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

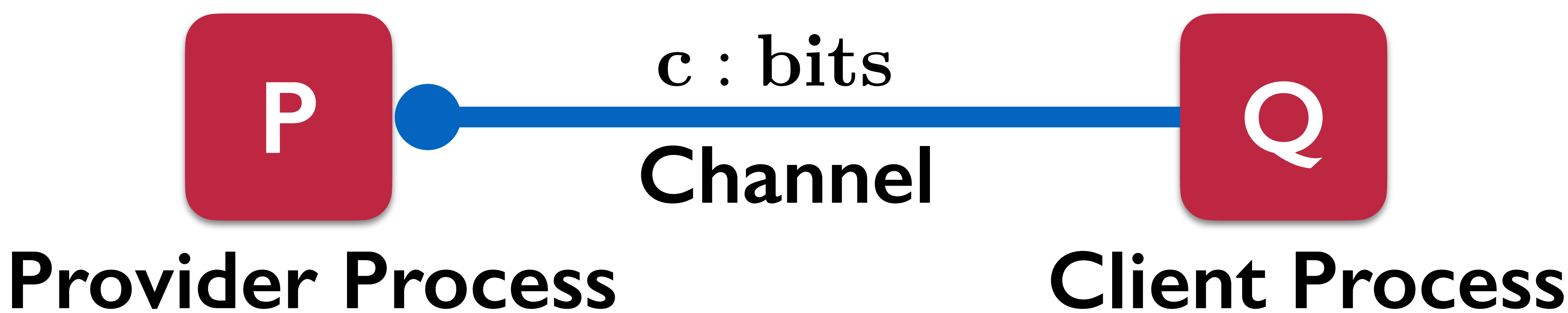
What are Session Types?

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

What are Session Types?

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

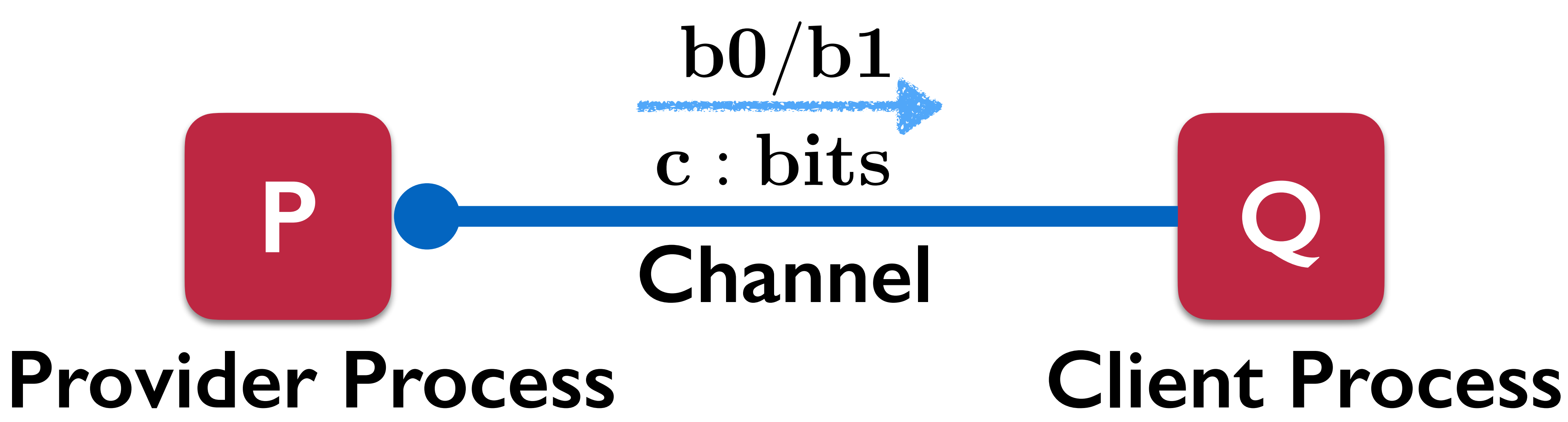
$$\mathrm{bits} = \oplus\{\mathrm{b0} : \mathrm{bits}, \mathrm{b1} : \mathrm{bits}\}$$



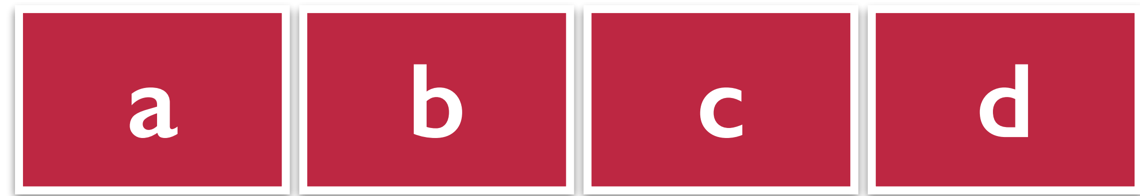
What are Session Types?

