Design and Implementation of Concurrent C0

Max Willsey
Advisor: Frank Pfenning

Carnegie Mellon University
School of Computer Science
Senior Honors Thesis

May 4, 2016

CC0 in 1 Sentence

Concurrent C0 uses session types to make concurrent message passing safer and more efficient than in other programming languages.

CC0 in 1 Sentence

Concurrent C0 uses session types to make concurrent message passing safer and more efficient than in other programming languages.

```
typedef <?int; !bool;> protocol;
```

CC0 in 1 Sentence

Concurrent C0 uses session types to make concurrent message passing safer and more efficient than in other programming languages.

```
typedef <?int; !bool;> protocol;
```

Provider:

```
protocol $c even() {
  int x = recv($c);
  send($c, x%2==0);
  close($c);
}
```

CC0 in 1 Sentence

Concurrent C0 uses session types to make concurrent message passing safer and more efficient than in other programming languages.

```
typedef <?int; !bool;> protocol;
```

Provider:

```
protocol $c even() {
  int x = recv($c);
  send($c, x%2==0);
  close($c);
}
```

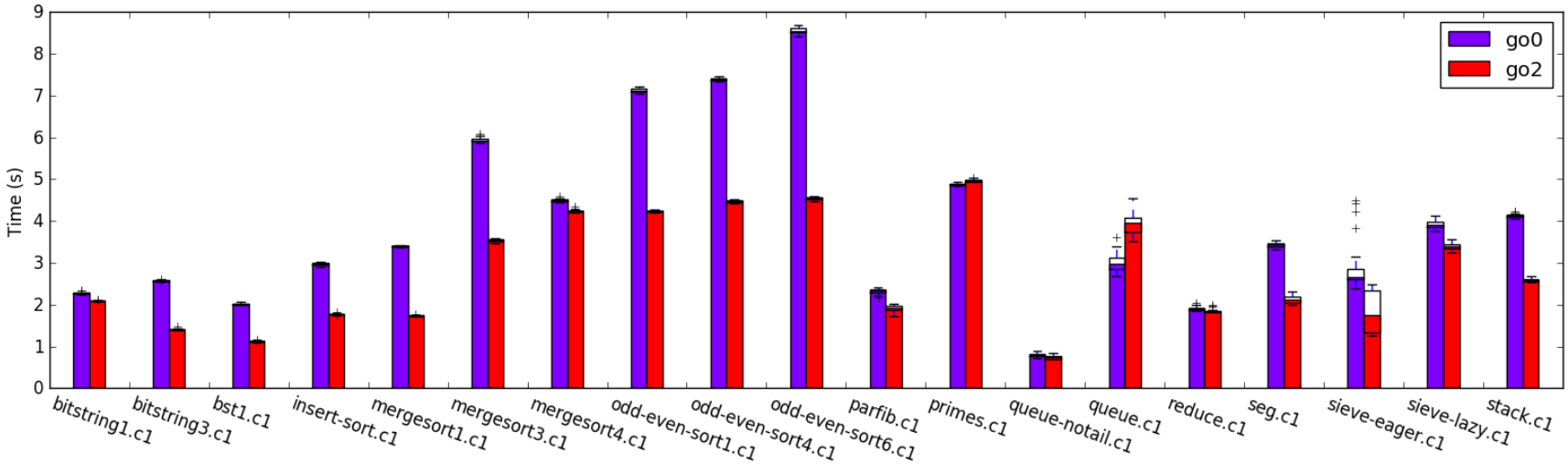
Client:

```
protocol $c = even();
send($c, 4);
assert(recv($c));
wait($c);
```

CC0 in 1 Sentence

Concurrent C0 uses session types to make concurrent message passing safer and more efficient than in other programming languages.

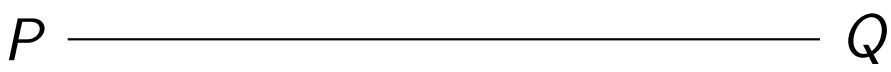
1.38× faster



Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.



