L \text{auction}, \text{lost} : \text{money} \otimes \downarrow^S_L \uparrow^S_L \text{auction} \}, \text{cancel} : \downarrow^8_L \uparrow^S_L \text{auction} \} \} \]

Equi-synchronizing: Release contract at the same type.

Example: Type of an Auction Contract
Example: Type of an Auction Contract

\[
auction = \uparrow^S_L \downarrow^{11}_L \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \uparrow^1 \downarrow^S_L \text{auction}, \\
\text{cancel} : \uparrow^8 \downarrow^S_L \text{auction} \}, \\
\text{ended} : \& \{ \text{collect} : \text{id} \supset \\
\oplus \{ \text{won} : \text{lot} \otimes \uparrow^3 \downarrow^S_L \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow^S_L \text{auction} \}, \\
\text{cancel} : \uparrow^8 \downarrow^S_L \text{auction} \} \}
\]

Action can be open (running) or closed (ended).
Example: Type of an Auction Contract

\[
auction = \uparrow_L S \otimes \{ \text{running} : \& \{ \text{bid} : \text{id} \multimap \text{money} \rightarrow \uparrow_1 \downarrow_L S \text{ auction}, \text{cancel} : \uparrow_8 \downarrow_L S \text{ auction} \}, \text{ended} : \& \{ \text{collect} : \text{id} \multimap \}
\]

\[
\quad \oplus \{ \text{won} : \text{lot} \otimes \uparrow_3 \downarrow_L S \text{ auction}, \text{lost} : \text{money} \otimes \downarrow_L S \text{ auction} \}, \text{cancel} : \uparrow_8 \downarrow_L S \text{ auction} \}
\]
Example: Type of an Auction Contract

\[ \text{auction} = \uparrow^S_L a^{11} \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \overset{1}{\rightarrow} \downarrow^S_L \text{auction}, \\
\text{cancel} : \uparrow^8_L \downarrow^S_L \text{auction}, \\
\text{ended} : \& \{ \text{collect} : \text{id} \supset \\
\quad \oplus \{ \text{won} : \text{lot} \otimes \uparrow^3_L \downarrow^S_L \text{auction}, \\
\quad \text{lost} : \text{money} \otimes \downarrow^S_L \text{auction}, \\
\quad \text{cancel} : \uparrow^8_L \downarrow^S_L \text{auction} \} \} \]
Example: Type of an Auction Contract

\[
auction = \uparrow^S_L \downarrow^{11} \oplus \{\text{running} : \&\{\text{bid} : \text{id} \supset \text{money} \to \downarrow^1_L \text{auction}, \\
\text{cancel} : \downarrow^8_L \text{auction}\}, \\
\text{ended} : \&\{\text{collect} : \text{id} \supset \\
\oplus\{\text{won} : \text{lot} \otimes \downarrow^3_L \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow^S_L \text{auction}\}, \\
\text{cancel} : \downarrow^8_L \text{auction}\}\}
\]
Example: Type of an Auction Contract
At the beginning, you have to pay 11 units to cover the worst-case gas cost.

\[
auction = \uparrow^S_L \downarrow^{11} \oplus \{\text{running} : \&\{\text{bid} : \text{id} \supset \text{money} \twoheadrightarrow \uparrow^1 \downarrow^S_L \text{auction}, \\
\text{cancel} : \uparrow^8 \downarrow^S_L \text{auction}\}, \\
\text{ended} : \&\{\text{collect} : \text{id} \supset \\
\oplus\{\text{won} : \text{lot} \otimes \uparrow^3 \downarrow^S_L \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow^S_L \text{auction}\}, \\
\text{cancel} : \uparrow^8 \downarrow^S_L \text{auction}\}\}
\]

Example: Type of an Auction Contract
Example: Type of an Auction Contract

\[
\text{auction} = \uparrow^S_L \downarrow^{11} \oplus \{ \text{running} : \& \{ \text{bid : id } \supset \text{money} \rightarrow \uparrow^1 \downarrow_S^S \text{ auction}, \\
                     \text{cancel} : \uparrow^8 \downarrow_L^S \text{ auction} \}, \\
              \text{ended} : \& \{ \text{collect : id } \supset \\
                             \oplus \{ \text{won : lot } \otimes \uparrow^3 \downarrow_L^S \text{ auction}, \\
                               \text{lost : money } \otimes \downarrow_L^S \text{ auction} \}, \\
                             \text{cancel} : \uparrow^8 \downarrow_L^S \text{ auction} \} \}
\]
At the beginning, you have to pay 11 units to cover the worst-case gas cost.

\[
\text{auction} = \uparrow_{L}^{S} \downarrow^{11} \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow_{L}^{S} \text{auction}, \\
\text{cancel} : \uparrow^{8} \downarrow_{L}^{S} \text{auction} \}, \\\n\text{ended} : \& \{ \text{collect} : \text{id} \supset \\
\oplus \{ \text{won} : \text{lot} \otimes \uparrow^{3} \downarrow_{L}^{S} \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow_{L}^{S} \text{auction} \}, \\
\text{cancel} : \uparrow^{8} \downarrow_{L}^{S} \text{auction} \} \}
\]

If the worst-case path is not taken then the leftover is returned.

