Event Structures as Psi-calculus

Håkon Normann*

Dept. of Informatics – Univ. of Oslo, P.O. Box 1080 Blindern, 0316 Oslo. E-mail: haakno@ifi.uio.no

Abstract

Psi-calculi have been recently introduced as a generalization of π-calculi involving nominal data structures and powerful conditionals and assertions [BJPV11]. Instantiations of psi-calculi become standard variants of pi-calculus like the cryptographic, polyadic, or distributed extensions. We are interested in this paper in how psi-calculi could accommodate the event structures model of concurrency, with a final goal of capturing the Dynamic Condition Response graphs model (DCR-graphs). Event names in event-based models of concurrency are unique, and can thus be thought of nominals, whereas the execution of an event can be seen as a transmission of some sort. The dependencies between events that an event structure defines can be captured with rather simple assertions on the nominal data structures, and similarly the notion of computation in event structures.

These are the basic ideas the we follow in this work to give an encoding of event structures into an instance of psi-calculi. The drawback is that psi-calculi have interleaving semantics using rewrite rules, whereas the event structures are a true concurrency model. Irrespective of this aspect we give a result for the encoding that shows that the concurrency embodied by the event structure is captured in the translation psi-process through the standard interleaving diamond. Another feature of true concurrency models is that they are well behaved wrt. action refinement. For this we give another result showing that action refinement is preserved by our translation; under a properly defined refining function on psi-process, which we define similarly to the refinement function on the event structures. A corollary of the refinement is that we get a composition result for a restricted form of parallel operation on event structures.

We believe that the techniques that we use in the translation of event structures can be easily extended to translate condition-response event structures (CRES) and DCR-graphs into instances of psi-calculi.

1 Psi-calculi and event structures

We recall only the notions from psi-calculi and event structures that we use in this short version. A long version containing complete proofs, more definitions and explanations can be found on the authors homepage (`http://folk.uio.no/haakno/`).

A Psi-calculus is built over *data* terms M, N. These are used in the communication primitives as $\overline{M}N.P$ to say that the process sends data N over the channel M, and $\underline{K}(\lambda\widetilde{x})L.Q$, to say that channel K receives data matching the pattern $\lambda\widetilde{x}.L$. Interaction is under the conditions:
(1) The two channels M and K are equivalent, as defined by a predicate $M \leftrightarrow K$
(2) N matches the input pattern, i.e., $N = L[\widetilde{x} := \widetilde{T}]$ for some sequence of terms $\widetilde{T}$

Psi-calculi are parametrised by the following entities, and we instantiate these to obtain our particular psi-calculus for capturing event structures in Section 2. Define three nominal data types: $\mathbf{T}$, i.e., the (data) terms ranged over by N, M; $\mathbf{C}$, i.e., the conditions ranged over by φ; $\mathbf{A}$, i.e., the assertions ranged over by Ψ. Define four equivariant operators: $\dot{\leftrightarrow}\colon \mathbf{T} \times \mathbf{T} \to \mathbf{C}$, *channel equivalence*; $\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$, *composition*; $\mathbf{1} : \mathbf{A}$, *unit*; $\vdash \subseteq \mathbf{A} \times \mathbf{C}$, *entailment*; respecting the following conditions: $\dot{\leftrightarrow}$ must be symmetric and transitive; $\otimes$ must be associative, commutative, compositional, and maintain identity.

*I would like to thank my supervisors Thomas Hildebrandt and Cristian Prisacariu for their help.

Transitions are of kind $\Psi \triangleright P \xrightarrow{\alpha} P'$, meaning that when the environment provides the assertion Ψ then a well formed agent P can do an α to become P'. The part of the operational semantics used in this paper is given below, where Ψ_Q in PAR is the assertion exposed by Q.

$$\frac{\Psi \vdash M \leftrightarrow K}{\Psi \triangleright \overline{M}N.P \xrightarrow{\overline{K}N} P}\ \textsc{Out} \qquad \frac{\Psi \triangleright P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \triangleright case\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}\ \textsc{Case} \qquad \frac{\Psi_Q \otimes \Psi \triangleright P \xrightarrow{a} P'}{\Psi \triangleright P|Q \xrightarrow{a} P'|Q}\ \textsc{Par}$$

For event structures we follow that notation of [NPW81].

