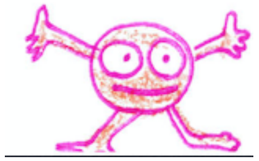
Session-Based Programming and Verifications

Protocol Conformance of Collaborative Federate Query using Multiparty Session Types

Nobuko Yoshida, TASE, China, 30th July 2024

UNIVERSITY OF OXFORD


OXFORD
UNIVERSITY

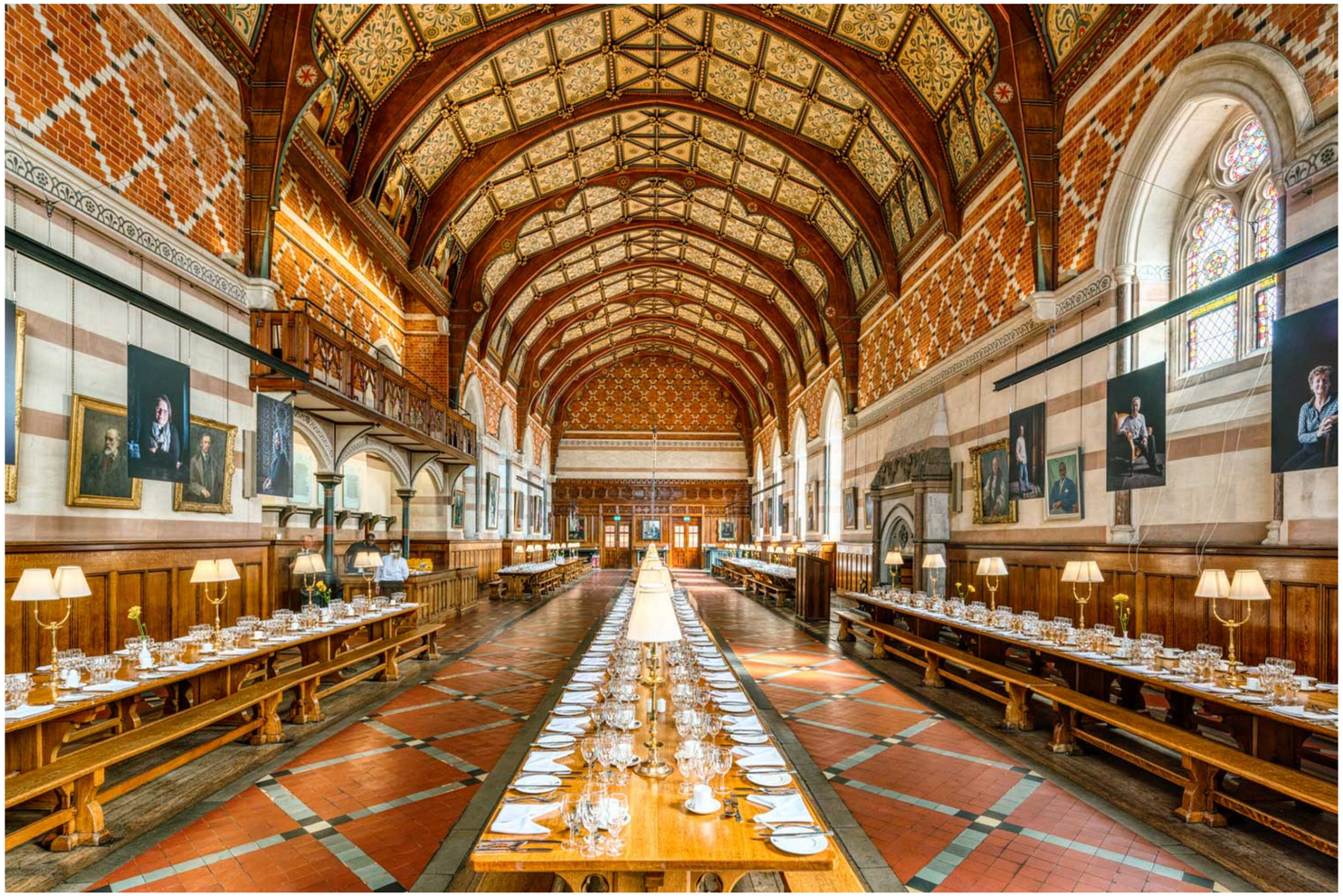
Colleges

University

Departments

39 Colleges

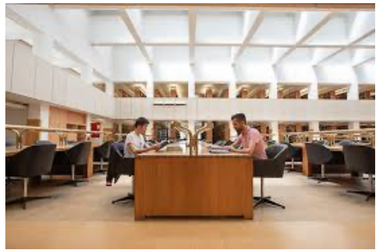
History



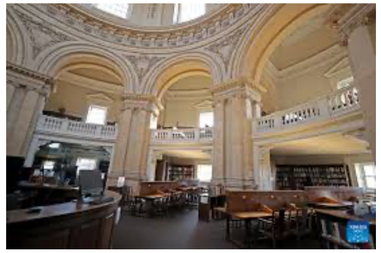
College Library

28 University Libraries

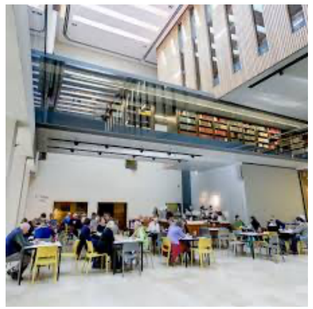


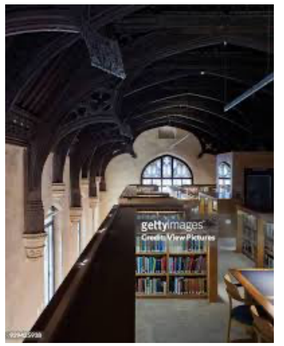











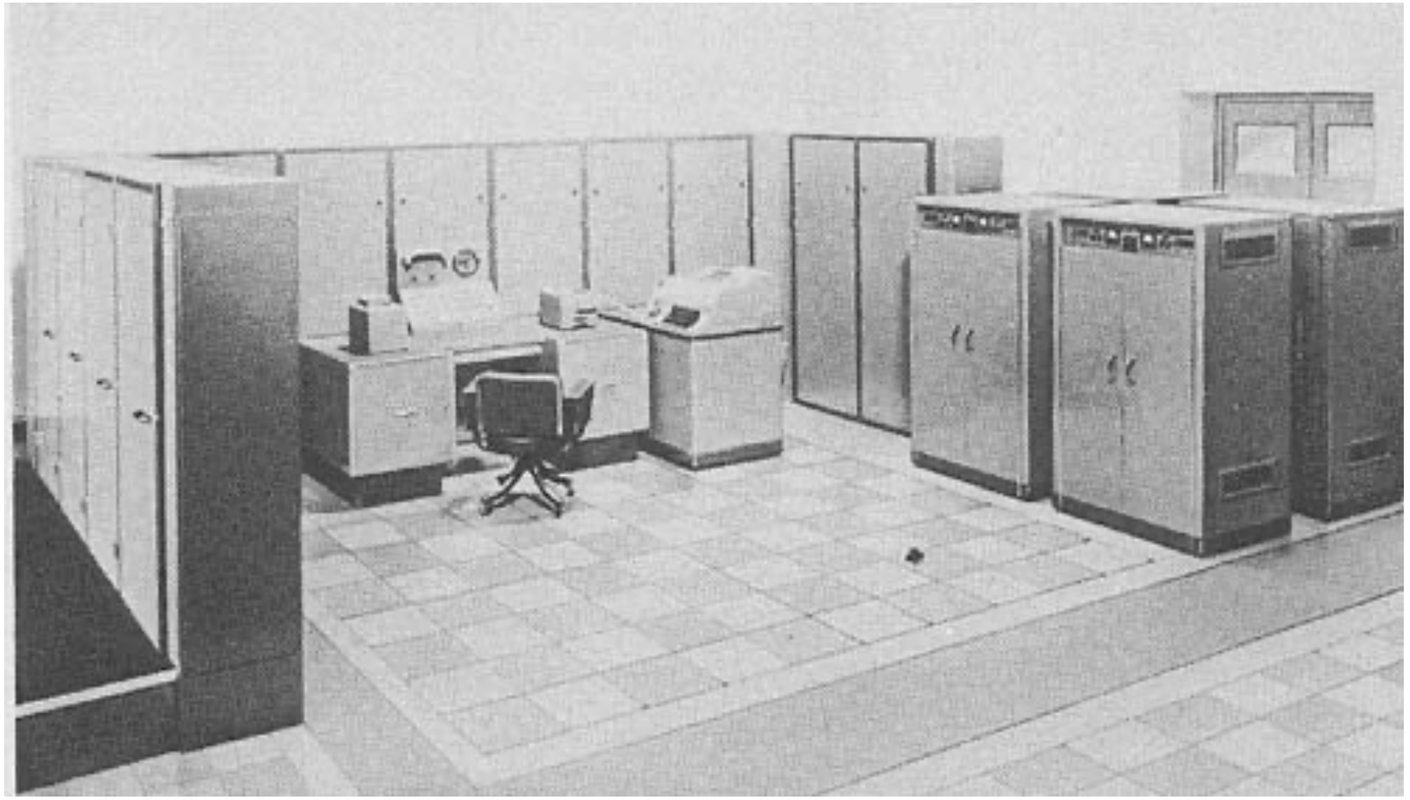



400+ University Clubs and Societies

67% of all graduate students are International

Computer Science Department

History of the Oxford University Computing Laboratory



1956 Oxford buys a Ferranti Mercury

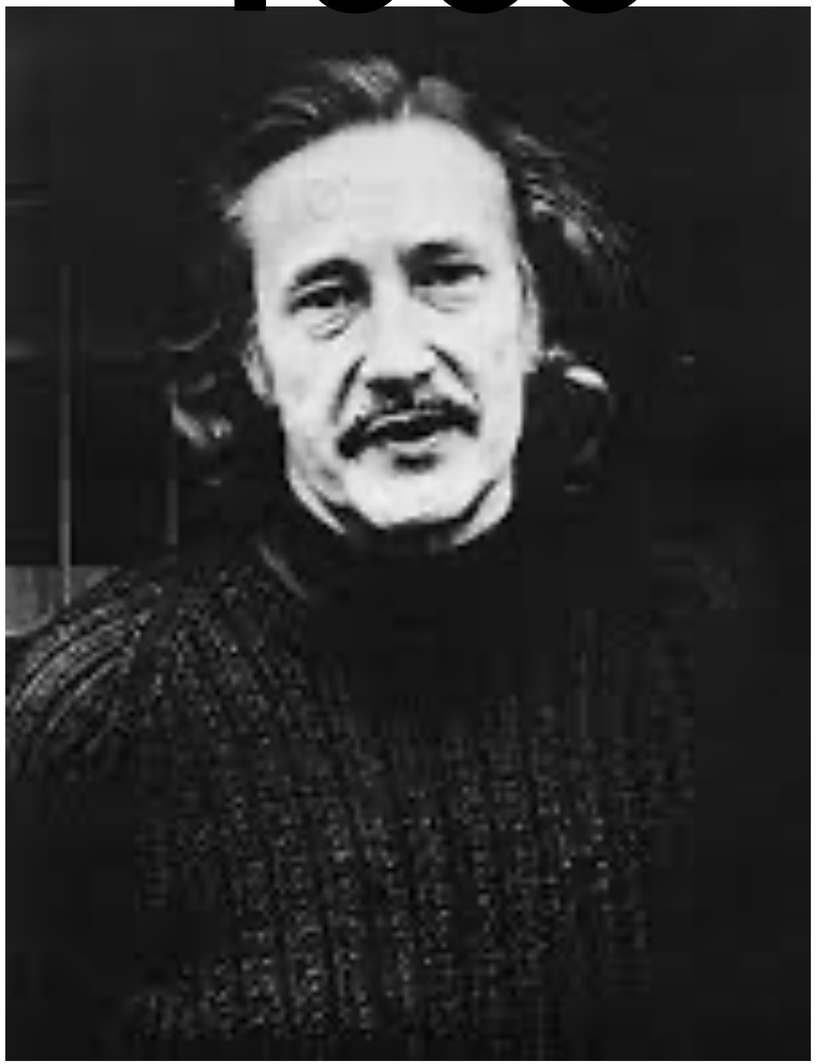
1957 NAG

(Numerical Algorithms Group)

Leslie Fox

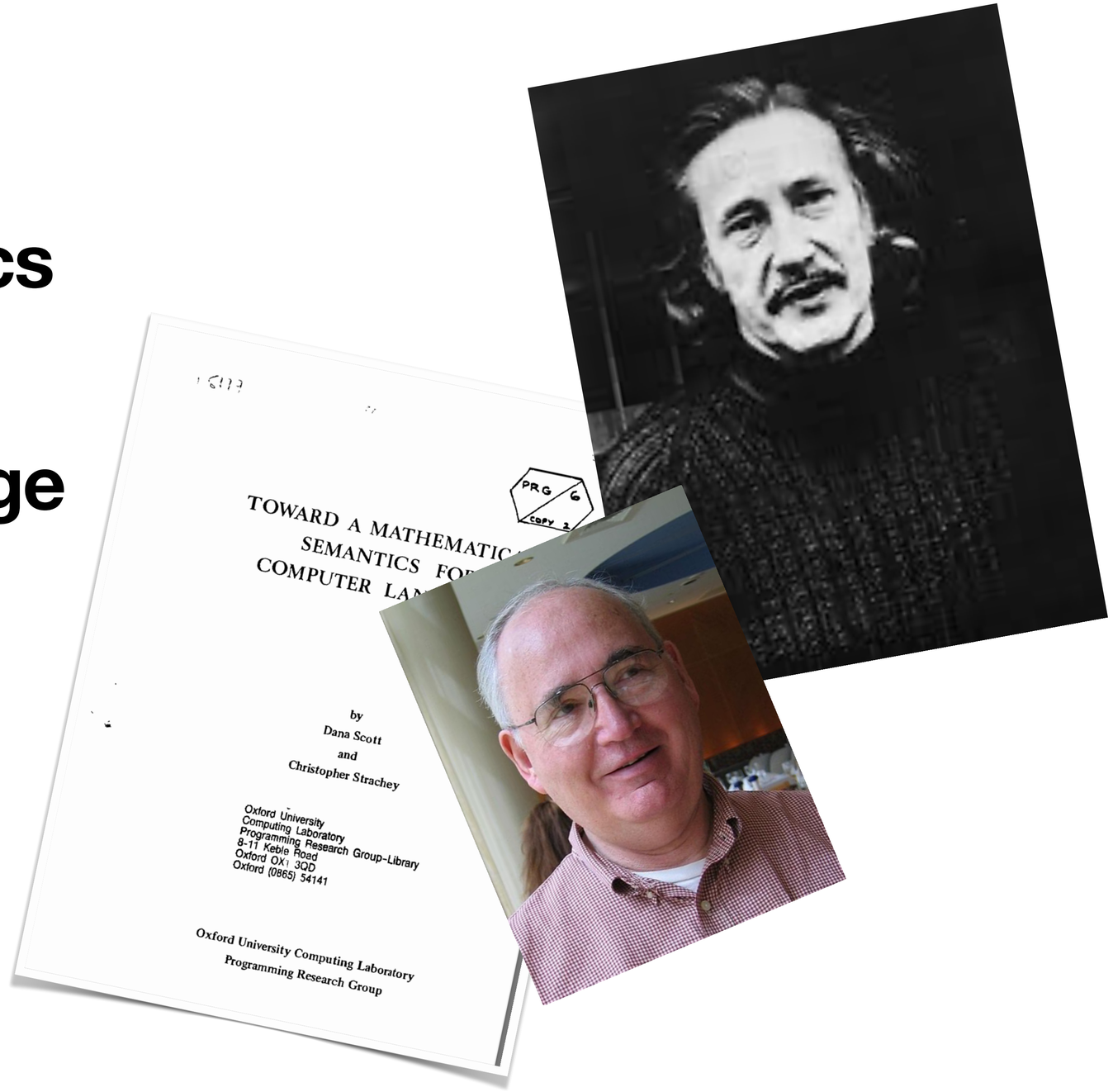
1966 PRG

(Programming Research Group)



CHRISTOPHER STRACHEY

Together with Danna Scott, Strachey's Denotational Semantics Approach has proved foundational to Programming Language Theory



Forms the basis for our research :)

$$[\![\texttt{Int}]\!] = \texttt{Int} \quad [\![\texttt{Bool}]\!] = \texttt{Bool} \quad [\![\texttt{Unit}]\!] = \texttt{Unit}$$

$$[\![\forall[\overline{r};\ \overline{\gamma}](\overline{\tau}) \to\ \rho\ \kappa]\!] = \overline{\forall r} \to \overline{[\![\gamma]\!] \to [\![\tau]\!]} \to [\![\kappa]\!]$$

$$[\![\texttt{Cap}\ \rho\ \tau_1\ \tau_2\ \kappa_1\ \kappa_2]\!] = [\![\tau_1]\!] \to ([\![\tau_2]\!] \to [\![\kappa_1]\!]) \to [\![\kappa_2]\!]$$

$$[\![\tau / \square]\!] = \forall a.(([\![\tau]\!] \to a) \to a)$$

$$[\![\tau / \kappa_1 \Rightarrow \kappa_2]\!] = ([\![\tau]\!] \to [\![\kappa_1]\!]) \to [\![\kappa_2]\!]$$

$$[\![\rho_1 : \kappa_{11} \Rightarrow \kappa_{12} \sqsubseteq \rho_2 : \kappa_{21} \Rightarrow \kappa_{22}]\!] = \forall a.((a \to [\![\kappa_{21}]\!]) \to [\![\kappa_{22}]\!]) \to ((a \to [\![\kappa_{11}]\!]) \to [\![\kappa_{12}]\!])$$

$$[\![n]\!] = n$$

$$[\![0]\!] = \Lambda a.\lambda m.m$$

$$[\![e_1 \oplus e_2]\!] = \Lambda a.\lambda m.e_1\ a\ (e_2\ a\ m)$$

$$[\![\Gamma \mid \rho \vdash \texttt{return}\ v : \kappa]\!] = \lambda k.k\ [\![v]\!]$$

$$[\![\Gamma \mid \rho \vdash \texttt{val}\ x = s_0;\ s_1 : \kappa]\!] = \lambda k.[\![x_0]\!]\ (\lambda x.[\![s]\!]\ k)$$

$$[\![\Gamma \mid \rho \vdash v_0[\rho_0;\ e](v) : \kappa]\!] = [\![v_0]\!]\ \overline{[\![\rho_0]\!]}\ \overline{[\![e]\!]}\ \overline{[\![v]\!]}$$

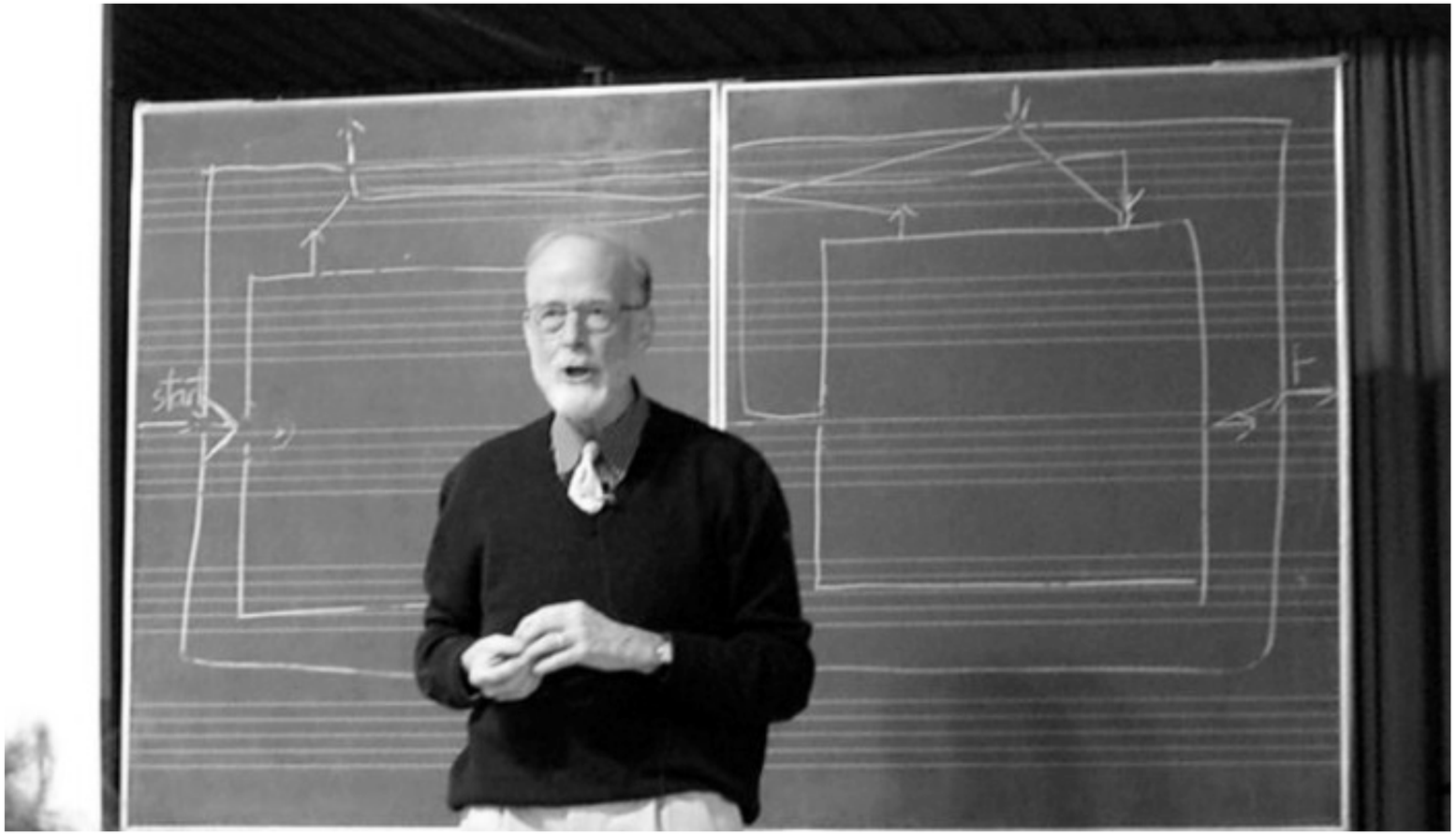
$$[\![\Gamma \mid \rho \vdash \texttt{do}\ v_0[e](v) : \kappa]\!] = [\![e]\!]\ [\![\kappa]\!]\ ([\![v_0]\!]\ [\![v_1]\!])$$

$$[\![\Gamma \mid \rho \vdash \begin{array}{l}\texttt{try}\ \{[r,n](\overline{c}) \Rightarrow s_0\}\ \texttt{with} \\ \left\{\overline{c(x,\ k) \Rightarrow s_1},\ \texttt{return}\ y \Rightarrow s_2\right\}\end{array} : \kappa]\!] = (\Lambda r.\lambda n.\overline{\lambda c}.[\![s_0]\!])\ [\![\kappa]\!]\ (\text{LIFT})\ \overline{(\lambda x.\lambda k.[\![s_1]\!])}\ (\lambda y.\lambda k.k\ [\![s_2]\!])$$

$$[\![x]\!] = x \quad [\![1]\!] = 1$$

$$[\![\{[\overline{r};\overline{n\ :\ \gamma}](\overline{x\ :\ \tau})\ \texttt{at}\ \rho \Rightarrow s\}]\!] = \overline{\Lambda r}.\overline{\lambda n}.\overline{\lambda x}.[\![s]\!]$$

Tony Hoare (Takes Over PRG 1977-1999)



Hoare was a prolific computer scientist. His biggest achievements are Hoare Logic & CSP, but he is also known for quicksort (and, infamously, null pointers)

An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

Abstract. In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

Key words and phrases: axiomatic method, theory of programming, proving properties of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR Categories: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

1. Introduction

Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning. Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs. The exact choice of axioms will to some extent depend on the choice of programming language. For illustrative purposes, this paper is confined to a very simple language, which is effectively a subset of all current procedure-oriented languages.

2. Computer Arithmetic

The first requirement in valid reasoning about a program is to know the properties of the elementary operations which it invokes, for example, addition and multiplication of integers. Unfortunately, the set of numbers familiar to arithmetic is not the same in several respects as the set of numbers manipulated by computers. Their truth is largely independent of the size of the set; however, it is largely dependent on the choice of the set. Furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never exists.
(2) Firm boundary: the result of an overflowing operation is computed modulo the size of the set of integers represented.
(3) Modulo arithmetic: the result of an overflowing addition and multiplication, tables for a trivially small model in which 0, 1, 2, and 3 are the only integers represented.

