ABCD Meeting, London, 17/12/2018

Asynchronous Timed Session Types & Processes

Laura Bocchi University of Kent

Maurizio Murgia University of Kent
Vasco Thudichum Vasconcelos University of Lisbon
Nobuko Yoshida Imperial College London

Agenda

- Session types ♥ Time
  - Synchronous [Bartoletti,Cimoli&Murgia@FORTE'13]
  - Multiparty asynchronous [Bocchi,Yang,Yoshida@CONCUR'14]
    - restriction to types/protocols that could be used for type-checking
    - limitations to the expressiveness of the calculus
- Today
  - Designing timed protocols : asynchronous timed **duality**
  - Checking timed programs : a time-sensitive **calculus** & **typing** system

Time & trouble

- A timed protocol is not correct by definition

$$!\mathtt{Int}.?\mathtt{String}$$

$$!\mathtt{Int}(x \leq 3).?\mathtt{String}(x \leq 2)$$

- Usually this is handled by adding some conditions

feasibility [Bocchi, Yang, Yoshida@CONCUR'14],
interaction-enabledness (CTA) [Bocchi, Lange, Yoshida@CONCUR'15],
compliance [Bartoletti, Cimoli, Murgia@FORTE'16]
formation + duality [Bocchi, Murgia, Yoshida, Vasconcelos'18]

$$!\mathtt{Int}(x \leq 3).?\mathtt{String}(x \leq 3)$$

Duality & progress

- Duality characterises well-behaved systems

$S = !\mathtt{Int}(x \leq 1, x)\,.\,?\mathtt{String}(x \leq 2)$ $\quad\quad$ $\overline{S} = ?\mathtt{Int}(y \leq 1, y)\,.\,!\mathtt{String}(y \leq 2)$

Synchronous

$$S \mid \overline{S} \xrightarrow{0.4} S \mid \overline{S} \xrightarrow{\mathtt{Int}} ?\mathtt{String}(x \leq 2) \mid !\mathtt{String}(y \leq 2) \xrightarrow{2} \xrightarrow{\mathtt{String}}$$

Duality & progress

- Duality characterises well-behaved systems

$$S = !\texttt{Int}(x \leq 1, x).?\texttt{String}(x \leq 2) \qquad \overline{S} = ?\texttt{Int}(y \leq 1, y).!\texttt{String}(y \leq 2)$$

Synchronous

$$S \mid \overline{S} \xrightarrow{0.4} S \mid \overline{S} \xrightarrow{\texttt{Int}} ?\texttt{String}(x \leq 2) \mid !\texttt{String}(y \leq 2) \xrightarrow{2} \xrightarrow{\texttt{String}}$$

Asynchronous

$$S \mid \overline{S} \xrightarrow{0.4} S \mid \overline{S} \xrightarrow{!\texttt{Int}} ?\texttt{String}(x \leq 2) \mid ?\texttt{Int}(y \leq 1, y).!\texttt{String}(y \leq 2)$$

$$\xrightarrow{0.6} \xrightarrow{?\texttt{Int}} ?\texttt{String}(x \leq 2) \mid !\texttt{String}(y \leq 2) \xrightarrow{2} \xrightarrow{!\texttt{String}} ?\texttt{String}(x \leq 2)$$

x 2.6 y 2

Receive & asynchrony (1/2)

$S = \mu \mathtt{t}.!\mathtt{Int}(x \leq 1, x).?\mathtt{String}(x \leq 2).\mathtt{t}$ 1

```
func S (a chan<- int, b <-chan string, start time.Time){
 for {
  time.Sleep(400 * time.Millisecond)
  t := time.Now()
  a<-10
  fmt.Printf("sent int 10 at time %s\n", t.Sub(start))
  select{
      case c:=<-b :
         t := time.Now()
         fmt.Printf("received string %s at time %s\n", c, t.Sub(start))
      case <-time.After(2 * time.Second):
        fmt.Println("S Failed! String not received within deadline")
  }
 }
}
```