Definition 1.1 (prime event structures)**.** *A prime event structure is a tuple $\varepsilon = (E, <, \sharp)$ where E is a set of events, $< \subseteq E \times E$ is an (irreflexive) partial order (the causality relation) satisfying the principle of finite causes, i.e., $\forall e \in E : \{d \in E | d < e\}$ is finite, $\sharp \subseteq E \times E$ is an irreflexive, symmetric relation (the conflict relation) satisfying the principle of conflict heredity, i.e., $\forall d, e, f \in E : d < e \wedge d\sharp f, \Rightarrow e\sharp f,$.*

A prime event structure models a concurrent system, intuitively, by using $d < e$ to mean that d is a prerequisite of e, and $d\sharp e$ to mean that d and e cannot both happen in same run, i.e., a choice/branching point. Casual independence (concurrency) between events $d||e$ is modelled as the absence of casual dependence or conflict, i.e., $\neg(d < e \vee e < d \vee d\sharp e)$. The computation of an event structure ε is captured by the subsets of events, called configurations C_ε, each containing the events that happened in some partial run.

2 Encoding event structures in psi-calculi

Definition 2.1 (event psi-calculus)**.** *We define a psi-calculus instance, which we call* event psi*, parametrized by a nominal set E, to be understood as* events*. This means providing the following definitions of the key elements of a psi-calculus instance:*

$$\boldsymbol{T} \stackrel{def}{=} E \qquad \boldsymbol{C} \stackrel{def}{=} \mathcal{P}(E) \times \mathcal{P}(E) \qquad \boldsymbol{A} \stackrel{def}{=} \mathcal{P}(E) \qquad \dot{\leftrightarrow} \stackrel{def}{=} = \qquad \otimes \stackrel{def}{=} \cup \qquad \boldsymbol{1} \stackrel{def}{=} \emptyset$$

$$\vdash \stackrel{def}{=} \Psi \vdash \varphi \text{ iff } (\pi_L(\varphi) \subseteq \Psi) \wedge (\Psi \cap \pi_R(\varphi) = \emptyset) \qquad \Psi \vdash a \dot{\leftrightarrow} b \text{ iff } a = b$$

We have that T, C, and A are nominal sets. Channel equivalence maintains symmetry and transitivity since $=$ is upholding these rules. The $\otimes$ is compositional, associative and commutative as $\cup$ is. and as $\emptyset \cup S = S$, for any set S. We have that identity is maintained.

We use only one simple nominal set for T which intuitively is understood to be the set of all events. The conditions C consists of pairs of subsets of events. The assertions A is intuitively understood as capturing the set of all executed events (and thus ranges over T). Channel equivalence is equality of event names. Composition of two assertions is the union of the sets. Identity is the empty-set. The entailment $\vdash$ intuitively captures when events may fire, thus describing when events are enabled by a configuration in event structures, as well as how it affects channel equivalence.

Definition 2.2 (from event structures to event psi)**.** *We define a function* PSI *which given an event structure $\varepsilon = (E, <, \sharp)$ and a Configuration C_ε of ε, returns an event psi process $P_E = |_{e \in E} P_e$ with $P_e = (\!|\{e\}|\!)$ if $e \in C_\varepsilon$, otherwise $P_e = case\ \varphi_e : \overline{e}e.(\!|\{e\}|\!)$, where we have $\varphi_e = (<e, \sharp e)$ with $<e = \{e' | (e', e) \in <\}$ the set of all events e has as conditions, and $\sharp e = \{e' | (e', e) \in \sharp\}$ the events e is in conflict with.*

A process generated by the PSI function is built up by smaller event processes put in parallel. Where each event process is either in state executed or not executed depending on whether it is in the configuration or not.

