Session Types in ATS

Hanwen Wu, Boston University

Nov 2015, NEPLS

Overview

- **Session Types** enforce correct implementation of communication protocols in distributed programming. Global progress is guaranteed.
- Session Types originates from Honda, Vasconcelos, and Kubo in 1998. Since then, it is further extended and developed by a variety of researchers including Caires, Gay, Pfenning, Wadler, Yoshida, etc.
- **ATS** is a statically typed functional language with DML-style dependent types and linear types.
- **Session types** can be readily implemented **in ATS**.

Introduction

Idea: assign proper types to communication channels that conform to the protocol.

an example channel's type (has been simplified):

```
chsnd(int) :: chrcv(bool) :: nil
```

It will be, and can only be used for first sending an integer, then receiving a boolean, and finally terminating communication.

Introduction

Idea: provided functions will use channels and update their types to conform to protocol.

an example function type (has been simplified):

```
fun send
{a:vt@ype} {ss:type}
(!chan(chsnd(a)::ss) >> chan(ss), a): void
```

a channel's type before function call (has been simplified):

```
chsnd(int) :: chrcv(bool) :: nil
```

the channel's type after function call (has been simplified):

```
chrcv(bool) :: nil
```

Session Types in ATS

Types of some built-in functions provided by ATS.

```
fun channeg_create {ss:type}
      (chanpos(ss) -<lincloptr1> void): channeg(ss)

fun chanpos_send {a:vt@ype} {ss:type}
      (!chanpos(chsnd(a)::ss) >> chanpos(ss), a): void

fun chanpos_recv {a:vt@ype} {ss:type}
      (!chanpos(chrcv(a)::ss) >> chanpos(ss)): a

fun channeg_nil_close (channeg(chnil)): void
```

Note:

- Positive channels are endpoints hold by the server side.
- Negative channels are endpoints hold by the client side.

Session Types in ATS

Types for some functions in the demo.

```
fun counter (int): channeg (rpt int)
fun c_loop (chanpos (rpt int), int): void

fun filter (channeg (rpt int), int): channeg (rpt int)
fun f_loop (chanpos (rpt int), channeg (rpt int), int): void
```

Note:

- `rpt int` means “the server side repeatedly send integers”.

