Dummy title

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent convallis orci arcu, eu mollis dolor. Aliquam eleifend suscipit lacinia. Maecenas quam mi, porta ut lacinia sed, convallis ac dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements Anonymous acknowledgements

1 Introduction

2 The Session Calculus

We introduce the simple synchronous session calculus that our type system will be used on.

2.1 Processes and Sessions

▶ **Definition 2.1** (Expressions and Processes)**.** *We define processes as follows:*

$$\mathsf{P} ::= \mathsf{p}!\ell(\mathsf{e}).\mathsf{P} \;\mid\; \sum_{i\in I} \mathsf{p}?\ell_i(x_i).\mathsf{P}_i \;\mid\; \mathsf{if\ e\ then\ P\ else\ P} \;\mid\; \mu\boldsymbol{X}.\mathsf{P} \;\mid\; \boldsymbol{X} \;\mid\; \boldsymbol{0}$$

where e *is an expression that can be a variable, a value such as* `true`*,* 0 *or* -3*, or a term built from expressions by applying the operators* `succ`*,* `neg`*,* $\neg$*, non-deterministic choice* $\oplus$ *and* $>$*.*

$\mathsf{p}!\ell(\mathsf{e}).\mathsf{P}$ is a process that sends the value of expression e with label ℓ to participant p, and continues with process P. $\sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i$ is a process that may receive a value from any $\ell_i \in I$, binding the result to x_i and continuing with P_i, depending on which ℓ_i the value was received from. $\mathbf{X}$ is a recursion variable, $\mu\mathbf{X}.\mathsf{P}$ is a recursive process, $\mathsf{if\ e\ then\ P\ else\ P}$ is a conditional and $\mathbf{0}$ is a terminated process.

Processes can be composed in parallel into sessions.

▶ **Definition 2.2** (Multiparty Sessions)**.** *Multiparty sessions are defined as follows.*

$$\mathcal{M} \;::=\; \mathsf{p} \triangleleft \mathsf{P} \quad\mid\quad (\mathcal{M} \mid \mathcal{M}) \quad\mid\quad \mathcal{O}$$

$\mathsf{p} \triangleleft \mathsf{P}$ denotes that participant p is running the process P, $\mid$ indicates parallel compositon. We write $\prod_{i\in I} \mathsf{p}_i \triangleleft \mathsf{P}_i$ to denote the session formed by p_i running P_i in parallel for all $i \in I$. $\mathcal{O}$ is an empty session with no participants, that is, the unit of parallel composition.

▶ Remark 2.3. Note that $\mathcal{O}$ is different than $\mathsf{p} \triangleleft \mathbf{0}$ as p is a participant in the latter but not the former. This differs from previous work, e.g. in [1] the unit of parallel composition is $\mathsf{p} \triangleleft \mathbf{0}$. For a detailed discussion see **??**.

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:3
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2.2 Structural Congruence and Operational Semantics

We define a structural congruence relation $\equiv$ on sessions which expresses the commutativity, associativity and unit of the parallel composition operator.

$$\text{[SC-SYM]} \quad \mathsf{p} \triangleleft \mathsf{P} \mid \mathsf{q} \triangleleft \mathsf{Q} \equiv \mathsf{q} \triangleleft \mathsf{Q} \mid \mathsf{p} \triangleleft \mathsf{P} \qquad \text{[SC-ASSOC]} \quad (\mathsf{p} \triangleleft \mathsf{P} \mid \mathsf{q} \triangleleft \mathsf{Q}) \mid \mathsf{r} \triangleleft \mathsf{R} \equiv \mathsf{p} \triangleleft \mathsf{P} \mid (\mathsf{q} \triangleleft \mathsf{Q} \mid \mathsf{r} \triangleleft \mathsf{R})$$

$$\text{[SC-O]} \quad \mathsf{p} \triangleleft \mathsf{P} \mid \mathsf{q} \triangleleft \mathcal{O} \equiv \mathsf{p} \triangleleft \mathsf{P}$$

Table 1 Structural Congruence over Sessions

We now give the operational semantics for sessions by the means of a labelled transition system. We have two kinds of transitions, *silent* (τ) and *observable* (β). Correspondingly, we have two kinds of *transition labels*, τ and $(\mathsf{p}, \mathsf{q})\ell$ where p, q are participants and ℓ is a message label. We omit the semantics of expressions, they are standard and can be found in [1, Table 1]. We write $e \downarrow v$ when expression e evaluates to value v.