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

$$\mathbf{bits} = \oplus\{\mathbf{b0} : \mathbf{bits}, \mathbf{b1} : \mathbf{bits}\}$$

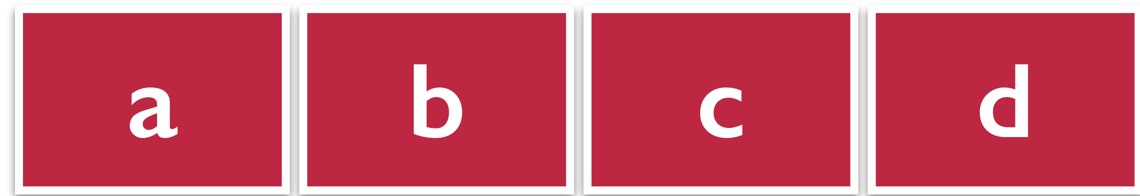


Example: Queues



$$
\begin{aligned}
\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\\
\mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\
\mathbf{some} : A \otimes \mathrm{queue}_A\}\}
\end{aligned}
$$

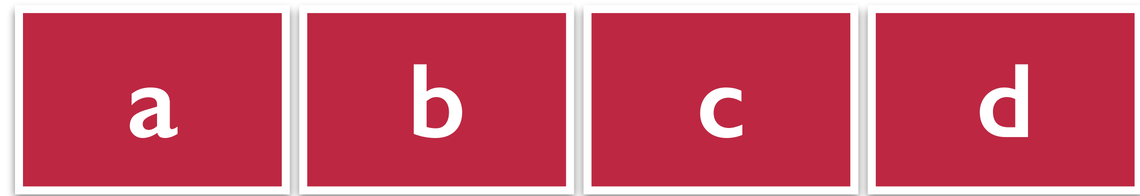
Example: Queues



offers choice of ins/del

$$\begin{aligned}\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ \mathbf{del} : \oplus\{&\mathbf{none} : \mathbf{1},\\ &\mathbf{some} : A \otimes \mathrm{queue}_A\}\}\end{aligned}$$

Example: Queues

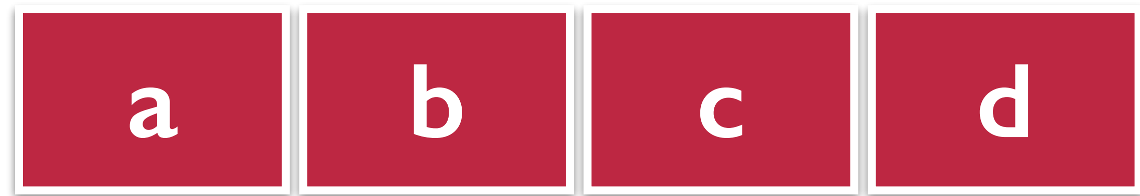


offers choice of ins/del

recv element of type A

$$\begin{aligned}\mathrm{queue}_A = \&\{&\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ &\mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ &\qquad\qquad \mathbf{some} : A \otimes \mathrm{queue}_A\}\}\end{aligned}$$

Example: Queues



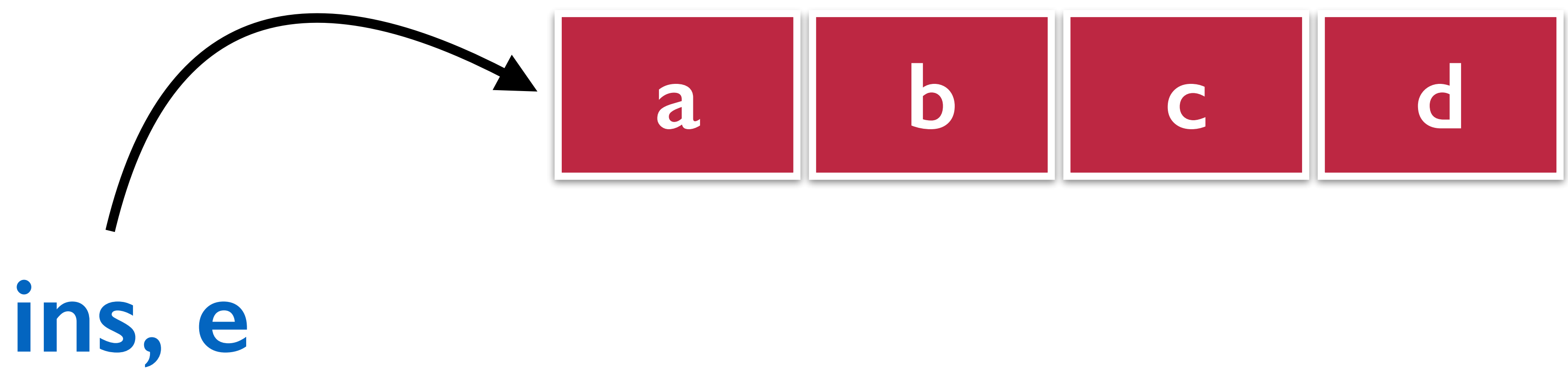
offers choice of ins/del

recv element of type A

behave as queue again

$$\begin{aligned}\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ \mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ \mathbf{some} : A \otimes \mathrm{queue}_A\}\}\end{aligned}$$

Example: Queues



offers choice of ins/del

recv element of type A

behave as queue again

$$
\begin{aligned}
\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A, \\
\mathbf{del} : \oplus\{&\mathbf{none} : \mathbf{1}, \\
&\mathbf{some} : A \otimes \mathrm{queue}_A\}\}
\end{aligned}
$$

