University Of Groningen

FORMAL MODELLING OF COMMUNICATING SYSTEM

Session and Session Types

Frans Juanda Simanjuntak S3038971

supervised by
Dr. Jorge Perez Parra

April 11, 2017

1 Introduction

Nowadays, the development of networked computing has made significant changes in the IT world. It allows the computers or nodes communicating and working together over a network rather than running as unconnected, stand-alone devices. This concept has attracted many experts and communities who are interested in networked computing to develop programming practice based on communication among processes.

Recently, many programming languages and formalisms have been proposed to describe the concept of software based on communication. CSP, Ada, POOL, ABCL and SmallTalk are the examples of programming languages for communicating system and CCS, CSP, and π-calculus are some of the formalisms which have been created so far.

In order to describe the information interchange between two or more communicating devices, a term called **session** was introduced. This term has been implemented in various formalisms and such formalism called **session types** was developed to describe communication protocol over session channels.

This summary presents a resume of a paper from Mariangiola, et.al., [DCd10] that discussed about the overview of session and session types. The summary starts with presenting the concept of session and session type in Section 2. Then it continues with the typing system in section 3 followed by some extensions of session and session types, also the implementation in Section 4.

2 Session and Session Types

A session can be defined as a private connection between a finite number of participants. In order to explain this term easily, let us recall the example of user and vending machine with some modifications. Let us assume that there is a vending machine which allows user to buy items such as coffee, cola, and ice cream with card only (similar to the vending machine at campus). This vending machine also provides a new feature called *checking remaining credit*. When a user go to the vending machine and buy something, for instance, a cup of coffee, by the time he inserted his card and started selecting an item, unconsciously he already created a session in order to be able to communicate with the vending machine. The interaction between user and vending machine can be described via formalisms. *Session types* are one of type formalism that can be used to describe the communication protocol over private session channels.

To illustrate the concept of session and its formal representation with session types, let us consider the following protocol which describes the interaction between user and vending machine:

- First, user insert the card to the vending machine.
- The vending machine answers with either success or failure.
  - If the answer is **success**, the vending machine will ask either for *buying item* or *checking remaining credits*. For *buying item*, the user communicates an item and then the vending machine answers either dispense or cancel transaction if the credit is not sufficient. For *checking credits*, the vending machine will show the remaining credit.
  - If the answer is **failure**, then the interaction between user and vending machine is terminated.

The above illustration can be described with a global description as follows.

```
User ⟶ VM:identifier
VM ⟶ User:
        {
          success:   User ⟶ VM:
            {buy:     User ⟶ VM:item.
                       VM ⟶ User:
                       { dispense:end
                              ||
                         cancel: end
                         }
                    ||
                checkcredit:     User ⟶ VM
                                 VM ⟶ User: credit info.
                }
          ||   failure: end
}
```

We have created a global description that depicts the interaction between user and vending machine, and now this interaction can be easily converted to session types. Before we do the conversion, first thing first, we should be familiar with the syntax of language primitives for structured communication-based programming which was proposed by Honda, et.al., [HVK98].

There are four important symbols we should know: **!** for input, **?** for output, $\bigoplus$ for selecting a choice, and **&** for offering a branching of choices. We must remove the arrows from the global description and replace them with ! or ? and also add new symbols $\bigoplus$ or & to describe the choices. For instances, the syntax **User** $\longrightarrow$ **VM:String** can be converted to **!String** if we interpret the session from user perspective and this interaction can be described as **?String** from vending machine perspective.

A session types system also checks whether the communication operations in user match the communication operations in vending machine. This is known as **duality** which intuitively describes the opposite behaviors between two communicating agents. The duality of session types is the basis of *communication safety* which aims to guarantee that only expected data type are exchanged and *session fidelity* which aims to ensure that different data types are allowed but in a particular sequence.

