Type-Driven Design of Communicating Systems using Idris

Jan de Muijnck-Hughes Edwin Brady

@jfdm
jfdm@st-andrews.ac.uk
http://jfdm.github.io

9 November 2016

University of St Andrews | FOUNDED 1413

It is always good to start with a joke...

Jan “Knock Knock”

It is always good to start with a joke. . .

Jan “Knock Knock”

Audience “Who’s there?”

It is always good to start with a joke. . .

Jan	“Knock Knock”
Audience	“Who’s there?”
Jan	“Amosquito! dummy!”

It is always good to start with a joke. . .

Jan	“Knock Knock”
Audience	“Who’s there?”
Jan	“Amosquito! dummy!”
Audience	“Amosquito! dummy!, who?”

It is always good to start with a joke. . .

Jan “Knock Knock”

Audience “Who’s there?”

Jan “Amosquito! dummy!”

Audience “Amosquito! dummy!, who?”

Mallory “Amos”

It is always good to start with a joke. . .

Jan	"Knock Knock"
Audience	"Who's there?"
Jan	"Amosquito! dummy!"
Audience	"Amosquito! dummy!, who?"
Mallory	"Amos"

Knock-Knock is a 'well known' joke.

- Doesn't follow the known specification.
- Messages are in the wrong order and format.
- Unknown participants $\Longrightarrow$ unknown channels.
- Messages might arrive late. . .

It is always good to start with a joke...

Jan	“Knock Knock”
Audience	“Who’s there?”
Jan	“Amosquito! dummy!”
Audience	“Amosquito! dummy!, who?”
Mallory	“Amos”
Edwin	“Not this stupid joke again!”

`Knock-Knock` is a ‘well known’ joke.

- Doesn’t follow the known specification.
- Messages are in the wrong order and format.
- Unknown participants $\implies$ unknown channels.
- Messages might arrive late...

Knock Knock: Specifications

Informal Narration.

1. $A \rightarrow B$: "Knock, Knock"
2. $B \rightarrow A$: "Who's there?"
3. $A \rightarrow B$: msg
4. $B \rightarrow A$: msg ++ " who?"
5. $A \rightarrow B$: msg ++ $resp$

Knock Knock: Specifications

Informal Narration.

1. $A \rightarrow B$: “Knock, Knock”
2. $B \rightarrow A$: “Who’s there?”
3. $A \rightarrow B$: msg
4. $B \rightarrow A$: msg ++ “ who?”
5. $A \rightarrow B$: msg ++ $resp$

Global Type (MPST)

1. $\mathsf{A} \rightarrow \mathsf{B} : k\langle\texttt{String}\rangle\,.$
2. $\mathsf{B} \rightarrow \mathsf{A} : k\langle\texttt{String}\rangle\,.$
3. $\mathsf{A} \rightarrow \mathsf{B} : k\langle\texttt{String}\rangle\,.$
4. $\mathsf{B} \rightarrow \mathsf{A} : k\langle\texttt{String}\rangle\,.$
5. $\mathsf{A} \rightarrow \mathsf{B} : k\langle\texttt{String}\rangle\,.\,\mathsf{end}$

Knock Knock: Specifications

Informal Narration.

1. $A \rightarrow B$: “Knock, Knock”
2. $B \rightarrow A$: “Who’s there?”
3. $A \rightarrow B$: *msg*
4. $B \rightarrow A$: *msg* ++ “ who?”
5. $A \rightarrow B$: *msg* ++ *resp*

Global Type (MPST)

1. $\mathsf{A} \rightarrow \mathsf{B} : k\langle\texttt{String}\rangle\,.$
2. $\mathsf{B} \rightarrow \mathsf{A} : k\langle\texttt{String}\rangle\,.$
3. $\mathsf{A} \rightarrow \mathsf{B} : k\langle\texttt{String}\rangle\,.$
4. $\mathsf{B} \rightarrow \mathsf{A} : k\langle\texttt{String}\rangle\,.$
5. $\mathsf{A} \rightarrow \mathsf{B} : k\langle\texttt{String}\rangle\,.\,\mathsf{end}$

Session Types are great but not *perfect*

- Hard to reason on messages.
- Hard to reason on channel management.

Authentication Protocol: Simplified Kerberos

Establish a secure connection using a *Trusted Third Party*.

Authentication Protocol: Simplified Kerberos

Establish a secure connection using a *Trusted Third Party*.