It is interesting to note that the different systems satisfying axioms A1 to A9 may be rigorously distinguished from each other by choosing a particular one of a set of mutually exclusive supplementary axioms. For example, infinite arithmetic satisfies the axiom:

$$A10_\beta \quad \neg\exists x \forall y \quad (y \leq x),$$

where all finite arithmetics satisfy:

$$A10_\alpha \quad \forall x \quad (x < \max)$$

where "max" denotes the largest integer represented. Similarly, the three treatments of overflow may be distinguished by a choice of one of the following axioms relating to the value of $\max + 1$:

$$A11_\alpha \quad \neg\exists x \quad (x = \max + 1) \qquad \text{(strict interpretation)}$$

$$A11_\beta \quad \max + 1 = \max \qquad \text{(firm boundary)}$$

$$A11_\gamma \quad \max + 1 = 0 \qquad \text{(modulo arithmetic)}$$

Having selected one of these axioms, it is possible to use it in defining the properties of programs; however,

of axioms it is possible to deduce such simple

$$x = x + y \times 0$$

$$y < r \supset r + y \times q = (r - y) + y \times$$

The proof of the second of these is:

A5 $(r - y) + y \times (1 + q)$

A9 $= (r - y) + (y \times 1 + y$

A3 $= (r - y) + (y + y \times q)$

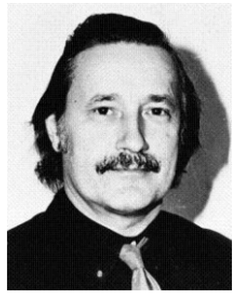
A6 $= ((r - y) + y) + y \times q$

$= r + y \times q$ provided $y \leqslant r$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to nonnegative numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

* Department of Computer Science

Christopher Strachey



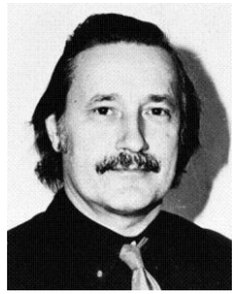
1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

Fundamental Concepts of
Programming Languages

Christopher Strachey



1916-1975

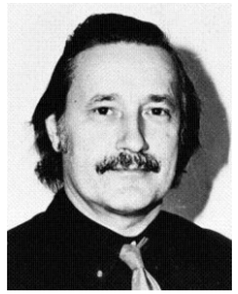
Computer Games, Literature and Music

A schoolmaster at Harrow

- Sequential and functional computation

Fundamental Concepts of Programming Languages

Christopher Strachey



1916-1975

Computer Games, Literature and Music

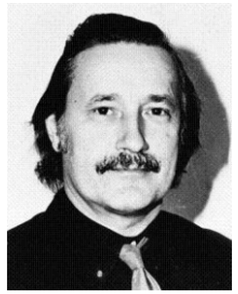
A schoolmaster at Harrow

- Sequential and functional computation

Fundamental Concepts of Programming Languages

  - ***Types*** = **abstract** and **digest computation (data types, polymorphism)**

Christopher Strachey



1916-1975

Computer Games, Literature and Music

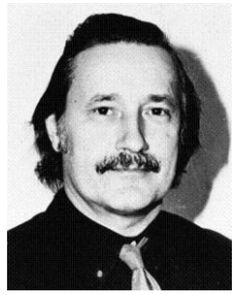
A schoolmaster at Harrow

- Sequential and functional computation

Fundamental Concepts of Programming Languages

  - *Types* = **abstract** and **digest computation (data types, polymorphism)**
  - **Structured programming = High-level programming**

Christopher Strachey



1916-1975

Computer Games, Literature and Music

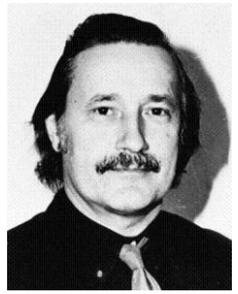
A schoolmaster at Harrow

- Sequential and functional computation

  Fundamental Concepts of Programming Languages

  - *Types* = **abstract** and **digest computation (data types, polymorphism)**
  - **Structured programming = High-level programming**
- **Session types** (concurrency & communication)

Christopher Strachey



1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

- Sequential and functional computation

  Fundamental Concepts of Programming Languages

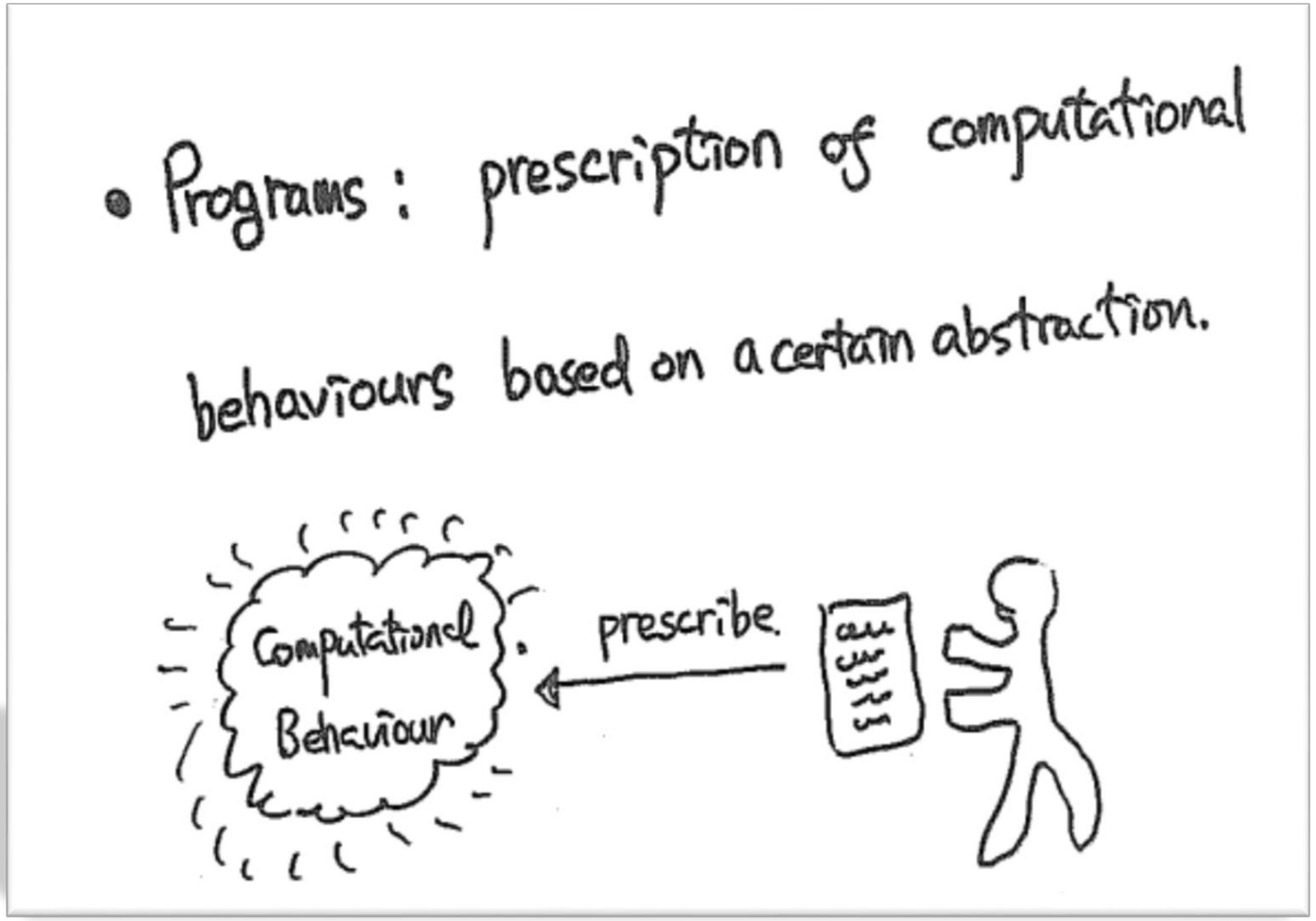
  - ***Types*** = **abstract** and **digest computation (data types, polymorphism)**
  - **Structured programming = High-level programming**
- **Session types** (concurrency & communication)
  - Structured programming = ***protocols***

Idioms for Interaction

—Invitation to Hacking on π-calculus—

Programming languages are tools which offer frameworks of abstraction for such activities – promoting or limiting them

- *Imperative*
- *Functional*
- *Logical*



On Programs and Programming

- The most fundamental element of a PL

  in this context is a set of operations

  it is based on:

  Imperative: assignment, jump.

  Functional: β-reduction.

  Logical : unification.

- Another element is how we can combine,

  or structure, these operations:

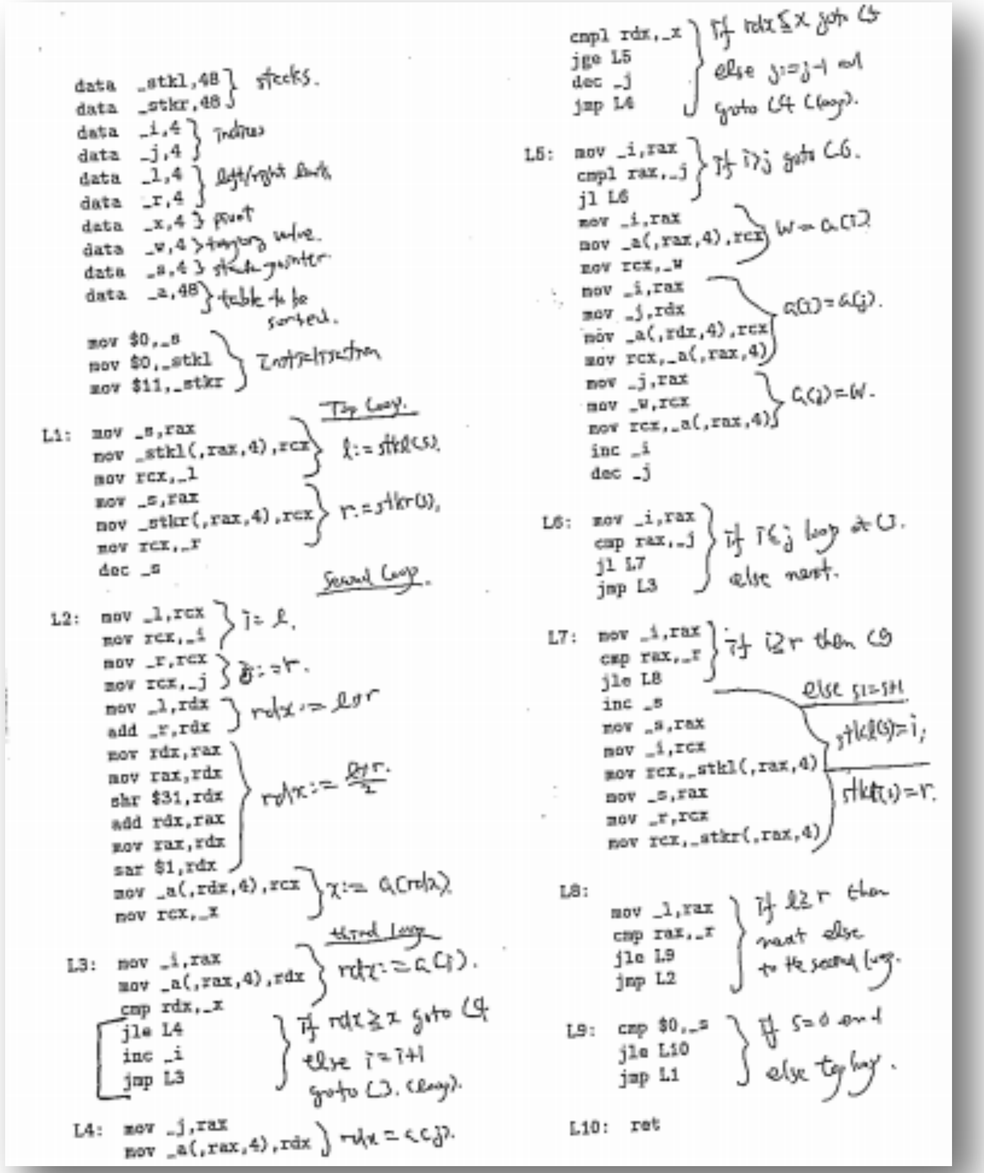
  Imperative: sequential composition, if-then-else,

  while, procedures, modules, ...

  Functional: application, product, union,

  recursion, modules, ...

UNSTRUCTURED:

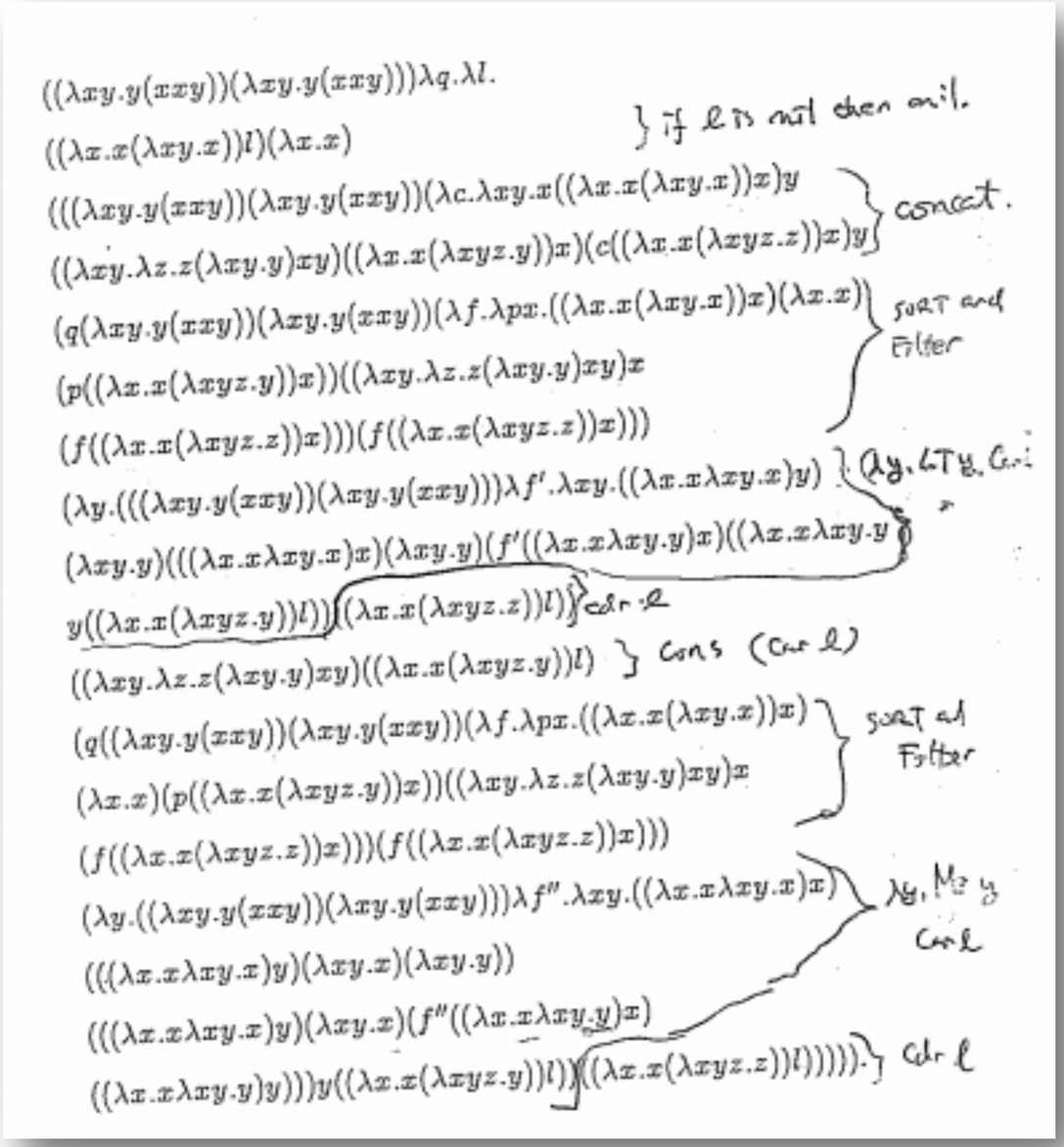


```
data _stkl,48 } stacks.
data _stkr,48 }
data _i,4 } indices
data _j,4 }
data _l,4 } left/right bounds
data _r,4 }
data _x,4 } pivot
data _w,4 } temporary value.
data _s,4 } stack pointer
data _a,48 } table to be sorted.

mov $0,_s
mov $0,_stkl } initialization
mov $11,_stkr

                    Top Loop.
L1: mov _s,rax
    mov _stkl(,rax,4),rcx } l := stkl(s).
    mov rcx,_l
    mov _s,rax
    mov _stkr(,rax,4),rcx } r := stkr(s).
    mov rcx,_r
    dec _s
                    Second Loop.
L2: mov _l,rcx } i := l.
    mov rcx,_i
    mov _r,rcx } j := r.
    mov rcx,_j
    mov _l,rdx } rdx := l or r
    add _r,rdx
    mov rdx,rax
    mov rax,rdx
    shr $31,rdx
    add rdx,rax
    mov rax,rdx
    sar $1,rdx } rdx := (l or r)/2
    mov _a(,rdx,4),rcx } x := a(rdx)
    mov rcx,_x
                    third loop
L3: mov _i,rax
    mov _a(,rax,4),rdx } rdx := a(i).
    cmp rdx,_x
    jle L4 } if rdx ≥ x goto L4
    inc _i  } else i := i+1
    jmp L3  } goto L3. (loop).
L4: mov _j,rax
    mov _a(,rax,4),rdx } rdx := a(j).
    cmpl rdx,_x } if rdx ≤ x goto L5
    jge L5      } else j := j-1 and
    dec _j      } goto L4 (loop).
    jmp L4

L5: mov _i,rax
    cmpl rax,_j } if i ≥ j goto L6.
    jl L6
    mov _i,rax
    mov _a(,rax,4),rcx } w := a(i)
    mov rcx,_w
    mov _i,rax
    mov _j,rdx
    mov _a(,rdx,4),rcx } a(i) := a(j).
    mov rcx,_a(,rax,4)
    mov _j,rax
    mov _w,rcx } a(j) := w.
    mov rcx,_a(,rax,4)
    inc _i
    dec _j

L6: mov _i,rax
    cmp rax,_j } if i < j loop at L3.
    jl L7      } else next.
    jmp L3

L7: mov _i,rax
    cmp rax,_r } if i ≥ r then L8
    jle L8
    inc _s                  else s := s+1
    mov _s,rax              stkl(s) := i;
    mov _i,rcx              stkr(s) := r.
    mov rcx,_stkl(,rax,4)
    mov _s,rax
    mov _r,rcx
    mov rcx,_stkr(,rax,4)

L8:
    mov _l,rax } if l ≥ r then
    cmp rax,_r } next else
    jle L9     } to the second loop.
    jmp L2

L9: cmp $0,_s } if s = 0 end
    jle L10   } else top loop.
    jmp L1

L10: ret
```

STRUCTURED:

```
Var a: array[MAX] of int;

Procedure sort(l,r: int);
  Var i,j, x: int;
  i := l; j := r;
  x := [(l+r) div 2];          • Choose a pivot.
  repeat
    while a[i]<x do i := i+1 end
    while a[j]>x do j := j+1 end.   } Partition into two parts.
    if i≤j then swap(i,j); i := i+1; j := j-1; end
  until i>j;
  if l<j then sort(l,j);        } Recursively
  if l<r then sort(i,r);        } sort two parts.
end

Procedure swap(i,j: int)
    var w: int;
    w := a[i]; a[i] := a[j]; a[j] := w;
end
```

Quicksort in pure lambda:



Quicksort with combinators:

```
Y (λf. λl.
    (Isnil l)
    (Concat (f Filter (λy. LTy (Car l))
                      (Cdr l))
            (Cons (Car l)
                  (f Filter (λy. Mey
                                 (Car l)
                                 (Cdr l))))))
```

I = λx.x T = λxy.x F = λxy.y Y = (λxy.y(xxy))(λxy.y(xxy))

Cons = λxyz.zFxy Isnil = λx.xT

Car = λx.x(λxyz.y) Cdr = λx.x(λxyz.z)

Concat = Y (λc.λxy. x (Isnil x) y (Cons (Car x)(c(Cdr x)y)

Filter = Y(λf. λpx.(Isnil x) I(p(Car x))(Cons x (Filter Cdr x))
                                                (Filter (Cdr x))

Iszero = λx. xT Pred = λx.xF

LT = Y (λf. λxy. (Iszero y) F ((Iszero x) F (f (Pred x)(Pred y))

ME = Y (λf. λxy. (Iszero x)((Iszero y) I F)((Iszero y)(T) y))

Quicksort in ML:

```
fun qs nil:int list = nil
  | qs (x::r) = let val small =
                        filter (fn y => y<x) r
                    and large =
                        filter (fn y => y>=x) r
                in qs small @ [x] @ qs large
                end

fun filter p nil = nil
  | filter p (x::r) =
      if p x then x:: filter p r
             else filter p r
```

What is a concept of *Types* in Programming?

- Types can avoid runtime error

Boolean, Natural Number

100 x 200

100 x true

What is a concept of *Types* in Programming?

- Types can avoid runtime error

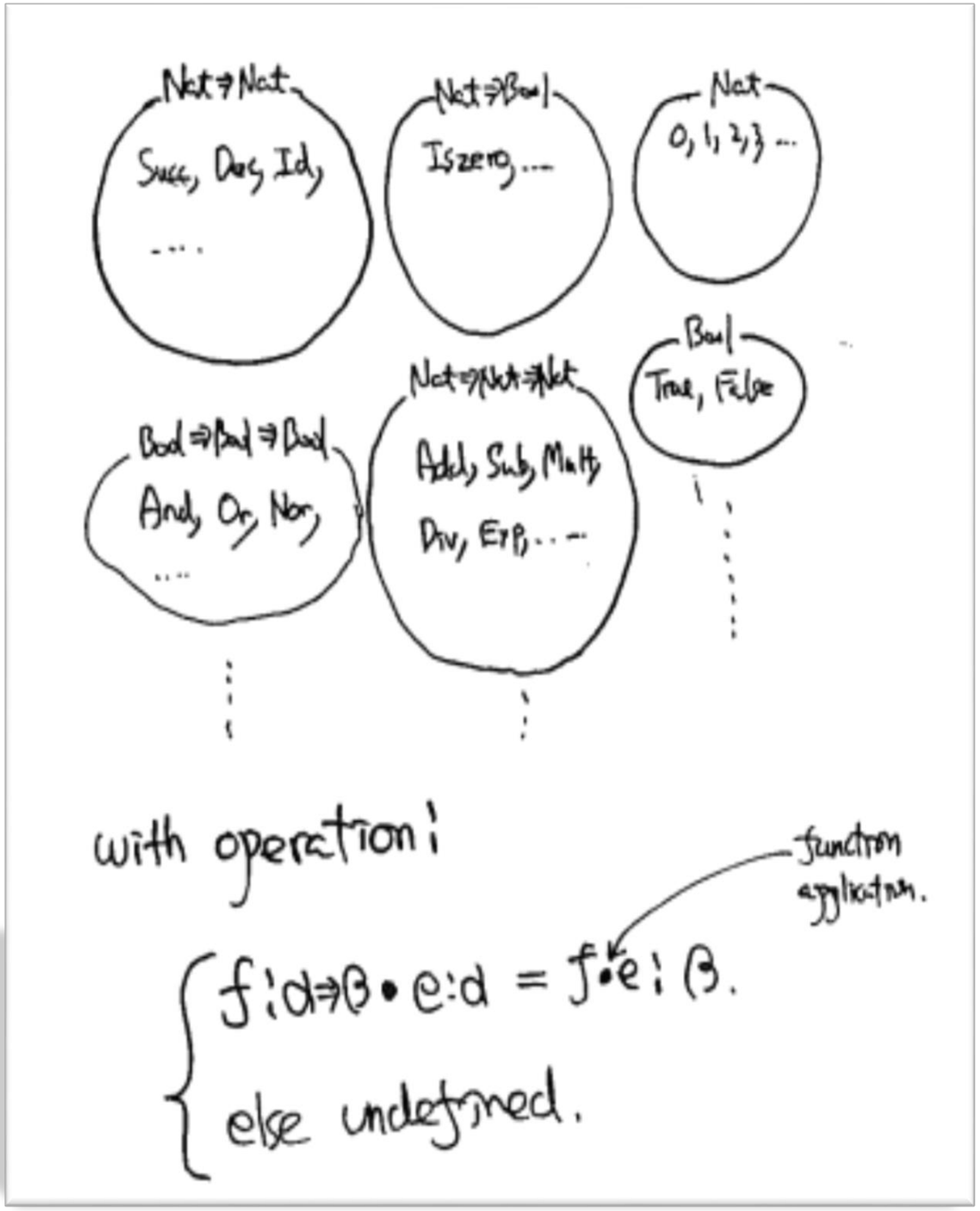
Boolean, Natural Number (Data Types)

100 x 200 **Correct**

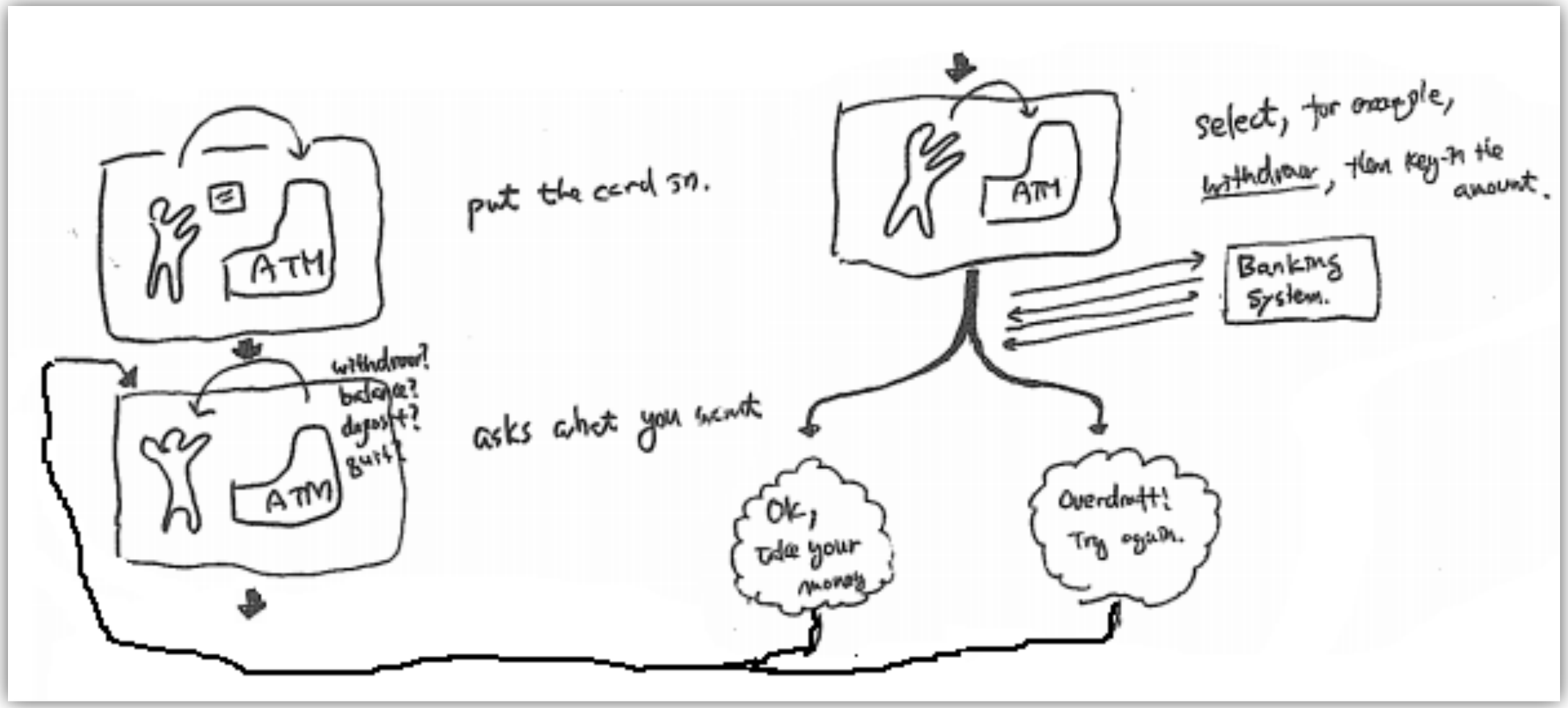
100 x true **Wrong**

Types for *Protocol*?

