CS 430/530
Formal Semantics

Zhong Shao

Yale University
Department of Computer Science

Concurrency; Dynamic Classification

April 22, 2025

Process Calculus: Actions & Events

To begin with, we will focus on *sequential processes*, which simply await the arrival of one of several possible actions, known as an event.

Proc	P	::=	`await`(E)	$\$\, E$	synchronize
Evt	E	::=	`null`	$\mathbf{0}$	null
			`or`$(E_1; E_2)$	$E_1 + E_2$	choice
			`que`$[a](P)$	$?a;P$	query
			`sig`$[a](P)$	$!a;P$	signal

The variable a ranges over symbols serving as *channels* that mediate communication among the processes.

An illustrative example of Milner's is a simple vending machine that may take in a 2p coin, then optionally either allow a request for a cup of tea, or take another 2p coin, then allow a request for a cup of coffee.

$$V = \$\,(?\texttt{2p};\$\,(!\texttt{tea};V + ?\texttt{2p};\$\,(!\texttt{cof};V))) \tag{39.2}$$

Process Calculus: Actions & Events

We will not distinguish between events that differ only up to *structural congruence*, which is defined to be the strongest equivalence relation closed under these rules:

$$\frac{E \equiv E'}{\$\, E \equiv \$\, E'} \tag{39.1a}$$

$$\frac{E_1 \equiv E_1' \quad E_2 \equiv E_2'}{E_1 + E_2 \equiv E_1' + E_2'} \tag{39.1b}$$

$$\frac{P \equiv P'}{?a;P \equiv ?a;P'} \tag{39.1c}$$

$$\frac{P \equiv P'}{!a;P \equiv !a;P'} \tag{39.1d}$$

$$\frac{}{E + \mathbf{0} \equiv E} \tag{39.1e}$$

$$\frac{}{E_1 + E_2 \equiv E_2 + E_1} \tag{39.1f}$$

$$\frac{}{E_1 + (E_2 + E_3) \equiv (E_1 + E_2) + E_3} \tag{39.1g}$$

Process Calculus: Interactions

Processes become interesting when they are allowed to interact with one another to achieve a common goal. To account for interaction, we enrich the language of processes with *concurrent composition*:

Proc	P	::=	`await`(E)	$\$\, E$	synchronize
			`stop`	$\mathbf{1}$	inert
			`conc`$(P_1; P_2)$	$P_1 \otimes P_2$	composition

Process Calculus: Interactions

We will identify processes up to structural congruence, the strongest equivalence relation closed under these rules:

$$\frac{}{P \otimes \mathbf{1} \equiv P} \tag{39.3a}$$

$$\frac{}{P_1 \otimes P_2 \equiv P_2 \otimes P_1} \tag{39.3b}$$

$$\frac{}{P_1 \otimes (P_2 \otimes P_3) \equiv (P_1 \otimes P_2) \otimes P_3} \tag{39.3c}$$

$$\frac{P_1 \equiv P_1' \quad P_2 \equiv P_2'}{P_1 \otimes P_2 \equiv P_1' \otimes P_2'} \tag{39.3d}$$

Up to structural congruence every process has the form

$$\$\, E_1 \otimes \ldots \otimes \$\, E_n$$

for some $n \geq 0$, it being understood that when $n = 0$ this stands for the null process $\mathbf{1}$.

Process Calculus: Interactions

Interaction between processes consists of synchronization of two complementary actions. The dynamics of interaction is defined by two forms of judgment. The transition judgment $P \longmapsto P'$ states that the process P evolves to the process P' as a result of a single step of computation. The family of transition judgments, $P \overset{\alpha}{\mapsto} P'$, where α is an *action*, states that the process P may evolve to the process P' as long as the action α is permissible in the context in which the transition occurs. As a notational convenience, we often regard the unlabeled transition to be the labeled transition corresponding to the special silent action.

The possible actions are given by the following grammar:

Act	α	$::=$	$\texttt{que}[a]$	$a\,?$	query
			$\texttt{sig}[a]$	$a\,!$	signal
			$\texttt{sil}$	ε	silent

Process Calculus: Interactions

The *query action* $a\,?$ and the *signal action* $a\,!$ are complementary, and the *silent action* ε, is self-complementary. We define the *complementary* action to α to be the action $\overline{\alpha}$ given by the equations $\overline{a\,?} = a\,!$, $\overline{a\,!} = a\,?$, and $\overline{\varepsilon} = \varepsilon$.

