Algebraic Multiparty Protocol Programming

David Castro-Perez Nobuko Yoshida

Imperial College London

December 17, 2018

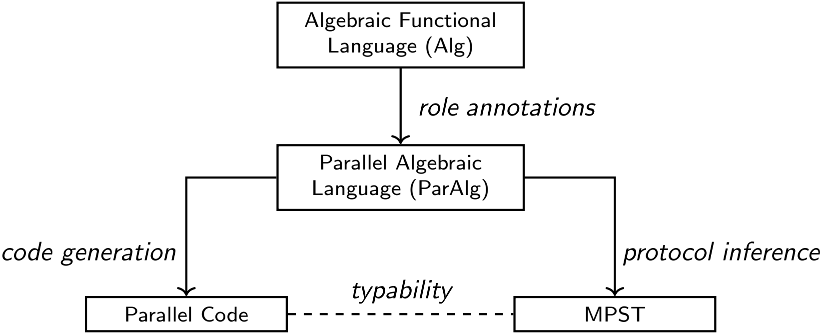
Parallel Programming

- Parallel programming is increasingly important: *many-core* architectures, GPUs, FPGAs, ...
- *Low-level* techniques are **error-prone**: deadlocks, data races, etc.
- *High-level* techniques **constraints programmers** to using a particular model, or a fixed set of parallel constructs.
- *Achieving (predictable) speedups is hard!*
- **Our goal:** generate **message-passing** parallel code from sequential implementations.
  - Not constrained by a fixed set of high-level parallel constructs.
  - Guarantee **correctness**
  - **Predictability**

Proposal : *Algebraic Multiparty Protocol Programming*

- *Algebra of programming* for specifying sequential algorithms.
  - Use **higher-order combinators**.
  - Use their equational theory for program optimisation and parallelisation.
- *Multiparty session types* for message-passing concurrency.
  - We provide an abstraction of the communication protocol of the generated parallel code as a **global type**.
  - We prove that we do not introduce concurrency errors, using the theory of *Multiparty Session Types* (MPST).
- **Key idea**: convert the *implicit data-flow* of the higher-order combinators to *explicit communication*.

Overview


Algebraic Functional Language (Alg)
role annotations
Parallel Algebraic Language (ParAlg)
code generation
protocol inference
typability
Parallel Code
MPST

Algebra of Programming

- Mathematical framework that codifies the basic laws of algorithmics. [Backus 78, Meertens 86, Bird 89].
- We define Algebraic Functional Language (Alg), a point-free functional programming language with a number of categorically-inspired combinators as syntactic constructs: composition, polynomial functors, recursion.
- Examples:
  - Function composition and identity:

$$e_1 \circ e_2 = \lambda x.\ e_1\ (e_2\ x) \qquad \mathrm{id} = \lambda x.\ x$$

$$e_1 \circ (e_2 \circ e_3) \equiv (e_1 \circ e_2) \circ e_3 \qquad \mathrm{id} \circ e \equiv e \circ \mathrm{id} \equiv e$$

  - *Split* and projections:

$$e_1 \vartriangle e_2 = \lambda x.\ (e_1\ x, e_2\ x) \quad \pi_i = \lambda(x_1,\ x_2).\ x_i$$

$$\pi_i \circ (e_1 \vartriangle e_2) \equiv e_i \qquad (e_1 \vartriangle e_2) \circ e \equiv (e_1 \circ e) \vartriangle (e_2 \circ e)$$

Algebra of Programming

- Mathematical framework that codifies the basic laws of algorithmics. [Backus 78], "Squiggol".
- We define Algebraic Functional Language (Alg), a point-free functional programming language with a number of categorically-inspired combinators as syntactic constructs: composition, polynomial functors, recursion.
- Examples:
  - Function composition and identity:

$$e_1 \circ e_2 = \lambda x.\ e_1\ (e_2\ x) \qquad \mathrm{id} = \lambda x.\ x$$

$$e_1 \circ (e_2 \circ e_3) \equiv (e_1 \circ e_2) \circ e_3 \qquad \mathrm{id} \circ e \equiv e \circ \mathrm{id} \equiv e$$

  - *Split* and projections:

$$e_1 \vartriangle e_2 = \lambda x.\ (e_1\ x, e_2\ x) \quad \pi_i = \lambda(x_1,\ x_2).\ x_i$$

$$\pi_i \circ (e_1 \vartriangle e_2) \equiv e_i \qquad (e_1 \vartriangle e_2) \circ e \equiv (e_1 \circ e) \vartriangle (e_2 \circ e)$$

