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)

Parallel Algebraic Language (ParAlg)

role annotations

code generation

typability

Parallel Code

protocol inference

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:

\[ e_1 \circ e_2 = \lambda x. \ e_1 (e_2 x) \quad \text{id} = \lambda x. \ x \]

\[ e_1 \circ (e_2 \circ e_3) \equiv (e_1 \circ e_2) \circ e_3 \quad \text{id} \circ e \equiv e \circ \text{id} \equiv e \]

- *Split* and projections:

\[ e_1 \triangle e_2 = \lambda x. \ (e_1 \ x, \ e_2 \ x) \quad \pi_i = \lambda (x_1, \ x_2). \ x_i \]

\[ \pi_i \circ (e_1 \triangle e_2) \equiv e_i \quad (e_1 \triangle e_2) \circ e \equiv (e_1 \circ e) \triangle (e_2 \circ e) \]
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:
  \[ e_1 \circ e_2 = \lambda x. \ (e_1 \ x) \quad \text{id} = \lambda x. \ x \]
  \[ e_1 \circ (e_2 \circ e_3) \equiv (e_1 \circ e_2) \circ e_3 \quad \text{id} \circ e \equiv e \circ \text{id} \equiv e \]

- Split and projections:
  \[ e_1 \triangle e_2 = \lambda x. \ (e_1 \ x, e_2 \ x) \quad \pi_i = \lambda (x_1, x_2). \ x_i \]
  \[ \pi_i \circ (e_1 \triangle e_2) \equiv e_i \quad (e_1 \triangle e_2) \circ e \equiv (e_1 \circ e) \triangle (e_2 \circ e) \]
Example: Cooley-Tukey FFT

▶ Discrete Fourier Transform

\[ X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} = E_k + e^{-\frac{2\pi i}{N} k} O_k \]

\[ X_{k+\frac{N}{2}} = E_k - e^{-\frac{2\pi i}{N} k} O_k \]

\( E_k \) = dft of the even-indexed part of \( x_n \)
\( O_k \) = dft of the odd-indexed part of \( x_n \)

▶ Alg expression

\[ \text{dft}_n = (\text{add} \bigtriangleup \text{sub}) \circ ((\text{dft}_{n/2} \circ \pi_1) \bigtriangleup (\exp \circ \text{dft}_{n/2} \circ \pi_2)) \]
Evaluating $\text{dft}_n$

$$((\text{add } \triangle \text{ sub}) \circ ((\text{dft}_{n/2} \circ \pi_1) \triangle (\exp \circ \text{dft}_{n/2} \circ \pi)))(x, y)$$
Evaluating $\text{dft}_n$

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

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

\[
((\text{add } \Delta \text{ sub}) \circ ((\text{dft}_{n/2} \circ \pi_1) \Delta (\exp \circ \text{dft}_{n/2} \circ \pi_2)))(x, y)
\]

\[
= (\text{add } \Delta \text{ sub}) (((\text{dft}_{n/2} \circ \pi_1) \Delta (\exp \circ \text{dft}_{n/2} \circ \pi_2))(x, y))
\]

\[
= (\text{add } \Delta \text{ sub}) ((\text{dft}_{n/2} \circ \pi_1)(x, y), (\exp \circ \text{dft}_{n/2} \circ \pi_2)(x, y))
\]
Evaluating \( \text{dft}_n \)

\[
\begin{align*}
((\text{add} \triangle \text{sub}) \circ ((\text{dft}_{n/2} \circ \pi_1) \triangle (\exp \circ \text{dft}_{n/2} \circ \pi_2))))(x, y) \\
= (\text{add} \triangle \text{sub}) (((\text{dft}_{n/2} \circ \pi_1) \triangle (\exp \circ \text{dft}_{n/2} \circ \pi_2))(x, y)) \\
= (\text{add} \triangle \text{sub}) ((\text{dft}_{n/2} \circ \pi_1)(x, y), (\exp \circ \text{dft}_{n/2} \circ \pi_2)(x, y)) \\
= (\text{add} \triangle \text{sub}) (\text{dft}_{n/2} x, \exp (\text{dft}_{n/2} y))
\end{align*}
\]
Evaluating $\text{dft}_n$