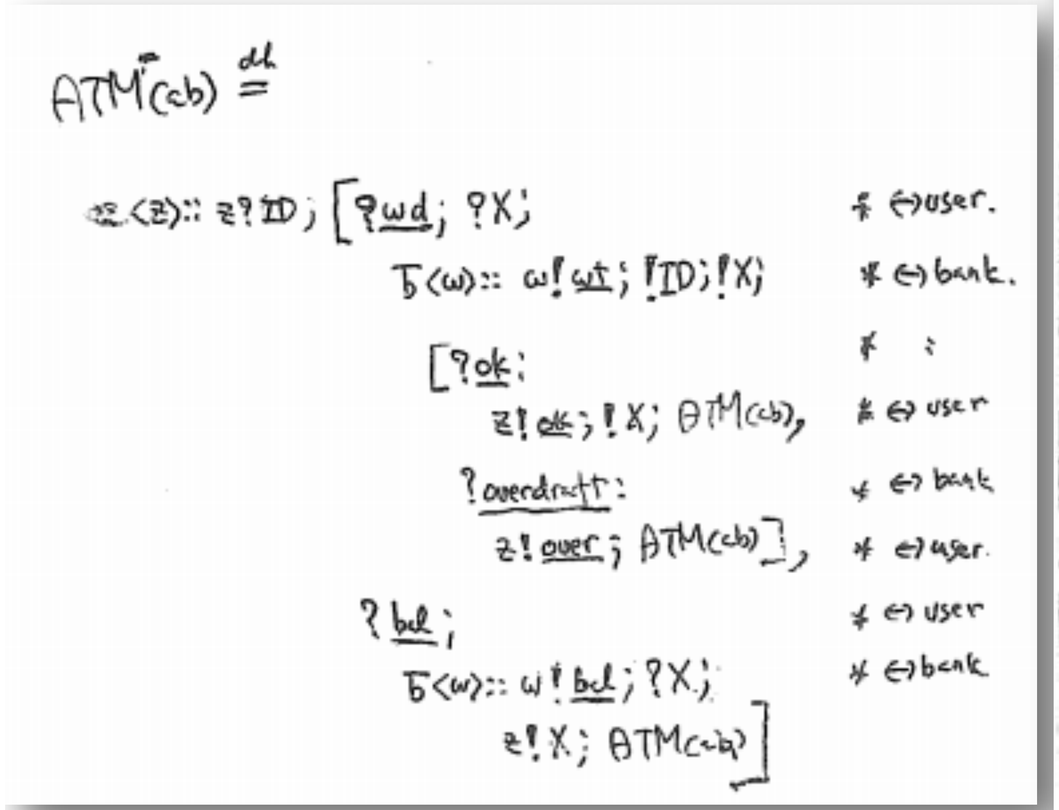
Functional Types

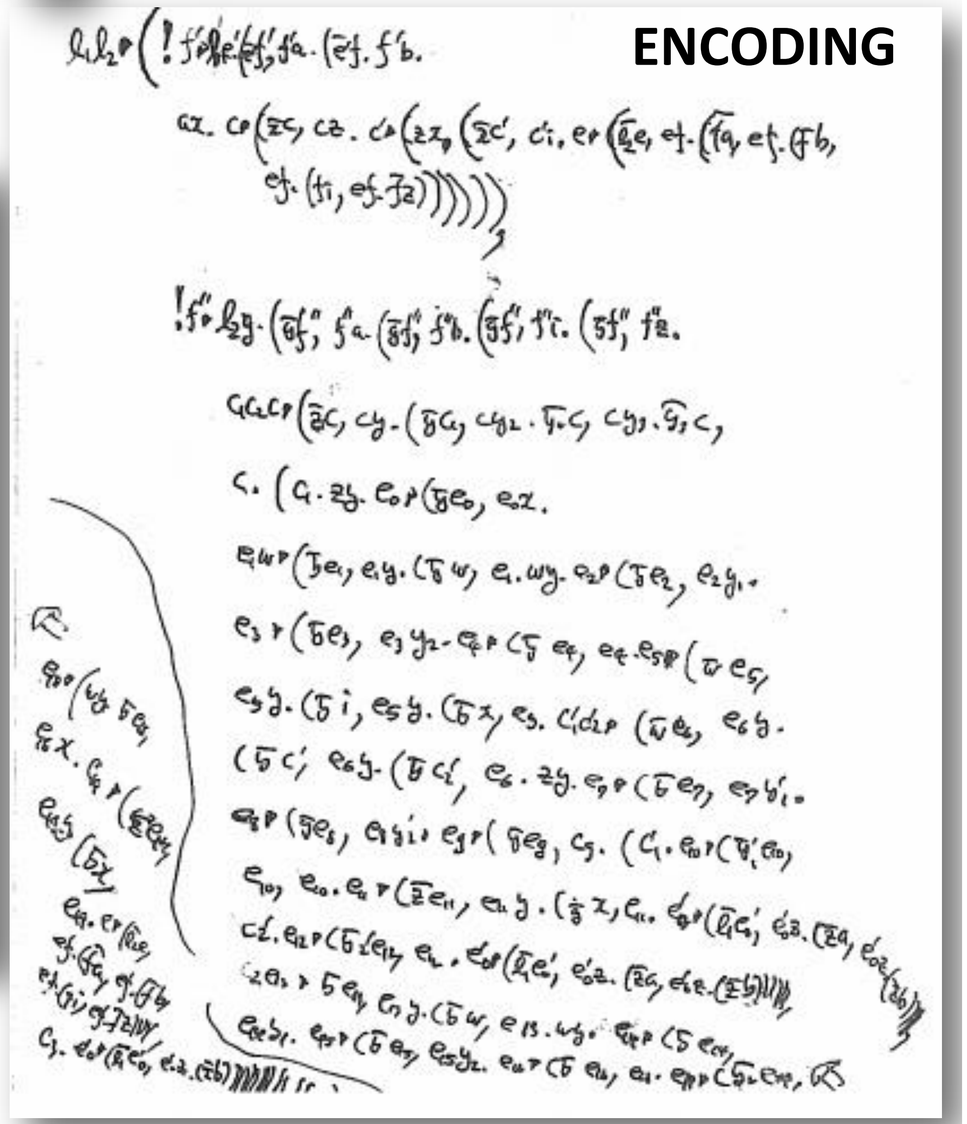


Implementing ATM



Implementing ATM





Communications are Ubiquitous

- Increasingly, **communications** are the way to organise software and systems.
- Industry trend – programming languages with **explicit message-passing primitives.**

microservices

Problems: Ambiguity

- Protocol descriptions are **ambiguous**
- **SMTP: simple mail transfer protocol**
  - They are written in English, often very long

RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS

Problems: Ambiguity

- Protocol descriptions are **ambiguous**
- **SMTP: simple mail transfer protocol**
  - They are written in English, often very long

3.1. MAIL

There are three steps to SMTP mail transactions. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follows giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

MAIL <SP> FROM:<reverse-path> <CRLF>

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply.

The <reverse-path> can contain more than just a mailbox. The <reverse-path> is a reverse source routing list of hosts and source mailbox. The first host in the <reverse-path> should be the host sending this command.

The second step in the procedure is the RCPT command.

RCPT <SP> TO:<forward-path> <CRLF>

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times.

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4k users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]

The Go Gopher

Do not communicate by sharing memory; share memory by communicating

– Go Philosophy

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4K users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]

deadlock

channel errors

More than a half of concurrency bugs in Go are caused by communications.

The Go Gopher

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4k users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]

More than a half of concurrency bugs in Go are caused by communications.

Session Types

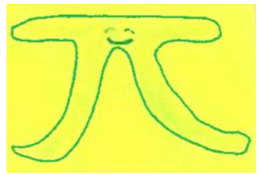
- Prevent concurrency bugs.
- Can abstract, implement and manage communications as **Protocols.**
- **Clean, Cheap** and **Retrofittable.**

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Milner, Honda, NY

Binary Session Types — ESOP'98



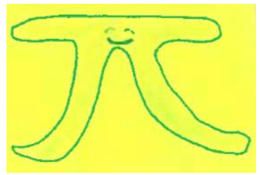
↓

Joined W3C Standardization — 2002

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

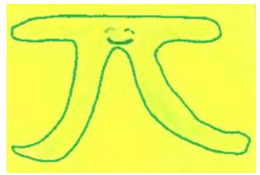
Choreography Description Language (W3C)

```
package HelloWorld {
    roleType YouRole, WorldRole;
    participantType You{YouRole}, World{WorldRole};
    relationshipType YouWorldRel between YouRole and WorldRole;
    channelType WorldChannelType with roleType WorldRole;

    choreography Main {
        WorldChannelType worldChannel;

        interaction operation=hello from=YouRole to=WorldRole
                relationship=YouWorldRel channel=worldChannel {
            request messageType=Hello;
        }
    }
}
```

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

↓

largest open source company in the world

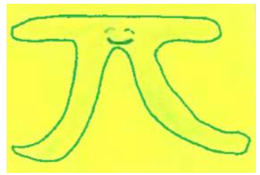
Red Hat

Scribble

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08

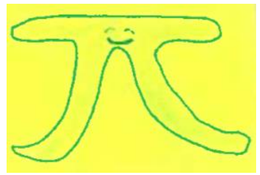
25M US$ large cyber infrastructures

middleware standardisations

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08







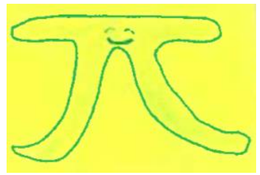



Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

POPL Influential Paper Award 2018

Multiparty Session Types — POPL' 08

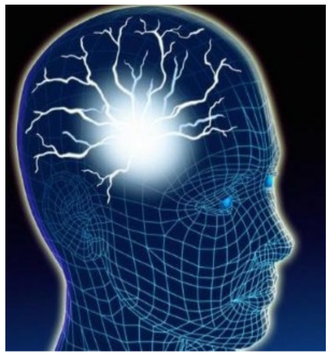
Distributed systems are:

focus on the

communication

not on

computation



Types for distributed processes

- Primitives – send, receive, internal and external choice, recursion
- How to combine types (hint: specify the communication flow):

send(int).receive(bool)

- What kind of errors does this prevent?
  - communication errors = communication mismatch + deadlock
- Challenge – when types are compatible ?

send(int).
send(int).
receive(bool)

receive(int).
receive(int).
send(bool)

Types for distributed processes

- Primitives – send, receive, internal and external choice, recursion
- How to combine types (hint: specify the communication flow):

send(int).receive(bool)

- What kind of errors does this prevent?
  - communication errors = communication mismatch + deadlock
- Challenge – when types are compatible ?

send(int).
send(nat).
receive(bool)

Type Error!

receive(int).
receive(int).
send(bool)

Types for distributed processes

- Primitives – send, receive, internal and external choice, recursion
- How to combine types (hint: specify the communication flow):

send(int).receive(bool)

- What kind of errors does this prevent?
  - communication errors = communication mismatch + deadlock
- Challenge – when types are compatible ?

send(int).
receive(int).
receive(bool)

Protocol Error!

receive(int).
receive(int).
send(bool)

Types for distributed processes

- Primitives – send, receive, internal and external choice, recursion
- How to combine types (hint: specify the communication flow):

send(int).receive(bool)

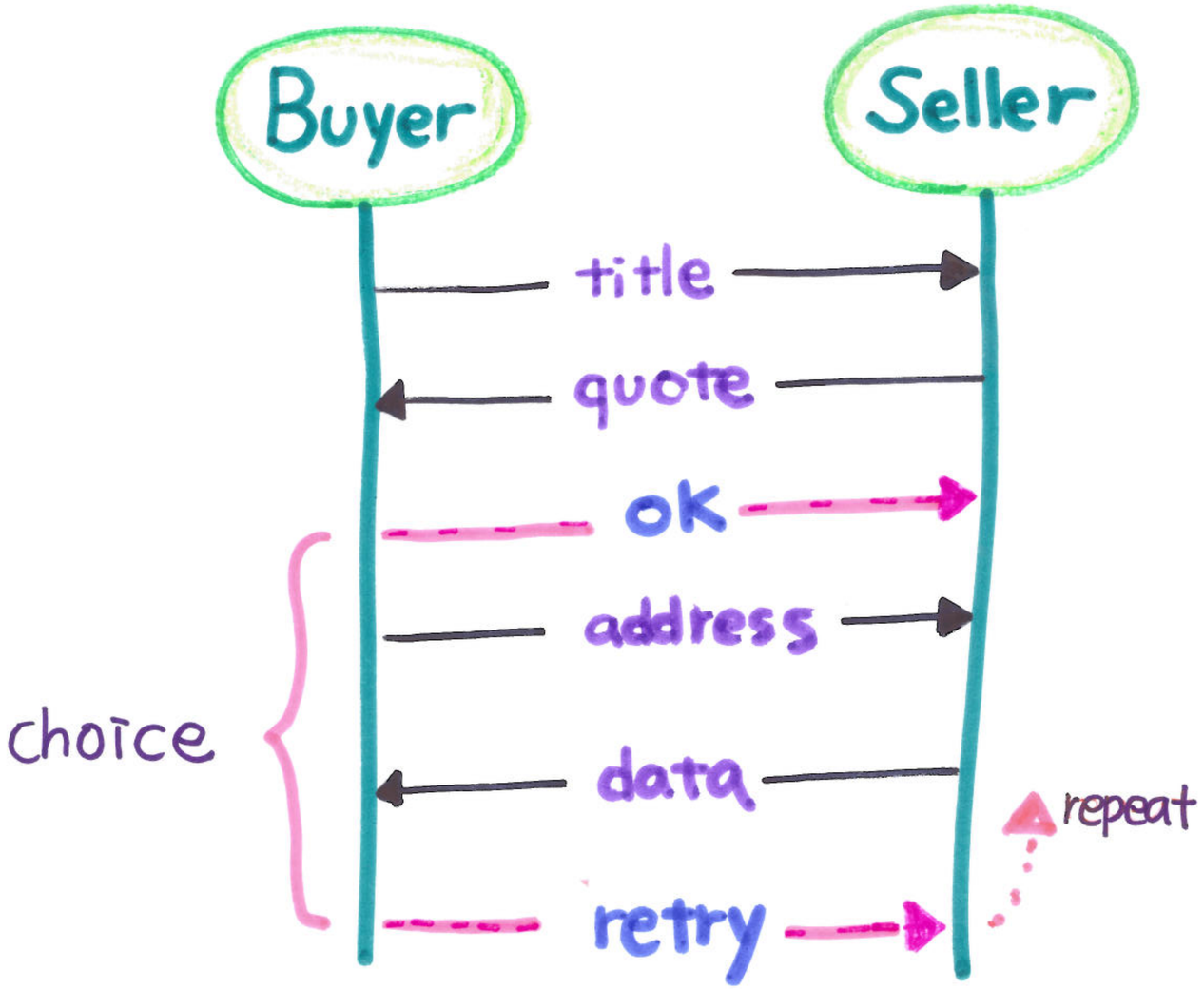
- What kind of errors does this prevent?
  - communication errors = communication mismatch + deadlock
- Challenge – when types are compatible ?

send(int).
send(int).
~~**receive(bool)**~~

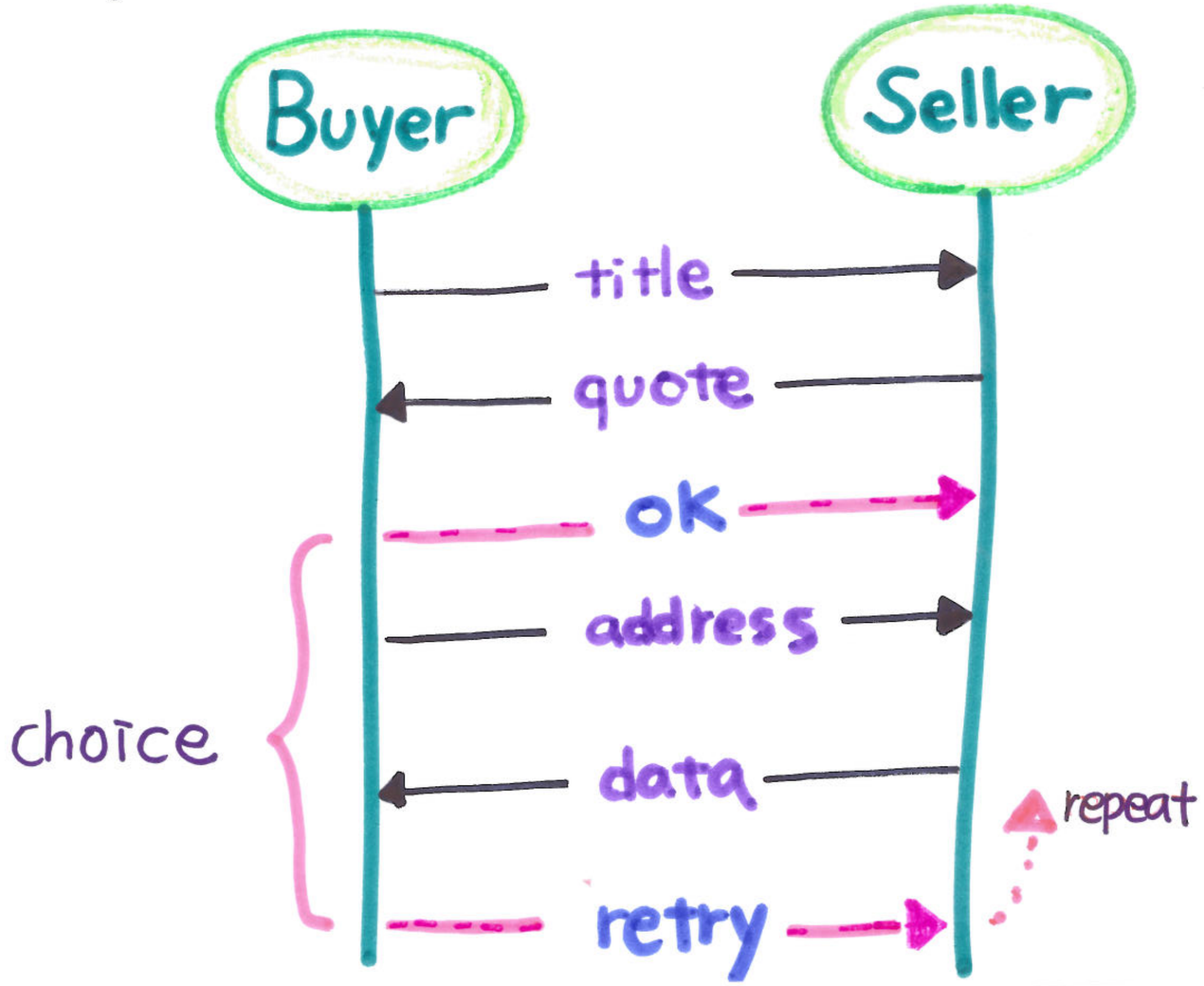
Deadlock!

receive(int).
receive(int).
send(bool)

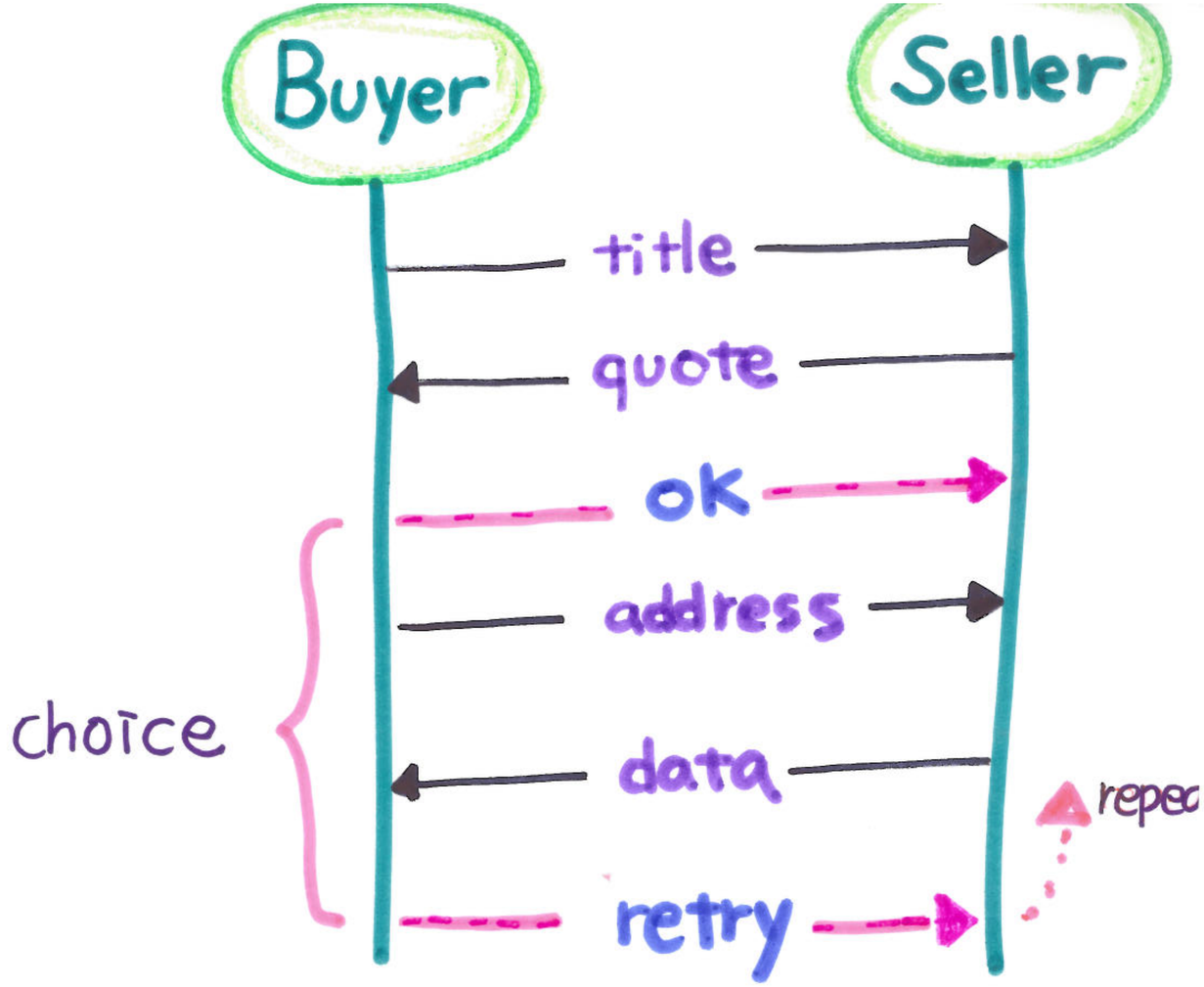
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