Example: Cooley-Tukey FFT

- Discrete Fourier Transform

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} = E_k + e^{-\frac{2\pi i}{N} k} O_k \\ X_{k+\frac{N}{2}} &= E_k - e^{-\frac{2\pi i}{N} k} O_k \\ E_k &= \text{dft of the even-indexed part of } x_n \\ O_k &= \text{dft of the odd-indexed part of } x_n \end{aligned}$$

- Alg expression

$$\mathbf{dft}_n = (\underbrace{\text{add}}_{+} \triangle \underbrace{\text{sub}}_{-}) \circ (\underbrace{(\mathbf{dft}_{n/2} \circ \pi_1)}_{E_k} \triangle (\underbrace{\text{exp}}_{e^{-\frac{2\pi i}{N} k}} \circ \underbrace{\mathbf{dft}_{n/2} \circ \pi_2}_{O_k}))$$

Evaluating $\mathbf{dft}_n$

$$((\text{add} \vartriangle \text{sub}) \circ ((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)))(x, y)$$

Evaluating $\mathbf{dft}_n$

$$((\mathrm{add} \vartriangle \mathrm{sub}) \circ ((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)))(x, y)$$

$$= (\mathrm{add} \vartriangle \mathrm{sub})\ (((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2))(x, y))$$

Evaluating $\mathbf{dft}_n$

$$
\begin{aligned}
& ((\mathrm{add} \vartriangle \mathrm{sub}) \circ ((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)))(x, y) \\
= {} & (\mathrm{add} \vartriangle \mathrm{sub})\ (((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2))(x, y)) \\
= {} & (\mathrm{add} \vartriangle \mathrm{sub})\ ((\mathbf{dft}_{n/2} \circ \pi_1)\ (x, y), (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)\ (x, y))
\end{aligned}
$$

Evaluating $\mathbf{dft}_n$

$$((\text{add} \vartriangle \text{sub}) \circ ((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)))(x, y)$$

$$= (\text{add} \vartriangle \text{sub}) (((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2))(x, y))$$

$$= (\text{add} \vartriangle \text{sub}) ((\mathbf{dft}_{n/2} \circ \pi_1) (x, y), (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2) (x, y))$$

$$= (\text{add} \vartriangle \text{sub}) (\mathbf{dft}_{n/2}\ x,\ \exp (\mathbf{dft}_{n/2}\ y))$$

Evaluating $\mathbf{dft}_n$

$$
\begin{aligned}
& ((\text{add} \bigtriangleup \text{sub}) \circ ((\mathbf{dft}_{n/2} \circ \pi_1) \bigtriangleup (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)))(x, y) \\
= {} & (\text{add} \bigtriangleup \text{sub}) (((\mathbf{dft}_{n/2} \circ \pi_1) \bigtriangleup (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2))(x, y)) \\
= {} & (\text{add} \bigtriangleup \text{sub}) ((\mathbf{dft}_{n/2} \circ \pi_1)\,(x, y), (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)\,(x, y)) \\
= {} & (\text{add} \bigtriangleup \text{sub}) (\mathbf{dft}_{n/2}\ x,\ \exp\,(\mathbf{dft}_{n/2}\ y)) \\
= {} & (\,\text{add}\,(\mathbf{dft}_{n/2}\ x, \exp\,(\mathbf{dft}_{n/2}\ y))\,, \text{sub}\,(\mathbf{dft}_{n/2}\ x, \exp\,(\mathbf{dft}_{n/2}\ y))\,)
\end{aligned}
$$

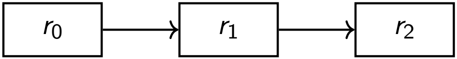
Evaluating $\mathbf{dft}_n$

$$
\begin{aligned}
X_k &= E_k + e^{-\frac{2\pi i}{N}k} O_k \\
X_{k+\frac{N}{2}} &= E_k - e^{-\frac{2\pi i}{N}k} O_k
\end{aligned}
$$

$$
\begin{aligned}
& ((\text{add} \vartriangle \text{sub}) \circ ((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)))(x, y) \\
= & (\text{add} \vartriangle \text{sub}) (((\mathbf{dft}_{n/2} \circ \pi_1) \vartriangle (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2))(x, y)) \\
= & (\text{add} \vartriangle \text{sub}) ((\mathbf{dft}_{n/2} \circ \pi_1)\ (x, y), (\exp \circ \mathbf{dft}_{n/2} \circ \pi_2)\ (x, y)) \\
= & (\text{add} \vartriangle \text{sub})\ (\mathbf{dft}_{n/2}\ x,\ \exp\ (\mathbf{dft}_{n/2}\ y)) \\
= & (\underbrace{\text{add}\ (\mathbf{dft}_{n/2}\ x, \exp\ (\mathbf{dft}_{n/2}\ y))}_{X_k}, \underbrace{\text{sub}\ (\mathbf{dft}_{n/2}\ x, \exp\ (\mathbf{dft}_{n/2}\ y))}_{X_{k+\frac{N}{2}}})
\end{aligned}
$$