$$\frac{}{\$\,(!a;P + E) \overset{a\,!}{\mapsto} P} \tag{39.4a}$$

$$\frac{}{\$\,(?a;P + E) \overset{a\,?}{\mapsto} P} \tag{39.4b}$$

$$\frac{P_1 \overset{\alpha}{\mapsto} P_1'}{P_1 \otimes P_2 \overset{\alpha}{\mapsto} P_1' \otimes P_2} \tag{39.4c}$$

$$\frac{P_1 \overset{\alpha}{\mapsto} P_1' \quad P_2 \overset{\overline{\alpha}}{\mapsto} P_2'}{P_1 \otimes P_2 \longmapsto P_1' \otimes P_2'} \tag{39.4d}$$

Process Calculus: Interactions

As an example, let us consider the vending machine V, given by Equation (39.2), interacting with the user process U defined as follows:

$$U = \$\,!\mathtt{2p};\$\,!\mathtt{2p};\$\,?\mathtt{cof};\mathbf{1}.$$

$$V = \$\,(?\mathtt{2p};\$\,(!\mathtt{tea};V + ?\mathtt{2p};\$\,(!\mathtt{cof};V)))$$

Here is a trace of the interaction between V and U:

$$\begin{aligned} V \otimes U &\longmapsto \$\,(!\mathtt{tea};V + ?\mathtt{2p};\$\,!\mathtt{cof};V) \otimes \$\,!\mathtt{2p};\$\,?\mathtt{cof};\mathbf{1} \\ &\longmapsto \$\,!\mathtt{cof};V \otimes \$\,?\mathtt{cof};\mathbf{1} \\ &\longmapsto V \end{aligned}$$

These steps are justified by the following pairs of labeled transitions:

$$U \overset{\mathtt{2p}\,!}{\longmapsto} U' = \$\,!\mathtt{2p};\$\,?\mathtt{cof};\mathbf{1}$$

$$V \overset{\mathtt{2p}\,?}{\longmapsto} V' = \$\,(!\mathtt{tea};V + ?\mathtt{2p};\$\,!\mathtt{cof};V)$$

$$U' \overset{\mathtt{2p}\,!}{\longmapsto} U'' = \$\,?\mathtt{cof};\mathbf{1}$$

$$V' \overset{\mathtt{2p}\,?}{\longmapsto} V'' = \$\,!\mathtt{cof};V$$

$$U'' \overset{\mathtt{cof}\,?}{\longmapsto} \mathbf{1}$$

$$V'' \overset{\mathtt{cof}\,!}{\longmapsto} V$$

Process Calculus: Replication

Some presentations of process calculi forego reliance on defining equations for processes in favor of a *replication* construct, which we write as $*\,P$. This process stands for as many concurrently executing copies of P as needed. Implicit replication can be expressed by the structural congruence

$$*\,P \equiv P \otimes *\,P. \tag{39.5}$$

Understood as a principle of structural congruence, this rule hides the steps of process creation and gives no hint as to how often it should be applied. We could alternatively build replication into the dynamics to model the details of replication more closely:

$$*\,P \longmapsto P \otimes *\,P. \tag{39.6}$$

$$\frac{}{*\,\$\,(!a;P + E) \overset{a\,!}{\longmapsto} P \otimes *\,\$\,(!a;P + E)} \tag{39.10a}$$

$$\frac{}{*\,\$\,(?a;P + E) \overset{a\,?}{\longmapsto} P \otimes *\,\$\,(?a;P + E)} \tag{39.10b}$$

Process Calculus: Allocating Channels

Proc P ::= $\texttt{new}(a.P)$ $\nu\, a.P$ new channel

The channel a is bound within the process P. To simplify notation, we sometimes write $\nu\, a_1, \ldots, a_k.P$ for the iterated declaration $\nu\, a_1 \ldots \nu\, a_k.P$.

We then extend structural congruence with the following rules:

$$\frac{P =_\alpha P'}{P \equiv P'} \tag{39.11a}$$

$$\frac{P \equiv P'}{\nu\, a.P \equiv \nu\, a.P'} \tag{39.11b}$$

$$\frac{a \notin P_2}{(\nu\, a.P_1) \otimes P_2 \equiv \nu\, a.(P_1 \otimes P_2)} \tag{39.11c}$$

$$\frac{}{\nu\, a.\nu\, b.P \equiv \nu\, b.\nu\, a.P} \tag{39.11d}$$

$$\frac{(a \notin P)}{\nu\, a.P \equiv P} \tag{39.11e}$$

PiC Statics

To account for the scopes of channels, we extend the statics of **PiC** with a *signature* Σ comprising a finite set of active channels. The judgment $\vdash_\Sigma P$ proc states that a process P is well-formed relative to the channels declared in the signature Σ.

$$\frac{}{\vdash_\Sigma \mathbf{1}\ \mathsf{proc}} \tag{39.12a}$$

$$\frac{\vdash_\Sigma P_1\ \mathsf{proc} \quad \vdash_\Sigma P_2\ \mathsf{proc}}{\vdash_\Sigma P_1 \otimes P_2\ \mathsf{proc}} \tag{39.12b}$$

$$\frac{\vdash_\Sigma E\ \mathsf{event}}{\vdash_\Sigma \$E\ \mathsf{proc}} \tag{39.12c}$$

$$\frac{\vdash_{\Sigma,a} P\ \mathsf{proc}}{\vdash_\Sigma \nu\, a.P\ \mathsf{proc}} \tag{39.12d}$$