Receive & asynchrony (2/2)

$$\overline{S} = \mu \mathtt{t}.?\mathtt{Int}(y \leq 1, y).!\mathtt{String}(y \leq 2).\mathtt{t}$$

```
func Sd (a <-chan int, b chan<- string, start time.Time){
    for{
      select{
        case c:=<-a :     t := time.Now()
                fmt.Printf("received int %d at time %s\n", c, t.Sub(start))
        case <-time.After(1 * time.Second): fmt.Println("Sd Failed! …")
       }
      time.Sleep(600 * time.Millisecond)
      t := time.Now()
      b<-"hello!"
      fmt.Printf("sent 'hello!' at time %s\n", t.Sub(start))
     }
}
```

https://play.golang.org/p/pagByVB4Gtd

Urgent receive semantics

- Urgent receive semantics: *messages are* ***received*** *as soon as*
  - *they are in a channel, and*
  - *the time constraint of the receiver is satisfied*
- Urgent receive semantics yields executions that are
  - are <u>a bit more synchronous</u> …
  - … but <u>as asynchronous as</u> when using (common) receive primitives

$$S \mid \bar{S} \xrightarrow{0.4} S \mid \bar{S} \xrightarrow{\texttt{!Int}} \texttt{?String}(x \leq 2) \mid \texttt{?Int}(y \leq 1, y).\texttt{!String}(y \leq 2)$$

$$\xrightarrow{0.6} \xrightarrow{\texttt{?Int}} \texttt{?String}(x \leq 2) \mid \texttt{!String}(y \leq 2) \xrightarrow{2} \xrightarrow{\texttt{!String}} \texttt{?String}(x \leq 2)$$

x 0 y 0

Urgent receive semantics

- Urgent receive semantics: *messages are* ***received*** *as soon as*
  - *they are in a channel, and*
  - *the time constraint of the receiver is satisfied*
- Urgent receive semantics yields executions that are
  - are <u>a bit more synchronous</u> …
  - … but <u>as asynchronous as</u> when using (common) receive primitives

Type Progress: $(\nu_1, S_1, M_1) \mid (\nu_2, S_2, M_2)$ <u>satisfies progress</u> *if any reachable state is either* ***success*** (`end` *types and empty queues*) *or* ***allows an action****, possibly after some delay.*

Theorem (Duality Progress). $(\nu_0, S, \varnothing) \mid (\nu_0, \overline{S}, \varnothing)$ *enjoys progress* (*when using urgent receive semantics*).

Subtyping

- Asymmetric as e.g., [Gay&Hole'05][Demangeon&Honda'11] [Chen,Dezani-Ciancaglini&Yoshida'14]

Definition (Timed Simulation). *Fix* $\mathbf{s}_1 = (\nu_1, S_1)$ *and* $\mathbf{s}_2 = (\nu_2, S_2)$. *A relation* $\mathcal{R} \in (\mathbb{V} \times \mathcal{S})^2$ *is a timed simulation if* $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{R}$ *implies:*

1. $S_1 = \mathtt{end}$ *implies* $S_2 = \mathtt{end}$
2. $\mathbf{s}_1 \xrightarrow{t!m_1} \mathbf{s}'_1$ *implies* $\exists \mathbf{s}'_2, m_2 : \mathbf{s}_2 \xrightarrow{t!m_2} \mathbf{s}'_2$, $(m_2, m_1) \in \mathcal{S}$, *and* $(\mathbf{s}'_1, \mathbf{s}'_2) \in \mathcal{R}$
3. $\mathbf{s}_2 \xrightarrow{t?m_2} \mathbf{s}'_2$ *implies* $\exists \mathbf{s}'_1, m_1 : \mathbf{s}_1 \xrightarrow{t?m_1} \mathbf{s}'_1$, $(m_1, m_2) \in \mathcal{S}$, *and* $(\mathbf{s}'_1, \mathbf{s}'_2) \in \mathcal{R}$
4. $\mathbf{s}_1 \overset{?}{\Rightarrow}$ *implies* $\mathbf{s}_2 \overset{?}{\Rightarrow}$ *and* $\mathbf{s}_2 \overset{!}{\Rightarrow}$ *implies* $\mathbf{s}_1 \overset{!}{\Rightarrow}$

$$!\mathtt{String}(x = 0) \quad <: \quad !\mathtt{String}(x \leq 2)$$
$$?\mathtt{String}(x \leq 2) \quad <: \quad ?\mathtt{String}(x = 0)$$
$$?\mathtt{String}(x \leq 2).!\mathtt{String}(x = 1) \quad \not<: \quad ?\mathtt{String}(x = 0).!\mathtt{String}(x = 1)$$
$$!\mathtt{String}(\mathtt{true}) \quad \not<: \quad ?\mathtt{String}(\mathtt{true})$$

Subtyping

- Asymmetric as e.g., [Gay&Hole'05][Demangeon&Honda'11] [Chen,Dezani-Ciancaglini&Yoshida'14]

Definition (Timed Simulation). *Fix* $\mathbf{s}_1 = (\nu_1, S_1)$ *and* $\mathbf{s}_2 = (\nu_2, S_2)$. *A relation* $\mathcal{R} \in (\mathbb{V} \times \mathcal{S})^2$ *is a timed simulation if* $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{R}$ *implies:*