Demo:
Prime Number Generator

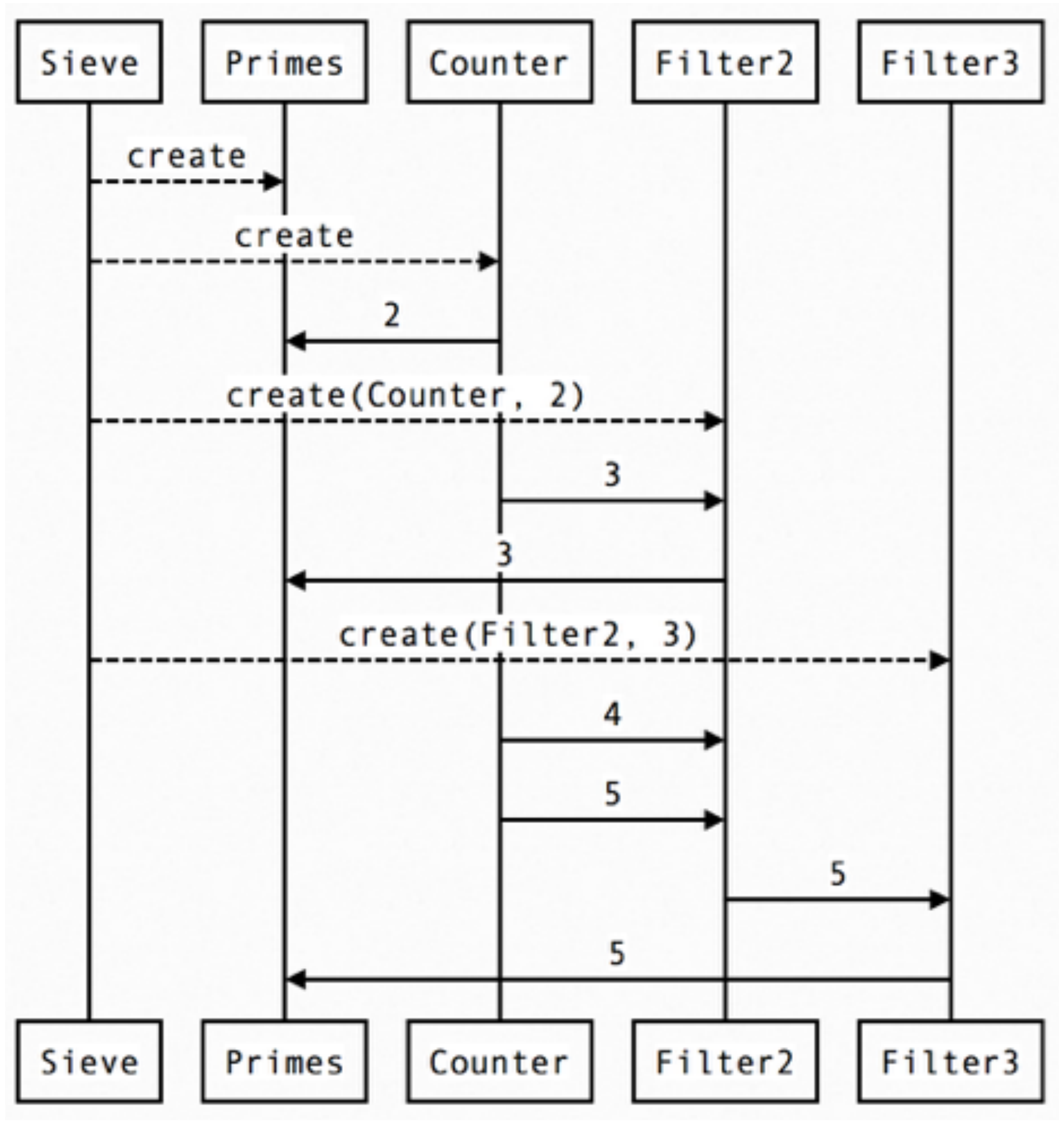
pseudo code:

```
counter_loop(counter, N):
  send(counter, N)
  counter_loop(counter, N+1)

filter_loop(in, out, P):
  N = recv(in)
  if N % P != 0
  then send(out, N)
  else filter_loop(in, out, P)

filter(in, P):
  mod_n = new channel
  spawn filter_loop(in, mod_n, P)
  return mod_n

primes_loop(in, primes):
  N = recv(in)
  send(primes, N)
  filter_n = filter(in, P)
  sieve_loop(filter_n, primes)
```



Demo Explained

- **Demo 1:**
  - Channel is untyped, ConcurrentML style. Its usage is completely unrestricted.
  - **Cons**: prone to incorrect usage of channel, which usually causes deadlock, and it is hard to debug.
- **Demo 2:**
  - Channel is session typed for sending/receiving infinite number of integers.
  - **Pros**: channel usage is forced to be correct, e.g. programmers can't mistakenly use a sending channel for receiving, etc.
  - **Cons**: session doesn't terminate
- **Demo 3:**
  - Channel is session typed for either continuing to send/receive or terminating.
  - **Pros**: channel is forced to be closed after all intended usages. e.g. programmer is forced to close the channel in the end.

Demo

http://steinwaywhw.github.io/nepls-15-demo/

requires ATS/Erlang/Elixir to be installed

Advantages

- Global progress (deadlock-free) is guaranteed.
- Session protocol is strictly enforced through type checking.
- Resource leaking is prevented through linear typed channels.
- Extensive support of distributed computing through compiling into Erlang.
- ATS co-programming with Erlang.
- Asynchronous session.
- Session type is part of the language instead of an embedding. Utilizing everything provided by ATS, e.g. dependent type (DML-style), linear type, and proofs.

Q&A

Thanks!
for more info
http://steinwaywhw.github.io/nepls-15-demo/

Backup

- Session types in ATS supports:
  - dependent types, linear types
  - high-order sessions (mobile sessions)
  - dyadic session for now
- Implementation details
  - Sessions are not symmetric. two endpoints are denoted negative(client) and positive(server) respectively. Though symmetric can be implemented in ATS, too.
  - Global progress is proved based on a formalization of multi-threaded linear lambda calculus extended with channels.
  - A channel is a process in Erlang.