μt. ! Title ; ? Quote ; ! { ok : ! Add ; ? Date , retry : t }



P has T

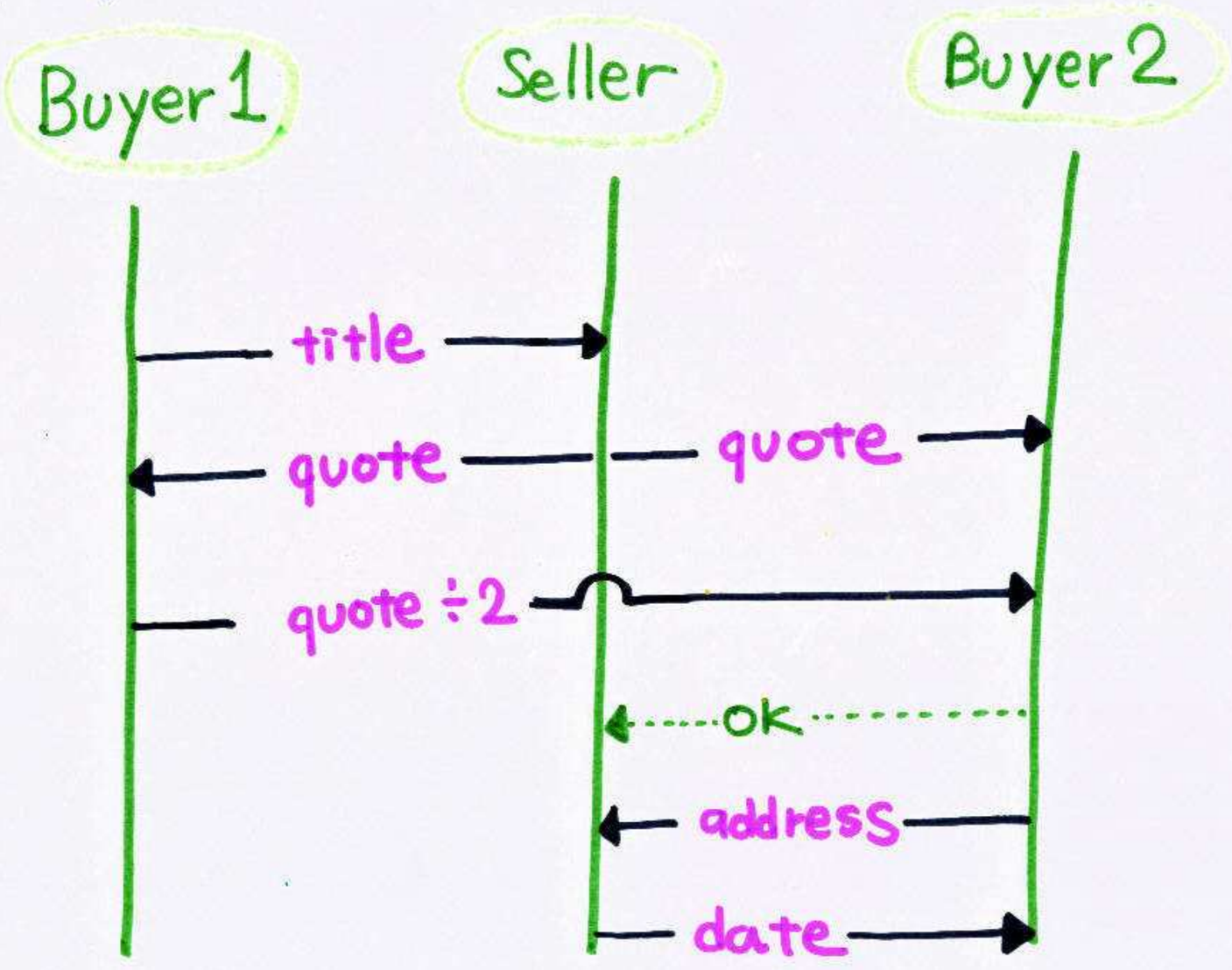
Q has $\overline{T}$ dual

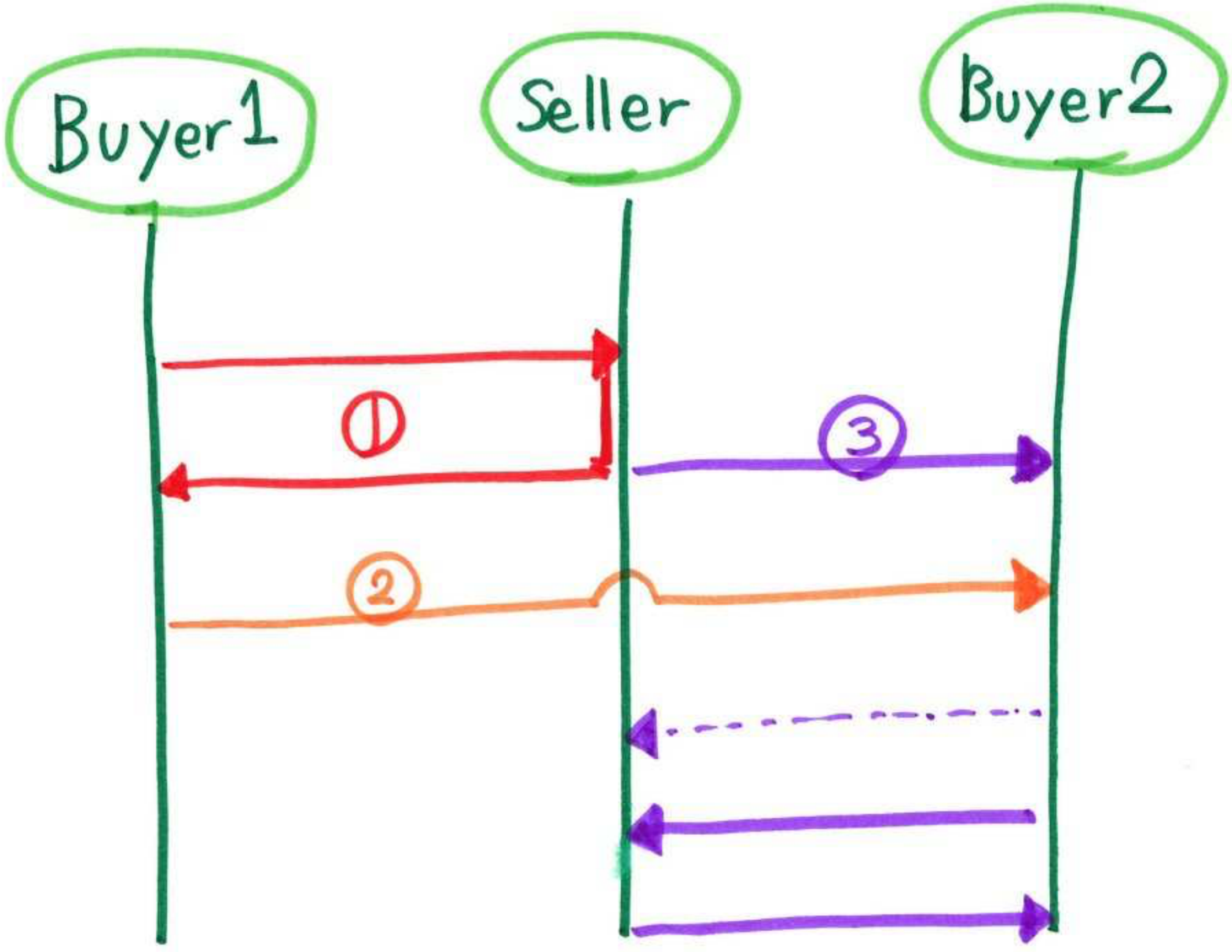
P | Q typable

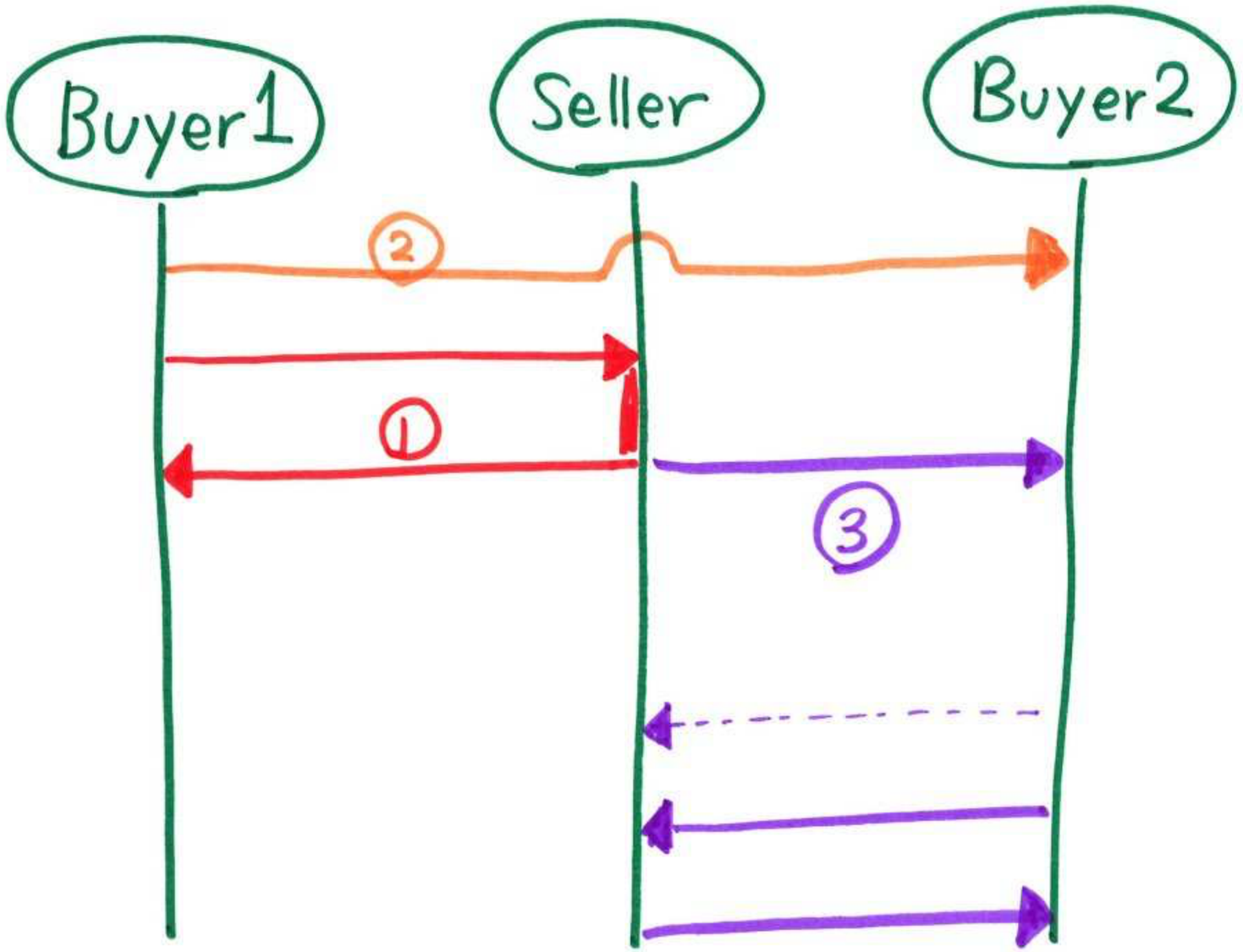
$\mu t.!\,\text{Title};?\,\text{Quote};!\{\textbf{ok}:!\,\text{Add};?\,\text{Date},\ \textbf{retry}:t\}$

$\mu t?\,\text{Title};!\,\text{Quote};?\{\textbf{ok}:?\,\text{Add};!\,\text{Date},\ \textbf{retry}:t\}$

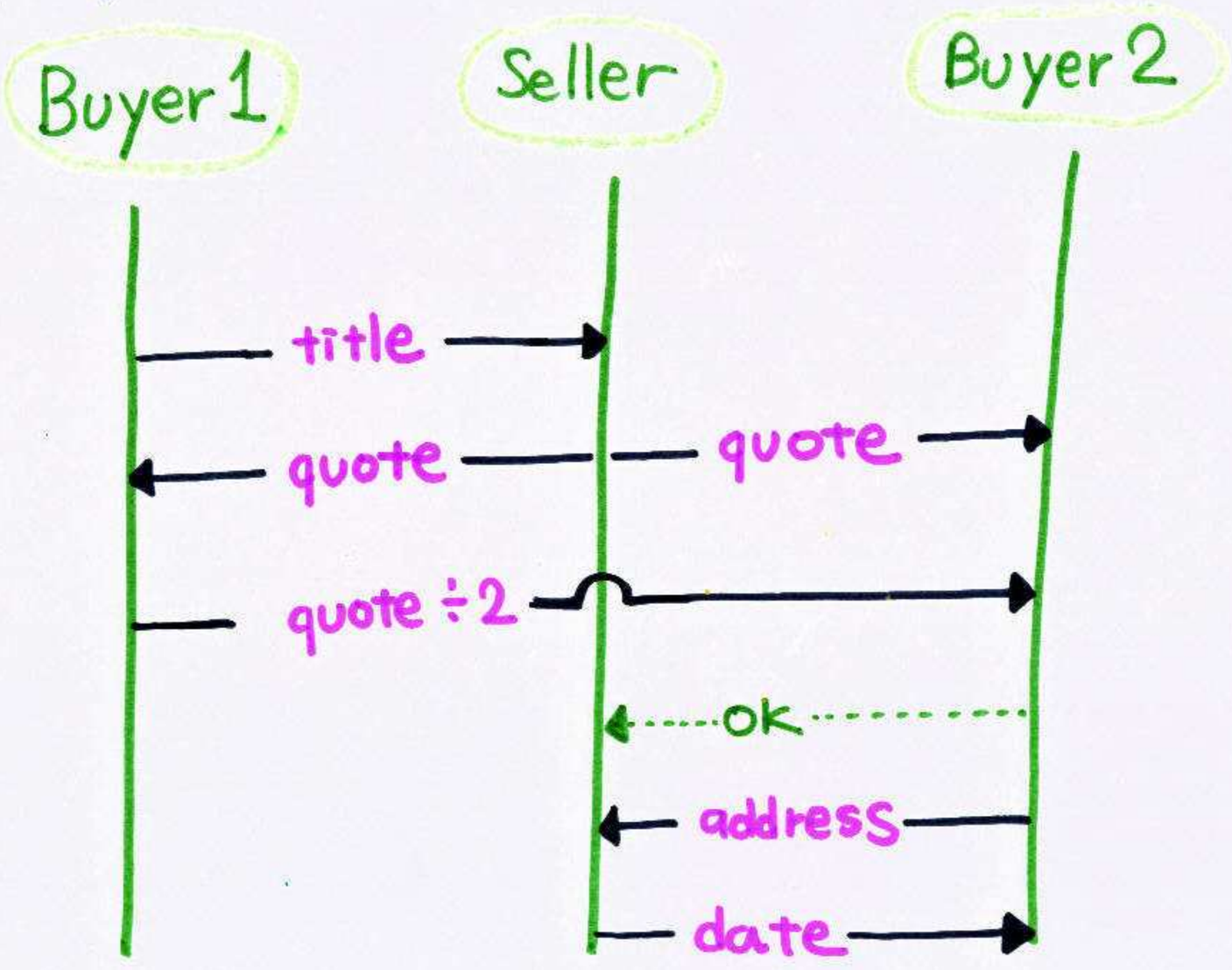
Multiparty Session Types



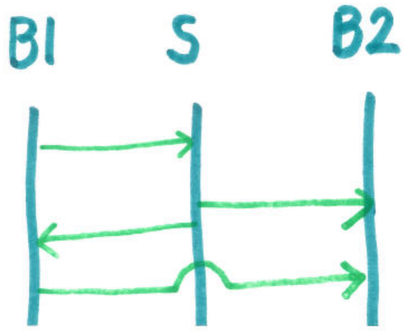

Buyer1
Seller
Buyer2
1
3
2


Buyer1
Seller
Buyer2
2
1
3

Multiparty Session Types



Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

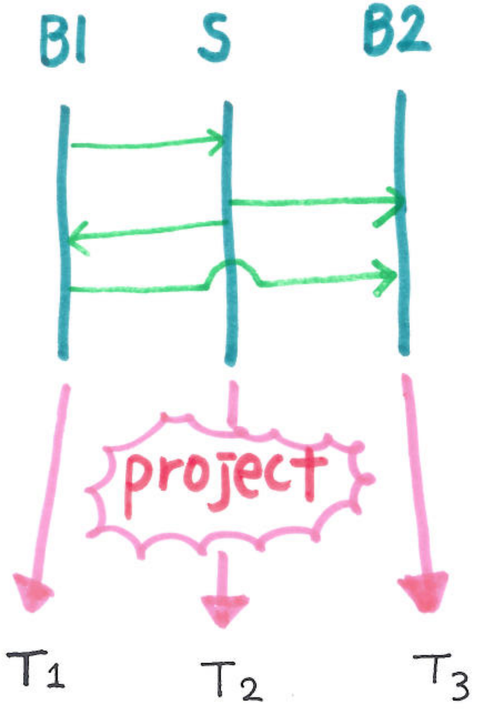
B1 → S Int.

S → B2 Char

STEP 1

Write Global Type

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

$B1 \rightarrow S$ Int.

$S \rightarrow B2$ Char

T

$B_1?\text{Int}.B_2!\text{Char}$

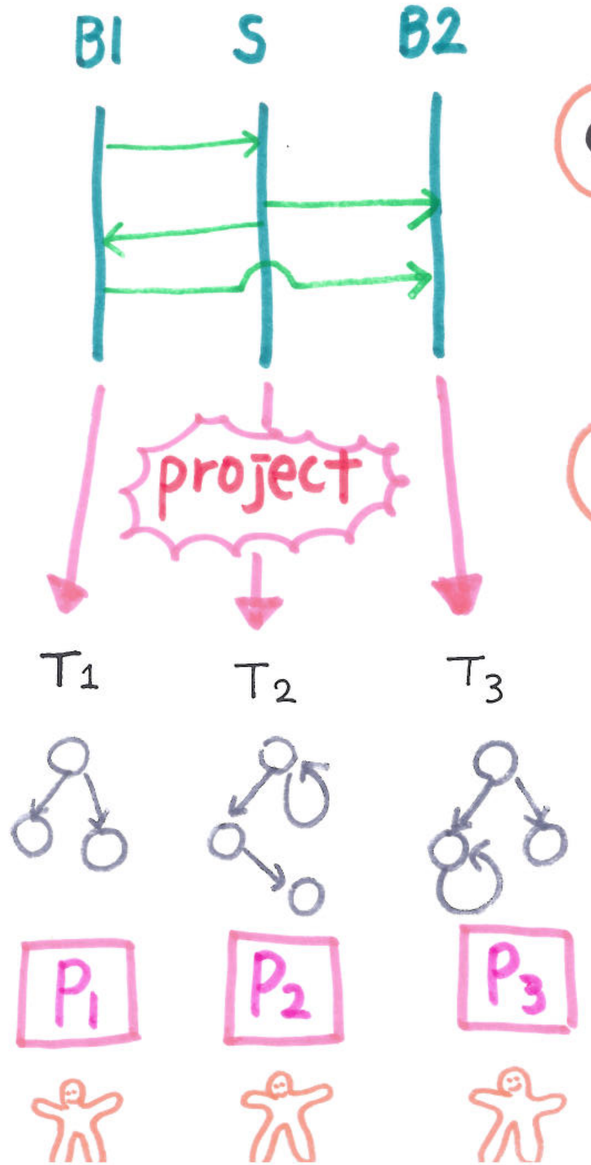
STEP 1

Write Global Type

STEP 2

Project to Local Types

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G $B1 \rightarrow S$ Int.
$S \rightarrow B2$ Char

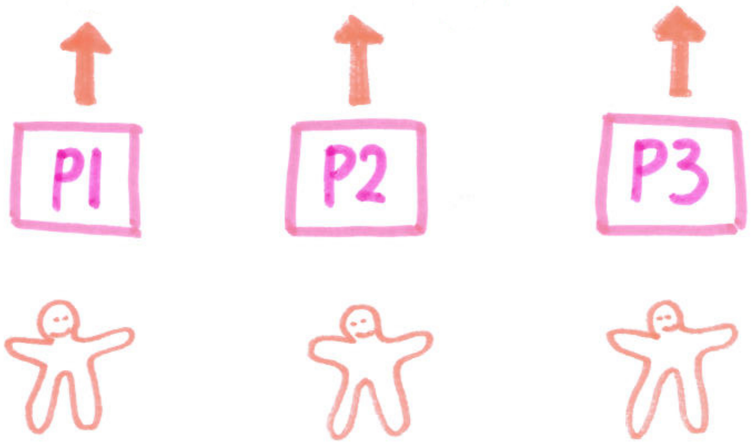
T $B_1?\text{Int}.B_2!\text{Char}$

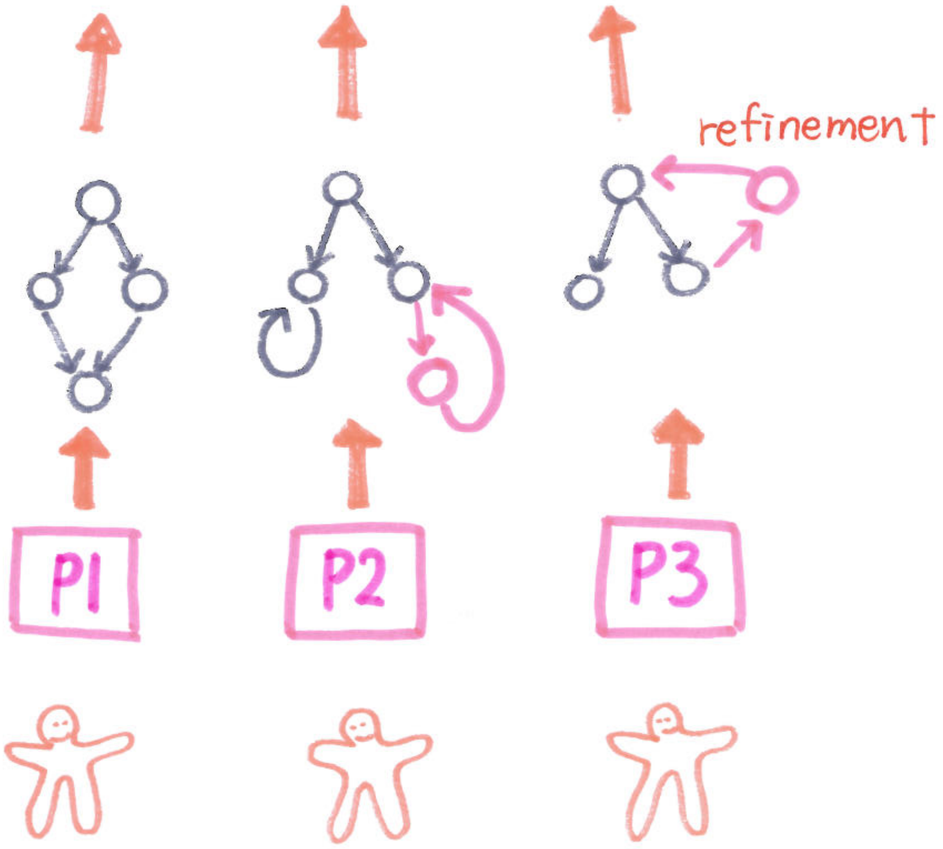
P $B_1?(x).B_2!\langle\text{"apple"}\rangle$