1. Sign into Service (AS)
    - Establish: $\mathrm{K}_{A,AS}$
    - $\mathit{Alice} \rightarrow \mathit{AS} : \mathrm{ID}(A)$
    - AS generates
        - ticket with TTL: $\mathcal{T}_{ttl} \leftarrow \{\mathrm{ID}(A) \mid\mid \mathrm{K}_{A,TGS}\}_{\mathrm{K}_{AS,TGS}}$
        - Session Key $\mathrm{K}_{A,TGS}$
    - $\mathit{AS} \rightarrow \mathit{Alice} : \{\mathrm{K}_{A,TGS} \mid\mid \mathcal{T}_{ttl}\}_{\mathrm{K}_{A,AS}}$

Authentication Protocol: Simplified Kerberos

Establish a secure connection using a *Trusted Third Party*.

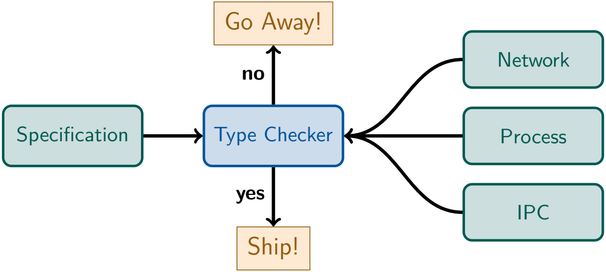
1. Sign into Service (AS)
   - Establish: $\mathsf{K}_{A,AS}$
   - $Alice \rightarrow AS : \mathrm{ID}(A)$
   - AS generates
     - ticket with TTL: $\mathcal{T}_{ttl} \leftarrow \{\mathrm{ID}(A) \mid\mid \mathsf{K}_{A,TGS}\}_{\mathsf{K}_{AS,TGS}}$
     - Session Key $\mathsf{K}_{A,TGS}$
   - $AS \rightarrow Alice : \{\mathsf{K}_{A,TGS} \mid\mid \mathcal{T}_{ttl}\}_{\mathsf{K}_{A,AS}}$
2. Request Ticket from TGS to Talk to Bob
   - Establish: $\mathsf{K}_{A,TGS}$ & Alice generates: Timestamp t.
   - $A \rightarrow TGS : \mathcal{T}_{ttl} \mid\mid \mathrm{ID}(B) \mid\mid \{t\}_{\mathsf{K}_{A,TGS}}$
   - TGS generates Session Key $\mathsf{K}_{A,B}$ and obtains $\mathsf{K}_{B,TGS}$.
   - $TGS \rightarrow A : \{\mathrm{ID}(B) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{A,TGS}} \mid\mid \{\mathrm{ID}(A) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{B,TGS}}$

Authentication Protocol: Simplified Kerberos

Establish a secure connection using a *Trusted Third Party*.

1. Sign into Service (AS)
   - Establish: $\mathsf{K}_{A,AS}$
   - $Alice \rightarrow AS : \mathsf{ID}(A)$
   - AS generates
     - ticket with TTL: $\mathcal{T}_{ttl} \leftarrow \{\mathsf{ID}(A) \mid\mid \mathsf{K}_{A,TGS}\}_{\mathsf{K}_{AS,TGS}}$
     - Session Key $\mathsf{K}_{A,TGS}$
   - $AS \rightarrow Alice : \{\mathsf{K}_{A,TGS} \mid\mid \mathcal{T}_{ttl}\}_{\mathsf{K}_{A,AS}}$
2. Request Ticket from TGS to Talk to Bob
   - Establish: $\mathsf{K}_{A,TGS}$ & Alice generates: Timestamp t.
   - $A \rightarrow TGS : \mathcal{T}_{ttl} \mid\mid \mathsf{ID}(B) \mid\mid \{t\}_{\mathsf{K}_{A,TGS}}$
   - TGS generates Session Key $\mathsf{K}_{A,B}$ and obtains $\mathsf{K}_{B,TGS}$.
   - $TGS \rightarrow A : \{\mathsf{ID}(B) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{A,TGS}} \mid\mid \{\mathsf{ID}(A) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{B,TGS}}$
3. Ask Bob To Talk
   - $A \rightarrow B : \{\mathsf{ID}(A) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{B,TGS}} \mid\mid \{t\}_{\mathsf{K}_{A,B}}$
   - $B \rightarrow A : \{t+1\}_{\mathsf{K}_{A,B}}$

Type-Driven Verification of Communicating Systems



System to describe, reason, and build Communicating Systems:

- Inspired by Session Types
- Leverage Dependent Types, Algebraic Effects & States