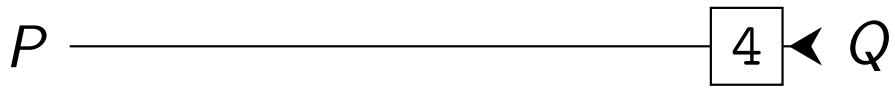

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.




```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.

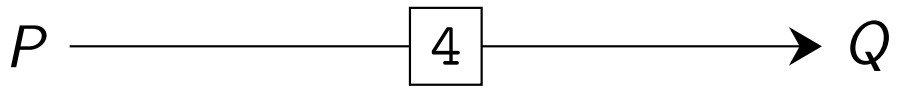

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.



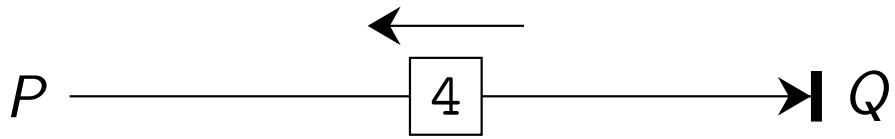

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.



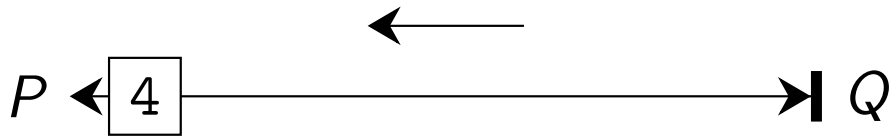

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.



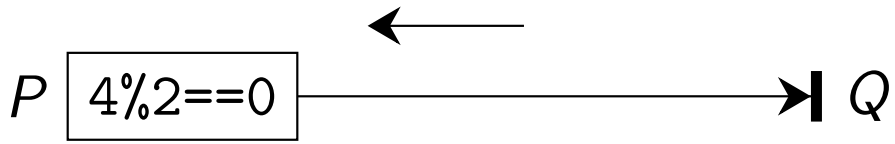

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.



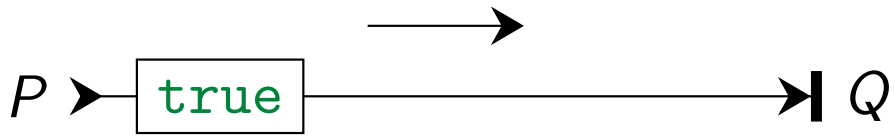

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.



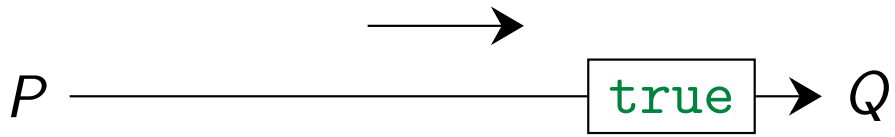

```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

Synchronization Points

Communication in CC0 is asynchronous, but we know the direction of communication, so we only need one buffer.




```
int x = recv($c);
send($c, x%2==0);
```

```
send($c, 4);
recv($c);
```

Contributions

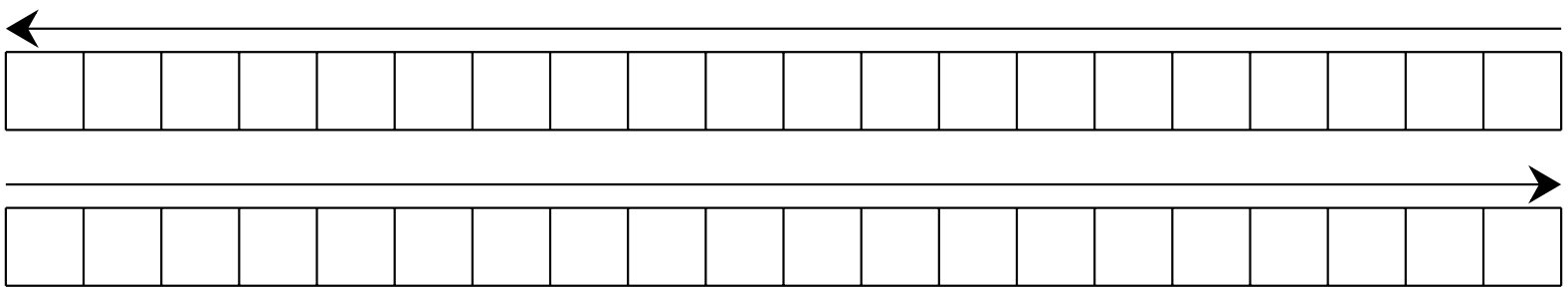
Type Width

```
typedef <?choice request> atm;
choice request {
    <?amount; !balance; atm>  Deposit;
    <?amount; !choice result> Withdraw;
};
choice result {
    <!payment; atm> Success;
    <atm>           Overdraft;
};
```