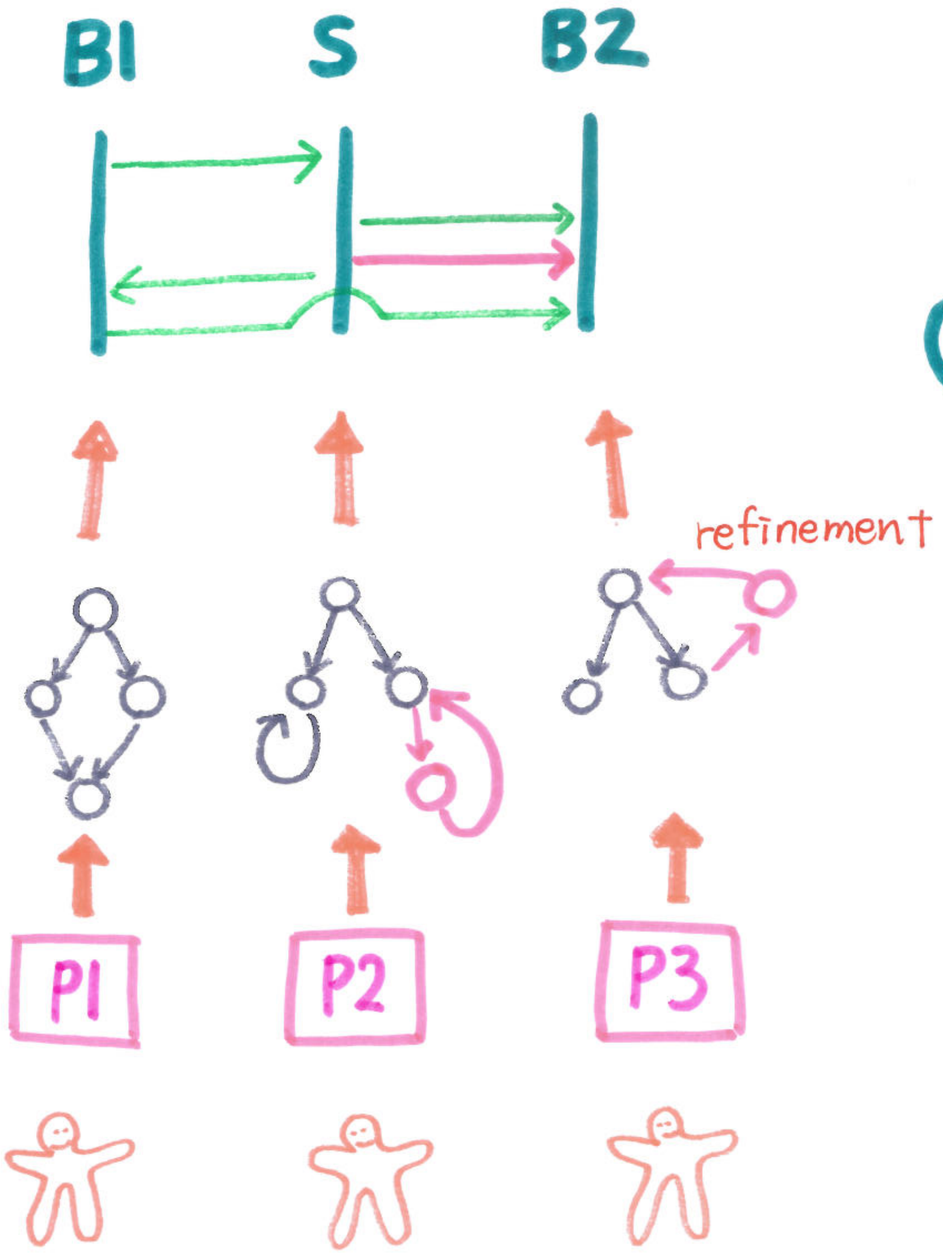
STEP 1
Write Global Type

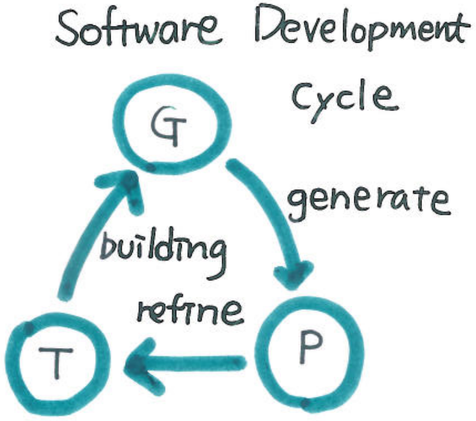
STEP 2
Project to Local Type

STEP 3
- Static Check
- Generate Code
- Run-time check


P1
P2
P3


refinement
P1
P2
P3

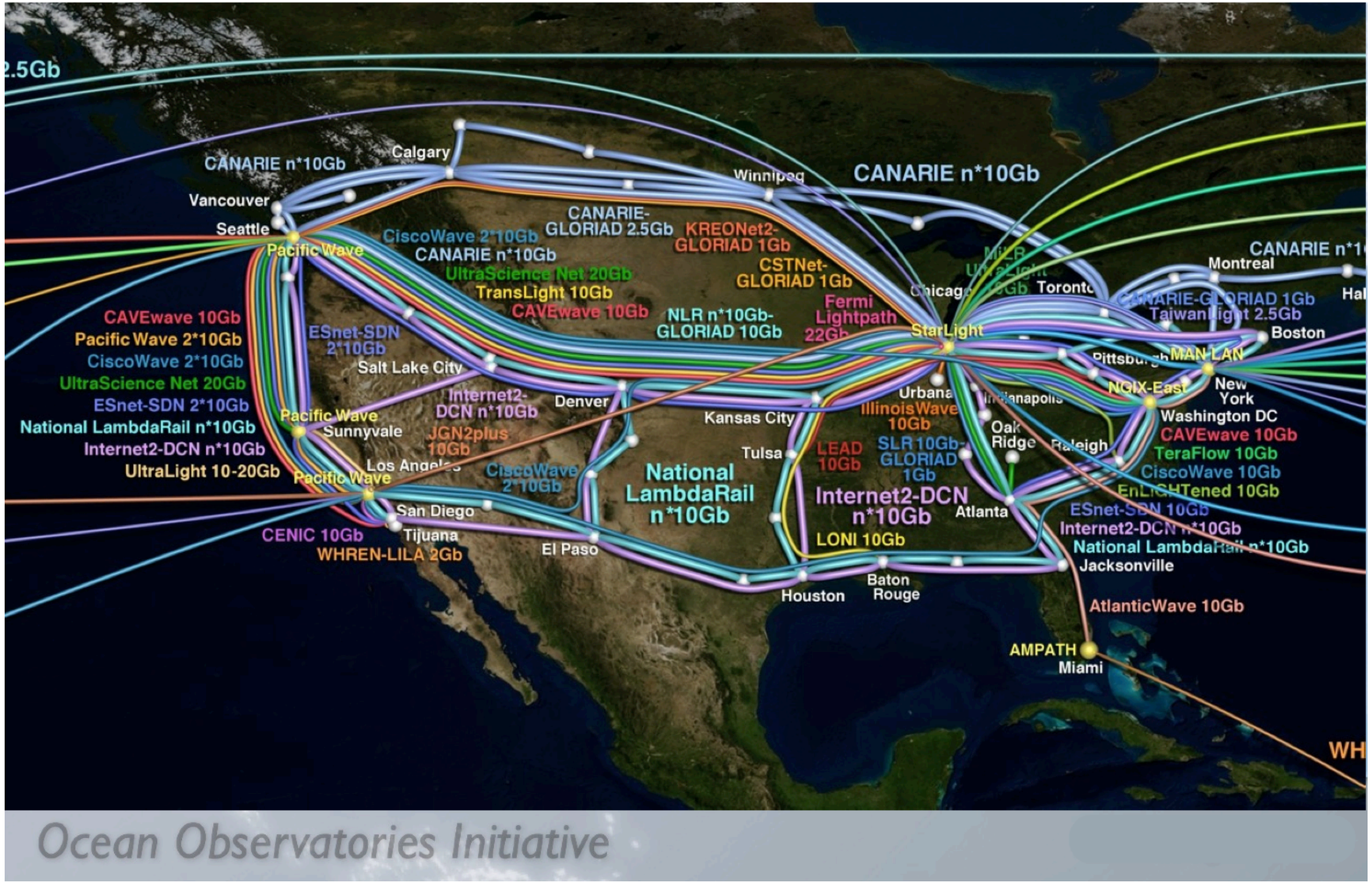




- Optimisation
- refinement
- Inference
- Testing

Mobility Reading Group http://mrg.cs.ox.ac.uk/

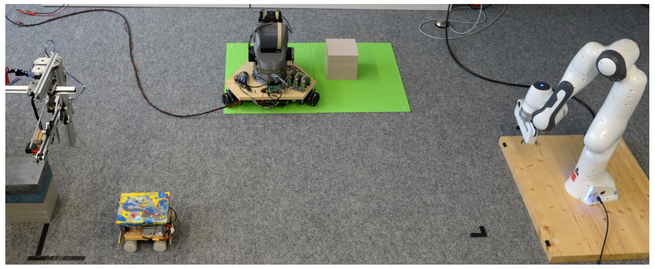
Some Applications on Multiparty Session Types



Distributed Tracing

OpenTelemetry

Robotics



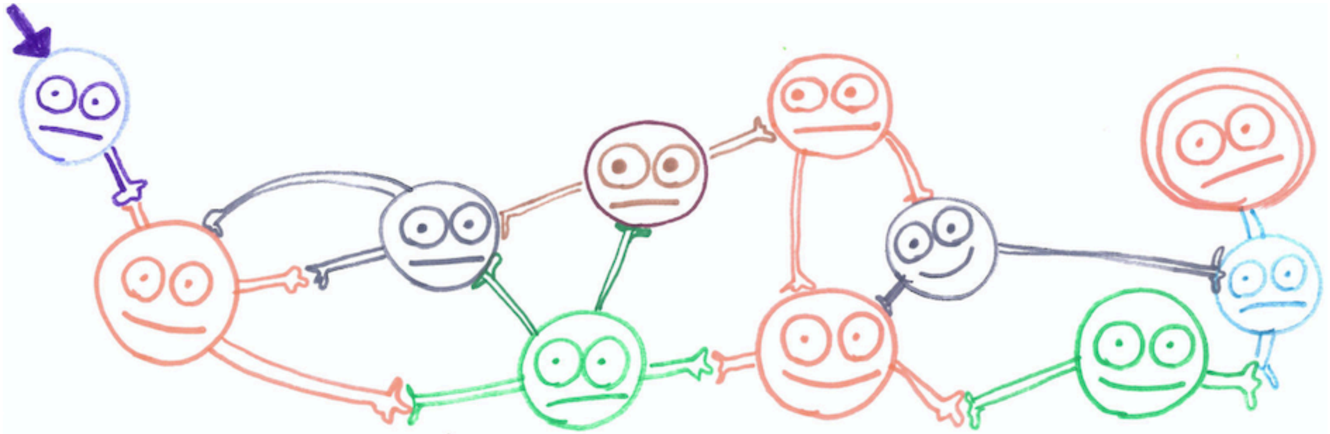
Mechanisation

Zooid

Multiparty Session Types in Rust

Deadlock-Free Message Reordering with Multiparty Session Types

[PPoPP 2022]



Zak Cutner, NY and Martin Vassor

Introduction

Rust Language

- Modern systems language focussed on safety and performance

Introduction

Rust Language

- Modern systems language focussed on safety and performance
- “Most loved language” for past five years on StackOverflow

Introduction

Rust Language

- Modern systems language focussed on safety and performance
- “Most loved language” for past five years on StackOverflow
- Particular emphasis on safe concurrency using message passing

Introduction

Rust Language

- Modern systems language focussed on safety and performance
- “Most loved language” for past five years on StackOverflow
- Particular emphasis on safe concurrency using message passing
- Affine type system is well-suited to session types

Ring Protocol

Example

Global Type

$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\texttt{i32}).\mathbf{t} \} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\texttt{i32}).\mathbf{t} \} \end{array} \right\} \right\}$$

Ring Protocol

Example

$$\mathsf{G} = \mu \mathtt{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\mathtt{i32}).\mathbf{t}\} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\mathtt{i32}).\mathbf{t}\} \end{array} \right\} \right\}$$

Ring Protocol

Example

$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array} \right\} \right\}$$

Ring Protocol

Example

$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array} \right\} \right\}$$

Ring Protocol

Example

$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\texttt{i32}).\mathbf{t} \} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\texttt{i32}).\mathbf{t} \} \end{array} \right\} \right\}$$

Ring Protocol

Example

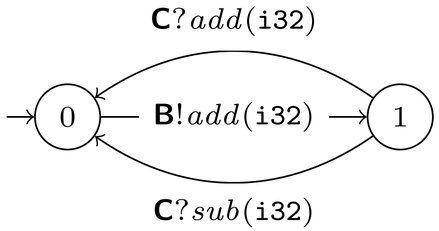
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array} \right\} \right\}$$

Ring Protocol

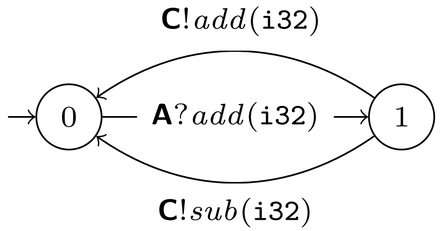
Example

$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\mathtt{i32}).\mathbf{t}\} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\mathtt{i32}).\mathbf{t}\} \end{array} \right\} \right\}$$

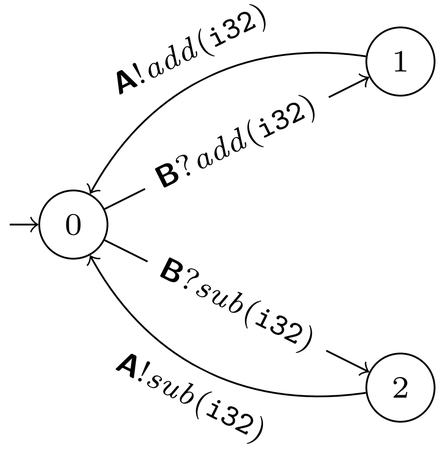
PROJECTION



PROJECTION



PROJECTION

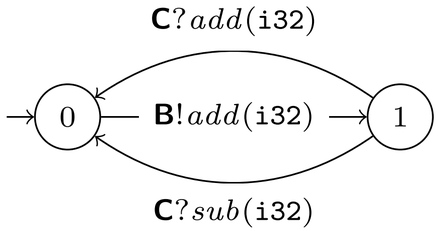


Ring Protocol

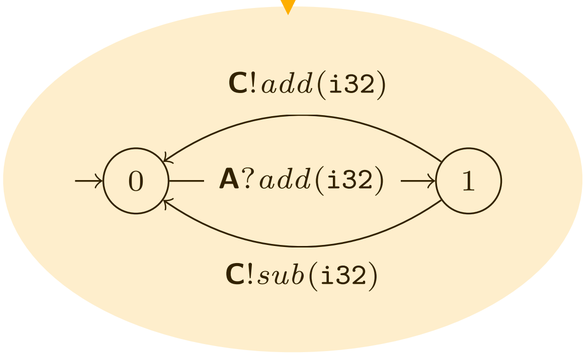
Example

$$\mathsf{G} = \mu \mathtt{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\mathtt{i32}).\mathtt{t} \} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\mathtt{i32}).\mathtt{t} \} \end{array} \right. \right\}$$

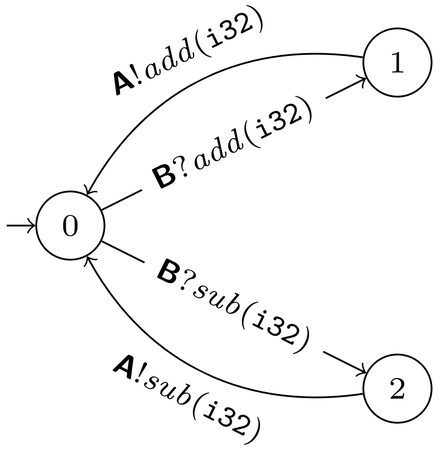
PROJECTION



PROJECTION

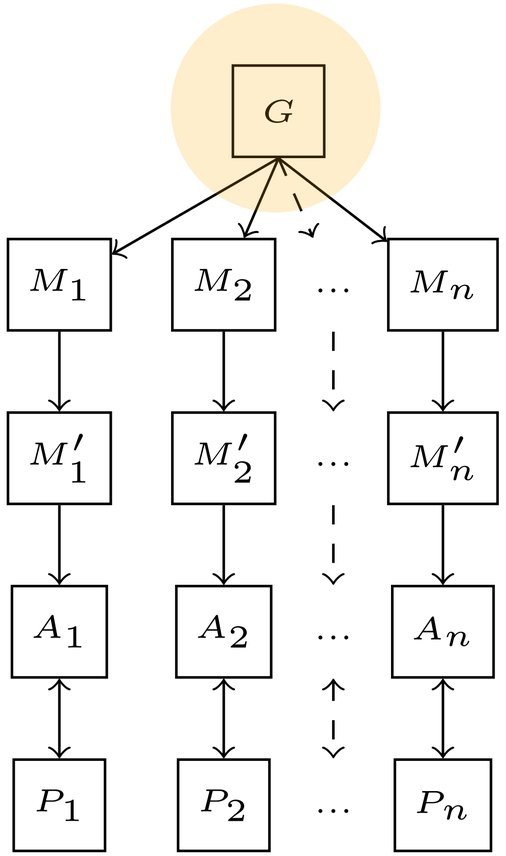


PROJECTION



Workflow

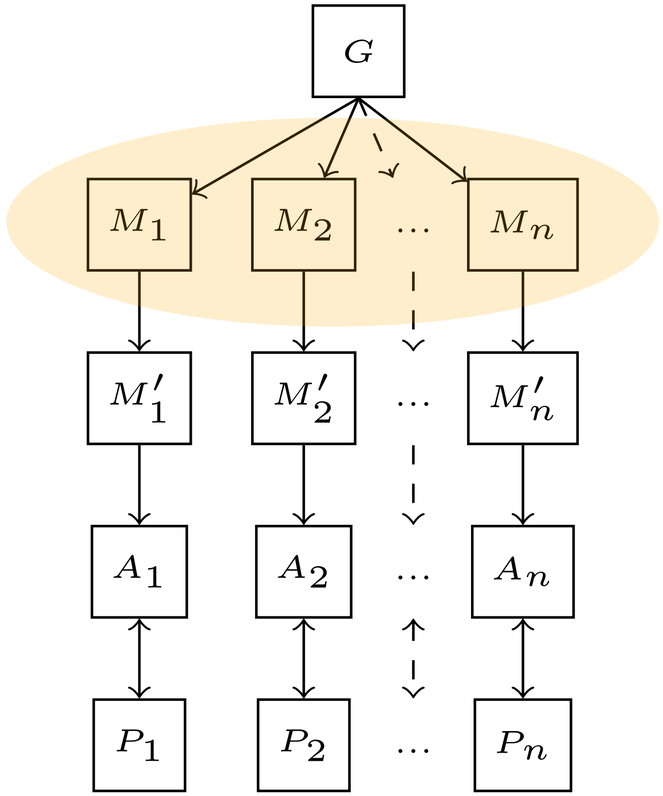
Top-Down Approach



Global Type

Workflow

Top-Down Approach

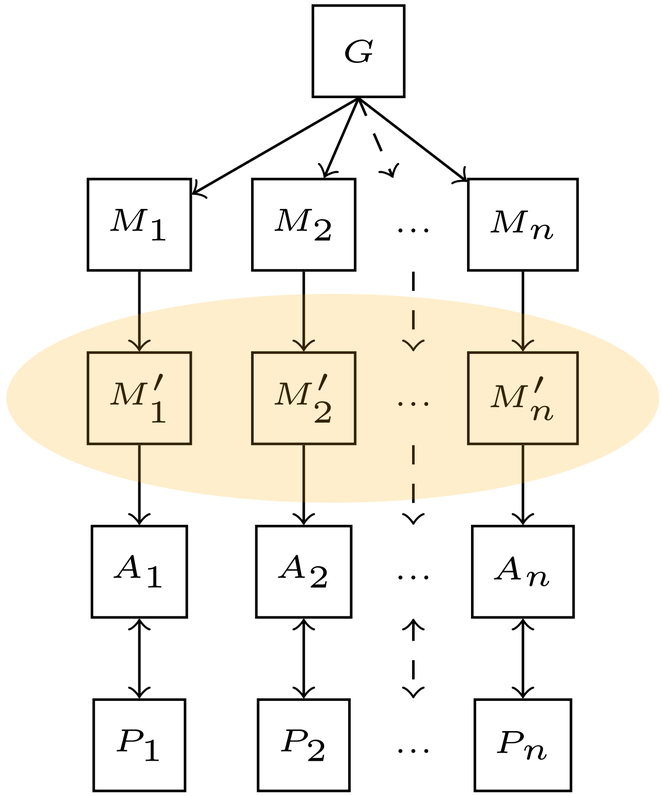


Global Type

Projected FSM

Workflow

Top-Down Approach



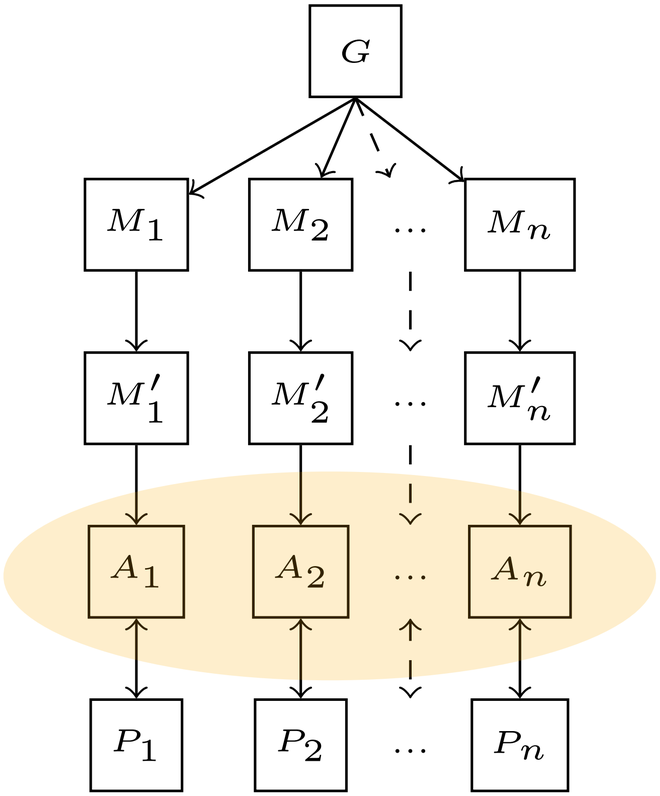
Global Type

Projected FSM

Optimised FSM

Workflow

Top-Down Approach



Global Type

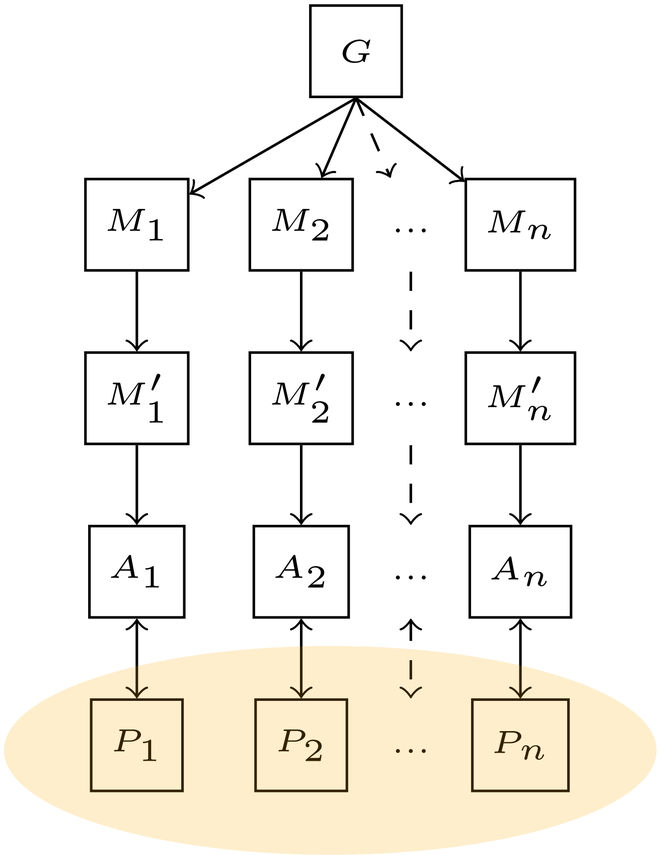
Projected FSM

Optimised FSM

Rust API

Workflow

Top-Down Approach



Global Type

Projected FSM

Optimised FSM

Rust API

Rust Implementation

Ring Protocol

Example

Global Type

$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\texttt{i32}).\mathbf{t} \} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\texttt{i32}).\mathbf{t} \} \end{array} \right\} \right\}$$

Ring Protocol

Example

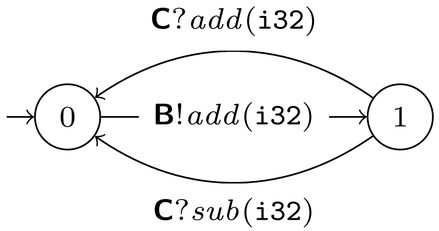
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array} \right\} \right\}$$

Ring Protocol

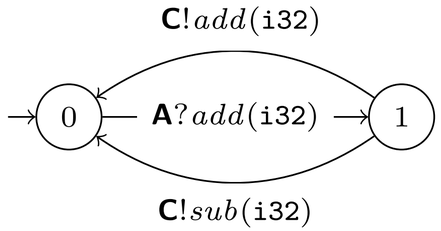
Example

$$\mathsf{G} = \mu \mathtt{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\mathtt{i32}).\mathtt{t} \} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\mathtt{i32}).\mathtt{t} \} \end{array} \right\} \right\}$$