Sessions Modelling Language

- Describing Sessions i.e. Global Types
  - Automatic trace generation.
- Using Idris control structures.
  - Do Notation—Linearity
  - Case Splits—Branches
  - Recursion—Recursion
- Fine-grained Channel Management
  - Creation, Use, Destruction
- Actor Management
  - When and What Actors can do.
- Reason on Description
  - 'Resource'-Dependent State Changes
  - Predicates & Idris' Proof Search

```
data Session : (ty  : Type)
            -> (old : Context)
            -> (new : ty -> Context)
            -> Type
  where
    Activate...     Call...
    Deactivate...   Rec...
    NewChannel...   Done...
    RmChannel...    (>>=)...
    Startup...      Pure...
    Teardown...
    Send...
```

TCP ‘Handshake’: Naïve

1. $A \to B : (\mathsf{Syn}, x)$
2. $B \to A : (\mathsf{SynAck}, y, x+1)$
3. $A \to B : (\mathsf{Ack}, y+1, x+1)$

TCP 'Handshake': Naïve

1. $A \to B : (\mathsf{Syn}, x)$
2. $B \to A : (\mathsf{SynAck}, y, x + 1)$
3. $A \to B : (\mathsf{Ack}, y + 1, x + 1)$

1. $\mathsf{A} \to \mathsf{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat} \rangle\,.$
2. $\mathsf{B} \to \mathsf{A} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle\,.$
3. $\mathsf{A} \to \mathsf{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle\,.\,\mathsf{end}$

TCP ‘Handshake’: Naïve

1. $A \to B : (\text{Syn}, x)$
2. $B \to A : (\text{SynAck}, y, x + 1)$
3. $A \to B : (\text{Ack}, y + 1, x + 1)$

1. $\mathsf{A} \to \mathsf{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat} \rangle\,.$
2. $\mathsf{B} \to \mathsf{A} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle\,.$
3. $\mathsf{A} \to \mathsf{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle\,.\mathsf{end}$

```
Handshake : Session [A,B] [(A,B)] ()
Handshake = do
  activateAll
  chan <- channel A B
  startup chan
  send chan A B (TCPMsg, Nat)
  send chan B A (TCPMsg, Nat, Nat)
  send chan A B (TCPMsg, Nat, Nat)
  shutdown chan A
  deactivateAll
  end
```

TCP ‘Handshake’: Improved

1. $A \to B : (\mathsf{Syn}, x)$
2. $B \to A : (\mathsf{SynAck}, y, x + 1)$
3. $A \to B : (\mathsf{Ack}, y + 1, x + 1)$

1. $\mathsf{A} \to \mathsf{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat} \rangle\,.$
2. $\mathsf{B} \to \mathsf{A} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle\,.$
3. $\mathsf{A} \to \mathsf{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle\,.\,\mathsf{end}$

```
Handshake : Session [A,B] [(A,B)] ()
Handshake = do
  activateAll
  chan <- channel A B
  startup chan
  (_,x)   <- send chan A B (TCPMsg, Nat)
  (_,y,_) <- send chan B A (TCPMsg, Nat, (x' ** x' = S x))
  send chan A B (TCPMsg, (y' ** y' = S y), (x' ** x' = S x))
  shutdown chan A
  deactivateAll
  end
```

TCP 'Handshake': Better

1. $A \to B : (\mathrm{Syn}, x)$
2. $B \to A : (\mathrm{SynAck}, y, x + 1)$
3. $A \to B : (\mathrm{Ack}, y + 1, x + 1)$

1. $\mathrm{A} \to \mathrm{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}\rangle\,.$
2. $\mathrm{B} \to \mathrm{A} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat}\rangle\,.$
3. $\mathrm{A} \to \mathrm{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat}\rangle\,.\,\mathrm{end}$

```
Handshake : Session [A,B] [(A,B)] ()
Handshake = do
  activateAll
  chan <- channel A B
  startup chan
  (_,x)   <- send chan A B (TCPMsg, Nat)
  (_,y,_) <- send chan B A (TCPMsg, Nat, Next x)
  send chan A B (TCPMsg, Next y, Next x)
  shutdown chan A
  deactivateAll
  end
```

TCP ‘Handshake’: Best

1. $A \to B : (\text{Syn}, x)$
2. $B \to A : (\text{SynAck}, y, x+1)$
3. $A \to B : (\text{Ack}, y+1, x+1)$