Example: Queues

a	b	c	d	e

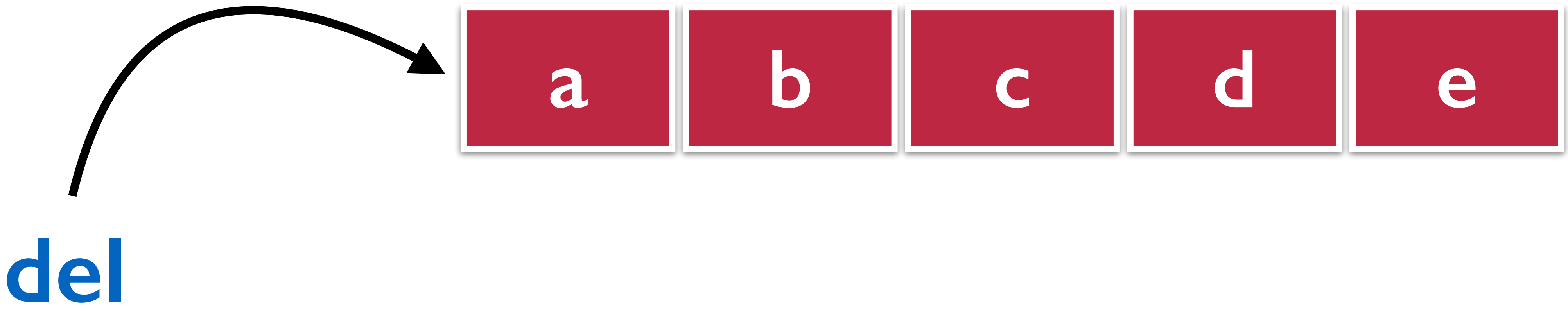
offers choice of ins/del

recv element of type A

behave as queue again

$$\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\ \mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\ \mathbf{some} : A \otimes \mathrm{queue}_A\}\}$$

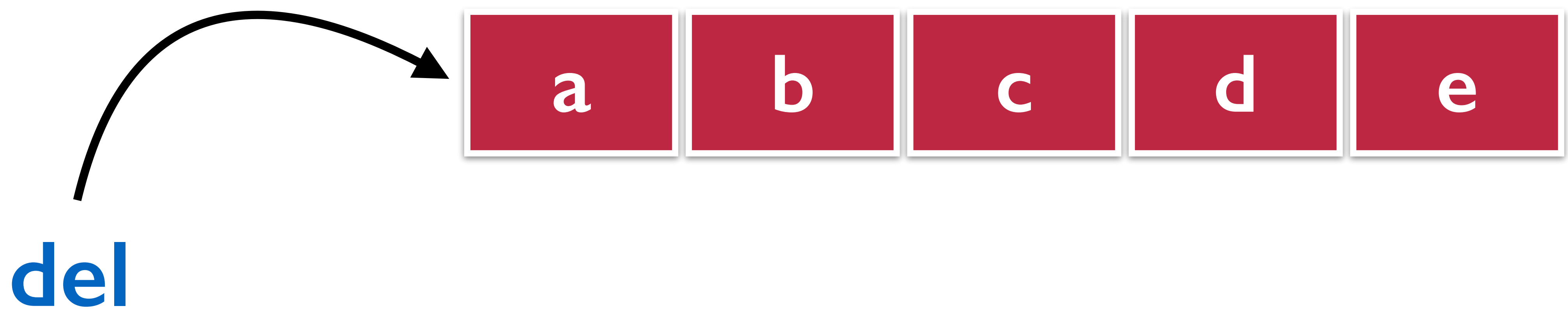
Example: Queues



offers choice of ins/del

$$\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ \mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ \mathbf{some} : A \otimes \mathrm{queue}_A\}\}$$

Example: Queues



offers choice of ins/del

$$\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\ \mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\ \mathbf{some} : A \otimes \mathrm{queue}_A\}\}$$

send none if queue is empty

Example: Queues

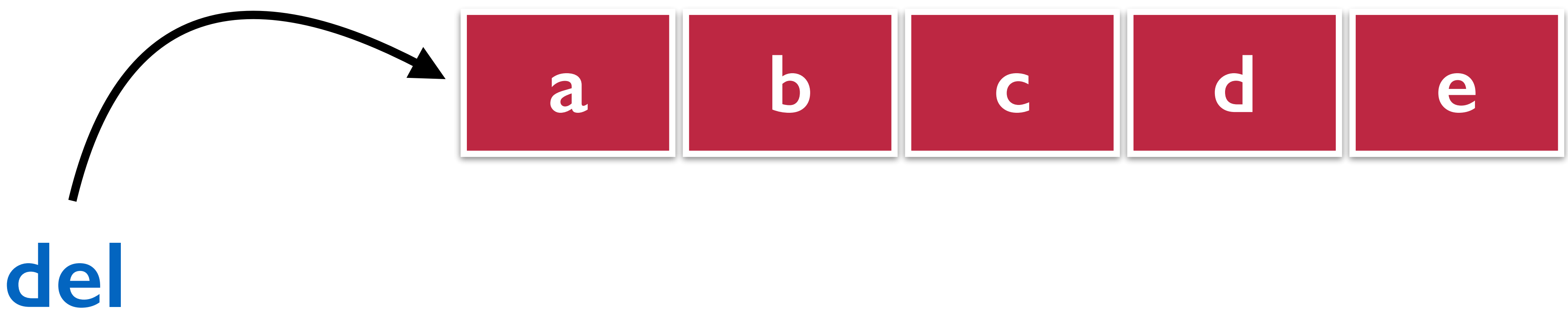
offers choice of ins/del

$$
\begin{aligned}
\text{queue}_A = \&\{&\mathbf{ins} : A \multimap \text{queue}_A,\\
&\mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\
&\qquad\quad \mathbf{some} : A \otimes \text{queue}_A\}\}
\end{aligned}
$$