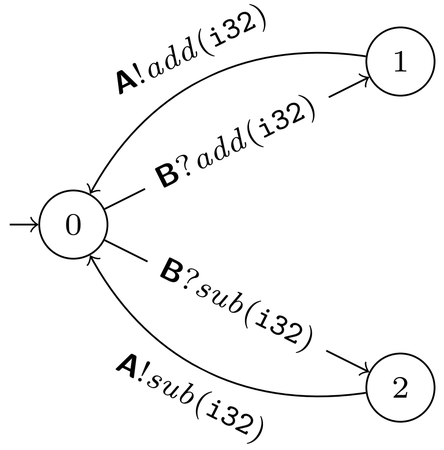
PROJECTION



PROJECTION

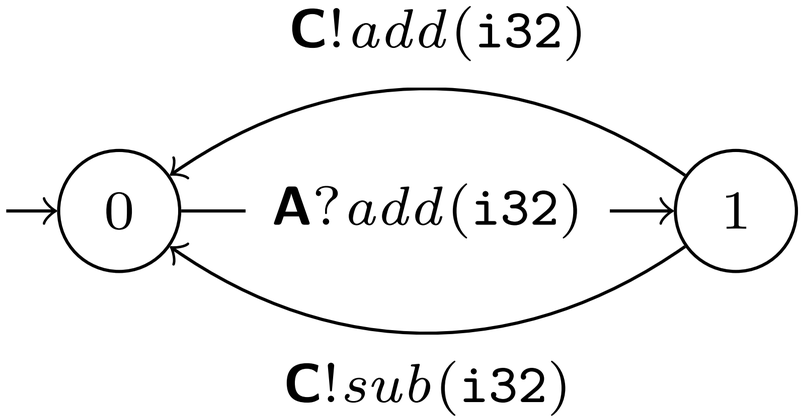


PROJECTION



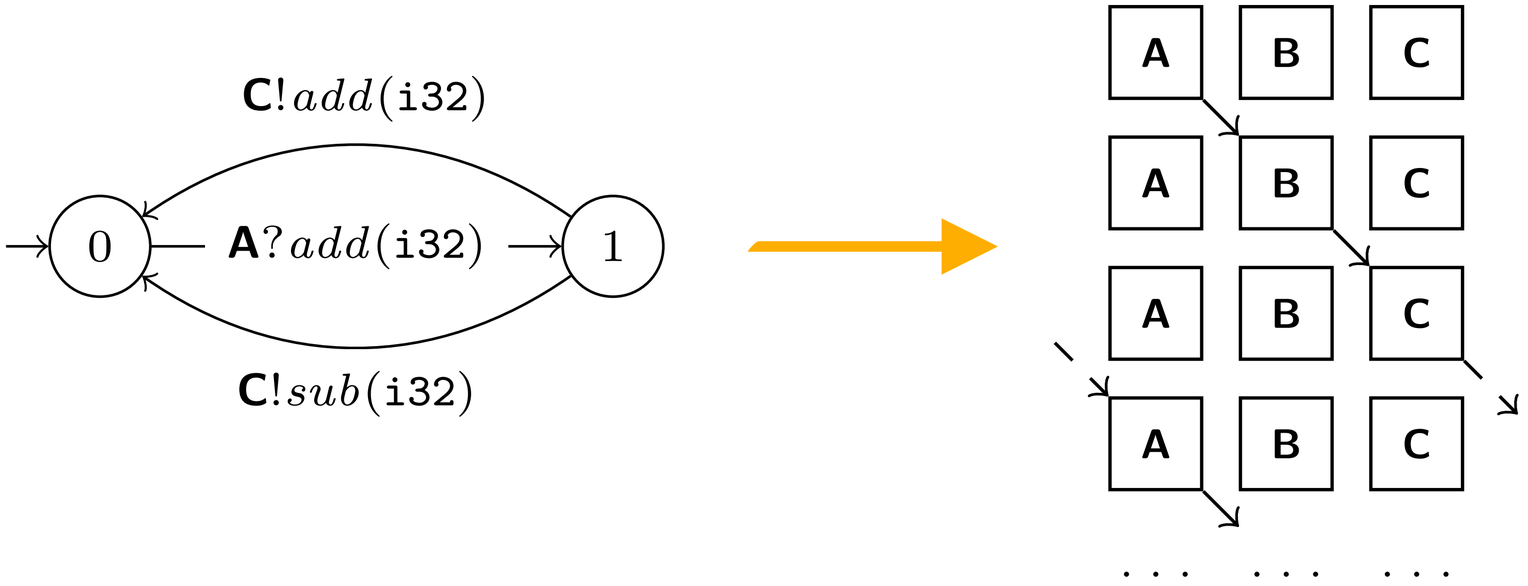
Ring Protocol

Example



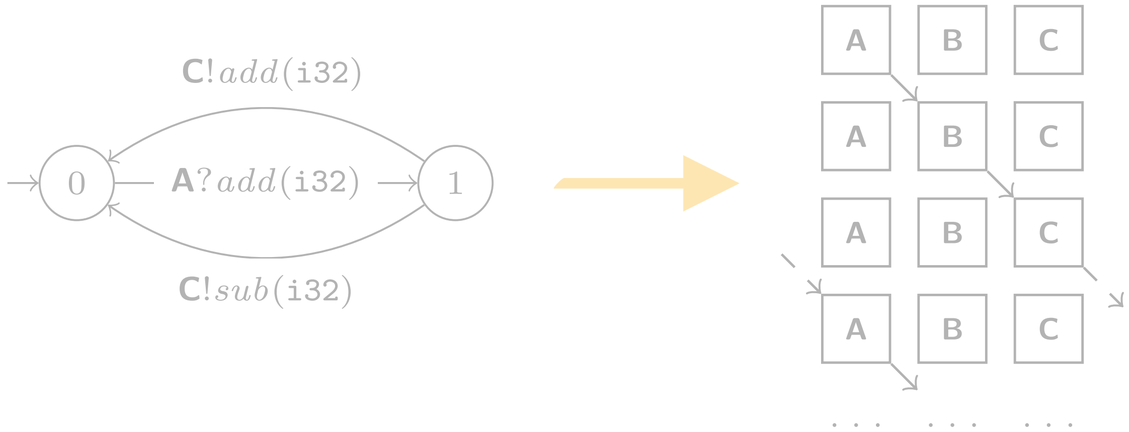
Ring Protocol

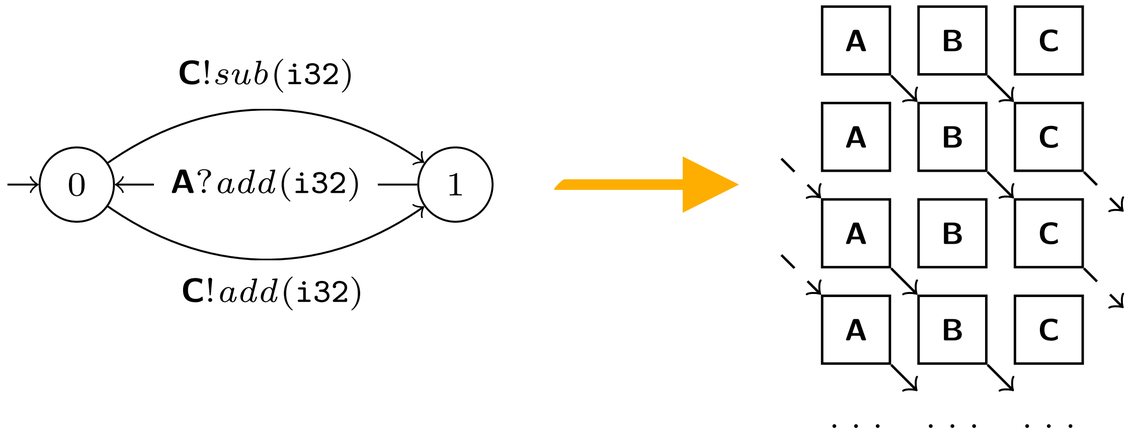
Example



Ring Protocol

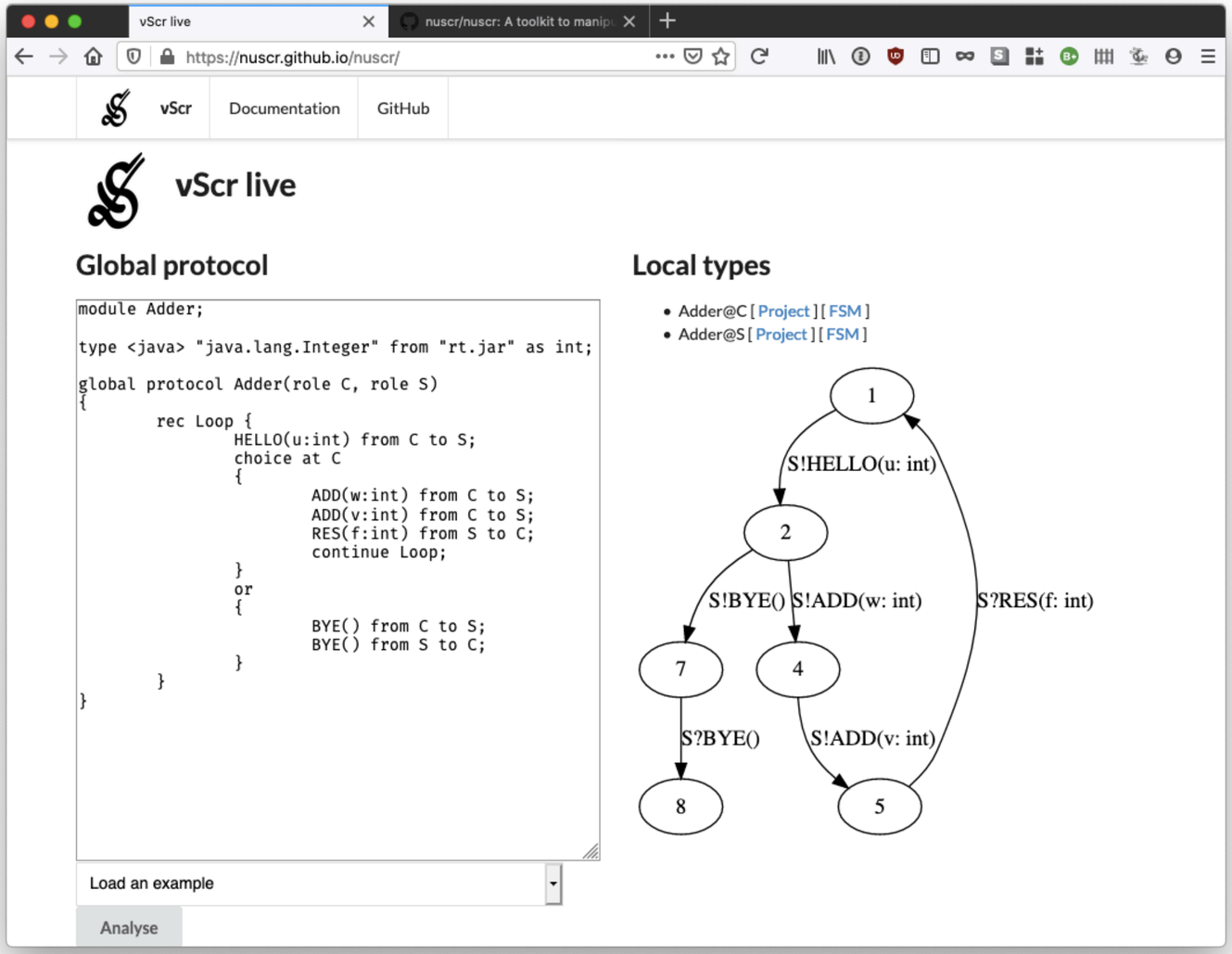
Example





vScr An Extensible Toolchain for Multiparty Session Types

- It’s small and easy to modify
- Available on opam
  - opam install nuscr
- Available on GitHub
  - https://github.com/nuscr
- Available on the web
  - https://nuscr.dev



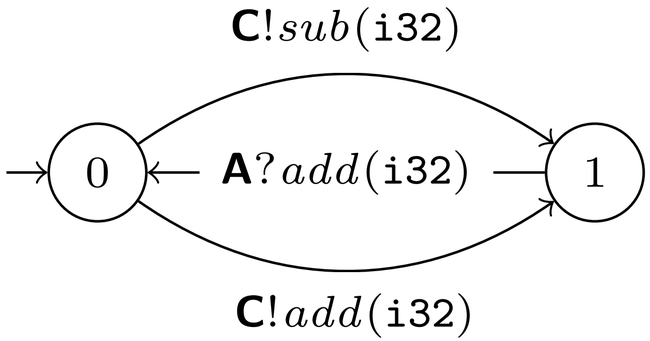
Scribble

Protocol Description Language

```
global protocol Ring(role A, role B, role C) {
    Add(i32) from A to B;
    choice at B {
        Add(i32) from B to C;
        Add(i32) from C to A;
        do Ring(A, B, C);
    } or {
        Sub(i32) from B to C;
        Sub(i32) from C to A;
        do Ring(A, B, C);
    }
}
```

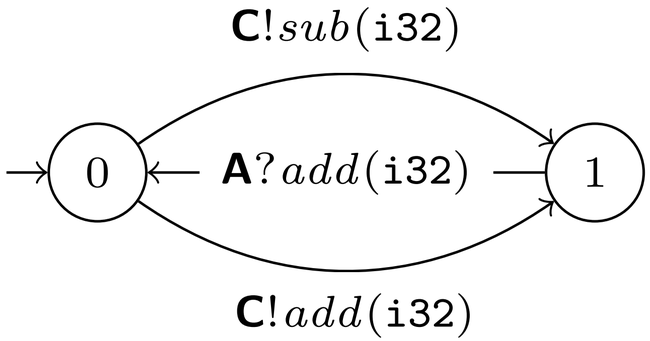
Ring Protocol

Rust API



Ring Protocol

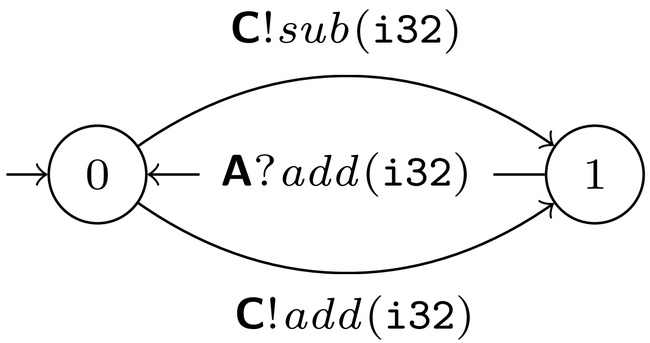
Rust API




```
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

Ring Protocol

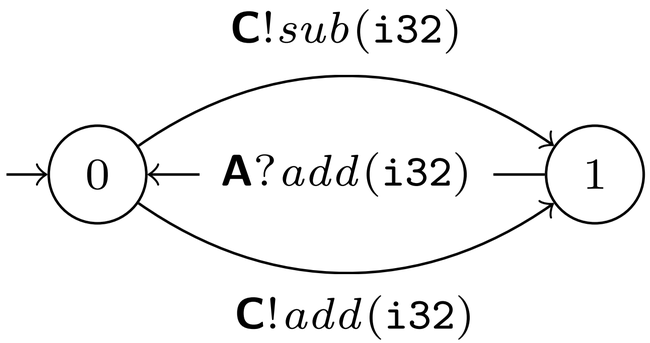
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

Ring Protocol

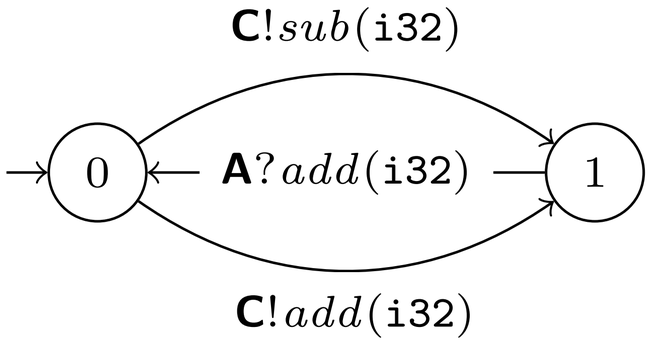
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

Ring Protocol

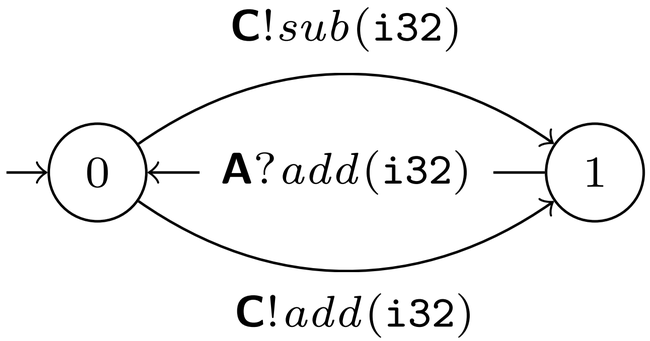
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);

#[derive(Message)]
enum Label {
    Add(Add),
    Sub(Sub),
}

struct Add(i32);
struct Sub(i32);
```

Ring Protocol

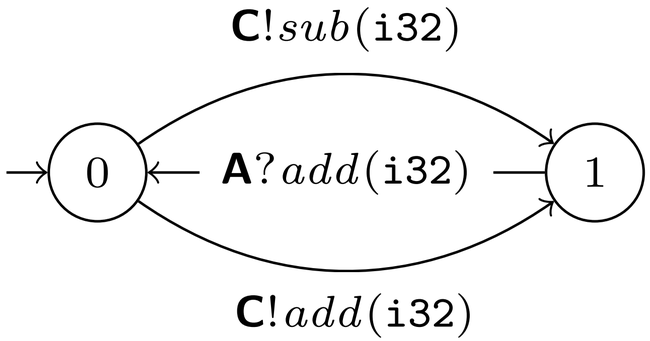
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);

#[derive(Message)]
enum Label {
    Add(Add),
    Sub(Sub),
}

struct Add(i32);
struct Sub(i32);

#[session]
type RingB = Select<C, RingBChoice>;

#[session]
enum RingBChoice {
    Add(Add, Receive<A, Add, RingB>),
    Sub(Sub, Receive<A, Add, RingB>),
}
```

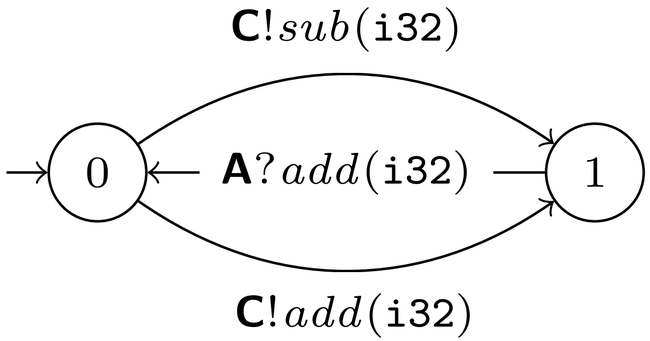
Ring Protocol

Rust API



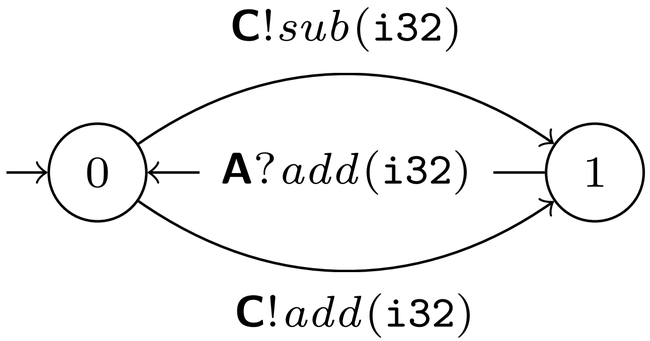
Ring Protocol

Implementation



Ring Protocol

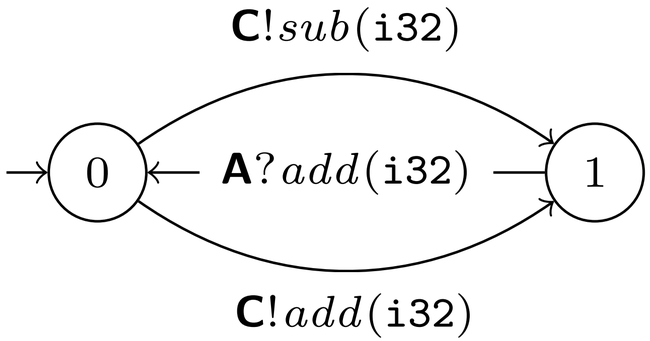
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

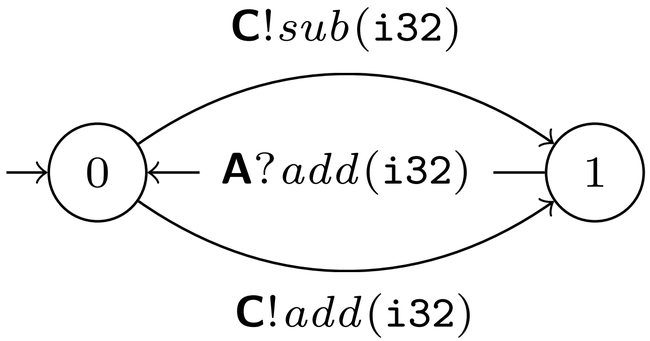
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

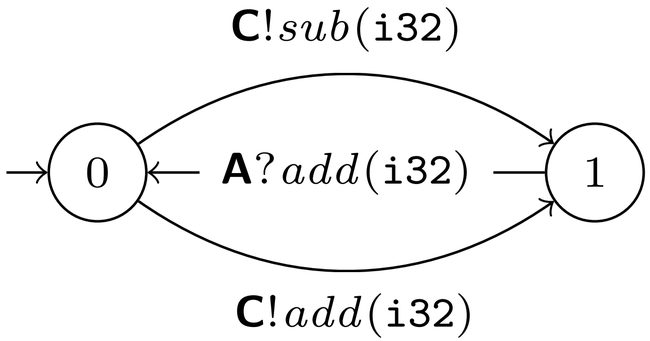
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

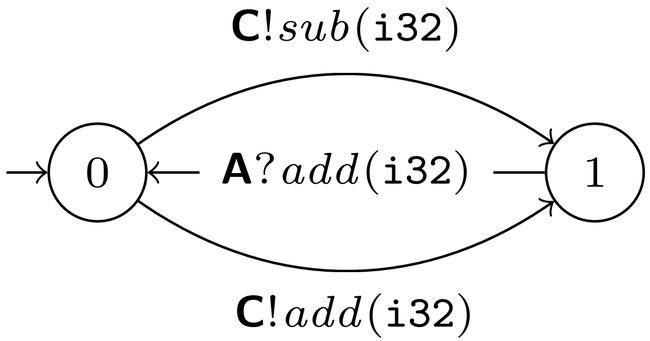
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

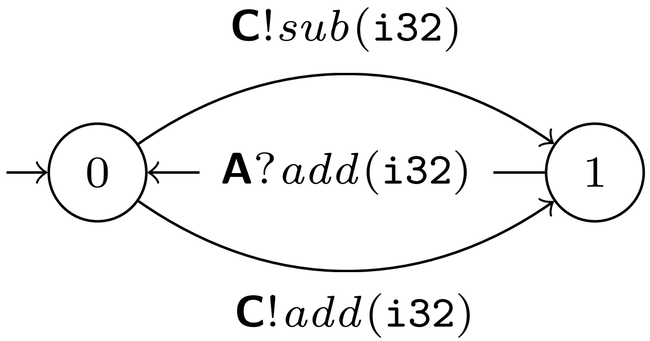
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

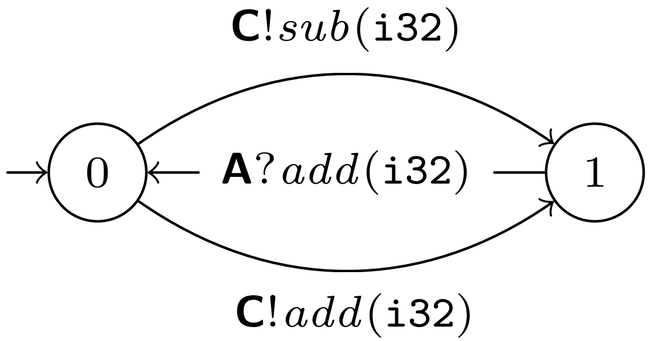
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

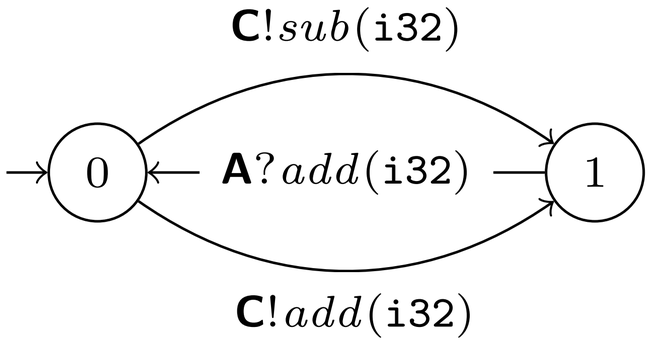
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

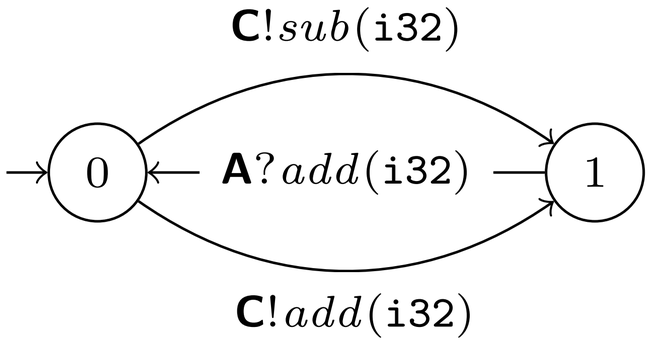
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

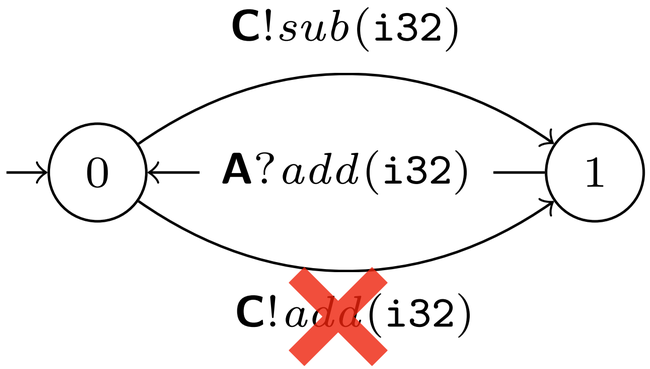
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

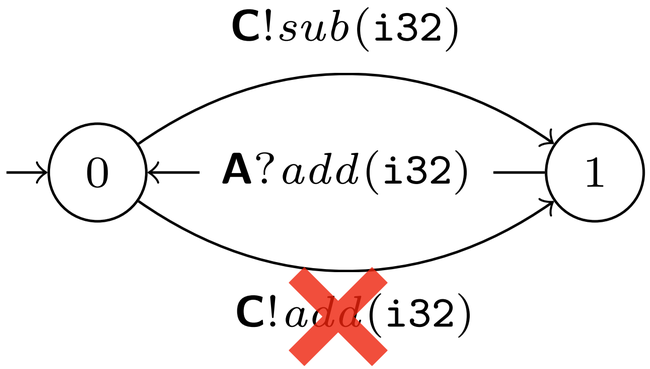
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

