Session Types with Arithmetic Refinements

Ankush Das*
Frank Pfenning
Carnegie Mellon University

Session 4A, Sept 1
CONCUR 2020
What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types
What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

![Diagram showing provider process P and client process Q connected through a channel](image)
What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

\[ \text{bits} = \oplus \{ \text{b0 : bits, b1 : bits} \} \]
What are Session Types?

- Implement message-passing concurrent programs
- Communication via typed bi-directional channels
- Communication protocol enforced by session types

\[ \text{bits} = \bigoplus \{ \text{b0 : bits}, \text{b1 : bits} \} \]
Example: Queues

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

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

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

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

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

offers choice of ins/del
recv element of type A
behave as queue again

ins, e
Example: Queues

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

```
queue_A = \{ \text{ins} : A \rightarrow \text{queue}_A, \\
              \text{del} : \oplus \{ \text{none} : 1, \\
                             \text{some} : A \otimes \text{queue}_A \} \}
```
Example: Queues

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

offers choice of ins/del

send none if queue is empty
Example: Queues

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

offers choice of ins/del

send none if queue is empty

del

send none if queue is empty
Example: Queues

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

offers choice of ins/del

send some otherwise

send some otherwise

Otherwise
Example: Queues

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

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

- offers choice of ins/del
- send element of type A
- behave as queue again
- send some otherwise

\[ a \quad b \quad c \quad d \quad e \]
Example: Queues

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

offers choice of ins/del

send element of type A

behave as queue again

send some otherwise

some, a
Example: Queues

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

When are the none and some branches taken? Can the size of queue be encoded in the type?
Count the total number of messages!

a b c d
Work done by Queue

Count the total number of messages!

ins, e

a b c d
Work done by Queue

Count the total number of messages!

\[ w_i = \text{Work done to process insertion} \]
\[ = 2n \text{ (n is the size of queue)} \]
\[ = \text{‘ins’ and ‘e’ travel to end of queue} \]
Work done by Queue

Count the total number of messages!

\[ w_i = \text{Work done to process insertion} \]
\[ = 2n \quad (n \text{ is the size of queue}) \]
\[ = \text{‘ins’ and ‘e’ travel to end of queue} \]

Insertion: How do you refer to \( n \) in the queue type?
Refined Queue Type

\[
\text{queue}_A[n] = \&\{\text{ins} : A \rightarrow \text{queue}_A[n + 1], \\
\text{del} : \oplus\{\text{none} : ?\{n = 0\}. 1, \\
\text{some} : ?\{n > 0\}. A \otimes \text{queue}_A[n - 1]\}\}
\]
Refined Queue Type

\[
\text{queue}_A[n] = \&\{\text{ins} : A \rightarrow \text{queue}_A[n + 1], \\
\quad \text{del} : \oplus\{\text{none} : ?\{n = 0\}. 1, \\
\quad \text{some} : ?\{n > 0\}. A \otimes \text{queue}_A[n - 1]\}\}
\]
Refined Queue Type

\[
\text{queue}_A[n] = \&\{\text{ins} : A \rightarrow \text{queue}_A[n + 1], \\
\text{del} : \oplus\{\text{none} : ?\{n = 0\}. 1, \\
\text{some} : ?\{n > 0\}. A \otimes \text{queue}_A[n - 1]\}\}
\]

- ‘none’ branch: send (proof of) constraint \(\{n=0\}\)
- ‘some’ branch: send (proof of) constraint \(\{n>0\}\)
- Domain of constraints: Presburger Arithmetic
Three Key Results
Three Key Results

Typing System with Arithmetic Refinements

Negative Result:
Type equality with refinements is undecidable
(even though type equality for simple session types
and Presburger arithmetic are both decidable!)

Positive Result:
A sound algorithm for type equality
(works exceptionally well in practice!)
Three Key Results

Negative Result: 
Type equality with refinements is undecidable 
(even though type equality for simple session types 
and Presburger arithmetic are both decidable!)

Positive Result: 
A sound algorithm for type equality 
(works exceptionally well in practice!)
Two New Type Operators
Two New Type Operators

\[ ?\{\phi\}. A \]

- Provider sends (proof of) \( \phi \), then continues to provide \( A \)
- Client receives (proof of) constraint \( \phi \)
Two New Type Operators

\(?\{\phi}\). A

- Provider sends (proof of) $\phi$, then continues to provide $A$
- Client receives (proof of) constraint $\phi$

\(!\{\phi}\). A

- Provider receives (proof of) $\phi$, then continues to provide $A$
- Client sends (proof of) constraint $\phi$
Two New Type Operators

\(?\{\varphi}\). A

- Provider sends (proof of) \(\varphi\), then continues to provide A
- Client receives (proof of) constraint \(\varphi\)