ParAlg: Alg + role annotations

- We call Parallel Algebraic Language (ParAlg) to Alg extended with *role* annotations.
- $\vdash e \Rightarrow p : A \to B \mid \mathcal{C}$: "Alg expression e synthethises ParAlg expression p, with type $A \to B$ and choices $\mathcal{C}$".
- E.g.
  - $A = a@r_0 \times b@r_1$ is the product $a \times b$, where a is at r_0 and b at r_1.
  - $p = e_2@r_2 \circ e_1@r_1$ is the composition of $e_2 \circ e_1$, where e_2 is applied at r_2, and e_1 at r_1.



ParAlg: Inferring Global Types

- A *global type*, in *Multiparty Session Types*, is a global description of a communication protocol between multiple participants.
- Inferring a global type from ParAlg implies representing the *implicit dataflow* with *explicit communication*.
- $\mathcal{C} \vDash p \Leftarrow A \sim G$: "Expression p with domain A, in a choice context $\mathcal{C}$ behaves as global type G."

ParAlg	global type
$e_0@r_0 \circ e_1@r_1 : a@r \to c@r_0$	$r \to r_1 : a.\ r_1 \to r_0 : b.\ \mathsf{end}$
$e_0@r_0 \vartriangle e_1@r_1 : a@r \to b@r_0 \times c@r_1$	$r \to r_0 : a.\ r \to r_1 : a.\ \mathsf{end}$
$e_0@r_0 \triangledown e_1@r_1 : (a+b)@r \to c@r_0 \cup c@r_1$	$r \to \{r_0, r_1\}\{\mathsf{inj}_1.\ r \to r_0 : a.\ \mathsf{end},$ $\mathsf{inj}_2.\ r \to r_1 : b.\ \mathsf{end}\}$

ParAlg: Size-2 FFT protocol

$$(\mathsf{add} \quad \triangle\ \mathsf{sub} \quad) \circ ((\mathbf{dft}_{n/2} \quad \circ\ \pi_1) \triangle (\ \ \mathsf{exp} \circ \mathbf{dft}_{n/2} \quad \circ\ \pi_2))$$

ParAlg: Size-2 FFT protocol

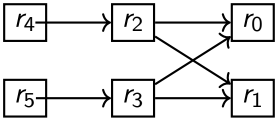
$$(\text{add}@r_0 \bigtriangleup \text{sub}@r_1) \circ ((\mathbf{dft}_{n/2}@r_2 \circ \pi_1) \bigtriangleup (\{\text{exp} \circ \mathbf{dft}_{n/2}\}@r_3 \circ \pi_2))$$

ParAlg: Size-2 FFT protocol

$$(\mathsf{add}@r_0 \bigtriangleup \mathsf{sub}@r_1) \circ ((\mathbf{dft}_{n/2}@r_2 \circ \pi_1) \bigtriangleup (\{\exp \circ \mathbf{dft}_{n/2}\}@r_3 \circ \pi_2))$$

Global type assuming that the domain is: $V@r_4 \times V@r_5$:

$r_4 \to r_2 : V.$
$r_5 \to r_3 : V.$
$r_2 \to r_0 : V.$
$r_2 \to r_1 : V.$
$r_3 \to r_0 : V.$
$r_3 \to r_1 : V.\mathsf{end}$

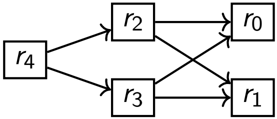


ParAlg: Size-2 FFT protocol

$$(\mathrm{add}@r_0 \vartriangle \mathrm{sub}@r_1) \circ ((\mathbf{dft}_{n/2}@r_2 \circ \pi_1) \vartriangle (\{\exp \circ \mathbf{dft}_{n/2}\}@r_3 \circ \pi_2))$$

Global type assuming that the domain is: $(V \times V)@r_4$:

$r_4 \to r_2 : V.$
$r_4 \to r_3 : V.$
$r_2 \to r_0 : V.$
$r_2 \to r_1 : V.$
$r_3 \to r_0 : V.$
$r_3 \to r_1 : V.\mathrm{end}$



Message Passing Monad(I)