Example: Type of an Auction Contract

Gas cost is given by a cost semantics and the type system ensures 11 is the worst-case.
At the beginning, you have to pay 11 units to cover the worst-case gas cost.

\[
\text{auction} = \uparrow^S_L \downarrow^{11} \oplus \{\text{running} : \&\{\text{bid} : \text{id} \Rightarrow \text{money} \rightarrow \downarrow^1 L \text{auction}, \\
\text{cancel} : \downarrow^8 L \text{auction}\}, \\
\text{ended} : \&\{\text{collect} : \text{id} \Rightarrow \\
\oplus\{\text{won} : \text{lot} \otimes \downarrow^3 L \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow^S L \text{auction}, \\
\text{cancel} : \downarrow^8 L \text{auction}\}\}
\]

If the worst-case path is not taken then the leftover is returned.

Gas cost is given by a cost semantics and the type system ensures 11 is the worst-case.

Example: Type of an Auction Contract
Implementation of a Running Auction

\[
\text{auction} = \uparrow^{S}_{L} \downarrow^{11}_{L} \oplus \{\text{running} : \&\{\text{bid} : \text{id} \supset \text{money} \rightarrow \uparrow^{1}_{L} \downarrow^{S}_{L} \text{auction}\},
\]

\( (b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash \text{run} :: (sa : \text{auction}) \)

\[
\begin{align*}
sa & \leftarrow \text{run} \ b \leftarrow M \ l = \\
la & \leftarrow \text{accept} \ sa \ ; \\
l.a.\text{running} \ ; \\
\text{case la} \\
(bid \Rightarrow r \leftarrow \text{recv} \ la \ ; \\
m & \leftarrow \text{recv} \ la \ ; \\
sa & \leftarrow \text{detach} \ la \ ; \\
m.\text{value} \ ; \\
v & \leftarrow \text{recv} \ m \ ; \\
b' = \text{addbid} \ b \ (r, v) \ ; \\
M' & \leftarrow \text{add} \leftarrow M \ m \ ; \\
sa & \leftarrow \text{run} \ b' \leftarrow M' \ ml)
\end{align*}
\]
Implementation of a Running Auction

\[\text{auction} = \uparrow^{S}_{L} \downarrow^{11} \oplus \{\text{running} : \& \{\text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow^{1} \downarrow^{S}_{L} \text{auction}\}, \]

\[(b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash \text{run} :: (sa : \text{auction})\]

\[sa \leftarrow \text{run} \quad b \leftarrow M \quad l =\]

\[la \leftarrow \text{accept} \quad sa ;\]

\[la.\text{running} ;\]

\[\text{case} \quad la\]

\[(\text{bid} \Rightarrow r \leftarrow \text{recv} \quad la ;\]

\[m \leftarrow \text{recv} \quad la ;\]

\[sa \leftarrow \text{detach} \quad la ;\]

\[m.\text{value} ;\]

\[v \leftarrow \text{recv} \quad m ;\]

\[b' = \text{addbid} \quad b \leftarrow (r,v) ;\]

\[M' \leftarrow \text{add} \leftarrow M \quad m ;\]

\[sa \leftarrow \text{run} \quad b' \leftarrow M' \quad ml)\]
Implementation of a Running Auction

\[
auction = \uparrow_S^L \oplus \{\text{running} : \& \{\text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow^1_L S \text{auction}\},
\]

\[(b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash run :: (sa : \text{auction})
\]

\[
sa \leftarrow \text{run} b \leftarrow M l =
\]

\[
la \leftarrow \text{accept} sa ;
\]

\[
la.\text{running} ;
\]

\[
\text{case } la
\]

\[
(bid \Rightarrow r \leftarrow \text{recv} la ;
\]

\[
m \leftarrow \text{recv} la ;
\]

\[
sa \leftarrow \text{detach} la ;
\]

\[
m.\text{value} ;
\]

\[
v \leftarrow \text{recv} m ;
\]

\[
b' = \text{addbid} b (r, v) ;
\]

\[
M' \leftarrow \text{add} \leftarrow M m ;
\]

\[
\text{sa} \leftarrow \text{run} b' \leftarrow M' ml
\]
Implementation of a Running Auction

\[
\text{auction} = \uparrow^S_L \uparrow^{11} \oplus \{\text{running} : \& \{\text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow^1_L \text{sauction}\},
\]