terminate

send none if queue is empty

Example: Queues

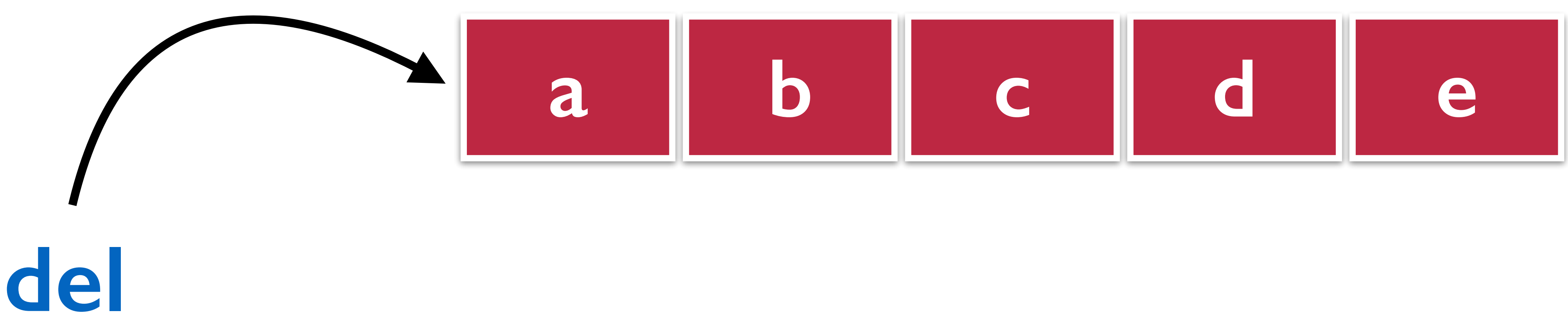


offers choice of ins/del

$$
\begin{aligned}
\mathrm{queue}_A = \&\{&\mathbf{ins} : A \multimap \mathrm{queue}_A,\\
&\mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\
&\qquad\qquad \mathbf{some} : A \otimes \mathrm{queue}_A\}\}
\end{aligned}
$$

send some otherwise

Example: Queues



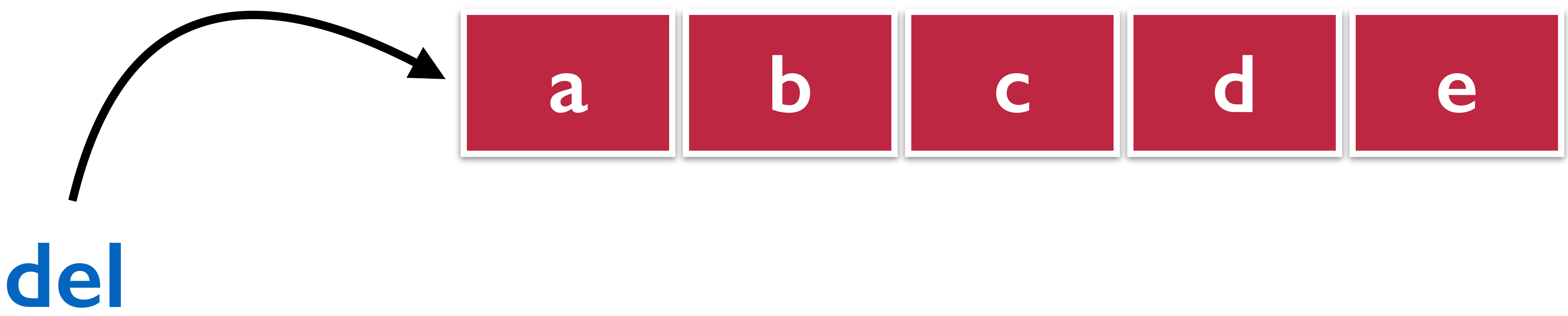
offers choice of ins/del

send element of type A

$$\begin{aligned}\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ \mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ \mathbf{some} : A \otimes \mathrm{queue}_A\}\}\end{aligned}$$

send some otherwise

Example: Queues



offers choice of ins/del

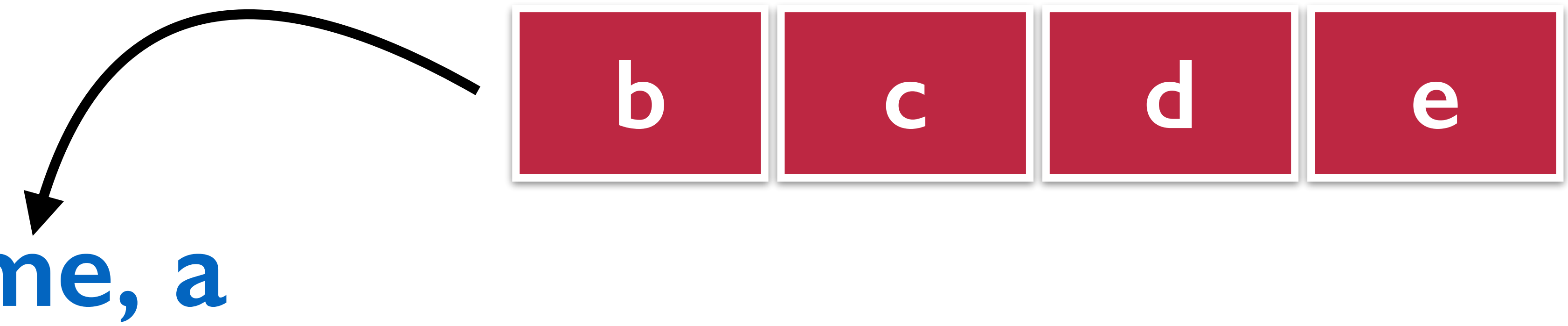
send element of type A

behave as queue again

$$\begin{aligned}\mathrm{queue}_A = \&\{&\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ &\mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ &\qquad\quad \mathbf{some} : A \otimes \mathrm{queue}_A\}\}\end{aligned}$$