\[
((\text{add } \triangle \text{ sub}) \circ ((\text{dft}_{n/2} \circ \pi_1) \triangle (\exp \circ \text{dft}_{n/2} \circ \pi_2)))(x, y) \\
= (\text{add } \triangle \text{ sub}) ((((\text{dft}_{n/2} \circ \pi_1) \triangle (\exp \circ \text{dft}_{n/2} \circ \pi_2))(x, y)) \\
= (\text{add } \triangle \text{ sub}) ((\text{dft}_{n/2} \circ \pi_1)(x, y), (\exp \circ \text{dft}_{n/2} \circ \pi_2)(x, y)) \\
= (\text{add } \triangle \text{ sub}) (\text{dft}_{n/2} x, \exp (\text{dft}_{n/2} y)) \\
= (\text{add } (\text{dft}_{n/2} x, \exp (\text{dft}_{n/2} y)) , \text{sub } (\text{dft}_{n/2} x, \exp (\text{dft}_{n/2} y)))
\]
Evaluating $\text{dft}_n$

\[
X_k = E_k + e^{-\frac{2\pi i}{N} k} O_k \\
X_{k + \frac{N}{2}} = E_k - e^{-\frac{2\pi i}{N} k} O_k
\]

\[
(((\text{add } \triangle \text{ sub}) \circ (((\text{dft} \frac{n}{2} \circ \pi_1) \triangle (\exp \circ \text{dft} \frac{n}{2} \circ \pi_2))))(x, y))
\]

\[
= (\text{add } \triangle \text{ sub}) (((\text{dft} \frac{n}{2} \circ \pi_1) \triangle (\exp \circ \text{dft} \frac{n}{2} \circ \pi_2))(x, y))
\]

\[
= (\text{add } \triangle \text{ sub}) ((\text{dft} \frac{n}{2} \circ \pi_1) (x, y), (\exp \circ \text{dft} \frac{n}{2} \circ \pi_2) (x, y))
\]

\[
= (\text{add } \triangle \text{ sub}) (\text{dft} \frac{n}{2} x, \exp (\text{dft} \frac{n}{2} y))
\]

\[
= (\text{add } (\text{dft} \frac{n}{2} x, \exp (\text{dft} \frac{n}{2} y)), \text{sub } (\text{dft} \frac{n}{2} x, \exp (\text{dft} \frac{n}{2} y)))
\]

\[
X_k \\
X_{k + \frac{N}{2}}
\]
We call Parallel Algebraic Language (ParAlg) to Alg extended with *role* annotations.

⊢ e ⇒ p : A → B | C: “Alg expression e synthethises ParAlg expression p, with type A → B and choices C”.

E.g.

- A = a@r_0 × b@r_1 is the product a × b, where a is at r_0 and b at r_1.
- p = e_2@r_2 ◦ e_1@r_1 is the composition of e_2 ◦ 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*.
- $C \vdash p \iff A \sim G$: “Expression $p$ with domain $A$, in a choice context $C$ behaves as global type $G$.”

<table>
<thead>
<tr>
<th>ParAlg</th>
<th>global type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0 @ r_0 \circ e_1 @ r_1 : a @ r \rightarrow c @ r_0$</td>
<td>$r \rightarrow r_1 : a. r_1 \rightarrow r_0 : b. \text{end}$</td>
</tr>
<tr>
<td>$e_0 @ r_0 \triangledown e_1 @ r_1 : a @ r \rightarrow b @ r_0 \times c @ r_1$</td>
<td>$r \rightarrow r_0 : a. r \rightarrow r_1 : a. \text{end}$</td>
</tr>
<tr>
<td>$e_0 @ r_0 \triangledown e_1 @ r_1 : (a + b) @ r \rightarrow c @ r_0 \cup c @ r_1$</td>
<td>$r \rightarrow {r_0, r_1}{\text{inj}_1. r \rightarrow r_0 : a. \text{end, inj}_2. r \rightarrow r_1 : b. \text{end}}$</td>
</tr>
</tbody>
</table>
ParAlg: Size-2 FFT protocol

\[(\text{add } \triangle \text{ sub }) \circ ((\text{dft}_{n/2} \circ \pi_1) \triangle (\exp \circ \text{dft}_{n/2} \circ \pi_2))\]
ParAlg: Size-2 FFT protocol