Contributions

Type Width

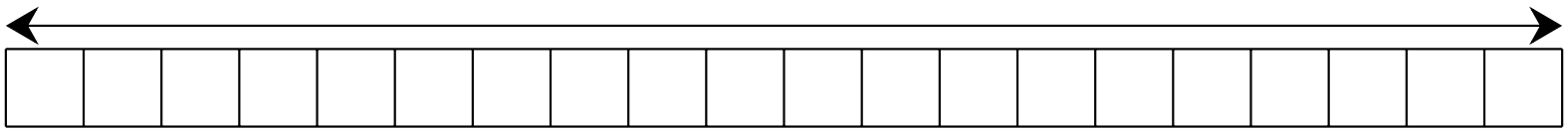
```
typedef <?choice request> atm;
choice request {
    <?amount; !balance; atm>  Deposit;
    <?amount; !choice result> Withdraw;
};
choice result {
    <!payment; atm> Success;
    <atm>           Overdraft;
};
```



Contributions

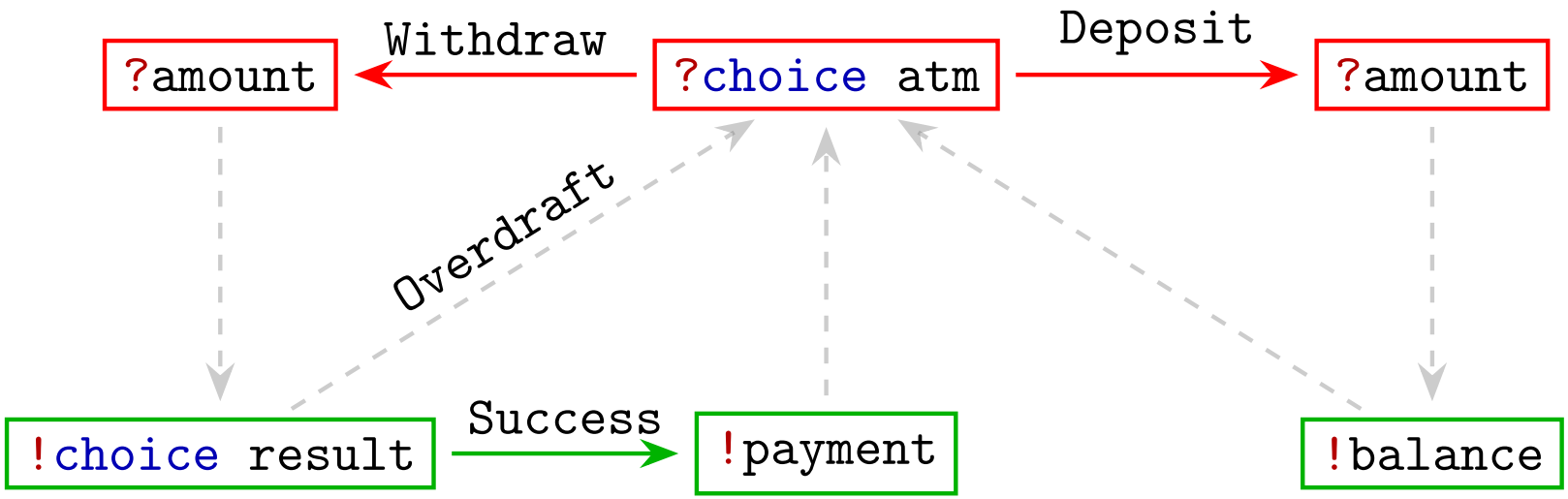
Type Width

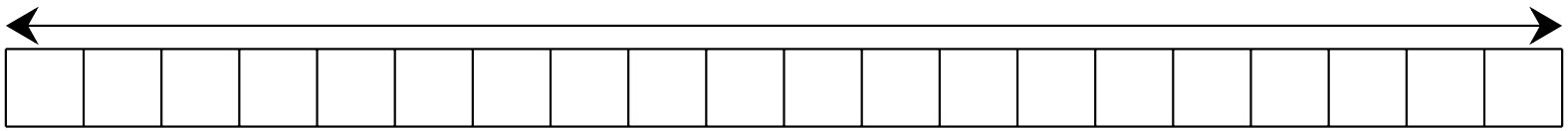
```
typedef <?choice request> atm;
choice request {
    <?amount; !balance; atm>  Deposit;
    <?amount; !choice result> Withdraw;
};
choice result {
    <!payment; atm> Success;
    <atm>           Overdraft;
};
```