1. $S_1 = \texttt{end}$ *implies* $S_2 = \texttt{end}$
2. $\mathbf{s}_1 \xrightarrow{t!m_1} \mathbf{s}_1'$ *implies* $\exists \mathbf{s}_2', m_2 : \mathbf{s}_2 \xrightarrow{t!m_2} \mathbf{s}_2'$, $(m_2, m_1) \in \mathcal{S}$, *and* $(\mathbf{s}_1', \mathbf{s}_2') \in \mathcal{R}$
3. $\mathbf{s}_2 \xrightarrow{t?m_2} \mathbf{s}_2'$ *implies* $\exists \mathbf{s}_1', m_1 : \mathbf{s}_1 \xrightarrow{t?m_1} \mathbf{s}_1'$, $(m_1, m_2) \in \mathcal{S}$, *and* $(\mathbf{s}_1', \mathbf{s}_2') \in \mathcal{R}$
4. $\mathbf{s}_1 \overset{?}{\Rightarrow}$ *implies* $\mathbf{s}_2 \overset{?}{\Rightarrow}$ *and* $\mathbf{s}_2 \overset{!}{\Rightarrow}$ *implies* $\mathbf{s}_1 \overset{!}{\Rightarrow}$

Theorem (Safe/Progressing Substitution). *Let* $S' <: \overline{S}$ *then*

1) $(\nu_0, S, \emptyset) \mid (\nu_0, S', \emptyset) \lesssim (\nu_0, S, \emptyset) \mid (\nu_0, \overline{S}, \emptyset)$

2) $(\nu_0, S, \emptyset) \mid (\nu_0, S', \emptyset)$ *enjoys progress.*

In [Bartoletti,Bocchi,Murgia@CONCUR'18] asymmetric refinement does not preserve behaviour/progress (it was "local" and did not assume duality)

Implementing dual types

"An SMTP server SHOULD have a timeout of at least 5 minutes while it is awaiting the next command from the sender" [RFC 5321]

$S = ?\texttt{Com}(x < 5, x).S'$ $\qquad$ $C = !\texttt{Com}(y < 5, y).C'$

- This protocol can be implemented *e.g., in* ***Go***, ***Erlang*** *(timeout pattern)*, ***Real-Time Java***, …

```
select{
      case <-b : \\ proceed as S'
      case <-time.After(5 * time.Second): \\ explode
 }
```

- This protocol cannot be correctly implemented with the calculus in [Bocchi,Yang&Yoshida'14]

Implementing dual types

$$S = ?\texttt{Com}(x < 5, x).S' \qquad C = !\texttt{Com}(y < 5, y).C'$$

i want to implement **these** types

$$\texttt{delay}(4.90).a(b).P'_s \mid \texttt{delay}(4.99).\overline{a}(\texttt{HELO}).P'_c$$

$$\longrightarrow \quad a(b).P'_s \mid \texttt{delay}(0.09).\overline{a}(\texttt{HELO}).P'_c$$

Wait-freedom [Bocchi,Yang,Yoshida'14]: the solutions of the constraint of a receive action must be all after any solution of the corresponding send action

$$S = ?\texttt{Com}(x = 5, x).S' \qquad C = !\texttt{Com}(y < 5, y).C'$$

Programs

time-consuming

$$P ::= \overline{a}v\,.\,P$$
$$\mid\ a \triangleleft \mathtt{l}\,.\,P$$
$$\mid\ \mathtt{if}\ v\ \mathtt{then}\ P\ \mathtt{else}\ P$$
$$\mid\ P \mid P$$
$$\mid\ 0$$
$$\mid\ \mathtt{def}\ D\ \mathtt{in}\ P$$
$$\mid\ X\langle \vec{a}; \vec{a} \rangle$$
$$\mid\ (\nu ab)\ P$$
$$\mid\ ab : h$$

$$\mid\ \mathtt{delay}(\delta)\,.\,P$$
$$\mid\ a^n(b)\,.\,P$$
$$\mid\ a^n \triangleright \{\mathtt{l}_i : P_i\}_{i\in I}$$

$a^n(b)\,.\,P$

- $n = 0$ non-blocking
- $n = \infty$ blocking
- $n \in \mathbb{R}_{>0}$ blocking with timeout

$$C = !\mathtt{Com}(y < 5, y).C'$$
$$\mathtt{delay}(x = 4.90).a^0(b).P'_s$$
$$a^5(b).P'_s$$