send some otherwise

Example: Queues



offers choice of ins/del

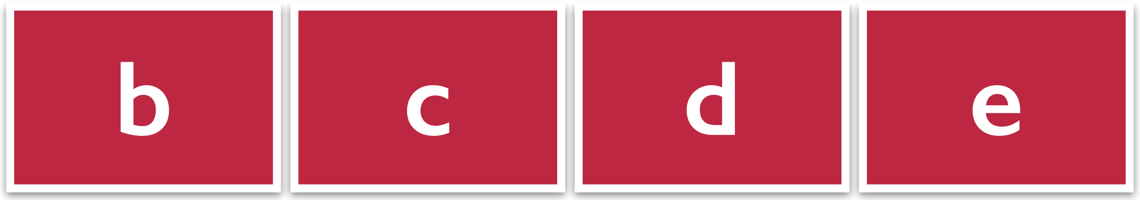
send element of type A

behave as queue again

$$\begin{aligned}\mathrm{queue}_A = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A,\\ \mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ \mathbf{some} : A \otimes \mathrm{queue}_A\}\}\end{aligned}$$

send some otherwise

Example: Queues



$$\begin{aligned}\text{queue}_A = \&\{\mathbf{ins} : A \multimap \text{queue}_A,\\ \mathbf{del} : \oplus\{&\mathbf{none} : \mathbf{1},\\ &\mathbf{some} : A \otimes \text{queue}_A\}\}\end{aligned}$$

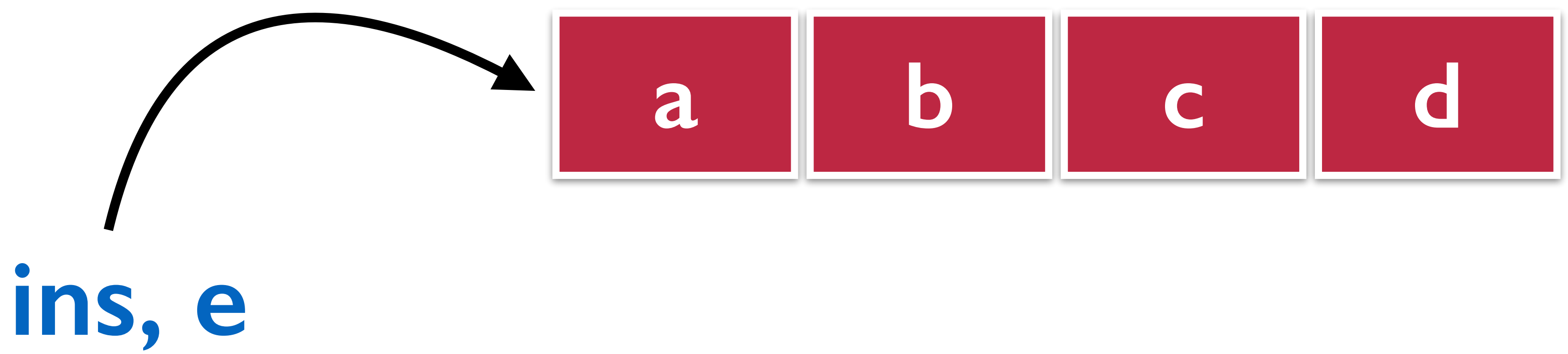
When are the none and some branches taken?
Can the size of queue be encoded in the type?

Work done by Queue

Count the total number of messages!

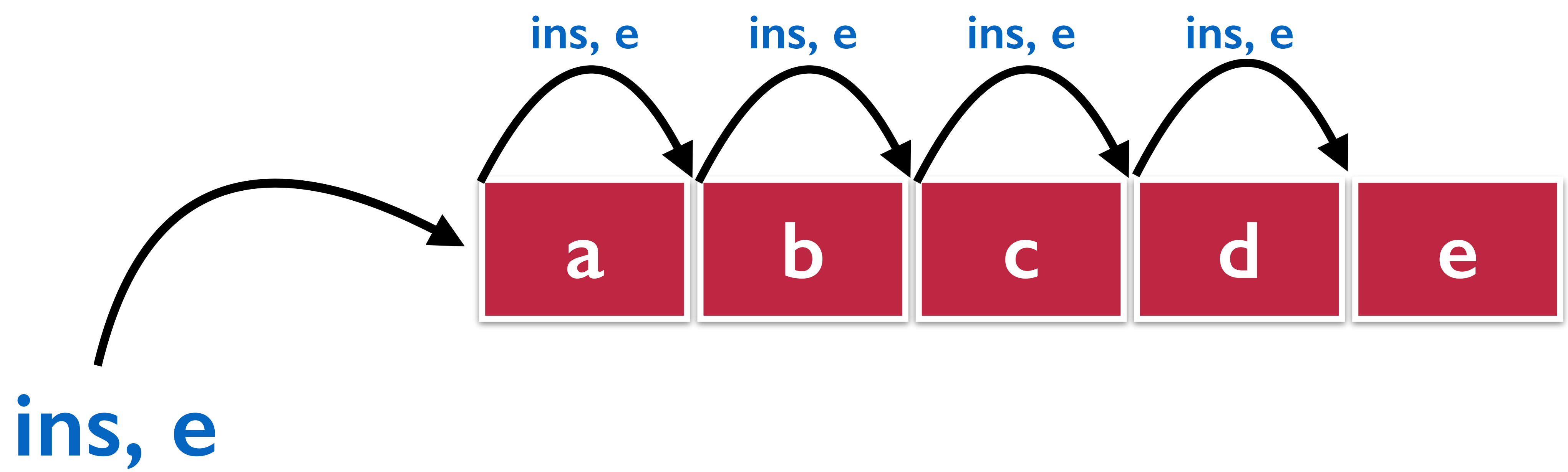
Work done by Queue

Count the total number of messages!