Contributions

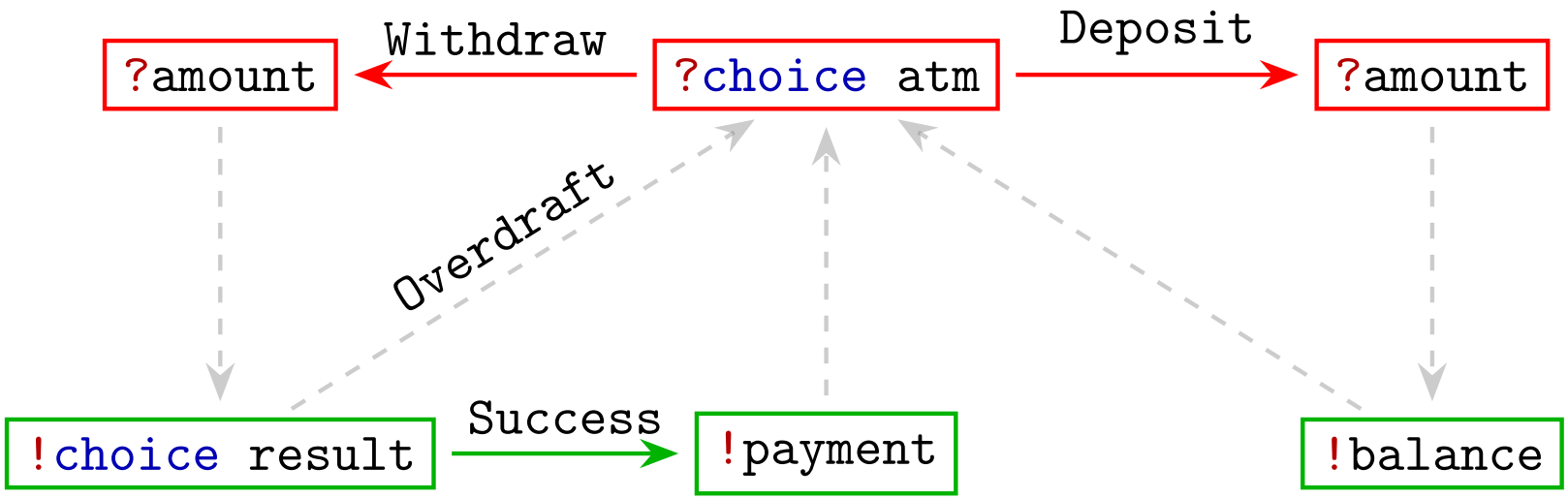
Type Width

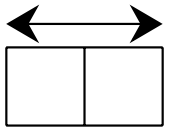




Contributions

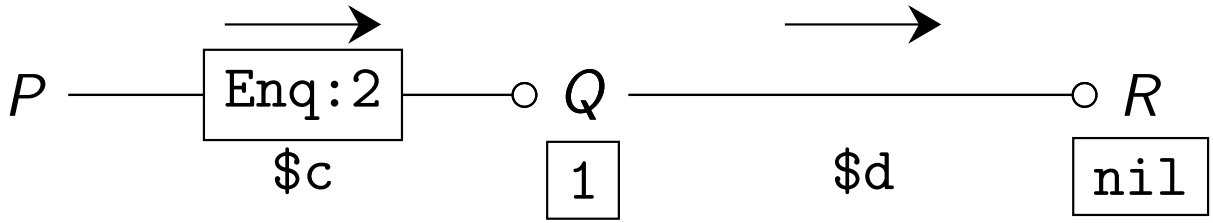
Type Width





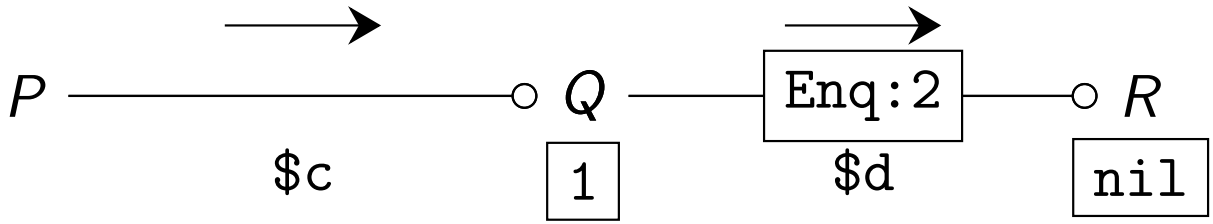
Contributions

Forwarding



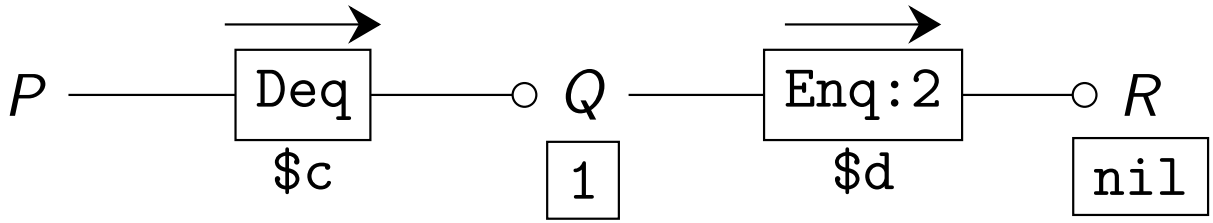
Contributions

Forwarding



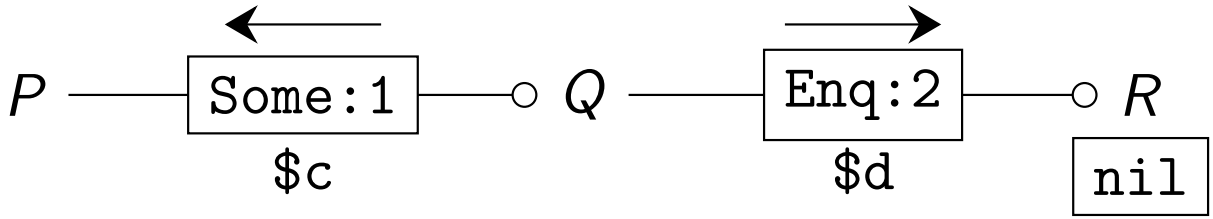
Contributions

Forwarding



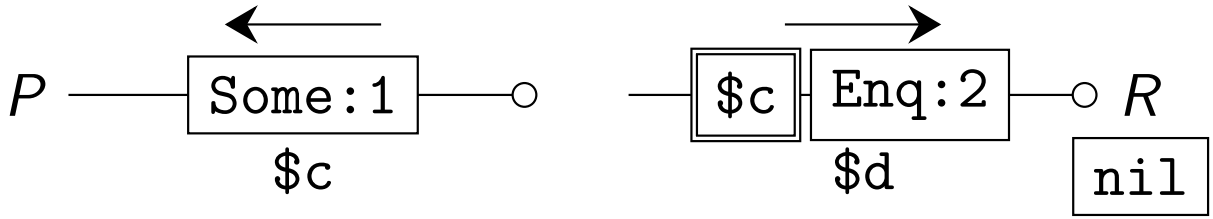
Contributions

Forwarding



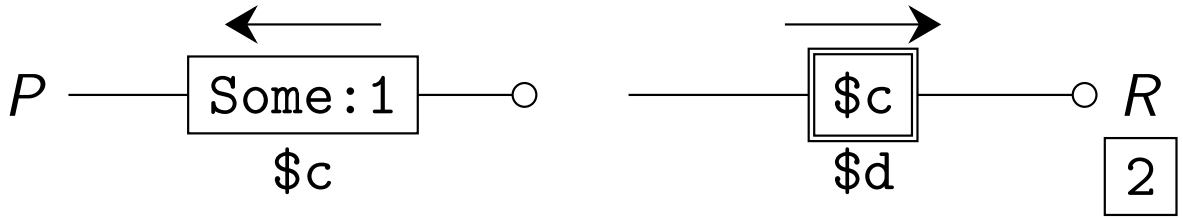
Contributions

Forwarding



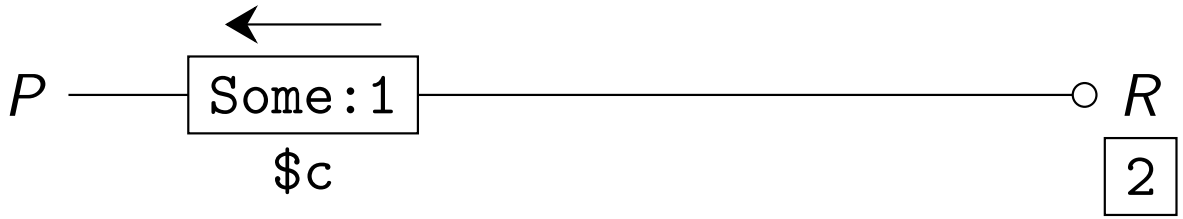
Contributions

Forwarding



Contributions

Forwarding



Takeaways

Session types give valuable insight into the structure of communication that's useful for safety and efficient implementation.

Takeaways

Session types give valuable insight into the structure of communication that's useful for safety and efficient implementation.

Takeaways

Session types give valuable insight into the structure of communication that's useful for safety and efficient implementation.

Questions?