For each event e we make a condition φ_e that contains two sets, the set of events e is depending on and the set of events e is in conflict with. When an event happens we will have a transition over the channel with the same name as the event. For event structures where the configuration is empty we only give the tuple as input.

Lemma 2.1 (correspondence configuration–frame)**.** *For any event structure ε and configuration C_ε the frame of the event-psi process $\text{PSI}(\varepsilon, C_\varepsilon)$ corresponds to the configuration C_ε.*

Lemma 2.2 (transitions maintain configurations)**.** *For some event structure ε and some configuration of it C_ε, then any transition from this configuration $C_\varepsilon \xrightarrow{e} C'_\varepsilon$ is matched by a transition $\text{PSI}(\varepsilon, C_\varepsilon) \xrightarrow{\overline{e}e} \text{PSI}(\varepsilon, C'_\varepsilon)$ in the corresponding psi process.*

Theorem 2.3 (preserving concurrency)**.** *For an event structure $(E, <, \sharp)$ with two concurrent events $e||e'$ then in the translation $\text{PSI}(E, <, \sharp)$ we find the behaviour forming the interleaving diamond, i.e., $\emptyset \rhd \text{PSI}(E, <, \sharp) \xrightarrow{e} P_1 \xrightarrow{e'} P_2$ and $\emptyset \rhd \text{PSI}(E, <, \sharp) \xrightarrow{e'} P_3 \xrightarrow{e} P_4$ with $P_2 = P_4$.*

We want to be able to refine the psi processes on the same line as event structures are refined in [vGG01]; i.e., for labeled prime event structures, a function $ref : Act \to \mathbf{E}_{prime}$ is called a refinement function iff $\forall a \in Act : ref(a)$ is non-empty, finite and conflict-free. Then ref(ε) is the prime event structure defined by: $E_{ref(\varepsilon)} := \{(e, e')|e \in E_\varepsilon, e' \in E_{ref(l_\varepsilon(e))}\}$; $(d, d') <_{ref(\varepsilon)} (e, e')$ iff $d <_\varepsilon e$ or $(d = e \wedge d' <_{ref(l_\varepsilon(d))} e')$; $(d, d')\sharp_{ref(\varepsilon)}(e, e')$ iff $d\sharp_\varepsilon e$; $l_{ref(\varepsilon)}(e, e') := l_{ref(l_\varepsilon(e))}(e')$.

Definition 2.3. *We define a function ref^P that refines an event-psi process to a new one over $T^P = \{(e, e')|e \in T, e' \in T_{ref(e)}\}$, with $\varphi^P = \{(<(e, e'), \sharp(e, e'))|(e, e') \in T^P\}$, where $<(e, e') = \{(d, d')|d \in < e \vee d = e \wedge d' \in <_{ref(d)} e\}$ and $\sharp(e, e') = \{(d, d')|d \in \sharp e\}$ to obtain $P_{ref} = |_{(e,e')\in T^P} P_{(e,e')}$ with $P_{(e,e')} = (\!|\{(e, e')\}|\!)$ if $e \in \Psi_P$, else $P_{(e,e')} = \overline{(e, e')}(e, e').(\!|\{(e, e')\}|\!)$.*

The new names are all possible pairs of a parent events name and one of its children events name. This can be the same as the parents name. We make new conditions for each of the new names (e, e'), where $<(e, e')$ is all pairs of names where first part is a condition for e, or if first part is same as e, second part is condition for e'. For $\sharp(e, e')$ we have that it is all pairs of names where first part is precondition for e. Then make new event processes for each new pair, where state is either executed or not executed depending on whether first part of event name was in the frame of the old event-psi.

Theorem 2.4 (refinement of event-psi corresponds to refinement in ES)**.** *For a prime event structure ε we have that:* $\text{PSI}(ref(\varepsilon)) = ref^P(\text{PSI}(\varepsilon))$.

References

[BJPV11] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.

[NPW81] Nielsen, Plotkin, and Winksel. Petri nets, event structures and domains, part I. *TCS: Theoretical Computer Science*, 13:85–108, 1981.

[vGG01] Rob J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4/5):229–327, 2001.