\(!\{\varphi\}\). A

- Provider receives (proof of) \(\varphi\), then continues to provide A
- Client sends (proof of) constraint \(\varphi\)

Since Presburger arithmetic is decidable and only closed programs are executed, no need to exchange proofs/constraints at runtime
Type Grammar

\[ A ::= \bigoplus_{\ell \in L} \ell : A | \bigwedge_{\ell \in L} \ell : A | A \otimes A | A \rightarrow A | 1 \]
\[ | V[e] | \phi . A | !\phi . A | \exists n. A | \forall n. A \]
Type Grammar

Standard session types

\[ A ::= \bigoplus_{\ell \in L} A \bigotimes_{\ell \in L} A \otimes A \rightarrow A \rightarrow 1 \]
\[ \mid V[e] \mid ?\phi . A \mid !\phi . A \mid \exists n . A \mid \forall n . A \]
Type Grammar

Standard session types

\[ A ::= \bigoplus_{\ell \in L} A \mid \bigwedge_{\ell \in L} A \mid A \otimes A \mid A \rightarrow A \mid 1 \mid V[\bar{e}] \mid ?\{\phi\}.A \mid !\{\phi\}.A \mid \exists n. A \mid \forall n. A \]
Type Grammar

Standard session types

\[ A ::= \bigoplus_{\ell \in L} \ell : A | \&_{\ell \in L} \ell : A | A \otimes A | A \rightarrow A | 1 \\
| V[e] | ?\{\phi\}.A | !\{\phi\}.A | \exists n. A | \forall n. A \]

Type variable indexed with arith. exps.
Send / Recv proof constraints
Type Grammar

Standard session types

\[ A ::= \begin{align*}
& \oplus\{\ell : A\}_{\ell \in L} \mid \&\{\ell : A\}_{\ell \in L} \mid A \otimes A \mid A \to A \mid 1 \\
& \mid V[e] \mid {?}\{\phi}\}.A \mid {!}\{\phi}\}.A \mid \exists n. A \mid \forall n. A
\end{align*} \]

Type variable indexed with arith. exps.
Send / Recv proof constraints
Send / Recv natural numbers
Three Key Results

Typing System with Arithmetic Refinements

Negative Result:
Type equality with refinements is undecidable
(even though type equality for simple session types
and Presburger arithmetic are both decidable!)

Positive Result:
A sound algorithm for type equality
(works exceptionally well in practice!)
Three Key Results

Typing System with Arithmetic Refinements

Negative Result:
Type equality with refinements is undecidable
(even though type equality for simple session types
and Presburger arithmetic are both decidable!)

Positive Result:
A sound algorithm for type equality
(works exceptionally well in practice!)
Two types are considered equal if they have the same communication behavior (structural type system).

Formally defined by establishing a type bisimulation.

Refinement session types can encode 2-counter machines (2CM)!

Equality reduces to determining if the 2CM halts, and is thus, undecidable!

Given a 2CM and input, we construct two types that only differ at the halting instruction.
Reduction from 2CM

Increment Instruction:
\[ m : \text{inc}(c_1) ; \text{goto} \ k \]

\[
A_m[c_1, c_2] = \bigoplus \{ \text{inc}_1 : A_k[c_1 + 1, c_2] \}
\]

\[
A'_m[c_1, c_2] = \bigoplus \{ \text{inc}_1 : A'_k[c_1 + 1, c_2] \}
\]
Decrement Instruction:
$m : \text{zero}(c_1) ? \text{goto } k : \text{goto } n$

\[
A_m[c_1, c_2] = \bigoplus \{ \text{zero}_1 : ?\{c_1 = 0\}. A_k[c_1, c_2], \text{dec}_1 : ?\{c_1 > 0\}. A_n[c_1 - 1, c_2] \}
\]

\[
A'_m[c_1, c_2] = \bigoplus \{ \text{zero}_1 : ?\{c_1 = 0\}. A'_k[c_1, c_2], \text{dec}_1 : ?\{c_1 > 0\}. A'_n[c_1 - 1, c_2] \}
\]
Halting Instruction:

\[ m : \text{halt} \]

\[ A_m[c_1, c_2] = A \]

\[ A'_m[c_1, c_2] = A' \]
Reduction from 2CM

Halting Instruction:
\( m : \text{halt} \)

\[
A_m[c_1, c_2] = A
\]

\[
A'_m[c_1, c_2] = A'
\]

\( A \not\equiv A' \)
Halting Instruction:
\[ m : \text{halt} \]

\[ A_m[c_1, c_2] = A \]

\[ A'_m[c_1, c_2] = A' \]

\[ A \neq A' \]

The types are equal iff the 2CM does not halt
Three Key Results

Typing System with Arithmetic Refinements

Negative Result:
Type equality with refinements is undecidable
(even though type equality for simple session types
and Presburger arithmetic are both decidable!)

