# Typing network communication

Ivan Nikitin

University of Glasgow

# Types

```php
/**
 * Undocumented function
 * @return void
 */
public static function test2() : void
{
    $sum = Prize::test3(1, 1);
    echo $sum;
}

/**
 * Undocumented function
 * @param int $a
 * @param int $b
 * @return int
 */
private static function test3(int $a, float $b) : float
{
    return $a + $b;
}
```

Expected type 'float'. Found 'int'. intelephense(10006)

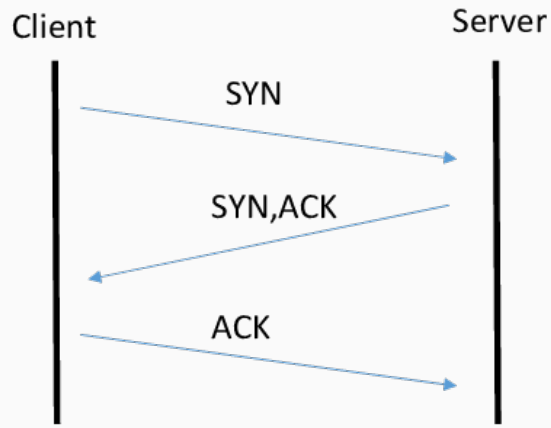
# Types

We can use static typing to:

- Protect from run-time errors.
- Document intention.
- Provide clear errors.

# Behavioural Types - Session Types

- Behaviour - the communication that occurs in terms of sequences of actions.
- TCP Handshake - Send a SYN, get a SYN-ACK, send an ACK.
- We can express this communication as a *type*.



# Behavioural Types - Session Types

$$\begin{aligned}\Gamma \;=\quad & \mathtt{s}[\mathtt{server}] : \mathtt{client} \oplus \mathit{syn}(\mathtt{SYN}).\mathtt{client}\&\mathit{synack}(\mathtt{SYN\text{-}ACK}) \\ & \qquad\qquad .\mathtt{client} \oplus \mathit{ack}(\mathtt{ACK}).\mathtt{end}, \\ & \mathtt{s}[\mathtt{client}] : \mathtt{server} \,\&\, \mathit{syn}(\mathtt{SYN}).\mathtt{server} \oplus \mathit{synack}(\mathtt{SYN\text{-}ACK}) \\ & \qquad\qquad .\mathtt{server}\&\mathit{ack}(\mathtt{ACK}).\mathtt{end}\end{aligned}$$

- $\oplus$ - Send (Select), & - Receive (Offer),
  . - continue as, `end` - end of communication.
- Each participant's local type is their view of the communication.
- E.g. from the server viewpoint - send a message of type `SYN` to the client, continue as receive a message of type `SYN-ACK` from the client, continue as send a message of type `ACK` to the client, continue as `end`.

# Behavioural Types - Session Types

An encoding of session types in a programming language gives us the ability to ensure that:

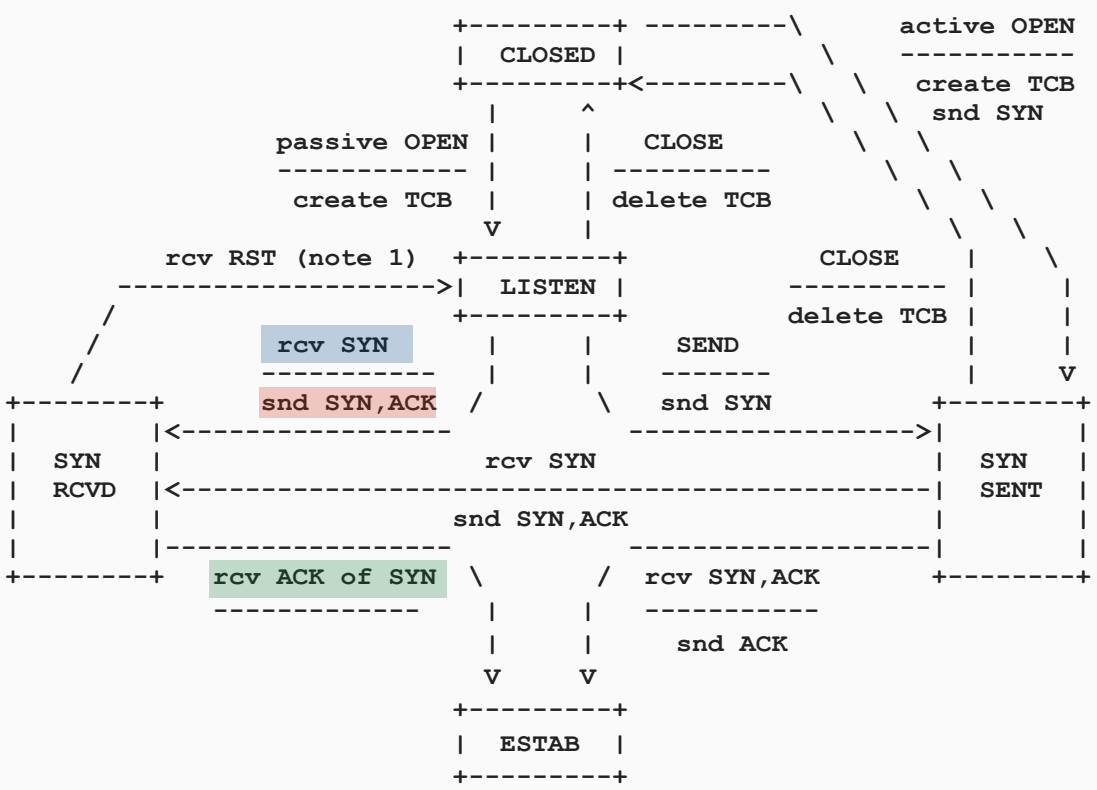
- Messages sent and received are in the expected order.
- Messages are of the expected type.
- Communication occurs between the expected participants.

# Behavioural Types - Session Types

An approach to encoding session types can be:

- Encode session type actions as a type. E.g.
  $\oplus - Select < Message, Continuation >$,
  $\& - Offer < Message, Continuation >$
- Channel parametrised over roles.
  $Channel < ParticipantOne, ParticipantTwo >$
- Channel has functions that use the session type.
  $select(message, session\_type) -> SessionType$
- Continuation passing style ensures adherence to the sequence of actions. Parametrisation over roles ensures the channel is used correctly.

# Behavioural Types - Session Types




$\Gamma$ = s[ss] : cs&syn(SynSet).
cs ⊕ syn_ack(SynAckSet).
cs&(AckSet).
cs ⊕ fin(Fin).end

```
// Instantiate the session type of the TCP server.
type ServerSessionType = OfferOne<RoleClient, Syn,
                         SelectOne<RoleClient, SynAck,
                         OfferOne<RoleClient, Ack,
                         SelectOne<RoleClient, FinAck,
                             End>>>>;
let st_server = ServerSystemSessionType::new();
// Create the session typed NetChannel
let mut chan = NetChannel::<RoleServer,
     RoleClient>::new(iter, tx, remote_addr);
// Recieve a SYN packet.
let (syn, cont) = chan.offer_one(st_server);
// Send the message along the channel, following our
     session type.
let cont = chan.select_one(cont, SynAck {...});
// Recieve a message of type ACK.
let (ack, cont) = chan.offer_one(cont);
// Send the FIN-ACK along the channel.
let cont = chan.select_one(cont, FinAck {...});
// Close
chan.close(cont);
```

# To Summarise

- We can encode the *behaviour* of a protocol as a *type*.
- Session types are a discipline used to express concurrent communication.
- An encoding of session types in a programming language leverages the compiler to make sure the protocol is implemented according to its session type.