```
method not found in `rumpsteak::Select<'_, B, C, RingBChoice<'_, B>>`
```

Summary

Multiparty Session Types in Rust

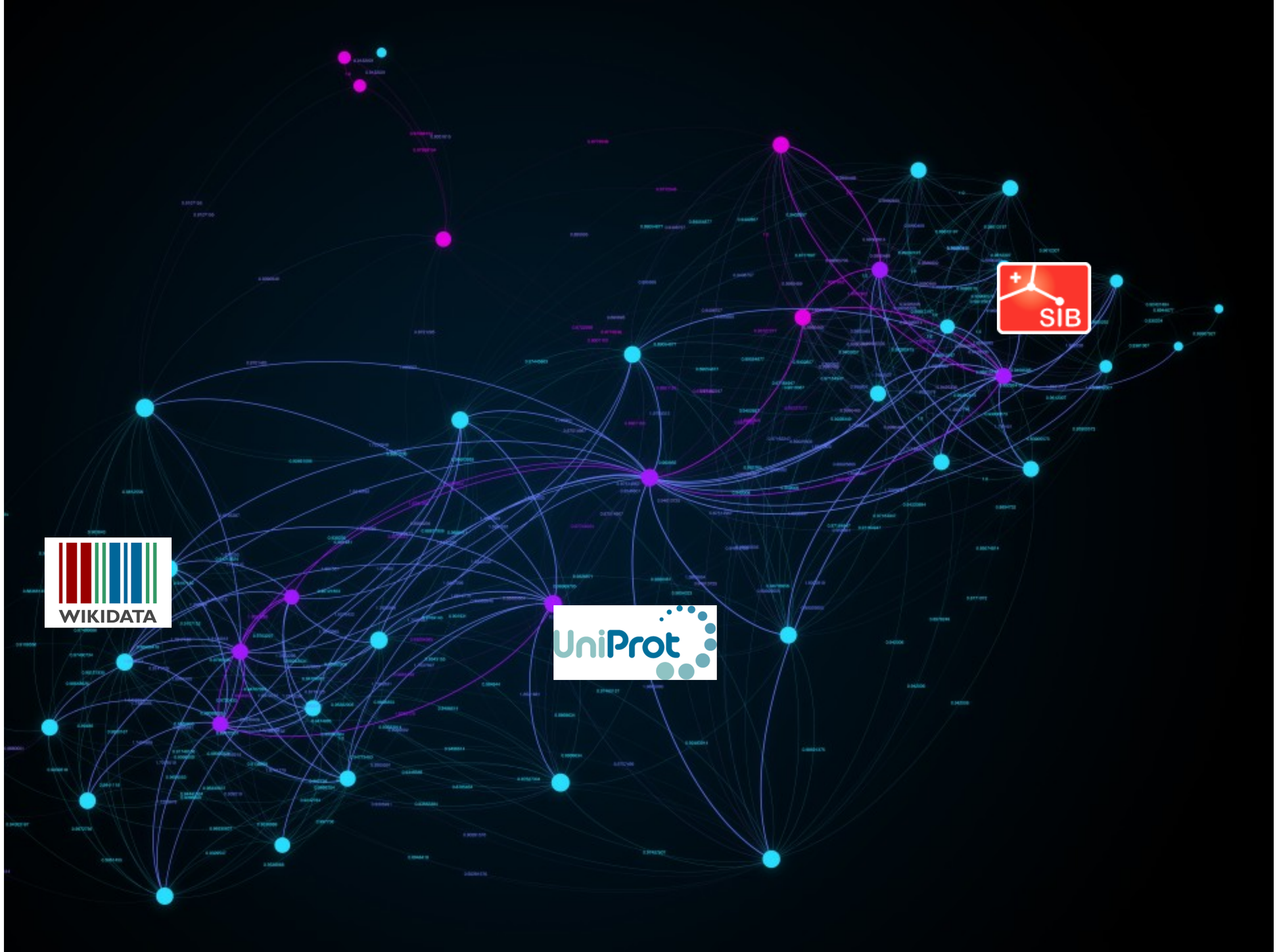
- Multiparty session type description language
  -  **nuScr https://github.com/nuscr**
- Applications of multiparty session types using communicating automata
  -  **https://github.com/zakcutner/rumpsteak**
  - More on **Rumpsteak**: 1st August **Huawei Workshop on Formal Methods**
  - **[ECOOP 2024] Refinements for Multiparty Message-Passing Protocols: Specification-agnostic theory and implementation** https://zenodo.org/records/12731834

Protocol Conformance of Collaborative Federate Query using Multiparty Session Types

Ari Hernawan & NY

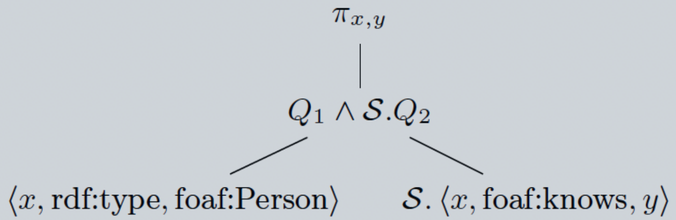
TASE 2024

Retrieving drug targets for human enzymes involved in sterol metabolism

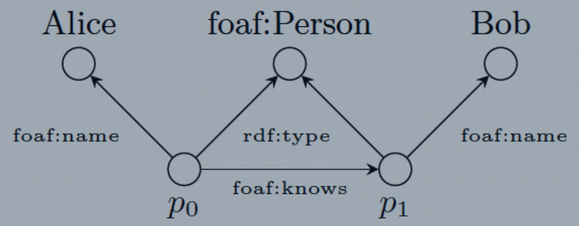

SIB
WIKIDATA
UniProt

SPARQL and RDF

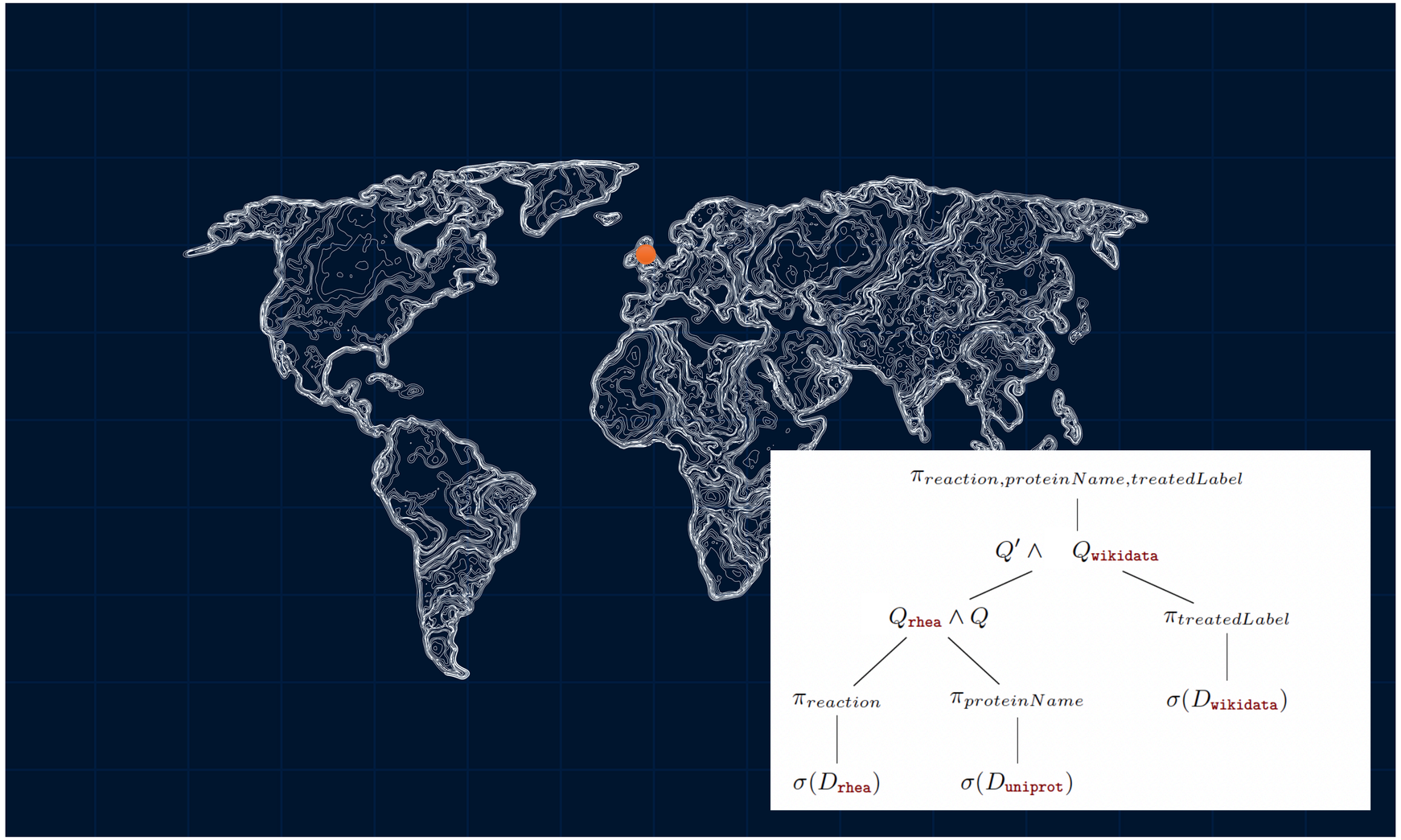
```
1  SELECT ?x ?y /* return variables */
2  WHERE {
3    ?x rdf:type foaf:Person . /* graph pattern is evaluated on local dataset */
4    SERVICE <https://remote>{
5      ?x foaf:knows ?y . /* graph pattern is evaluated on remote dataset */
6    }
7  }
```



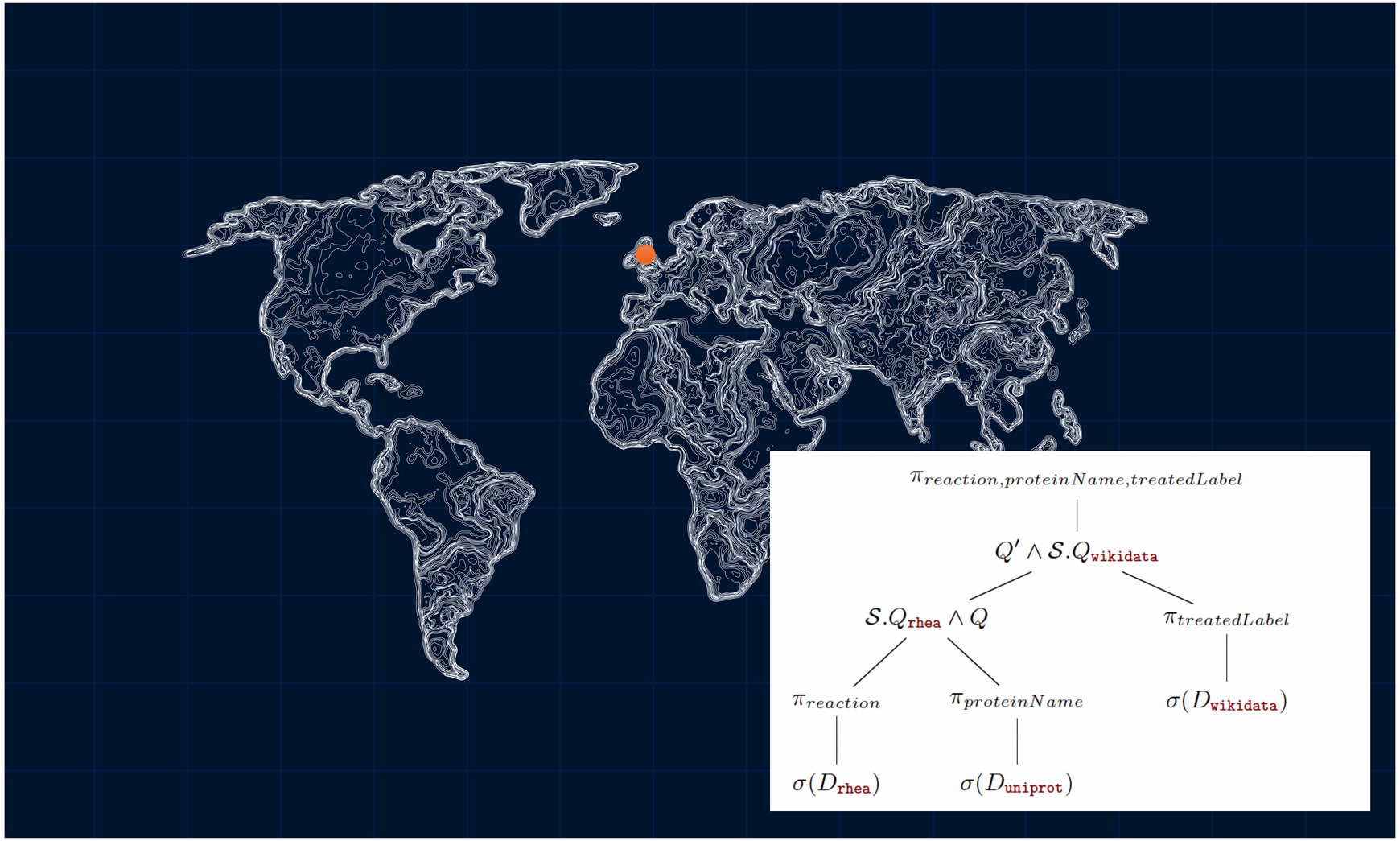
$$
\begin{aligned}
Q, Q' ::= & \; A && \textit{(triple pattern or atom)} \\
| & \; \mathcal{S}.Q' && \textit{(service federation)} \\
| & \; Q \wedge Q' && \textit{(group of basic graph pattern)}
\end{aligned}
$$



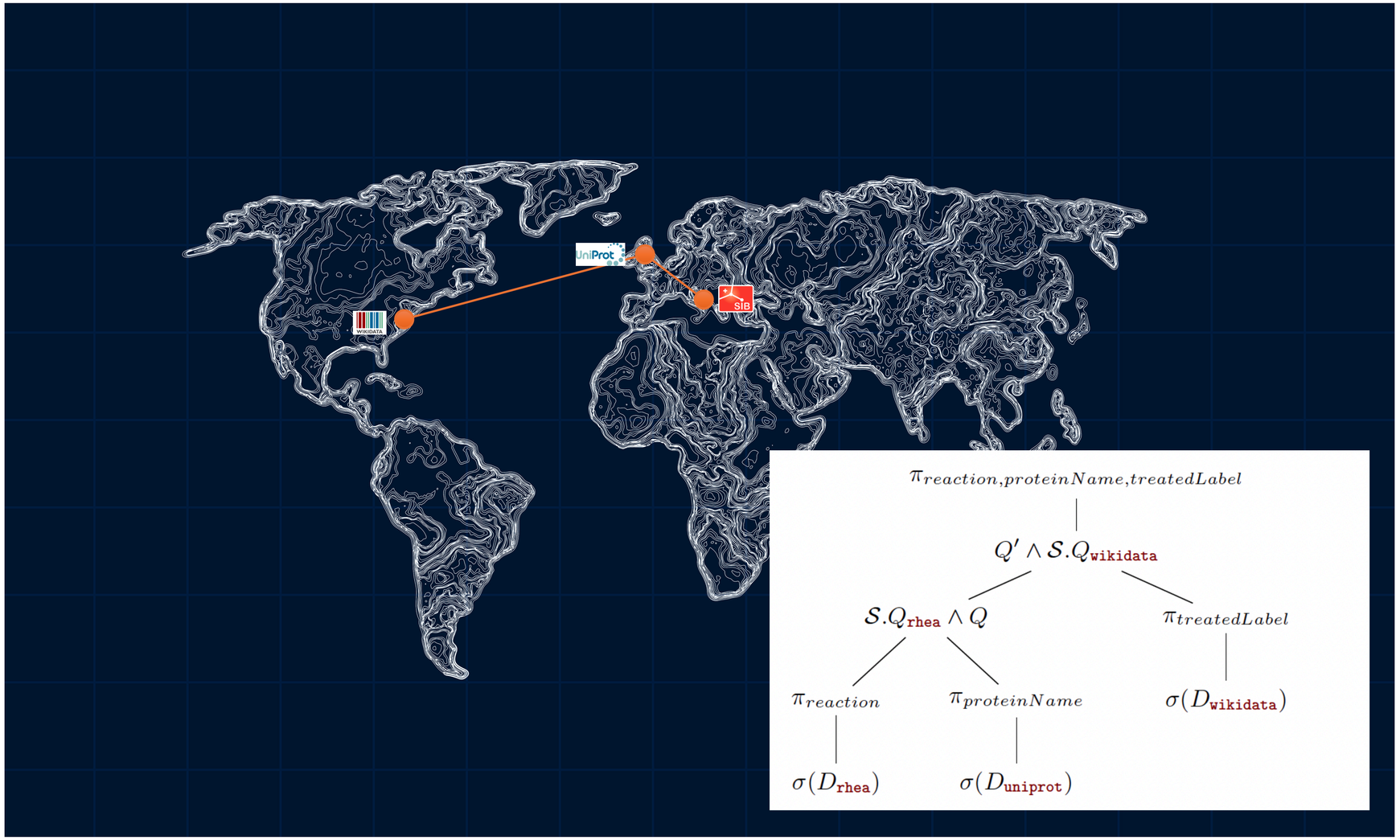
$$
\begin{aligned}
D =& \{V, E\} \\
V =& \{\text{Alice}, \text{Bob}, \text{foaf:Person}, p_0, p_1\} \\
E =& \{\langle p_0, \text{foaf:knows}, p_1 \rangle, \langle p_0, \text{rdf:type}, \text{foaf:Person} \rangle, \\
& \langle p_1, \text{rdf:type}, \text{foaf:Person} \rangle, \langle p_0, \text{foaf:name}, \text{Alice} \rangle, \langle p_1, \text{foaf:name}, \text{Bob} \rangle\}
\end{aligned}
$$



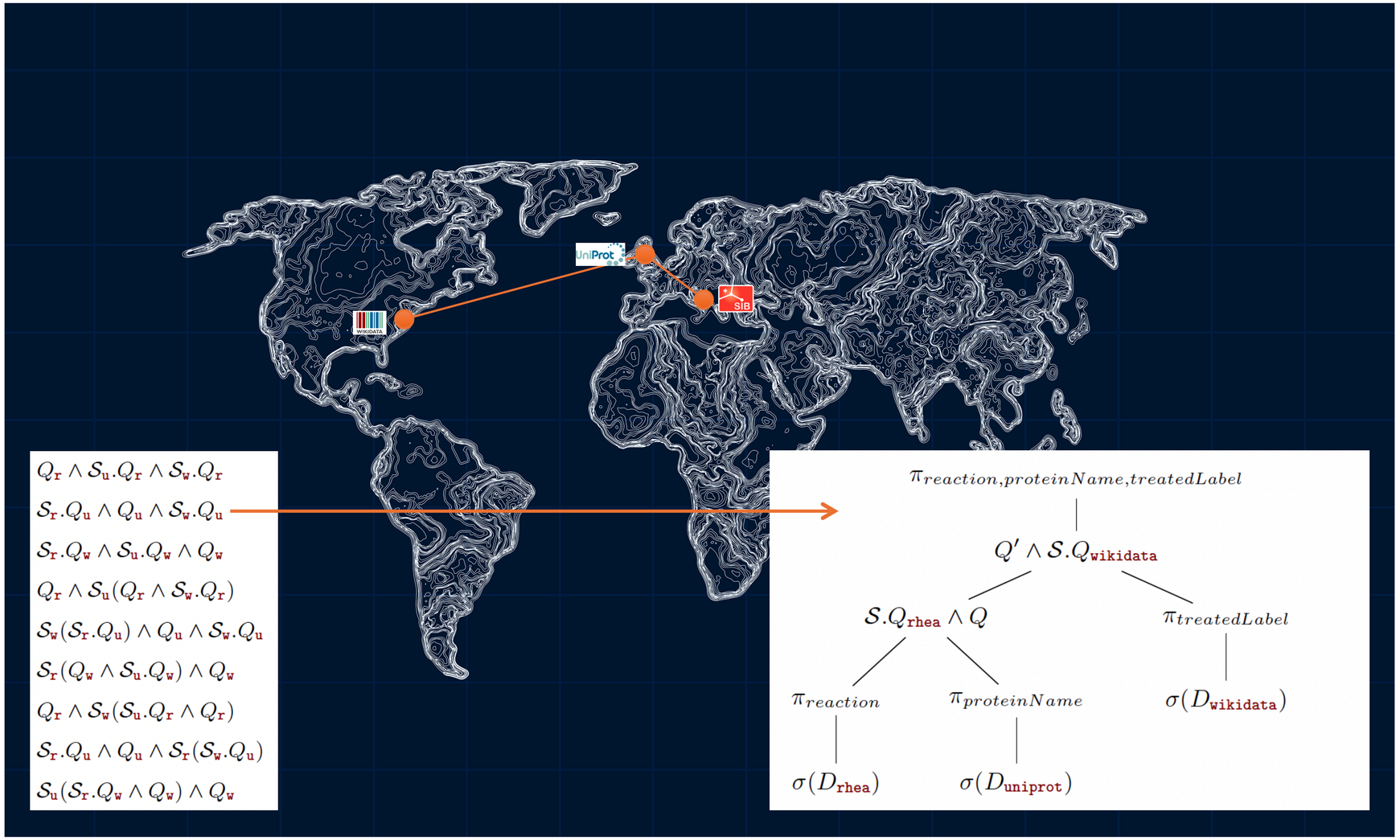
Retrieving drug targets for human enzymes involved in sterol metabolism



Retrieving drug targets for human enzymes involved in sterol metabolism

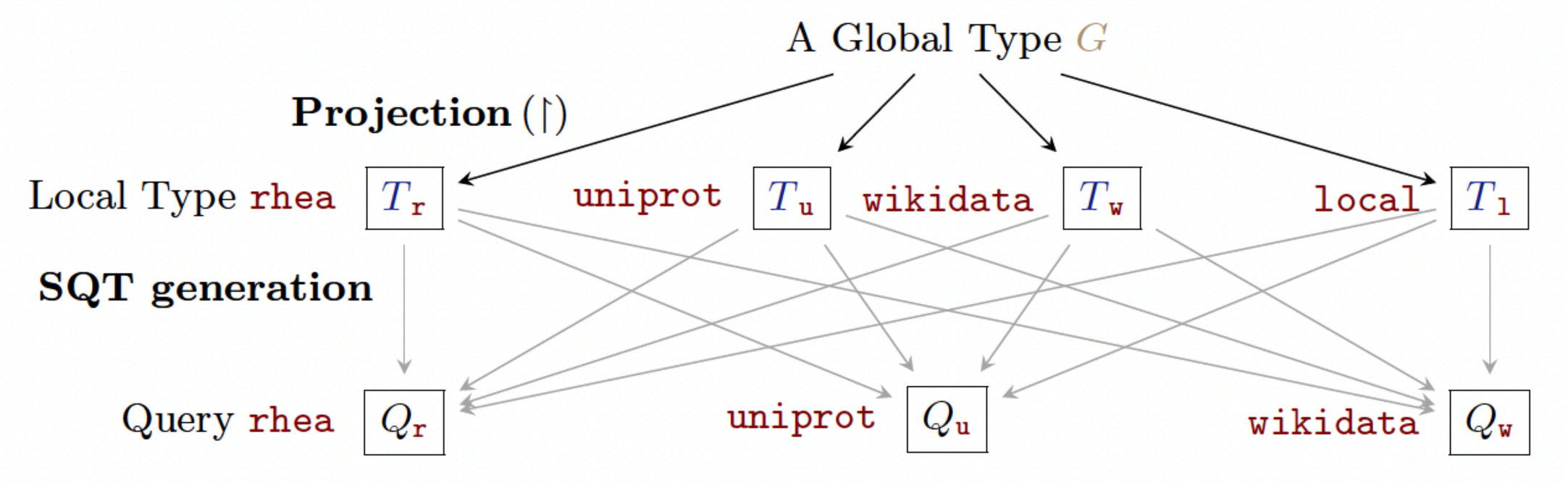


Retrieving drug targets for human enzymes involved in sterol metabolism



Retrieving drug targets for human enzymes involved in sterol metabolism

SQT Generation with a Global Type



Broadcasting Global Specifications

$$I = \{\texttt{rhea}, \texttt{uniprot}, \texttt{wikidata}\}$$
$$G = \texttt{local} \to \texttt{rhea}\{l_i : \texttt{local} \to \texttt{uniprot}\{l_i : \texttt{local} \to \texttt{wikidata}\{l_i : G_i\}\}\}_{i \in I}$$

$$\begin{aligned}
G_{\texttt{rhea}} &= \texttt{local} \to \texttt{rhea} : [U]; \texttt{uniprot} \to \texttt{rhea} : [U]; \\
&\quad \texttt{wikidata} \to \texttt{uniprot} : [U]; \textbf{end} \\
G_{\texttt{uniprot}} &= \texttt{rhea} \to \texttt{wikidata} : [U]; \texttt{wikidata} \to \texttt{uniprot} : [U]; \\
&\quad \texttt{local} \to \texttt{uniprot} : [U]; \texttt{wikidata} \to \texttt{uniprot} : [U]; \textbf{end} \\
G_{\texttt{wikidata}} &= \texttt{rhea} \to \texttt{wikidata} : [U]; \texttt{uniprot} \to \texttt{rhea} : [U]; \\
&\quad \texttt{local} \to \texttt{wikidata} : [U]; \textbf{end}
\end{aligned}$$

Algorithm for Constructing SPARQL Query Template

Algorithm 1 Construct SPARQL Query Template

function SQT($G \restriction \mathtt{p}, \mathtt{q}$) ▷ begin with $p = q$ where $\mathtt{p}, \mathtt{q} \in pt(G) \setminus \{\mathtt{local}\}$
  while $(G \setminus \mathtt{local} \xrightarrow{l} \bar{\mathtt{s}}) \restriction \mathtt{p} \neq \mathbf{end}; l = \mathtt{q}$ **do** ▷ $\bar{\mathtt{s}} = pt(G) \setminus \{\mathtt{local}\}$
    if $T_{\mathtt{p}} = \mathtt{r}?[U]$ **then** ▷ $\mathtt{p} \neq \mathtt{r}; \mathtt{r} \in pt(G)$
      if $\mathtt{r} = \mathtt{local}$ **then**
        $Q_{\mathtt{q}} \leftarrow [\![\mathtt{r}?[U]]\!]$ ▷ local query Q
      else
        $\mathcal{S}_{\mathtt{r}}.(Q_{\mathtt{q}}) \leftarrow$ SQT($G \restriction \mathtt{r}, \mathtt{q}$) ▷ federated query $\mathcal{S}.Q$
      end if
    end if
    $Q'_{\mathtt{q}} = Q_{\mathtt{q}} \wedge \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{q}} \wedge \cdots \wedge \mathcal{S}_{\mathtt{r}'}.Q_{\mathtt{q}}$ ▷ Conjunctive query
  end while
  return $Q'_{\mathtt{q}}$
end function

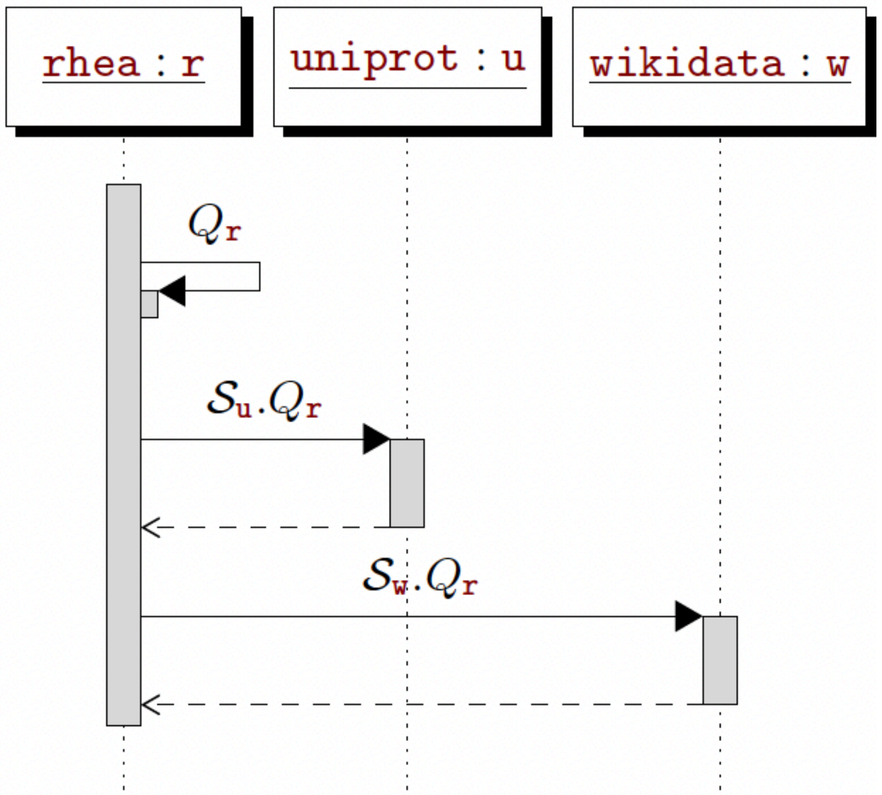
Experiments

Table 2: Construction time for single query

Number of participants in G	**Time** (second) **Projection** (total)	**SQT** (average)
10 participants	0.445	0.321
30 participants	2.720	0.724
60 participants	30.624	1.429

Table 3: Query elapsed time from different global protocol

Query Template	**Elapsed time** (second)
1. $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{r}]\!] = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}}$	23.160
