**Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages**

# Message passing, contracts
# Section and Practice Problems

Monday April 13, 2015

---

## 1 Synchronous sends and receives

(a) To make sure you understand the operational semantics of sends and receives, show the execution of the following program. The initial configuration consists of a single thread, and your answer should show the sequence of configurations that program execution will step through.

```
let c = newchan_int in
spawn (1 + recv from c);
send 6 to c
```

(b) Consider the following program. What are the possible final values of the main (i.e., initial) thread?

```
let c = newchan_int in
let d = newchan_int in
let g = newchan_int chan in
spawn send c to g;
spawn send d to g;
spawn send 3 to c;
spawn send 4 to c;
spawn send 5 to d;
spawn send 6 to d;
(recv from (recv from g)) + recv from c
```

(c) Write a program that creates a channel $c$, and then sends the Fibonacci sequence over the channel (i.e., it sends 0, 1, 1, 2, 3, 5, 8, ...). Your program will spawn a thread to compute and send the Fibonacci sequence, and the other thread will consume the values produced.

## 2 First-class synchronization

(a) Write a program that creates a channel $c$, and then sends the Fibonacci sequence over the channel (i.e., it sends 0, 1, 1, 2, 3, 5, 8, ...), but does not block waiting for the sequence to be consumed. Spawn one thread to compute and send the Fibonacci sequence, but make the other thread consume (i.e., receive) just a single value from the channel. What are the possible values received by the other thread?

(b) Consider the following program, which uses events. Using the translation given in class, translate it to the language with blocking sends and receives without events.

```
let c = newchan_int in
spawn (sendEvt 8 to c; send 9 to c);
let r = recvEvt from c in
(recv from c) + sync r
```

## 3 Dynamic types and contracts

(a) To make sure you understand the operational semantics of dynamic types and exceptions, show the execution of the following program under the semantics of the first section of the lecture notes.

$$\begin{array}{l}\mathsf{let}\ f = \lambda x.\, 42 + x\ \mathsf{in}\\ \mathsf{let}\ g = \lambda y.\, y\ (\mathsf{true}) + 42\ \mathsf{in}\\ g\ f\end{array}$$

(b) Modify the program from question (a) by adding appropriate error handlers to catch the type error and return the integer $42$ as the final result of the program. There are multiple places in the program where you can insert an error handler to achieve the desired result. Show three variations and their executions.

(c) Modify the program from question (a) by adding appropriate dynamic type checks to raise the error as early as possible. Show the execution of the modified program and compare it with the execution from your answer to question (a).

(d) Modify the program from question (a) by adding contracts that specify the types of the input and output of $f$ and $g$ (ignore blame labels for this question). Show the execution of the modified program.

(e) To make sure you understand the operational semantics of contracts and blame assignment, show the execution of the following program.

$$\begin{array}{l}\mathsf{let}\ f = \mathsf{monitor}(\lambda x.\, x + 1, \mathsf{is\_int?} \longmapsto \mathsf{is\_int?}\ , f, main)\ \mathsf{in}\\ \mathsf{let}\ g = \mathsf{monitor}(\lambda x.\, x\ (42) + 42, (\mathsf{is\_int?} \longmapsto \mathsf{is\_int?}\ ) \longmapsto \mathsf{is\_int?}\ , g, main)\ \mathsf{in}\\ \mathsf{let}\ h = \mathsf{monitor}(\lambda x.\, \lambda y.\, y\ 42, \mathsf{is\_int?} \longmapsto ((\mathsf{is\_int?} \longmapsto \mathsf{is\_bool?}\ ) \longmapsto \mathsf{is\_bool?}\ ), h, main)\ \mathsf{in}\\ h\ (g\ f)\ (\lambda z.\, z + 42)\end{array}$$

(f) The function contracts we discussed in the lecture and the lecture notes specify only the input and output behavior of functions that successfully terminate. One aspect of the behavior of a function they do not specify is the function's exceptional behavior. Consider the following variant of function contracts: $e_1 \overset{x.e_3}{\longmapsto} e_2$. Expressions $e_1$ and $e_2$ are the contract of the argument and result as before. The role of $e_3$ is that when $e_1 \overset{x.e_3}{\longmapsto} e_2$ protects a function $f$ and the evaluation of the body of $f$ raises an exception $\mathsf{Err}\ v$ then $v$ is substituted for $x$ in $e_3$. If the result of the expression $e_3\{v/x\}$ is $\mathsf{true}$ then we return $\mathsf{Err}\ v$. If the result is $\mathsf{false}$ then we return a contract violation blaming the provider of $f$. In essence $x.e_3$ is a predicate that describes the valid exceptional behavioral of a function. Extend the contract calculus from the lecture notes to describe the semantics of these new function contracts. Hint: modify the inference rules for function contracts from the lectures to insert the appropriate exception handlers.