Programs

$$P ::= \overline{a}v\,.\,P$$
$$\mid\ a \triangleleft \mathtt{l}\,.\,P$$
$$\mid\ \mathtt{if}\ v\ \mathtt{then}\ P\ \mathtt{else}\ P$$
$$\mid\ P \mid P$$
$$\mid\ 0$$
$$\mid\ \mathtt{def}\ D\ \mathtt{in}\ P$$
$$\mid\ X\langle \overrightarrow{a}; \overrightarrow{a} \rangle$$
$$\mid\ (\nu ab)\ P$$
$$\mid\ ab : h$$

time-consuming

$$\mid\ \mathtt{delay}(\delta)\,.\,P$$
$$\mid\ a^n(b)\,.\,P$$
$$\mid\ a^n \triangleright \{\mathtt{l}_i : P_i\}_{i \in I}$$

$$S = ?\mathtt{Com}(x < 5, x).S'$$
$$\mathtt{delay}(x = 4.99).\overline{a}(\mathtt{HELO}).P'_c$$
$$\mathtt{delay}(4.8 \leq x < 5).\overline{a}(\mathtt{HELO}).P'_c$$

There are also typing rules…

$$\frac{\Gamma \vdash b : T \quad \nu \models \delta \quad \Gamma \vdash P \triangleright \Delta, a : (\nu[\lambda \mapsto 0], S)}{\Gamma \vdash \overline{a}\, b.P \ \triangleright \ \Delta, a : (\nu, !T(\delta, \lambda).S)} \quad \text{[send]}$$

$$\frac{\begin{array}{c}\forall t : \nu + t \models \delta \Leftrightarrow t \leq n \\ \forall t \leq n : \ \Gamma, b : T \vdash P \ \triangleright \ \Delta + t, a : (\nu + t[\lambda \mapsto 0], S) \quad \Delta \ \textit{not } t\textit{-reading}\end{array}}{\Gamma \vdash a^n(b).P \ \triangleright \ \Delta, a : (\nu, ?T(\delta, \lambda).S)} \quad \text{[rcv]}$$

$$\frac{\forall n \in \delta : \ \Gamma \vdash \texttt{delay}(n).P \ \triangleright \ \Delta}{\Gamma \vdash \texttt{delay}(\delta).P \ \triangleright \ \Delta} \quad \text{[delay1]}$$

$$\frac{\Gamma \vdash P \ \triangleright \ \Delta + n \quad \Delta \ \textit{not } n\textit{-reading}}{\Gamma \vdash \texttt{delay}(n).P \ \triangleright \ \Delta} \quad \text{[delay2]}$$

What is a missed deadline?

$$S = ?\texttt{Com}({\color{blue}x < 5, x}).S' \qquad C = !\texttt{Com}({\color{blue}y < 5, y}).C'$$

$$\texttt{delay}(4.90).a(b).P'_s \mid \texttt{delay}(4.99).\overline{a}(\texttt{HELO}).P'_c$$

$$\longrightarrow \; a(b).P'_s \mid \texttt{delay}(0.09).\overline{a}(\texttt{HELO}).P'_c$$

is this **really** a violation of progress?

$$\longrightarrow \; {\color{red}\texttt{failed}} \mid \texttt{delay}(0.09).\overline{a}(\texttt{HELO}).P'_c$$

- Failing semantics:
  - See system's behaviour beyond failure of some parts (-> error handling)
  - Reveals relationship between **untimed progress** and **time safety**

Programs

time-consuming

$$
\begin{aligned}
P ::= {} & \overline{a}v \,.\, P \\
\mid {} & a \triangleleft \mathtt{l} \,.\, P \\
\mid {} & \mathtt{if}\ v\ \mathtt{then}\ P\ \mathtt{else}\ P \\
\mid {} & P \mid P \\
\mid {} & 0 \\
\mid {} & \mathtt{def}\ D\ \mathtt{in}\ P \\
\mid {} & X\langle \overrightarrow{a}; \overrightarrow{a} \rangle \\
\mid {} & (\nu ab)\ P \\
\mid {} & ab : h
\end{aligned}
$$

$$
\begin{aligned}
\mid {} & \mathtt{delay}(\delta) \,.\, P \\
\mid {} & a^n(b) \,.\, P \\
\mid {} & a^n \triangleright \{\mathtt{l}_i : P_i\}_{i \in I}
\end{aligned}
$$

$$
\begin{aligned}
\mid {} & \mathtt{delay}(n) \,.\, P \\
\mid {} & \mathtt{failed}
\end{aligned}
$$

run-time

Subject reduction?