Work done by Queue

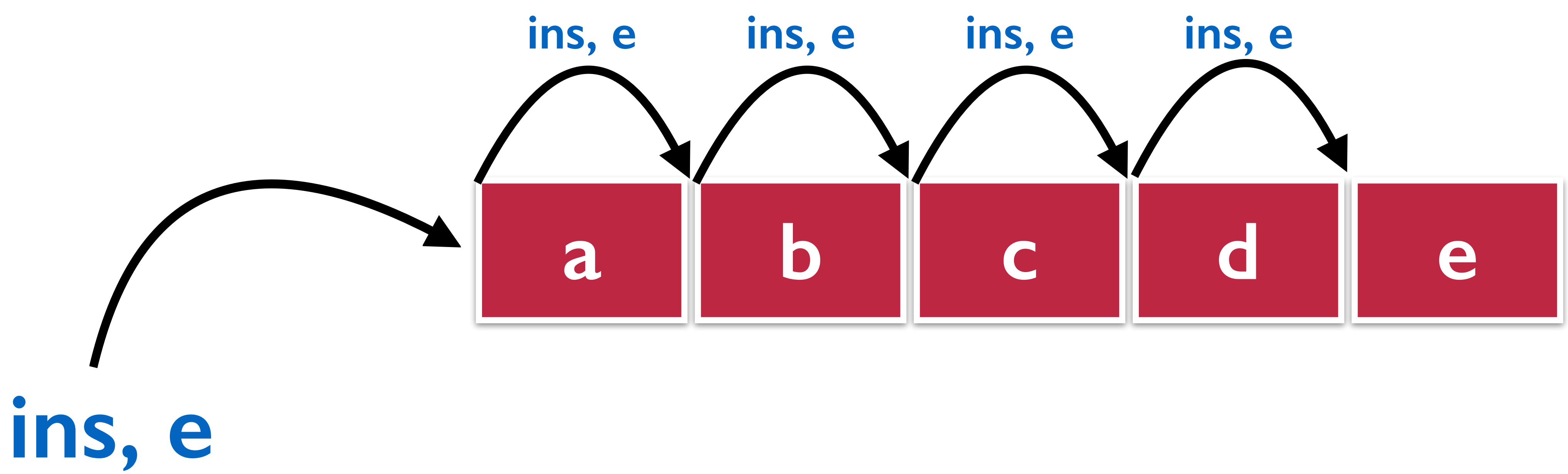
Count the total number of messages!



w_i = Work done to process insertion
= 2n (n is the size of queue)
= 'ins' and 'e' travel to end of queue

Work done by Queue

Count the total number of messages!



w_i = Work done to process insertion
= 2n (n is the size of queue)
= ‘ins’ and ‘e’ travel to end of queue

Insertion: How do you refer to n in the queue type?

Refined Queue Type

$$
\begin{aligned}
\mathrm{queue}_A[n] = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A[n+1],\\
\mathbf{del} : \oplus\{\mathbf{none} : ?\{n = 0\}.\, \mathbf{1},\\
\mathbf{some} : ?\{n > 0\}.\, A \otimes \mathrm{queue}_A[n-1]\}\}
\end{aligned}
$$

Refined Queue Type

$$\begin{aligned}\mathrm{queue}_A[n] = \&\{\mathbf{ins} : A \multimap \mathrm{queue}_A[n+1],\\ \mathbf{del} : \oplus\{\mathbf{none} : ?\{n = 0\}.\, \mathbf{1},\\ \mathbf{some} : ?\{n > 0\}.\, A \otimes \mathrm{queue}_A[n-1]\}\}\end{aligned}$$

Index Refinement
(Size of Queue)

Refined Queue Type

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

Index Refinement (Size of Queue)

Proof Constraints (Sent by queue)

- ‘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

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 New Type Operators

Two New Type Operators

$?\{\phi\}.\ A$

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

Two New Type Operators

$?\{\phi\}.\ A$

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

$!\{\phi\}.\ A$

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

Two New Type Operators

$?\{\phi\}.\ A$	$!\{\phi\}.\ A$
Provider sends (proof of) ϕ, then continues to provide A	Provider receives (proof of) ϕ, then continues to provide A
Client receives (proof of) constraint ϕ	Client sends (proof of) constraint ϕ

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

Type Grammar

$$
\begin{array}{rcl}
A & ::= & \oplus\{\ell : A\}_{\ell \in L} \mid \&\{\ell : A\}_{\ell \in L} \mid A \otimes A \mid A \multimap A \mid \mathbf{1} \\
& \mid & V[\overline{e}] \mid ?\{\phi\}.\, A \mid !\{\phi\}.\, A \mid \exists n.\, A \mid \forall n.\, A
\end{array}
$$

Type Grammar

Standard session types

$$A \quad ::= \quad \oplus\{\ell : A\}_{\ell \in L} \mid \&\{\ell : A\}_{\ell \in L} \mid A \otimes A \mid A \multimap A \mid \mathbf{1}$$

$$\mid \quad V[\overline{e}] \mid ?\{\phi\}.\,A \mid !\{\phi\}.\,A \mid \exists n.\,A \mid \forall n.\,A$$

Type Grammar

Standard session types

$$A \ ::= \ \oplus\{\ell : A\}_{\ell \in L} \mid \&\{\ell : A\}_{\ell \in L} \mid A \otimes A \mid A \multimap A \mid \mathbf{1}$$
$$\mid \ V[\overline{e}] \mid ?\{\phi\}.\, A \mid !\{\phi\}.\, A \mid \exists n.\, A \mid \forall n.\, A$$

Type variable indexed with arith. exps.

Type Grammar

Standard session types

$$A \quad ::= \quad \oplus\{\ell : A\}_{\ell \in L} \mid \&\{\ell : A\}_{\ell \in L} \mid A \otimes A \mid A \multimap A \mid \mathbf{1}$$
$$\mid \quad V[\overline{e}] \mid ?\{\phi\}.\, A \mid !\{\phi\}.\, A \mid \exists n.\, A \mid \forall n.\, A$$

Type variable indexed with arith. exps.