$$\text{[R-COMM]} \quad \frac{j \in I \quad e \downarrow v}{\mathsf{p} \triangleleft \sum_{i \in I} \mathsf{q}?\ell_i(x_i).\mathsf{P}_i \mid \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).\mathsf{Q} \mid \mathcal{N} \xrightarrow{(\mathsf{p},\mathsf{q})\ell_j} \mathsf{p} \triangleleft \mathsf{P}_j[v/x_j] \mid \mathsf{q} \triangleleft \mathsf{Q} \mid \mathcal{N}}$$

$$\text{[R-REC]} \quad \mathsf{p} \triangleleft \mu \mathbf{X}.\mathsf{P} \mid \mathcal{N} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathsf{P}[\mu \mathbf{X}.\mathsf{P}/\mathbf{X}] \mid \mathcal{N}$$

$$\text{[R-CONDT]} \quad \frac{e \downarrow \text{true}}{\mathsf{p} \triangleleft \text{ if } e \text{ then } \mathsf{P} \text{ else } \mathsf{Q} \mid \mathcal{N} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathsf{P} \mid \mathcal{N}}$$

$$\text{[R-CONDF]} \quad \frac{e \downarrow \text{false}}{\mathsf{p} \triangleleft \text{ if } e \text{ then } \mathsf{P} \text{ else } \mathsf{Q} \mid \mathcal{N} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathsf{Q} \mid \mathcal{N}}$$

$$\text{[R-STRUCT]} \quad \frac{\mathcal{N}_1' \equiv \mathcal{N}_1 \quad \mathcal{N}_1 \xrightarrow{\lambda} \mathcal{N}_2 \quad \mathcal{N}_2 \equiv \mathcal{N}_2'}{\mathcal{N}_1' \xrightarrow{\lambda} \mathcal{N}_2'}$$

Table 2 Operational Semantics of Sessions

In Table 2, [R-COMM] describes a synchronous communication from p to q via message label ℓ_j. [R-REC] unfolds recursion, [R-CONDT] and [R-CONDF] express how to evaluate conditionals, and [R-STRUCT] shows that the reduction respects the structural pre-congruence. We write $\mathcal{M} \to \mathcal{N}$ if $\mathcal{M} \xrightarrow{\lambda} \mathcal{N}$ for some transition label λ. We write $\to^*$ to denote the reflexive transitive closure of $\to$. We also write $\mathcal{M} \xrightarrow{\tau}^* \mathcal{N}$ when all t.

We have thus given labelled transition semantics for sessions. Later in this paper, we also define types and LTS semantics on them, establish *simulations* between sessions and their types, and use these simulations to prove properties about sessions. It turns out that τ transitions are not observed by types (blah blah consider transitions up to weak bisimilarity)). Hence we also define an *unfolding* relationship ($\Rrightarrow$) on sessions.

[R-COMM]
$$\frac{j \in I \quad e \downarrow v}{\mathsf{p} \triangleleft \sum_{i \in I} \mathsf{q}?\ell_i(x_i).\mathsf{P}_i \;\mid\; \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).\mathsf{Q} \;\mid\; \mathcal{N} \xrightarrow{(\mathsf{p},\mathsf{q})\ell_j} \mathsf{p} \triangleleft \mathsf{P}_j[v/x_j] \;\mid\; \mathsf{q} \triangleleft \mathsf{Q} \;\mid\; \mathcal{N}}$$

[R-REC]
$$\mathsf{p} \triangleleft \mu\mathbf{X}.\mathsf{P} \mid \mathcal{N} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathsf{P}[\mu\mathbf{X}.\mathsf{P}/\mathbf{X}] \;\mid\; \mathcal{N}$$

[R-CONDT]
$$\frac{e \downarrow \mathsf{true}}{\mathsf{p} \triangleleft \text{ if } e \text{ then } \mathsf{P} \text{ else } \mathsf{Q} \;\mid\; \mathcal{N} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathsf{P} \;\mid\; \mathcal{N}}$$

[R-CONDF]
$$\frac{e \downarrow \mathsf{false}}{\mathsf{p} \triangleleft \text{ if } e \text{ then } \mathsf{P} \text{ else } \mathsf{Q} \;\mid\; \mathcal{N} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathsf{Q} \;\mid\; \mathcal{N}}$$

[R-STRUCT]
$$\frac{\mathcal{N}_1' \equiv \mathcal{N}_1 \quad \mathcal{N}_1 \xrightarrow{\lambda} \mathcal{N}_2 \quad \mathcal{N}_2 \equiv \mathcal{N}_2'}{\mathcal{N}_1' \xrightarrow{\lambda} \mathcal{N}_2'}$$

Table 3 The unfolding relation

3 The Type System

4 LTS Semantics for Types

5 Properties of Local Types

6 Properties of Sessions

References

1 Silvia Ghilezan, Svetlana Jakšić, Jovanka Pantović, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Algebraic Methods in Programming*, 104:127–173, 2019. URL: `https://www.sciencedirect.com/science/article/pii/S2352220817302237`, `doi:10.1016/j.jlamp.2018.12.002`.