$$(\nu ab)(\nu cd)\ a^5(e)\,.\,\overline{d}e.0 \mid c^5(e)\,.\,\overline{b}e.0 \mid ab : \varnothing \mid ba : \varnothing \mid cd : \varnothing \mid cd : \varnothing$$

$$\longrightarrow (\nu ab)(\nu cd)(\texttt{failed} \mid \texttt{failed} \mid ab : \varnothing \mid ba : \varnothing \mid cd : \varnothing \mid cd : \varnothing)$$

Well typed

$$\varnothing \vdash P \triangleright a : (\nu_0, S),\ b : (\nu_0, \overline{S}),\ c : (\nu_0, S),\ d : (\nu_0, \overline{S})$$

$$S = \,!\texttt{Int}(x \leq 5, \varnothing)\,.\,\texttt{end}$$

Subject reduction does not hold in general

Subject reduction!

Definition (Live process). $\hat{P}$ *is live if, for each* $\hat{P}'$ *such that* $\hat{P} \longrightarrow^* \hat{P}'$:

$$\hat{P}' \equiv (\nu ab)\hat{Q} \wedge a \in \texttt{Wait}(\hat{Q}) \implies \exists \hat{Q}' : \hat{Q} \longrightarrow^* \hat{Q}' \wedge a \in \texttt{NEQueue}(\hat{Q}')$$

Theorem (Subject Reduction). *Let* $\texttt{erase}(P)$ *be live. If* $\emptyset \vdash P \triangleright \emptyset$ *and* $P \longrightarrow P'$ *then* $\emptyset \vdash P \triangleright \emptyset$.

Theorem (Time Safety). *If* $\texttt{erase}(P)$ *is live,* $\emptyset \vdash P \triangleright \emptyset$ *and* $P \longrightarrow^* P'$ *then* P' *is fail-free.*

In summary

- Duality, subtyping, & urgent receive.

In [Bartoletti,Bocchi,Murgia@CONCUR'18] asymmetric refinement does not preserve behaviour/progress (it was "local" and did not assume duality)

- Dual types cannot be (correctly) implemented with previous work on Multiparty Asynchronous Timed Session Types

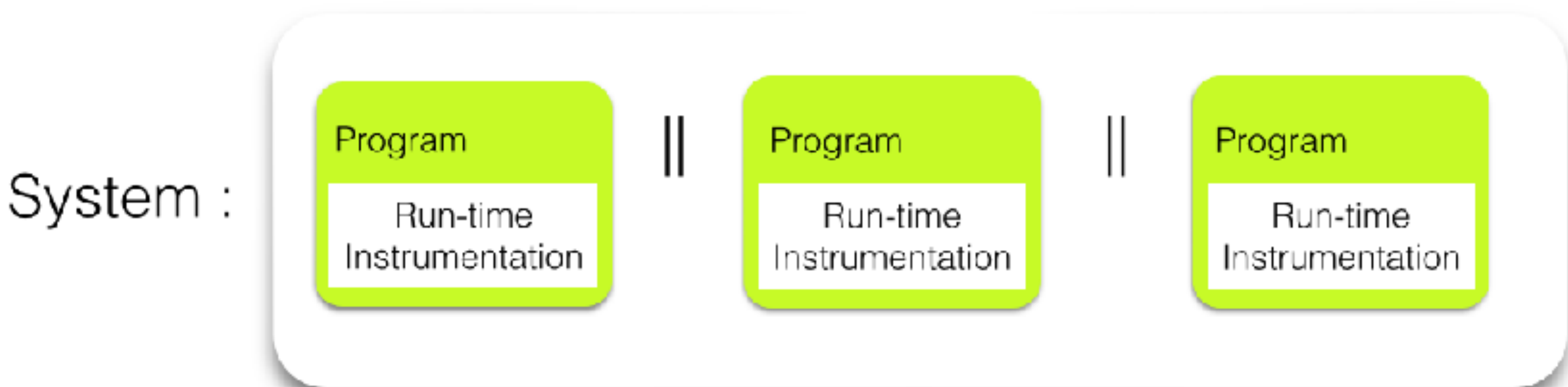
A **time-sensitive calculus** with: parametric receive, delays with arbitrary but constrained delays, and explicit failures upon timeout

A typing system for processes (with delegation) that satisfies subject reduction and time safety

Considerations on the meaning of progress and failure in a timed context

Future work

- Time-sensitive protocol design and implementation EP/N035372/1
  - expressiveness (flexible timing schedules) + run-time adjustments



Thank you!