Following the concept of *duality* in session types, the implementation of both user agent and vending machine can be done as follows:

- **User agent**

```
ses(u).u ! identifier.
u & { success: if ... then u ⊕ buy: u ! item.
                                    u & {dispense : ...
                                         || cancel : ...
                                         }
                  else u ⊕ checkcredit: u ? (y).0

        || failure: 0
}
```

- **Vending Machine agent**

ses(v).v ? (x).
if ... then $v \bigoplus$ success: v & { buy: v ? (y).
if ... then $v \bigoplus$ {dispense : ...
else $v \bigoplus$ cancel : ...
||
checkcredit: v ! z.0
else $v \bigoplus$ failure: 0
}

From the implementation above, the session name is represented with **ses**, the initiation of session is represented with $\overline{\textbf{ses}}$**(u)** and **ses(v)** and local computation is represented with **...**. Once the session is initiated, the session channels are automatically substituted by a new name κ, $p \in \{+,-\}$. Below are some rules of session types which were applied on the communication between user and vending machine:

- *Session Initiation*

  ($\overline{ses}$(u).User) — (ses(v).VM) $\longrightarrow$ ($\nu\kappa$)(User$\{\kappa^+/u\}$ — VM$\{\kappa^-/v\}$) where ($\nu\kappa$) represents channel hiding.

- *Receive/Send Values* using the polarised channels κ_1^p.

  We assume the action which was being performed is buy item.

  (κ^p ! *item*.User) — ($\kappa^{\overline{p}}$? *(y)*.VM) $\longrightarrow$ User — VM $\{item/y\}$

- *Select/Branching Actions.*

  We assume user selected buy action from available options.

  ($\kappa^p \bigoplus$ buy : User)—($\kappa^{\overline{p}}$ & { buy: VM_1 || checkcredit: VM_2}) $\longrightarrow$ User — VM_1, ($1 \leq$ i $\leq$ *2*)

- *Delegation.*

  Let us extend the scenario by allowing a new protocol, for example, the central vending machine server which aims to perform all actions and the vending machine just play the role as a forwarder between user and vending machine server. This process can be converted into session types as follows:

ses(v).v ? (x).
if ... then $v \bigoplus$ success: $\overline{ses2}(u)$. u ! x. v&{ buy: $u \bigoplus$ buy:
v ? (y). u ! y.
u&{dispense: $v \bigoplus$ {dispense : ...
|| cancel: $v \bigoplus$ cancel : ...
||
checkcredit: v ? (a). u!(a). u?(b). v!b.0
else $v \bigoplus$ failure: 0
}

This scenario can be simplified using *delegation* and the implementation would be:

ses(v).v ? (x).
if ... then $v \bigoplus$ success: $\overline{ses2}(u).\ u\ !\ x.\ u\ !\ v.0$
else $v \bigoplus$ failure: 0
}

3 Typing System

The more complex the interaction becomes, the more difficult to capture the whole interaction behavior as well as to write correct programs. However, this problem can be handled by *typing discipline*. The main sequence of typing system is $\Gamma \vdash \mathbf{P} \triangleright \Delta$ which should be read as *"Under the environment* Γ*,* Δ *a process P has a typing* Δ*"*. The description of each notation as follows: *Sorting* Γ specifies protocol at free names of P and *Typing* Δ specifies P's behavior at its free channels.

Honda and friends proposed fourteen typing rules in their paper "Language Primitives and Type For Structured Communication Based Programming" [HVK98]. These rules will handle all possible actions in session type. To get acquainted with the typing rules, let us pick three rules, for instance, *session initiation*, *receive/send values*, and *delegation* and apply these rules on the interaction between user and vending machine afterwards.

Using the typing rules, the initiation of session of both protocols can be written as:

$$\frac{\Gamma,\ \text{ses:}[S] \vdash P \triangleright \Delta,\ k\text{:}\ S}{\Gamma,\ \text{ses:}[S] \vdash \text{ses}(k).P \triangleright \Delta}$$

whose dual can be described as:

$$\frac{\Gamma,\ \text{ses:}[S] \vdash P \triangleright \Delta,\ k\text{:}\ \overline{S}}{\Gamma,\ \text{ses:}[S] \vdash \overline{ses}(k).P \triangleright \Delta}$$

The syntax **ses:[*S*]** exhibits that *the session name ses which is initiated by user will be able to open a session whose session channel* ***k*** *has type S (in this case the type is string because user sends its identity as string)*. This initiation can be matched by its dual, the vending machine. From the vending machine side, the intuition of the rules would be the vending machine is able to open a session whose session channel k has type $\overline{S}$.