Positive Result:
A sound algorithm for type equality
(works exceptionally well in practice!)
Three Key Results

Typing System with Arithmetic Refinements

Negative Result:
Type equality with refinements is undecidable
(even though type equality for simple session types
and Presburger arithmetic are both decidable!)

Positive Result:
A sound algorithm for type equality
(works exceptionally well in practice!)
Equality Algorithm

$$\text{ctr}[x, y] = \oplus \{ \text{lt} : \{ x < y \}. \text{ctr}[x + 1, y], \text{ge} : \{ x \geq y \}. 1 \}$$
Equality Algorithm

\[ \text{ctr}[x, y] = \oplus \{ \text{lt} : \{ x < y \}. \text{ctr}[x + 1, y], \text{ge} : \{ x \geq y \}. 1 \} \]

\[ \text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1] \]
Equality Algorithm

\[
\text{ctr}[x, y] = \oplus \{ \text{lt} : ?\{x < y\}. \text{ctr}[x + 1, y], \\
\text{ge} : ?\{x \geq y\}. 1 \}
\]

\[
\text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1]
\]

**Goal:** Find a counterexample
Equality Algorithm

\[
\text{ctr}[x, y] = \oplus \{ \text{lt} : ?\{x < y\}. \text{ctr}[x + 1, y], \text{ge} : ?\{x \geq y\}. 1 \} 
\]

\forall X, Y. \text{ctr}[X, Y] \equiv \text{ctr}[X + 1, Y + 1]

\text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1]

Goal: Find a counterexample

store the equality constraint and expand both types
Equality Algorithm

\[
\text{ctr}[x, y] = \oplus \{ \text{lt} : ?\{x < y\}. \text{ctr}[x + 1, y], \text{ge} : ?\{x \geq y\}. 1 \}
\]

∀X, Y. ctr[X, Y] ≡ ctr[X + 1, Y + 1]

Goal: Find a counterexample
Equality Algorithm

\[
\text{ctr}[x, y] = \oplus \{ \text{lt} : \{ x < y \} . \text{ctr}[x + 1, y], \\
\text{ge} : \{ x \geq y \} . 1 \}
\]

\[\forall X, Y. \text{ctr}[X, Y] \equiv \text{ctr}[X + 1, Y + 1]\]

\[
x < y \equiv x + 1 < y + 1
\]

\[
\{ x < y \} . \text{ctr}[x + 1, y] \equiv \{ x + 1 < y + 1 \} . \text{ctr}[x + 2, y + 1]
\]

\[\text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1]\]

Goal: Find a counterexample
Equality Algorithm

\[ ctr[x, y] = \bigoplus \{ \text{lt : } ?\{x < y\}. \ ctr[x + 1, y], \ \text{ge : } ?\{x \geq y\}. \ 1 \} \]

\[ \forall X, Y. \ ctr[X, Y] \equiv ctr[X + 1, Y + 1] \]

\[ x < y \equiv x + 1 < y + 1 \quad \text{ctr}[x + 1, y] \equiv \text{ctr}[x + 2, y + 1] \]

\[ ?\{x < y\}. \ ctr[x + 1, y] \equiv ?\{x + 1 < y + 1\}. \ ctr[x + 2, y + 1] \]

\[ \text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1] \]

**Goal:** Find a counterexample
Equality Algorithm

\[
\text{ctr}[x, y] = \oplus \{ \text{lt} : ?\{x < y\} \cdot \text{ctr}[x + 1, y], \\
\text{ge} : ?\{x \geq y\} \cdot 1 \}
\]

\[\forall X, Y. \text{ctr}[X, Y] \equiv \text{ctr}[X + 1, Y + 1]\]

\[x < y \equiv x + 1 < y + 1 \quad \text{ctr}[x + 1, y] \equiv \text{ctr}[x + 2, y + 1]\]

\[?\{x < y\} \cdot \text{ctr}[x + 1, y] \equiv ?\{x + 1 < y + 1\} \cdot \text{ctr}[x + 2, y + 1]\]

\[\text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1]\]

**Goal:** Find a counterexample
Equality Algorithm

\[ \text{ct}r[x, y] = \Theta \{ \text{lt} : ?\{ x < y \}. \text{ct}r[x + 1, y], \text{ge} : ?\{ x \geq y \}. 1 \} \]

\[ \forall X, Y. \text{ct}r[X, Y] \equiv \text{ct}r[X + 1, Y + 1] \]

substitute \( x+1 \) for \( X \), \( y \) for \( Y \)

\[ x < y \equiv x + 1 < y + 1 \quad \text{ct}r[x + 1, y] \equiv \text{ct}r[x + 2, y + 1] \]

\[ ?\{ x < y \}. \text{ct}r[x + 1, y] \equiv ?\{ x + 1 < y + 1 \}. \text{ct}r[x + 2, y + 1] \]

\[ \text{ct}r[x, y] \equiv \text{ct}r[x + 1, y + 1] \]