\((b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash \text{run} :: (sa : \text{auction})\)

\[
\begin{align*}
sa & \leftarrow \text{run} \ b \leftarrow M \ l = \\
& \quad la \leftarrow \text{accept} \ sa \\& \quad \text{la.running} \\
\text{case} \ la \\
& \quad (\text{bid} \Rightarrow r \leftarrow \text{recv} \ la \\
& \quad \quad m \leftarrow \text{recv} \ la \\
& \quad \quad sa \leftarrow \text{detach} \ la \\
& \quad \quad m.\text{value} \\
& \quad \quad v \leftarrow \text{recv} \ m \\
& \quad \quad b' = \text{addbid} \ b \ (r, v) \\
& \quad \quad M' \leftarrow \text{add} \leftarrow M \ m \\
& \quad \quad sa \leftarrow \text{run} \ b' \leftarrow M' \ ml)
\end{align*}
\]
### Implementation of a Running Auction

\[
auction = \uparrow^S_L \ominus \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow^1_S \text{auction} \} \},
\]

\[
(b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash \text{run} :: (sa : \text{auction})
\]

\[
\begin{align*}
\text{sa} & \leftarrow \text{run} \ b \leftarrow M \ l = \\
la & \leftarrow \text{accept} \ sa ; \\
la.\text{running} ; \\
\text{case la} \\
(bid \Rightarrow r & \leftarrow \text{recv} \ la ; \\
m & \leftarrow \text{recv} \ la ; \\
\text{sa} & \leftarrow \text{detach} \ la ; \\
m.\text{value} ; \\
v & \leftarrow \text{recv} \ m ; \\
b' & = \text{addbid} \ b (r, v) ; \\
M' & \leftarrow \text{add} \leftarrow M \ m ; \\
\text{sa} & \leftarrow \text{run} \ b' \leftarrow M' \ ml
\end{align*}
\]
Implementation of a Running Auction

\[
\text{auction} = \uparrow^S_L \cong \{ \text{running} : \& \{ \text{bid} : \text{id} \supset \text{money} \} \rightarrow \downarrow^1_L \downarrow^S_L \text{auction} \},
\]

\[
(b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash \text{run} :: (sa : \text{auction})
\]

\[
\text{sa} \leftarrow \text{run} \ b \leftarrow M \ l =
\]

\[
\text{la} \leftarrow \text{accept} \ \text{sa} ;
\]

\[
\text{la}.\text{running} ;
\]

\[
\text{case} \ \text{la}
\]

\[
(\text{bid} \Rightarrow r \leftarrow \text{recv} \ \text{la} ;
\]

\[
\ m \leftarrow \text{recv} \ \text{la} ;
\]

\[
\text{sa} \leftarrow \text{detach} \ \text{la} ;
\]

\[
\text{m}.\text{value} ;
\]

\[
\ v \leftarrow \text{recv} \ \text{m} ;
\]

\[
\ b' = \text{addbid} \ b \ (r, v) ;
\]

\[
\ M' \leftarrow \text{add} \leftarrow M \ m ;
\]

\[
\text{sa} \leftarrow \text{run} \ b' \leftarrow M' \ ml)
\]
Implementation of a Running Auction

\[
auction = \uparrow^S_L \downarrow^{11}_L \oplus \{\text{running} : \& \{\text{bid} : \text{id} \supset \text{money} \rightarrow \downarrow^1_S \text{auction}\},
\]

\[
(b : \text{bids}) ; (M : \text{money}), (ml : \text{monalisa}) \vdash \text{run} :: (sa : \text{auction})
\]

\[
\begin{align*}
\text{sa} & \leftarrow \text{run} \ b \leftarrow M \ l = \\
\text{la} & \leftarrow \text{accept} \ \text{sa} \\
\text{la.running} & ; \\
\text{case} \ \text{la} & \\
& (\text{bid} \Rightarrow r \leftarrow \text{recv} \ \text{la} \\
& \quad m \leftarrow \text{recv} \ \text{la} \\
& \quad \text{sa} \leftarrow \text{detach} \ \text{la} \\
& \quad m.\text{value} \\
& \quad v \leftarrow \text{recv} \ m \\
& \quad b' = \text{addbid} \ b \ (r, v) \\
& \quad M' \leftarrow \text{add} \leftarrow M \ m \\
& \quad \text{sa} \leftarrow \text{run} \ b' \leftarrow M' \ ml)
\end{align*}
\]
How to Use the Potential

**Payment schemes (amortized cost)**

- Ensure constant gas cost in the presence of costly operations
- Overcharge for cheap operations and store gas in contract
- Similar to storing ether in memory in EVM but part of contract