PiC Statics

The foregoing rules make use of an auxiliary judgment, $\vdash_\Sigma E$ event, stating that E is a well-formed event relative to Σ.

$$\frac{}{\vdash_\Sigma \mathbf{0} \text{ event}} \tag{39.13a}$$

$$\frac{\vdash_{\Sigma,a} P \text{ proc}}{\vdash_{\Sigma,a} ?a;P \text{ event}} \tag{39.13b}$$

$$\frac{\vdash_{\Sigma,a} P \text{ proc}}{\vdash_{\Sigma,a} !a;P \text{ event}} \tag{39.13c}$$

$$\frac{\vdash_\Sigma E_1 \text{ event} \quad \vdash_\Sigma E_2 \text{ event}}{\vdash_\Sigma E_1 + E_2 \text{ event}} \tag{39.13d}$$

The judgment $\vdash_\Sigma \alpha$ action states that α is a well-formed action relative to Σ:

$$\frac{}{\vdash_{\Sigma,a} a\,? \text{ action}} \tag{39.14a}$$

$$\frac{}{\vdash_{\Sigma,a} a\,! \text{ action}} \tag{39.14b}$$

$$\frac{}{\vdash_\Sigma \varepsilon \text{ action}} \tag{39.14c}$$

PiC Dynamics

The dynamics of the current fragment of **PiC** is correspondingly generalized to keep track of the set of active channels. The judgment $P \overset{\alpha}{\underset{\Sigma}{\mapsto}} P'$ states that P transitions to P' with action α relative to channels Σ. The dynamics of this extension is obtained by indexing the transitions by the signature, and adding a rule for channel declaration.

$$\$\,(!a;P + E) \overset{a\,!}{\underset{\Sigma,a}{\longmapsto}} P \tag{39.15a}$$

$$\$\,(?a;P + E) \overset{a\,?}{\underset{\Sigma,a}{\longmapsto}} P \tag{39.15b}$$

$$\frac{P_1 \overset{\alpha}{\underset{\Sigma}{\mapsto}} P_1'}{P_1 \otimes P_2 \overset{\alpha}{\underset{\Sigma}{\mapsto}} P_1' \otimes P_2} \tag{39.15c}$$

$$\frac{P_1 \overset{\alpha}{\underset{\Sigma}{\mapsto}} P_1' \quad P_2 \overset{\overline{\alpha}}{\underset{\Sigma}{\mapsto}} P_2'}{P_1 \otimes P_2 \underset{\Sigma}{\mapsto} P_1' \otimes P_2'} \tag{39.15d}$$

$$\frac{P \overset{\alpha}{\underset{\Sigma,a}{\longmapsto}} P' \quad \vdash_\Sigma \alpha \text{ action}}{\nu\, a.P \overset{\alpha}{\underset{\Sigma}{\mapsto}} \nu\, a.P'} \tag{39.15e}$$

PiC Communication

To account for interprocess communication, we enrich the language of processes to include *variables*, as well as *channels*, in the formalism. Variables range, as always, over types, and are given meaning by substitution. Channels, on the other hand, are assigned types that classify the data carried on that channel and are given meaning by send and receive events that generalize the signal and query events considered in Section 39.2. The abstract syntax of communication events is given by the following grammar:

Evt	E	$::=$	$\mathtt{snd}[a](e; P)$	$!\,a(e\,;P)$	send
			$\mathtt{rcv}[a](x.P)$	$?\,a(x.P)$	receive

The event $\mathtt{rcv}[a](x.P)$ represents the receipt of a value x on the channel a, passing x to the process P. The variable x is bound within P. The event $\mathtt{snd}[a](e; P)$ represents the transmission of e on a and continuing with P.

We modify the syntax of declarations to account for the type of value sent on a channel.

Proc	P	$::=$	$\mathtt{new}\{\tau\}(a.P)$	$\nu\, a \sim \tau.P$	typed channel

The process $\mathtt{new}[\tau](a.P)$ introduces a new channel a with associated type τ for use within the process P. The channel a is bound within P.