Goal: Find a counterexample
Formal Judgment

\[ \mathcal{V} ; C ; \Gamma \vdash A \equiv B \]
Formal Judgment

Free variables

\[ \forall \; C \; ; \; \Gamma \vdash A \equiv B \]
Formal Judgment

Free variables

Constraint satisfied by V

\[ \forall ; C ; \Gamma \vdash A \equiv B \]
Formal Judgment

\[ V ; C ; \Gamma \vdash A \equiv B \]

Free variables

Constraint satisfied by V

Equality constraints stored
Formal Judgment

Free variables

Constraint satisfied by $V$

Equality constraints stored

$\forall \; C \; ; \; \Gamma \vdash A \equiv B$

*Types A and B are equal under constraint C*
Formal Judgment

Free variables

Constraint satisfied by V

Equality constraints stored

\[ \forall \; C \; ; \; \Gamma \vdash A \equiv B \]

Types A and B are equal under constraint C

\[ x, y \; ; \; \top \vdash \text{ctr}[x, y] \equiv \text{ctr}[x + 1, y + 1] \]
\[\langle \mathcal{V}' ; C' ; \mathcal{V}_1[\overline{E_1}] \equiv \mathcal{V}_2[\overline{E_2}] \rangle \in \Gamma\]

\[
\forall \mathcal{V}. C \Rightarrow \exists \mathcal{V}'. C' \land \overline{E_1} = \overline{e_1} \land \overline{E_2} = \overline{e_2} \]

\[
\mathcal{V} ; C ; \Gamma \vdash \mathcal{V}_1[\overline{e_1}] \equiv \mathcal{V}_2[\overline{e_2}] \quad \text{def}
\]
Already encountered constraint

\[ \langle \mathcal{V}' ; C' ; V_1[E_1] \equiv V_2[E_2] \rangle \in \Gamma \]
\[ \forall \mathcal{V}. C \Rightarrow \exists \mathcal{V}'. C' \land E_1 = e_1 \land E_2 = e_2 \]
\[ \mathcal{V} ; C ; \Gamma \vdash V_1[e_1] \equiv V_2[e_2] \] \quad \text{def}
Already encountered constraint

\[
\langle \mathcal{V}' ; C' ; V_1[E_1] \equiv V_2[E_2] \rangle \in \Gamma
\]

\[
\forall \mathcal{V}. C \Rightarrow \exists \mathcal{V}'. C' \land E_1 = e_1 \land E_2 = e_2 \]

\[
\mathcal{V} ; C ; \Gamma \vdash V_1[e_1] \equiv V_2[e_2]
\]

def

**If we know** \(V_1[E_1] \equiv V_1[E_2]\),

can we prove \(V_1[e_1] \equiv V_2[e_2]\)?

**That is what the second premise achieves!**
Rast Programming Language
Goal of Rast

Lightweight Verification and Resource Analysis of Concurrent Programs

Execution Time

Memory Usage
Key Features of Rast

Session Types

Rast Language

Resource Analysis

Arithmetic Refinements

LICS 18, ICFP 18

FSCD 20, PPDP 20
## Evaluation

<table>
<thead>
<tr>
<th>Module</th>
<th>LOC</th>
<th>#Defs</th>
<th>T (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>143</td>
<td>8</td>
<td>1.325</td>
</tr>
<tr>
<td>integers</td>
<td>114</td>
<td>8</td>
<td>1.074</td>
</tr>
<tr>
<td>linlam</td>
<td>67</td>
<td>6</td>
<td>4.003</td>
</tr>
<tr>
<td>list</td>
<td>441</td>
<td>29</td>
<td>3.419</td>
</tr>
<tr>
<td>primes</td>
<td>118</td>
<td>8</td>
<td>1.646</td>
</tr>
<tr>
<td>segments</td>
<td>65</td>
<td>9</td>
<td>0.195</td>
</tr>
<tr>
<td>ternary</td>
<td>235</td>
<td>16</td>
<td>1.967</td>
</tr>
<tr>
<td>theorems</td>
<td>141</td>
<td>16</td>
<td>0.894</td>
</tr>
<tr>
<td>tries</td>
<td>308</td>
<td>9</td>
<td>5.283</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1632</td>
<td>109</td>
<td>19.806</td>
</tr>
</tbody>
</table>
Conclusion
Negative Result:  
Type equality with refinements is undecidable

Positive Result:  
A sound algorithm for type equality

Meta Result:  
Results generalize to any structural type system  
e.g. functional programming languages