**Explicit gas bounds**

- Add an additional argument that carries potential
- User arg \( N \sim \text{maximal number of players} \Rightarrow \text{gas bound is } 81*N + 28

**Enforce constant gas cost**

- Simply disable potential in contract state
- Require messages to only carry constant potential
Computation on a Blockchain

**Blockchain state:** shared processes waiting to be acquired

\[
\text{contr}_1(\bar{u}_1, \bar{v}_1) \quad \text{contr}_2(\bar{u}_2, \bar{v}_2) \quad \ldots \quad \text{contr}_n(\bar{u}_n, \bar{v}_n)
\]
Computation on a Blockchain

**Blockchain state:** shared processes waiting to be acquired

\[
\text{contr}_1(\bar{u}_1, \bar{v}_1) \quad \text{contr}_2(\bar{u}_2, \bar{v}_2) \quad \ldots \quad \text{contr}_n(\bar{u}_n, \bar{v}_n)
\]

Contracts store functional and linear data.
Computation on a Blockchain

**Blockchain state:** shared processes waiting to be acquired

Contracts store functional and linear data.

Channel name = address
Computation on a Blockchain

**Blockchain state:** shared processes waiting to be acquired

\[
\text{contr}_1(\bar{u}_1, \bar{v}_1) \quad \text{contr}_2(\bar{u}_2, \bar{v}_2) \quad \ldots \quad \text{contr}_n(\bar{u}_n, \bar{v}_n)
\]

Contracts store functional and linear data.

Channel name = address

**Transaction:** client submits code of a linear process

\[
\text{contr}_1(\bar{u}_1, \bar{v}_1) \quad \text{contr}_2(\bar{u}_2, \bar{v}_2) \quad \ldots \quad \text{contr}_n(\bar{u}_n, \bar{v}_n) \quad \text{client}
\]
Computation on a Blockchain

**Blockchain state:** shared processes waiting to be acquired

\[
\text{contr}_1(\bar{u}_1, \bar{v}_1) \quad \text{contr}_2(\bar{u}_2, \bar{v}_2) \quad \ldots \quad \text{contr}_n(\bar{u}_n, \bar{v}_n)
\]

Contracts store functional and linear data.

Channel name = address

**Transaction:** client submits code of a linear process

\[
\text{contr}_1(\bar{u}_1, \bar{v}_1) \quad \text{contr}_2(\bar{u}_2, \bar{v}_2) \quad \ldots \quad \text{contr}_n(\bar{u}_n, \bar{v}_n) \quad \text{client}
\]

- Client process can acquire existing contracts
- Client process can spawn new (shared) processes -> new contracts
- Client process needs to terminates in a new valid state
Computation on a Blockchain

**Blockchain state:** shared processes waiting to be acquired

contr₁(\( \bar{u}_1, \bar{v}_1 \))  contr₂(\( \bar{u}_2, \bar{v}_2 \))  ...  contrₙ(\( \bar{u}_n, \bar{v}_n \))

- Client process can acquire existing contracts
- Client process can spawn new (shared) processes -> new contracts
- Client process needs to terminates in a new valid state

**Transaction:** client submits code of a linear process

contr₁(\( \bar{u}_1, \bar{v}_1 \))  contr₂(\( \bar{u}_2, \bar{v}_2 \))  ...  contrₙ(\( \bar{u}_n, \bar{v}_n \))

Channel name = address

Contracts store functional and linear data.

Contract should have default clients.
Blockchain, Type Checking, and Verification

**Type checking is part of the attack surface**

- Contract code can be checked at publication time
- User code needs to be checked for each transaction
- Denial of service attacks are possible
- *Nomos type checking is linear in the size of the program*

**Verification of Nomos program is possible**

- Dynamic semantics specifies runtime behavior
- Directly applicable to verification in Coq
- Nomos’ type system guarantees some important properties
Nomos

A statically-typed, strict, functional language for digital contracts

• Automatic amortized resource analysis for static gas bounds
• Shared binary session types for transparent & safe contract interfaces
• Linear type system for accurately reflecting assets

References

• POPL ’17: AARA for OCaml (RaML)
• LICS ’18: Resource-Aware Session Types
• arXiv ’19: Nomos

Ongoing work: implementation

• Parser ✓
• Type checker ✓
• Interpreter
• Compiler
A statically-typed, strict, functional language for digital contracts

• Automatic amortized resource analysis for static gas bounds
• Shared binary session types for transparent & safe contract interfaces
• Linear type system for accurately reflecting assets

References

• POPL ’17: AARA for OCaml (RaML)
• LICS ’18: Resource-Aware Session Types
• arXiv ’19: Nomos

Ongoing work: implementation

• Parser ✓
• Type checker ✓
• Interpreter
• Compiler