Send / Recv proof constraints

Type Grammar

Standard session types

$$A \quad ::= \quad \oplus\{\ell : A\}_{\ell \in L} \mid \&\{\ell : A\}_{\ell \in L} \mid A \otimes A \mid A \multimap A \mid \mathbf{1}$$

$$\mid \quad V[\overline{e}] \mid ?\{\phi\}.\, A \mid !\{\phi\}.\, A \mid \exists n.\, A \mid \forall n.\, A$$

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!)

Co-inductive Type Equality

- 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 : inc(c_1) ; goto k

$$A_m[c_1, c_2] = \oplus\{\mathbf{inc}_1 : A_k[c_1 + 1, c_2]\}$$

$$A'_m[c_1, c_2] = \oplus\{\mathbf{inc}_1 : A'_k[c_1 + 1, c_2]\}$$

Reduction from 2CM

Decrement Instruction:
m : zero(c₁)? goto k : goto n

$$A_m[c_1, c_2] = \oplus\{\mathbf{zero_1} : ?\{c_1 = 0\}.\, A_k[c_1, c_2], \\ \mathbf{dec_1} : ?\{c_1 > 0\}.\, A_n[c_1 - 1, c_2]\}$$

$$A'_m[c_1, c_2] = \oplus\{\mathbf{zero_1} : ?\{c_1 = 0\}.\, A'_k[c_1, c_2], \\ \mathbf{dec_1} : ?\{c_1 > 0\}.\, A'_n[c_1 - 1, c_2]\}$$

Reduction from 2CM

Halting Instruction:
m : halt

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

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

Reduction from 2CM

Halting Instruction:
m : halt

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

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

A ≠ A'

Reduction from 2CM

Halting Instruction:
m : halt

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

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

A ≠ 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

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

Equality Algorithm

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

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

Equality Algorithm

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

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

Goal: Find a counterexample

Equality Algorithm

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

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

store the equality constraint and expand both types

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

Goal: Find a counterexample

Equality Algorithm

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

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

lt branch

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

Goal: Find a counterexample

Equality Algorithm

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

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

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

Goal: Find a counterexample

Equality Algorithm

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

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

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

Goal: Find a counterexample

Equality Algorithm

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

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

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

Goal: Find a counterexample

Equality Algorithm

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

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

substitute *x+1* for *X*, *y* for *Y*

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

Goal: Find a counterexample

Formal Judgment

$$\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash A \equiv B$$

Formal Judgment

Free variables

$$\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash A \equiv B$$

Formal Judgment

Constraint satisfied by V

Free variables

$$\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash A \equiv B$$

Formal Judgment

Free variables

Constraint satisfied by V

Equality constraints stored

$$\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash A \equiv B$$

Formal Judgment

Free variables

Constraint satisfied by V

Equality constraints stored

$$\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash A \equiv B$$

Types A and B are equal under constraint C

Formal Judgment

Free variables

Constraint satisfied by V

Equality constraints stored

$$\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash A \equiv B$$

Types A and B are equal under constraint C

$$x, y \; ; \; \top \vdash \mathrm{ctr}[x, y] \equiv \mathrm{ctr}[x + 1, y + 1]$$

Closing the Loop

$$\frac{\langle \mathcal{V}' \; ; \; C' \; ; \; V_1[\overline{E_1}] \equiv V_2[\overline{E_2}] \rangle \in \Gamma \qquad \forall \mathcal{V} . C \Rightarrow \exists \mathcal{V}' . C' \wedge \overline{E_1} = \overline{e_1} \wedge \overline{E_2} = \overline{e_2}}{\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]} \; \mathsf{def}$$

Closing the Loop

Already encountered constraint

$$\frac{\langle \mathcal{V}' \; ; \; C' \; ; \; V_1[\overline{E_1}] \equiv V_2[\overline{E_2}] \rangle \in \Gamma \qquad \forall \mathcal{V} . C \Rightarrow \exists \mathcal{V}' . C' \wedge \overline{E_1} = \overline{e_1} \wedge \overline{E_2} = \overline{e_2}}{\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]} \; \mathrm{def}$$

Closing the Loop

Already encountered constraint

$$\frac{\langle \mathcal{V}' \; ; \; C' \; ; \; V_1[\overline{E_1}] \equiv V_2[\overline{E_2}] \rangle \in \Gamma \qquad \forall \mathcal{V} . C \Rightarrow \exists \mathcal{V}' . C' \land \overline{E_1} = \overline{e_1} \land \overline{E_2} = \overline{e_2}}{\mathcal{V} \; ; \; C \; ; \; \Gamma \vdash V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]} \; \text{def}$$

