Programming with STOPS: Session Types, OwnershiP & Sizes

Imperial College London

Juliana Franco, Sophia Drossopoulou and Nobuko Yoshida | {j.vicente-franco,s.drossopoulou,n.yoshida} @imperial.ac.uk | Department of Computing

Motivation

The present and the future: In the near future, manufacturers will move to a manycore paradigm, where hundreds, or even thousands, of cores on a single chip are expected.

Multicores everywhere!
Quadcore in your pocket!

Nexus 5 equipped with quadcore CPU. (`http://www.google.com/nexus/5/`); **Tianhe-2**, developed by China's National University of Defense Technology, is the world's no 1 super computer since June 2013, with a total of 3,120,000 cores according to `http://www.top500.org`.

Programming languages that allow the development of applications that efficiently exploit such resources are needed.

Writing programs that scale with increasing number of cores should be as easy as writing programs for sequential computers.

Asanovic et al. [1]

Our aim

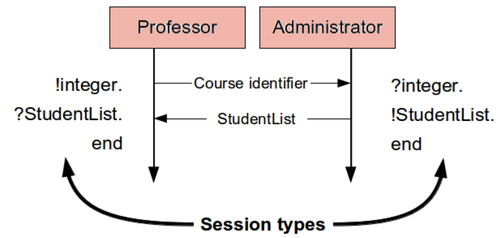
Design and implement a new object-oriented programming language that combines binary session types, ownership types and size annotations to **calculate communication costs**.

UPSCALE

`http://www.upscale-project.eu/`

Background: session types [3, 5]

- Session types describe message-passing communications.
- Session types make possible to ensure communication safety, absence of deadlocks and race conditions through static checking.



Binary session types example.

Background: ownership types [2, 4]

- In object oriented languages, aliasing is a *dangerous* and powerful feature.
- Ownership types were first introduced as *flexible alias protection* in order to limit the changes of objects via aliases.
- Each object owns zero or more objects and each object has a single owner.
- Applications: concurrency without race conditions, parallelism without locks, object cloning, garbage collection ...

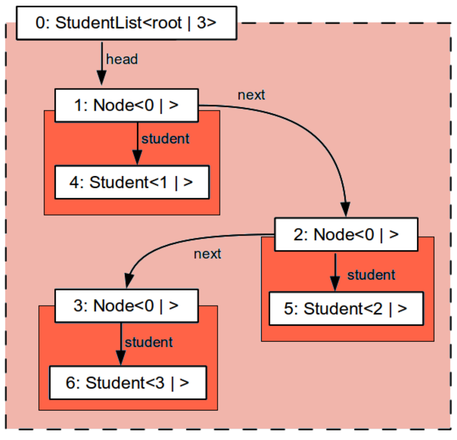
```
class StudentList<o> {
  Node<this> head;    }
class Node<o> {
  Node<o> next; Student<this> student; }
class Student<o> {
  String name; integer age; }
```

Our contribution: amalgamating session types and ownership types

(1) Extend the ownership types in order to express the size of data structures

```
class StudentList<o | N> {
  @has up_to N: Node<this | > head; }
class Node<o | > {
  Node<o | > next;
  @has 1: Student<this> student; }
class Student<o | > {
  String name; integer age; }
```

The heap topology for StudentList<root | 3> is:



(2) Extend the session types to express the number of repetitions of a given behaviour

The professor may ask for u objects of type StudentList<o | n> (owned by o and with a maximum size of n)

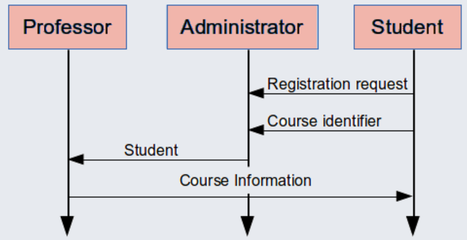
```
session Professor<o | u, n> =
  rec a. !integer. ?StudentList<o | n>. a[u]
```

(3) The communication cost of a channel governed by the Professor session type is:
The cost of sending u values of type **integer** plus the cost of receiving u objects of type StudentList<o | n> (the cost of sending n objects of type Node).

(4) Further questions:
Do we copy the students or do we send the reference? How many students does the professor receive?

What's next?

- Use the communication cost information to improve the performance of the program.
  Change the topology of the objects,
  change the order of operations, ...
- From STOPS to MultiSTOPS: extend the language for multiparty session types.



- Extend the language for distributed systems.
- Provide the implementation of well-known patterns of parallelization and communication.
- Optimise the code development process.

References

[1] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick.
A view of the parallel computing landscape.
Commun. ACM, 52:56–67, 2009.

[2] David G. Clarke, John M. Potter, and James Noble.
Ownership types for flexible alias protection.
In *OOPSLA '98*, pages 48–64. ACM, 1998.

[3] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo.
Language primitives and type disciplines for structured communication-based programming.
In *European Symposym on Programming*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

[4] James Noble, Jan Vitek, and John Potter.
Flexible alias protection.
In *ECOOP'98*, pages 158–185. Springer-Verlag, 1998.

[5] Vasco T. Vasconcelos.
Fundamentals of session types.
Information and Computation, 217:52–70, 2012.