- We translate ParAlg to the Message Passing Monad (Mp): `send` r x, `recv` r a, `branch` r m_1 m_2, `choice` x r f_1 f_2.
- The translation keeps track of:
  - Location of the data.
  - *Branches* in the control flow: which roles perform choices, and which roles are affected by which choice.
- For each role r in $p : A \to B$, we "project" its behaviour as a monadic action. E.g.

$$e_0@r_0 \circ e_1@r_1 : a@r \to c@r_0 \rightsquigarrow \begin{bmatrix} \mathtt{r} & \mapsto \lambda \mathtt{x}.\ \mathtt{send}\ \mathtt{r}_1\ \mathtt{x} \\ \mathtt{r}_0 & \mapsto \lambda \_.\ \mathtt{recv}\ \mathtt{r}_1\ \mathtt{b} \gg= \lambda \mathtt{x}.\ \mathtt{return}\ (\mathtt{e}_0\ \mathtt{x}) \\ \mathtt{r}_1 & \mapsto \lambda \_.\ \mathtt{recv}\ \mathtt{r}\ \mathtt{a} \gg= \lambda \mathtt{x}.\ \mathtt{send}\ \mathtt{r}_0\ (\mathtt{e}_1\ \mathtt{x}) \end{bmatrix}$$

Correctness

Theorem (Protocol Deadlock Freedom)

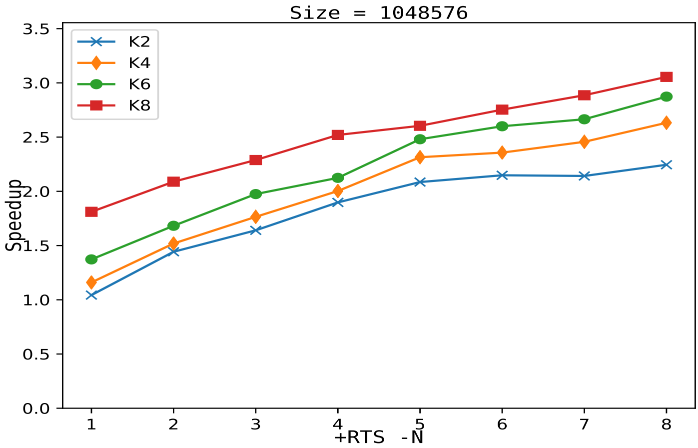
For all e, p, A, B, $\mathcal{C}$, *if* $\vdash e \Rightarrow p : A \to B \mid \mathcal{C}$, *then there exists a global type* G *s.t.* $\mathcal{C} \vDash p \Leftarrow A \sim G$, *and* G *is well-formed.*

Theorem (Deadlock Freedom of the Generated Code)

For all p, A, B, $\mathcal{C}$, G, r, *if* $\vdash e \Rightarrow p : A \to B \mid \mathcal{C}$ *and* $\mathcal{C} \vDash p \Leftarrow A \sim G$ *then* $[\![p]\!]^r_A : A \upharpoonright r \to \mathrm{Mp}\ (G \upharpoonright r)\ (B \upharpoonright r)$.