1. $\mathsf{A} \to \mathsf{B} : k\langle \texttt{TCPMsg}, \texttt{Nat}\rangle\,.$
2. $\mathsf{B} \to \mathsf{A} : k\langle \texttt{TCPMsg}, \texttt{Nat}, \texttt{Nat}\rangle\,.$
3. $\mathsf{A} \to \mathsf{B} : k\langle \texttt{TCPMsg}, \texttt{Nat}, \texttt{Nat}\rangle\,.\,\mathsf{end}$

```
Handshake : Session [A,B] [(A,B)] ()
Handshake = do
  activateAll
  chan <- channel A B
  startup chan
  (_,x)   <- send chan A B (TCPMsg SYN, Nat)
  (_,y,_) <- send chan B A (TCPMsg SYNACK, Nat, Next x)
  send chan A B (TCPMsg ACK, Next y, Next x)
  shutdown chan A
  deactivateAll
  end
```

Implementing Sessions: Sample Language Expressions

```
Activate : (a   : Actor)
        -> (idx : InContextP ACTOR (ActorHasState a DEAD) item ctxt)
        -> Session ()
                   ctxt
                   (\res => updateStateP ACTIVE ctxt idx)
```

```
Send : (c     : VarChannel chan)
    -> (s     : Actor)
    -> (r     : Actor)
    -> (mTy   : Type)
    -> (ok_s  : InContextP ACTOR (ActorHasState s ACTIVE) iS ctxt)
    -> (ok_r  : InContextP ACTOR (ActorHasState r ACTIVE) iR ctxt)
    -> (ok_c  : InContextP CHANNEL
                       (ChannelHasState chan c CONNECTED) iC ctxt)
    -> (vsend : ValidSend s r c mTy rTy iC)
    -> Session rTy ctxt (\res => ctxt)
```

Implementing Sessions: Proofs and Predicates

Predicated *De Bruijn* Index

```
data InContextP : (ty : Ty)       -> (p : Item ty -> Type)
               -> (x  : Item ty) -> (c : Context) -> Type
  where
    HereP  : p x -> InContextP ty p x (x :: rest)
    ThereP : InContextP ty p x rest
          -> InContextP ty p x (notitem :: rest)
```

Example Predicate

```
data ActorHasState : (actor : Actor )
                  -> (value : AState)
                  -> (item  : Item ACTOR)
                  -> Type
  where
    AState : ActorHasState a
                           value
                           (MkItem label (ReprActor a) value)
```

RFC 347 & 862

RFC 347 & 862

1. $A \to B : x$
2. $B \to A : x$

$$
\begin{aligned}
\mu\mathbf{t}\,.\ & \mathsf{A} \to \mathsf{B} \ :\ k\{ \\
& \quad \textbf{echo} \Rightarrow \mathsf{A} \to \mathsf{B} \ :\ k\langle \texttt{String} \rangle \\
& \qquad\qquad .\,\mathsf{B} \to \mathsf{A} \ :\ k\langle \texttt{String} \rangle \\
& \qquad\qquad .\,\mathbf{t} \\
& \quad \textbf{quit} \Rightarrow \mathsf{end}\}
\end{aligned}
$$

RFC 347 & 862

1. $A \rightarrow B : x$
2. $B \rightarrow A : x$

$$\begin{aligned}
&\mu\mathbf{t}\,.\,\mathrm{A} \rightarrow \mathrm{B} \;:\; k\{\\
&\quad \mathbf{echo} \Rightarrow \mathrm{A} \rightarrow \mathrm{B} \;:\; k\langle\texttt{String}\rangle\\
&\quad\qquad\qquad .\,\mathrm{B} \rightarrow \mathrm{A} \;:\; k\langle\texttt{String}\rangle\\
&\quad\qquad\qquad .\,\mathbf{t}\\
&\quad \mathbf{quit} \Rightarrow \mathrm{end}\}
\end{aligned}$$

```
Echo : Session ()
               [Client, Server]
               [(Client,Server)]
Echo = do
    activateAll

    net <- channel Client Server
    startup net
    call $ doEcho net
    shutdown net Server

    deactivateAll
    end
```

RFC 347 & 862: Looping

```
doEcho : (chan : CHAN Client Server)
      -> SubSession () (CommonContextCS chan)
doEcho net = do
  case !(send net Client Server (Maybe String)) of
    Just m => do
      send net Server Client $ Literal String m
      rec $ doEcho net
    Nothing => done
```

RFC 347 & 862: Looping

```
doEcho : (chan : CHAN Client Server)
      -> SubSession () (CommonContextCS chan)
doEcho net = do
  case !(send net Client Server (Maybe String)) of
    Just m => do
      send net Server Client $ Literal String m
      rec $ doEcho net
    Nothing => done
```

```
Rec : Inf (Session a ctxt ctxt') -> Session a ctxt ctxt'

Call : (sub : Session a ctxt' (const ctxt'))
    -> (prf : SubContext ctxt' ctxt)
    -> Session a ctxt ctxt
```