2. $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{u}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{u}]\!] = \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{u}} \wedge Q_{\mathtt{u}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{u}}$	12.255
3. $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{w}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{w}]\!] = \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{w}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{w}} \wedge Q_{\mathtt{w}}$	11.334
4. $[\![(G_2 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{r}]\!] = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{u}}(Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}})$	24.500
5. $[\![(G_2 \setminus \mathtt{l} \xrightarrow{\mathtt{u}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{u}]\!] = \mathcal{S}_{\mathtt{w}}(\mathcal{S}_{\mathtt{r}}.Q_{\mathtt{u}}) \wedge Q_{\mathtt{u}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{u}}$	14.824
6. $[\![(G_2 \setminus \mathtt{l} \xrightarrow{\mathtt{w}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{w}]\!] = \mathcal{S}_{\mathtt{r}}(Q_{\mathtt{w}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{w}}) \wedge Q_{\mathtt{w}}$	13.076
7. $[\![(G_3 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{r}]\!] = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}(\mathcal{S}_{\mathtt{u}}.Q_{\mathtt{r}} \wedge Q_{\mathtt{r}})$	23.084
8. $[\![(G_3 \setminus \mathtt{l} \xrightarrow{\mathtt{u}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{u}]\!] = \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{u}} \wedge Q_{\mathtt{u}} \wedge \mathcal{S}_{\mathtt{r}}(\mathcal{S}_{\mathtt{w}}.Q_{\mathtt{u}})$	19.786
9. $[\![(G_3 \setminus \mathtt{l} \xrightarrow{\mathtt{w}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \upharpoonright \mathtt{w}]\!] = \mathcal{S}_{\mathtt{u}}(\mathcal{S}_{\mathtt{r}}.Q_{\mathtt{w}} \wedge Q_{\mathtt{w}}) \wedge Q_{\mathtt{w}}$	4.241

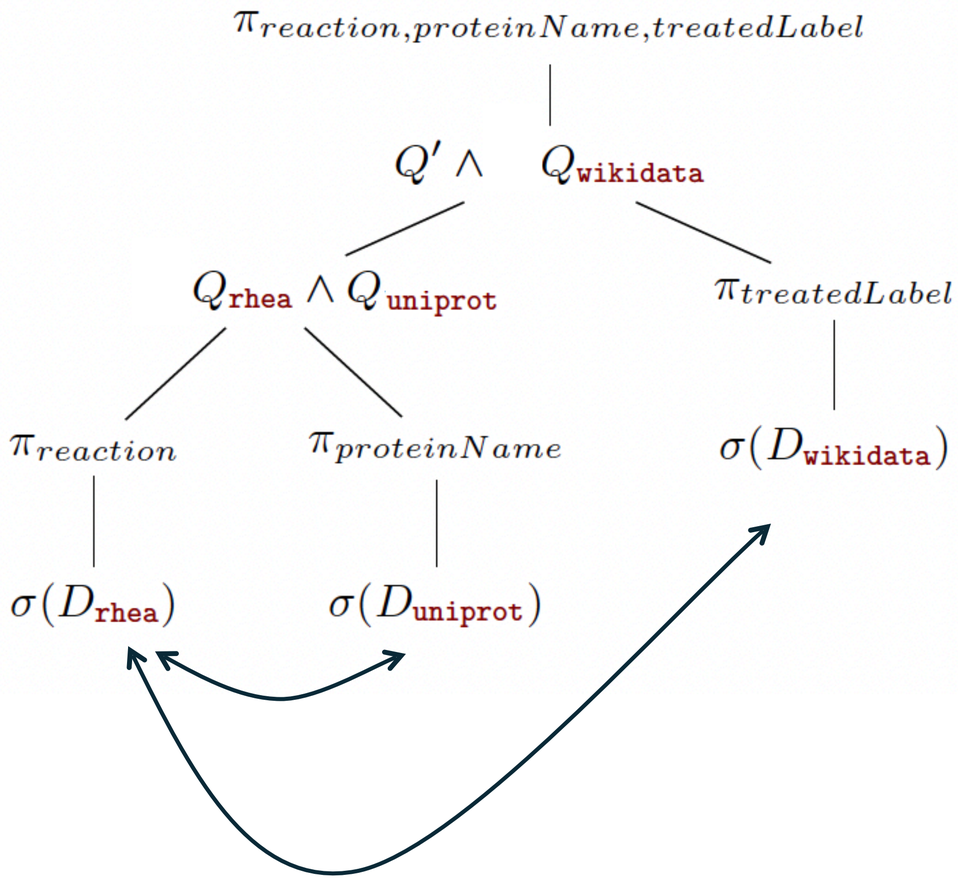


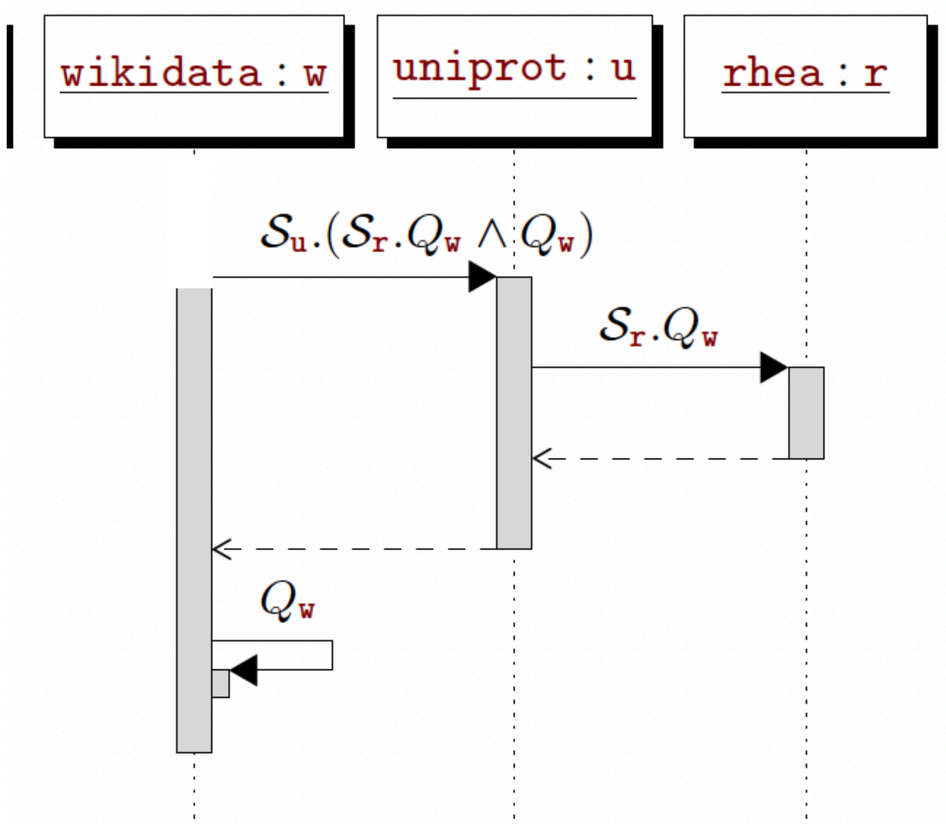
(a) Case 1: `local` (`l`) act as `rhea`

$T_{\mathtt{r}} = \mathtt{l}?[U];\mathtt{u}?[U];\mathtt{w}?[U];\mathbf{end}$

$T_{\mathtt{u}} = \mathtt{r}![U];\mathbf{end}$

$T_{\mathtt{w}} = \mathtt{r}![U];\mathbf{end}$



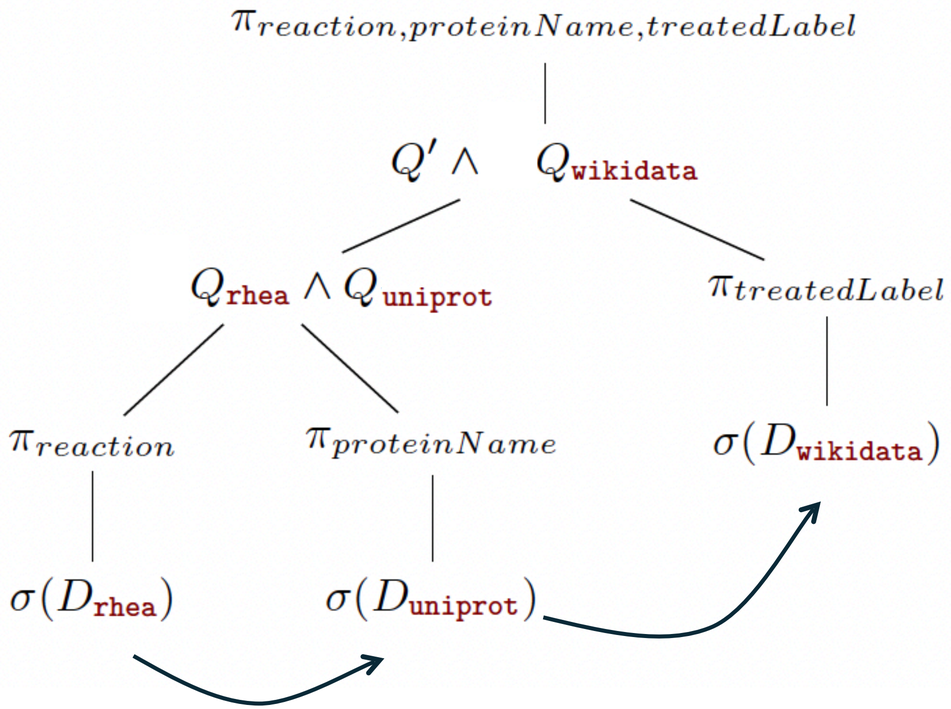


(b) Case 9 : `local` (`l`) act as `wikidata`

$T_{\mathtt{r}} = \mathtt{u}![U];\mathbf{end}$

$T_{\mathtt{u}} = \mathtt{r}?[U];\mathtt{w}![U];\mathbf{end}$

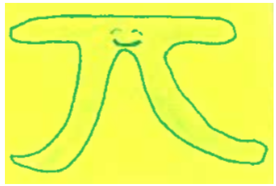
$T_{\mathtt{w}} = \mathtt{u}?[U];\mathtt{l}?[U];\mathbf{end}$



Conclusion

Specification Guided Concurrent and Distributed Programming

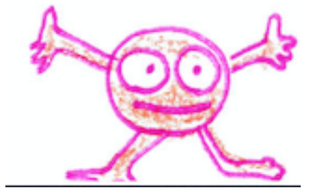
- Christopher Strachey and Structured Programming
- Introduction of Session Types
- Introduction of Multiparty Session Types
- **Case Study**: Multiparty Session Types in Rust
- **Case Study**: Federate Query with Multiparty Session Types



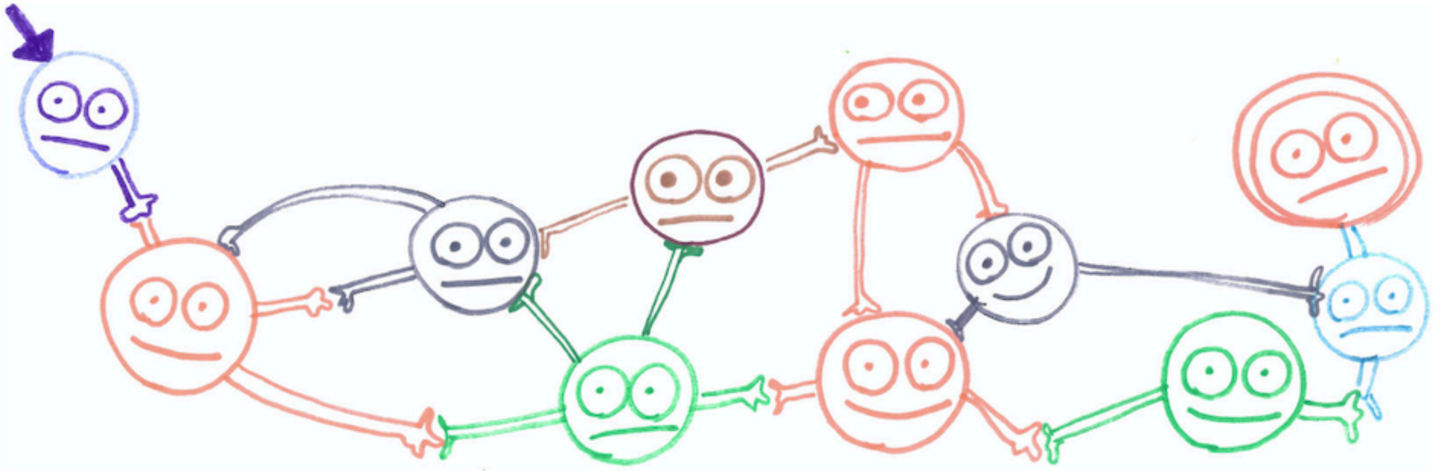




Thank you! Questions?



http://mrg.cs.ox.ac.uk/



CFSMs [1980 -] ITU notation SDL · MSCS ···

Def A CFSM $M = (Q, C, q0, \Sigma, \delta)$

Q a finite set of states

$C = \{ pq \in Participant^2 \mid p \neq q \}$

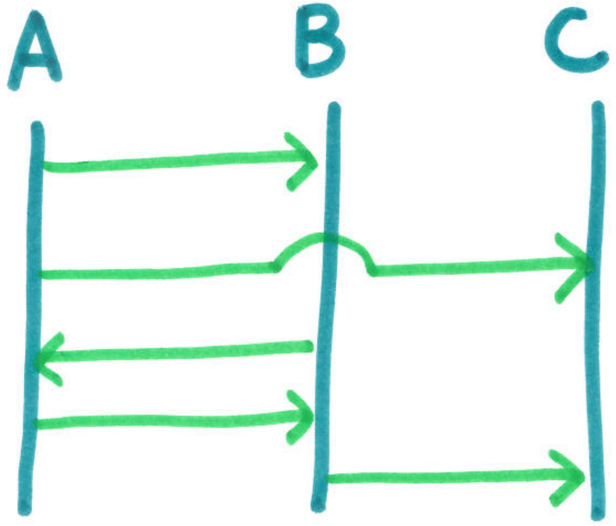
$q0$ initial state

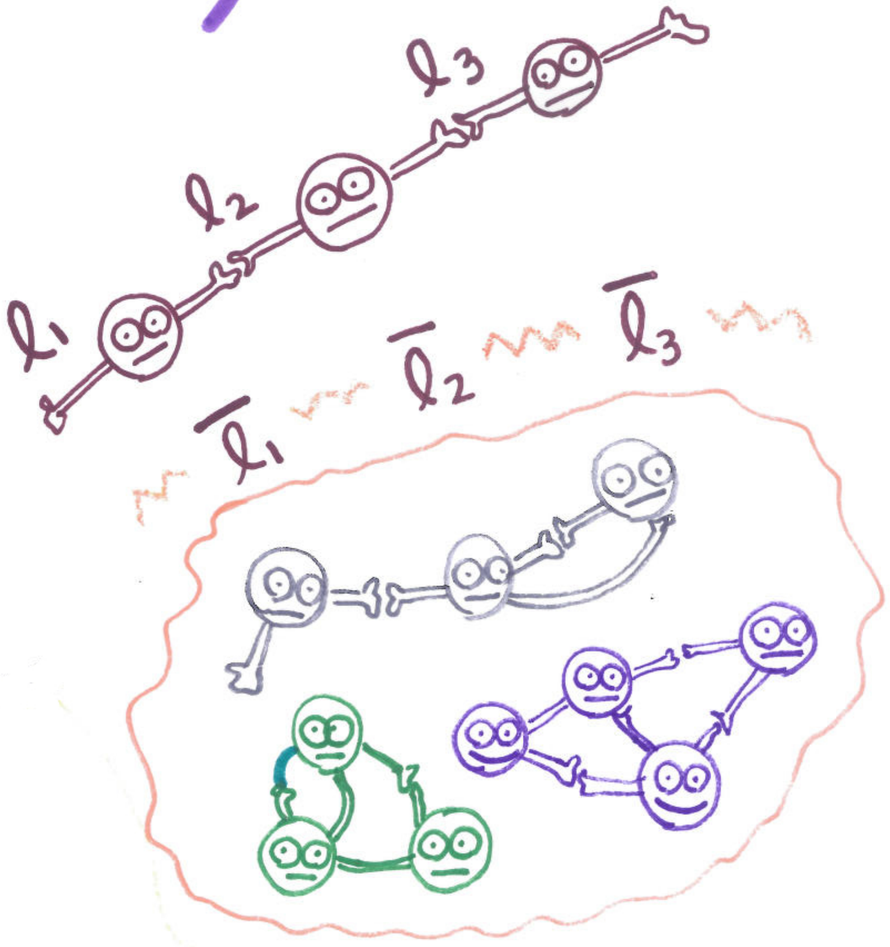
Σ a finite alphabet of messages

$\delta \subseteq Q \times (C \times \{!, ?\} \times \Sigma) \times Q$ a finite set of transitions

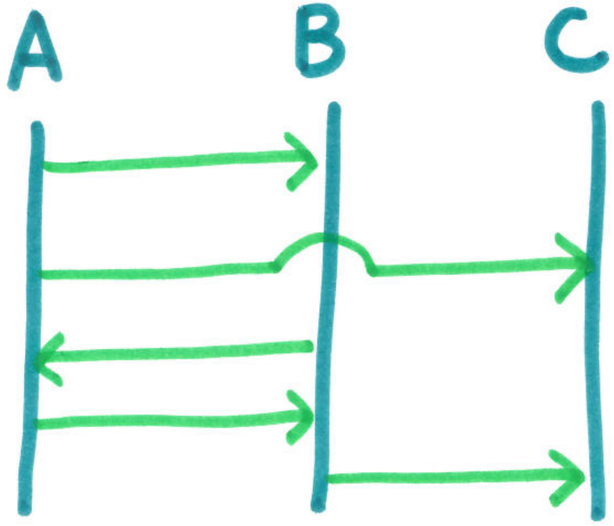
Def CS $S = (M_p)_{p \in Participant}$

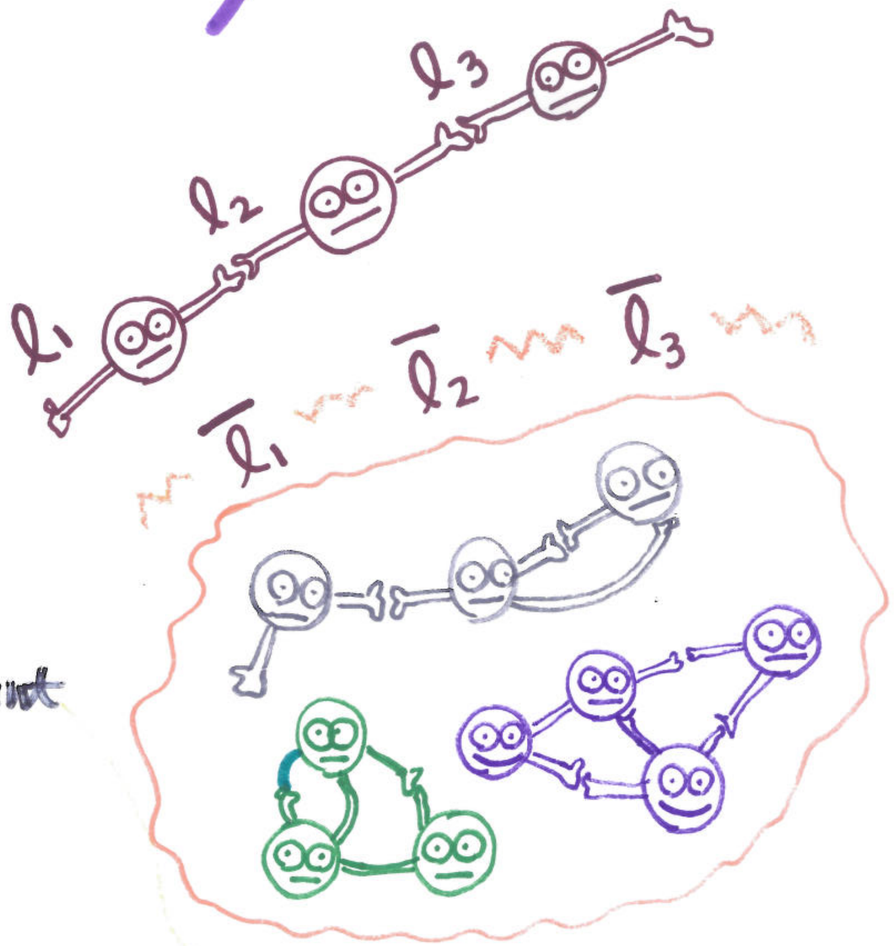
Multiparty Compatibility





Multiparty Compatibility





Def $S = (M_p)_{p \in Participant}$

$\forall s \,.\, s_0 \rightsquigarrow s$ (1-buffer execution)

if M_i does action ℓ

then $(M_j)_{j \in P \setminus i}$ do action $\bar{\ell}$

after some $\rightsquigarrow$

Multiparty Compatibility

Definition System $S = (M_p)_{p \in \mathcal{P}}$ is **MC** if for any 1-bound reachable state $s \in RS_1(S)$, and any output action $pq!a$ from s in M_p, there exists an alternation $\varphi.t$ from s in a system where $act(t) = pq?a$ and $p \notin act(\varphi)$

(Dual for Input)

$s \xrightarrow{t} s'$ configuration $s = (\vec{q}\,;\vec{w})$

states queues

q_p w_{pq}

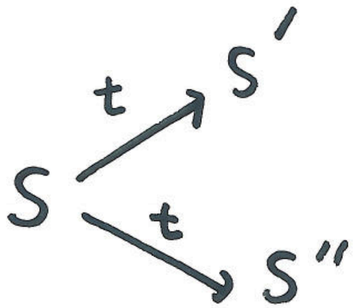
Send

$(\ldots q_p \ldots\,;\ldots w_{pq} \ldots) \xrightarrow{pq!\ell} (\ldots q_p' \ldots\,;\ldots w_{pq} \cdot \ell \ldots)$

Receive

$(\ldots q_q \ldots\,;\ldots \ell \cdot w_{pq} \ldots) \xrightarrow{pq?\ell} (\ldots q_q' \ldots\,;\ldots w_{pq} \ldots)$

Deteministic CFSM

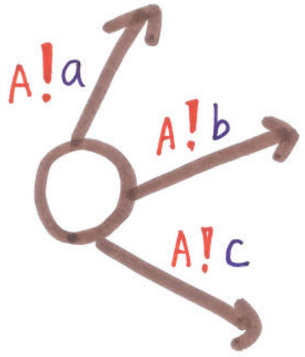


$\Rightarrow \quad s' = s''$

Basic CFSMs

A CFSM is **Basic** if **deterministic**

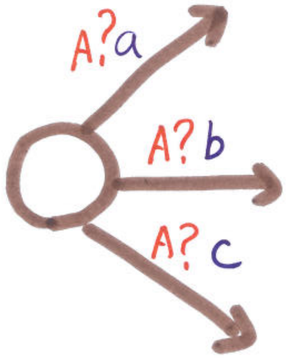
directed, has **no mixed states**



sending

✓

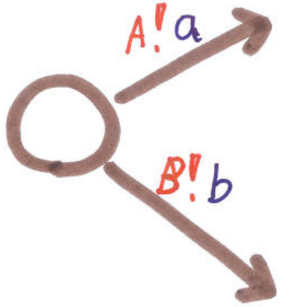
$T = A!\{a, b, c\}$



receiving

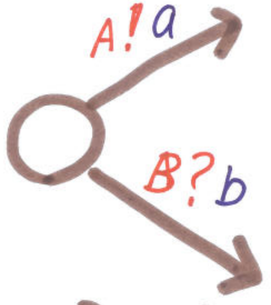
✓

$A?\{a, b, c\}$



non directed

✗



mixed

✗

k-Multiparty Compatibility [CAV'19]