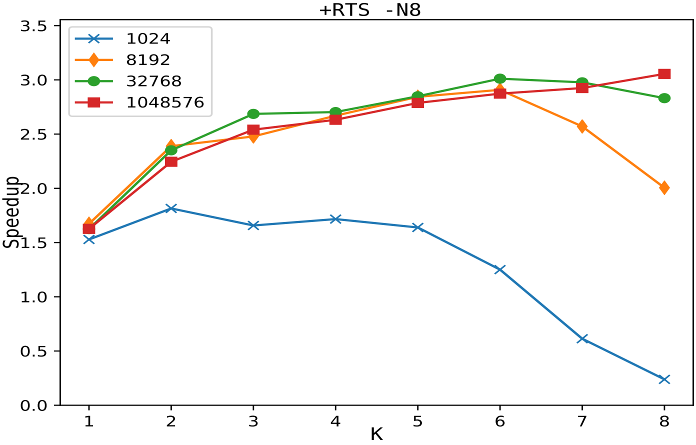
Speedups on a 4-Core Machine

FFT



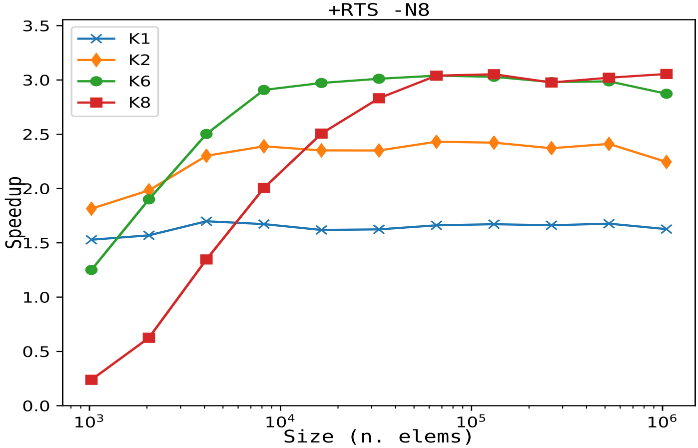
Speedups on a 4-Core Machine

FFT



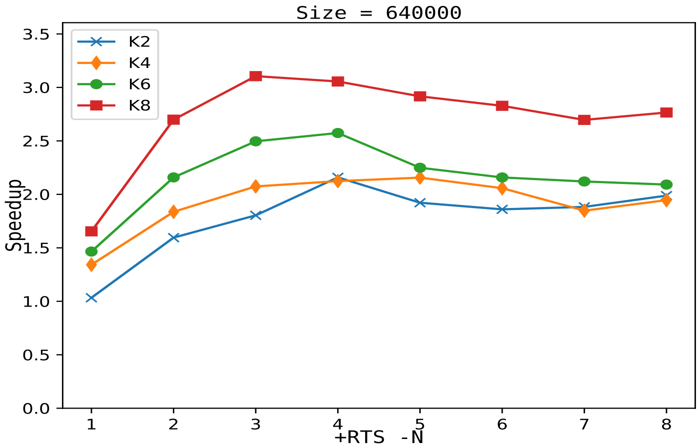
Speedups on a 4-Core Machine

FFT



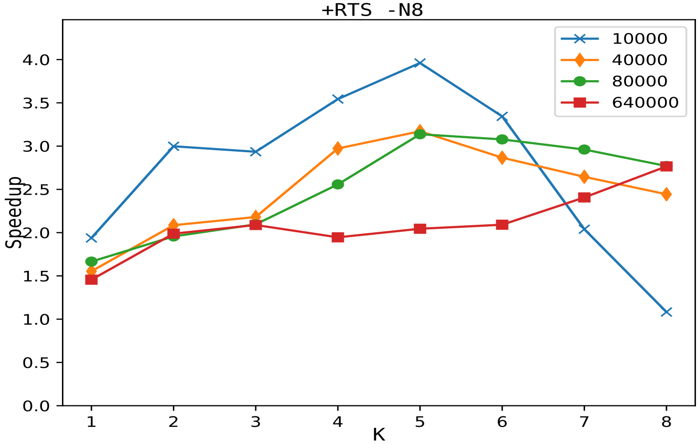
Speedups on a 4-Core Machine

Mergesort



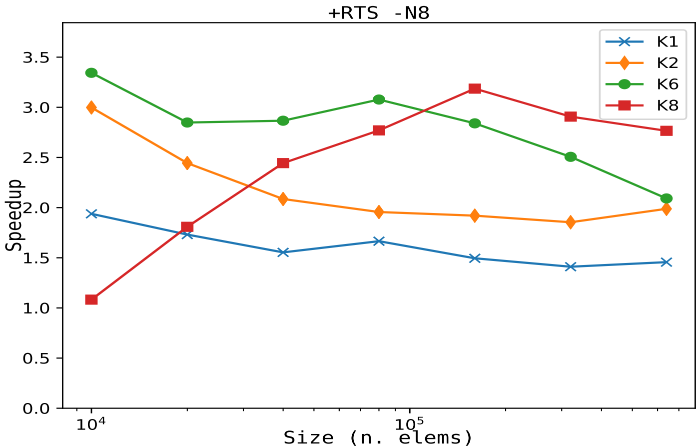
Speedups on a 4-Core Machine

Mergesort



Speedups on a 4-Core Machine

Mergesort



Conclusions

- We developed an algebraic approach to protocol inference and code generation.
- By adding role annotations, we interpret data-flow as communication.
- Different mappings of computations to roles yield different parallelisations: i.e. programmers can control how to parallelise their code by assigning parts of it to different roles.
- Global types provide valuable documentation about how a program was parallelised.

Future Work

- More examples, run on a machine with more cores.
- Explore code generation for GPUs/FPGAs.
- Support wider range of parallel patterns by using extensions to MPST: e.g. dynamic roles.
- Cost-models based on the inferred global type.
- Perform low-level code optimisations to the generated code, ensuring that the protocol is not modified.
- Implement semi-automatic strategies for rewriting programs and assigning roles.

Thank you!