Simplified Kerberos—Sans Crypto

```
Kerberos’ : Session () [A,B,T,K] [(A,B), (A,T), (A,K)]
Kerberos’ = do
  activateSet [A,K]

  kak <- channel A K  -- Contact Authentication Service
  startup kak
  aliceID <- send kak A K String
  (_, ticket) <- send kak K A (Literal String aliceID, String)
  shutdown kak A

  activate T

  kat <- channel A T  -- Request Ticket
  startup kat
  (_, bobID, t) <- send kat A T (Literal String ticket, String, Nat)
  (_, y) <- send kat T A ( (Literal String bobID, String)
                         , (Literal String aliceID, String))
  shutdown kat A
```

Simplified Kerberos—Sans Crypto—cont. . .

```
activate B   -- Talk to Bob
kab <- channel A B
startup kab
send kab A B ( Literal (Literal String aliceID, String) y
             , Literal Nat t)
send kab B A (Next t)
shutdown kab A

deactivateAll
end
```

Authentication Protocol: Simplified Kerberos

Establish a secure connection using a *Trusted Third Party*.

1. Sign into Service (AS)
   - Establish: $\mathsf{K}_{A,AS}$
   - $\mathit{Alice} \rightarrow \mathit{AS} : \mathrm{ID}(A)$
   - AS generates
     - ticket with TTL: $\mathcal{T}_{ttl} \leftarrow \{\mathrm{ID}(A) \mid\mid \mathsf{K}_{A,TGS}\}_{\mathsf{K}_{AS,TGS}}$
     - Session Key $\mathsf{K}_{A,TGS}$
   - $\mathit{AS} \rightarrow \mathit{Alice} : \{\mathsf{K}_{A,TGS} \mid\mid \mathcal{T}_{ttl}\}_{\mathsf{K}_{A,AS}}$
2. Request Ticket from TGS to Talk to Bob
   - Establish: $\mathsf{K}_{A,TGS}$ & Alice generates: Timestamp t.
   - $A \rightarrow \mathit{TGS} : \mathcal{T}_{ttl} \mid\mid \mathrm{ID}(B) \mid\mid \{t\}_{\mathsf{K}_{A,TGS}}$
   - TGS generates Session Key $\mathsf{K}_{A,B}$ and obtains $\mathsf{K}_{B,TGS}$.
   - $\mathit{TGS} \rightarrow A : \{\mathrm{ID}(B) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{A,TGS}} \mid\mid \{\mathrm{ID}(A) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{B,TGS}}$
3. Ask Bob To Talk
   - $A \rightarrow B : \{\mathrm{ID}(A) \mid\mid \mathsf{K}_{A,B}\}_{\mathsf{K}_{B,TGS}} \mid\mid \{t\}_{\mathsf{K}_{A,B}}$
   - $B \rightarrow A : \{t + 1\}_{\mathsf{K}_{A,B}}$

Codified Examples

'Real' Protocols

- `RFC 347 Echo`
- `RFC 862 Echo`
- `RFC 864 CharGen`
- `RFC 867 DayTime`
- `RFC 868 Time`

Not So Real Protocols

- Hello World.
- Greeter Program.
- String Length
- Natural Number Calculator
- TCP Handshake

So Sessions. . .

What can we do.

- Model interactions between components.
- Model multiple channels.
- Reason about session's emergent properties.
- Generate Local Traces.

What we don't do.

- Model beyond the specification.
- Guarantees towards protocol correctness.
- Loose specifications can lead to loose implementations.

Further Work

Short project, with much long term potential. . .

- Communication Contexts
  - Exploring how to link specifications using algebraic effects.
  - Constructing `Network`, `IPC`, & `Process` implementations.
  - Context Agnostic Contexts?
- More 'Real' & Complex Examples
  - Different Protocols, Workflows, & Processes
  - Multi-party Communications
  - `TCP`, `TLS`, `SPEKE`, `TFTP`, `PGP`. . . .
- Look beyond the interaction.
  - Formal verification of the Specification.
  - Applied-Π, CSP. . .

Summary

Dependent Types helps Session Types

Session Types, I think this is the beginning of a beautiful friendship.

- Implement *most* of Session Types.
- Reason on Messages & Channel Management
- Better means to reason on crypto messages.

Lots of interesting Future work

To Implementations, and Beyond!

- Want to link specifications with implementation using algebraic effects.
- Investigate how to prove non-functional properties *a la* ProVerif.