Another rules which can be applied on user and vending machine is **send and receive value**. Below are the rules for receiving values:

$$\frac{\Gamma,\ x\ :\ \mathrm{T} \vdash P \triangleright \Delta,\ k\text{:}\ S'}{\Gamma \vdash k\ ?\ (x).P \triangleright \Delta,\ k\text{:}\ ?\ \mathrm{T}.S'}$$

whose dual, the sender, can be derived by rule:

$$\frac{\Gamma \vdash P \triangleright \Delta,\ k\text{:}\ S''\ \Gamma \vdash v\ :\ \mathrm{T}}{\Gamma \vdash k\ !\ (v).P \triangleright \Delta,\ k\text{:}\ !\ \mathrm{T}.S''}$$

Using send identity to the vending machine, the above rules can be interpreted as:

- Vending machine will receive a value with type of T, then the conversation will continue with S′.
- User will send output to the vending machine with type of T and the conversation will continue with S″.

Delegation also has rules for both sending and receiving process. For the sending process the rule is:

$$\frac{\Gamma \vdash P \triangleright \Delta,\ k\text{:}\ S_1}{\Gamma \vdash\ k\ !\ (h).P \triangleright \Delta,\ k\text{:}\ !\ S_2.S_1, h : S_2}$$

and for the receiving process the rule is:

$$\frac{\Gamma \vdash Q \triangleright \Delta,\ k\text{:}\ S_1,\ h\text{:}\ S_2}{\Gamma \vdash\ k\ ?\ (h).Q \triangleright \Delta,\ k\text{:}\ ?\ S_2.S_1}$$

4 Extension and Implementation

Several extensions of the calculus were proposed, such as *Correspondence Assertion* to detect any miss behavior of agent, *Multiparty Sessions* to support interaction with many participants, *Concurent Constraint* to specify some constrains of session, *Code Moibility* to avoid many interactions, *Exception* to handle such exceptions, and *Resource Access Control Through Delegation* to enrich multiparty sessions with security. Another extensions of typing were also proposed such as *subtyping* to support more general type of value, *Bounded Polymosphism* to support flexibility of protocols, *Progress* to support asynchronous output, *Action Permutation* to improve efficiency by executing output before input, *Semantic Subtyping*, and *Hnnessy-Milner Logic*.

Session Type have also been implemented on functional and object oriented. The example of implementation in functional programming is *Haskell* which was done by Neubeur and Thiemann [NT04]. The implementation of session type in object oriented language can be found in the language **Sing#** (Variant of C#) [FAH+06], **SJ** which is an extension for Java syntax to support session type [HYH08], **Scribble** [1] which is a language to describe global and local behaviour, and **Bica**, an extension to the Java5 compiler [GVR+10].

References

[DCd10] Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. *Sessions and Session Types: An Overview*, pages 1–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[FAH+06] Manuel Fahndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, Jim Larus, and Steven Levi. Language support for fast and reliable message-based communication in singularity os. In *Proceedings of the EuroSys 2006 Conference*, page 177–190. Association for Computing Machinery, Inc., April 2006.

[GVR+10] Simon J. Gay, Vasco T. Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, pages 299–312, New York, NY, USA, 2010. ACM.

[HVK98] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. *Language primitives and type discipline for structured communication-based programming*, pages 122–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[HYH08] Raymond Hu, Nobuko Yoshida, and Kohei Honda. *Session-Based Distributed Programming in Java*, pages 516–541. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[NT04] Matthias Neubauer and Peter Thiemann. *An Implementation of Session Types*, pages 56–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[1] http://sourceforge.net/projects/pi4scribble/