\[(\text{add} \circ r_0 \triangle \text{sub} \circ r_1) \circ ((\text{dft}_{n/2} \circ r_2 \circ \pi_1) \triangle (\{\exp \circ \text{dft}_{n/2}\} \circ r_3 \circ \pi_2))\]
ParAlg: Size-2 FFT protocol

\[(\text{add} \circ r_0 \triangle \text{sub} \circ r_1) \circ ((\text{dft}_{n/2} \circ r_2 \circ \pi_1) \triangle (\{\exp \circ \text{dft}_{n/2}\} \circ r_3 \circ \pi_2))\]

Global type assuming that the domain is: \(V \circ r_4 \times V \circ r_5:\)

\[r_4 \to r_2 : V.\]
\[r_5 \to r_3 : V.\]
\[r_2 \to r_0 : V.\]
\[r_2 \to r_1 : V.\]
\[r_3 \to r_0 : V.\]
\[r_3 \to r_1 : V.\text{end}\]
ParAlg: Size-2 FFT protocol

\[(\text{add}_0 \triangle \text{sub}_{r_1}) \circ ((\text{dft}_{n/2} @ r_2 \circ \pi_1) \triangle (\{\exp \circ \text{dft}_{n/2}\} @ r_3 \circ \pi_2))\]

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

\[r_4 \rightarrow r_2 : V.\]
\[r_4 \rightarrow r_3 : V.\]
\[r_2 \rightarrow r_0 : V.\]
\[r_2 \rightarrow r_1 : V.\]
\[r_3 \rightarrow r_0 : V.\]
\[r_3 \rightarrow r_1 : V.\text{end}\]
Message Passing Monad (I)

- We translate ParAlg to the Message Passing Monad (Mp):
  send \( r \times \), 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 \( p : A \rightarrow B \), we “project” its behaviour as a monadic action. E.g.

\[
e_{0}@r_0 \circ e_{1}@r_1 : a@r \rightarrow c@r_0 \sim\sim \\
\begin{align*}
  & r \mapsto \lambda x. \text{send} \ r_1 \ x \\
  & r_0 \mapsto \lambda _. \text{recv} \ r_1 \ b \ggg \lambda x. \text{return} \ (e_0 \ x) \\
  & r_1 \mapsto \lambda _. \text{recv} \ r \ a \ggg \lambda x. \text{send} \ r_0 \ (e_1 \ x)
\end{align*}
\]
Correctness

Theorem (Protocol Deadlock Freedom)
For all $e$, $p$, $A$, $B$, $C$, if $\vdash e \Rightarrow p : A \to B \mid C$, then there exists a global type $G$ s.t. $C \vDash p \Leftrightarrow A \sim G$, and $G$ is well-formed.

Theorem (Deadlock Freedom of the Generated Code)
For all $p$, $A$, $B$, $C$, $G$, $r$, if $\vdash e \Rightarrow p : A \to B \mid C$ and $C \vDash p \Leftrightarrow A \sim G$ then $\llbracket p \rrbracket_A : A \upharpoonright r \to Mp(G \upharpoonright r)(B \upharpoonright r)$. 
Speedups on a 4-Core Machine

FFT

Size = 1048576
Speedups on a 4-Core Machine

FFT

+RTS -N8

1024
8192
32768
1048576

Speedup

K

0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Speedups on a 4-Core Machine

FFT
Speedups on a 4-Core Machine

Mergesort

Size = 640000

K2
K4
K6
K8
Speedups on a 4-Core Machine

Mergesort

![Graph showing speedups on a 4-core machine for Mergesort with different inputs. The x-axis represents K values from 1 to 8, and the y-axis represents speedup. The graph includes lines for +RTS -N8 with input sizes of 10000, 40000, 80000, and 640000.]
Speedups on a 4-Core Machine

Mergesort

![Graph showing speedup for Mergesort on a 4-core machine with different configurations. The x-axis represents the size of the data in million elements, and the y-axis represents the speedup. The graph includes lines and markers for different configurations labeled K1, K2, K6, and K8. The graph shows varying speedups across different data sizes.]
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!