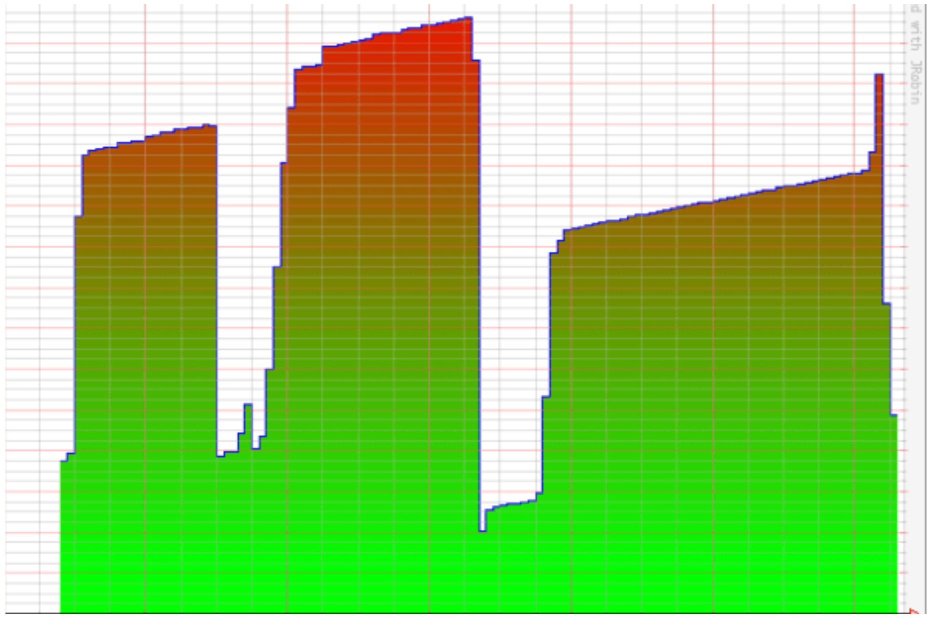
If we know $V_1[E_1] \equiv V_1[E_2]$,
can we prove $V_1[e_1] \equiv V_2[e_2]$?
That is what the second premise achieves!

Rast Programming Language

Goal of Rast

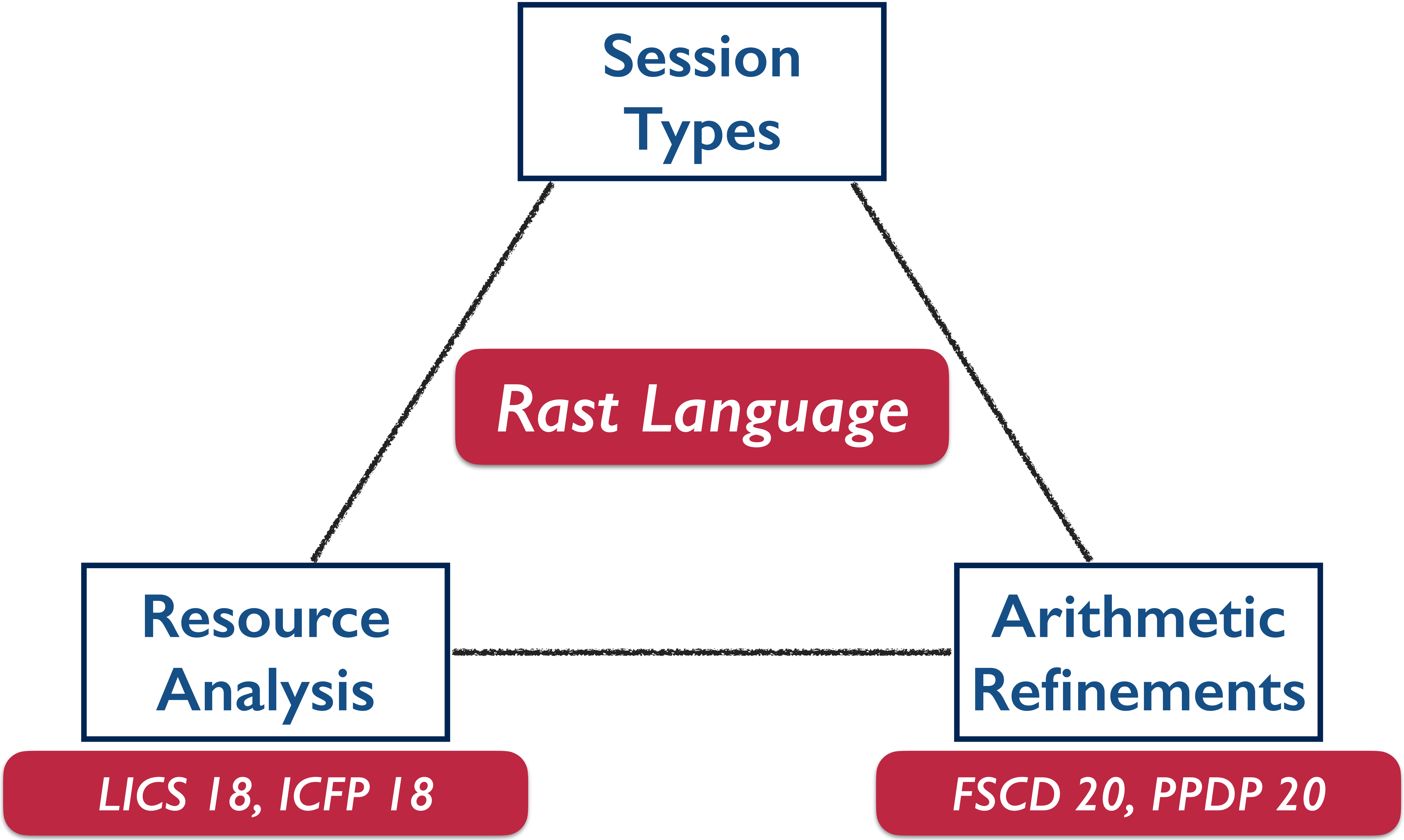
Lightweight Verification and Resource Analysis of Concurrent Programs

Execution Time



Memory Usage

Key Features of Rast



Evaluation

Module	LOC	#Defs	T (ms)
arithmetic	143	8	1.325
integers	114	8	1.074
linlam	67	6	4.003
list	441	29	3.419
primes	118	8	1.646
segments	65	9	0.195
ternary	235	16	1.967
theorems	141	16	0.894
tries	308	9	5.283
Total	**1632**	**109**	**19.806**

Conclusion

Conclusion

Typing System with Arithmetic Refinements

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