PiC Communication

The statics is extended to account for the type of a channel. The judgment $\Gamma \vdash_\Sigma P$ proc states that P is a well-formed process involving the channels declared in Σ and the variables declared in Γ. It is inductively defined by the following rules, wherein we assume that the typing judgment $\Gamma \vdash_\Sigma e : \tau$ is given separately.

$$\frac{}{\Gamma \vdash_\Sigma \mathbf{1}\ \mathsf{proc}} \tag{39.16a}$$

$$\frac{\Gamma \vdash_\Sigma P_1\ \mathsf{proc} \quad \Gamma \vdash_\Sigma P_2\ \mathsf{proc}}{\Gamma \vdash_\Sigma P_1 \otimes P_2\ \mathsf{proc}} \tag{39.16b}$$

$$\frac{\Gamma \vdash_{\Sigma, a \sim \tau} P\ \mathsf{proc}}{\Gamma \vdash_\Sigma \nu\, a \sim \tau . P\ \mathsf{proc}} \tag{39.16c}$$

$$\frac{\Gamma \vdash_\Sigma E\ \mathsf{event}}{\Gamma \vdash_\Sigma \$\, E\ \mathsf{proc}} \tag{39.16d}$$

PiC Communication

Rules (39.16) make use of the auxiliary judgment $\Gamma \vdash_\Sigma E \text{ event}$, stating that E is a well-formed event relative to Γ and Σ, which is defined as follows:

$$\frac{}{\Gamma \vdash_\Sigma \mathbf{0} \text{ event}} \tag{39.17a}$$

$$\frac{\Gamma \vdash_\Sigma E_1 \text{ event} \quad \Gamma \vdash_\Sigma E_2 \text{ event}}{\Gamma \vdash_\Sigma E_1 + E_2 \text{ event}} \tag{39.17b}$$

$$\frac{\Gamma, x : \tau \vdash_{\Sigma, a \sim \tau} P \text{ proc}}{\Gamma \vdash_{\Sigma, a \sim \tau} ?a(x.P) \text{ event}} \tag{39.17c}$$

$$\frac{\Gamma \vdash_{\Sigma, a \sim \tau} e : \tau \quad \Gamma \vdash_{\Sigma, a \sim \tau} P \text{ proc}}{\Gamma \vdash_{\Sigma, a \sim \tau} !a(e\,;P) \text{ event}} \tag{39.17d}$$

Rule (39.17d) makes use of a typing judgment for expressions that ensures that the type of a channel is respected by communication.

PiC Communication

The dynamics of communication extends that of synchronization by enriching send and receive actions with the value sent or received.

$$\begin{array}{llcll} \mathsf{Act} & \alpha & ::= & \mathtt{rcv}[a](e) & a\,?\,e \quad \text{receive} \\ & & & \mathtt{snd}[a](e) & a\,!\,e \quad \text{send} \\ & & & \mathtt{sil} & \varepsilon \quad \text{silent} \end{array}$$

Complementarity is defined as before, by switching the orientation of an action: $\overline{a\,?\,e} = a!e$, $\overline{a\,!\,e} = a\,?\,e$, and $\overline{\varepsilon} = \varepsilon$.

The statics ensures that the expression associated with these actions is a value of a type suitable for the channel:

$$\frac{\vdash_{\Sigma, a \sim \tau} e : \tau \quad e\ \mathsf{val}_{\Sigma, a \sim \tau}}{\vdash_{\Sigma, a \sim \tau} a\,!\,e\ \mathsf{action}} \tag{39.18a}$$

$$\frac{\vdash_{\Sigma, a \sim \tau} e : \tau \quad e\ \mathsf{val}_{\Sigma, a \sim \tau}}{\vdash_{\Sigma, a \sim \tau} a\,?\,e\ \mathsf{action}} \tag{39.18b}$$

$$\frac{}{\vdash_{\Sigma} \varepsilon\ \mathsf{action}} \tag{39.18c}$$

PiC Communication

The dynamics is defined by replacing the synchronization rules (39.15a) and (39.15b) with the following communication rules:

$$\frac{e \xmapsto[\Sigma, a \sim \tau]{} e'}{\$\,(!\,a(e\,;P) + E) \xmapsto[\Sigma, a \sim \tau]{} \$\,(!\,a(e'\,;P) + E)} \tag{39.19a}$$

$$\frac{e\ \mathsf{val}_{\Sigma, a \sim \tau}}{\$\,(!\,a(e\,;P) + E) \xmapsto[\Sigma, a \sim \tau]{a!e} P} \tag{39.19b}$$

$$\frac{e\ \mathsf{val}_{\Sigma, a \sim \tau}}{\$\,(?\,a(x.P) + E) \xmapsto[\Sigma, a \sim \tau]{a?e} [e/x]P} \tag{39.19c}$$

Using synchronous communication, both the sender and the receiver of a message are blocked until the interaction is completed. Therefore the sender must be notified whenever a message is received, which means that there must be an implicit reply channel from receiver to sender that carries the notification. This means that synchronous communication can be decomposed into a simpler *asynchronous send* operation, which sends a message on a channel without waiting for its receipt, together with *channel passing* to send an acknowledgment channel along with the message data.

PiC Communication

Asynchronous communication is defined by removing the synchronous send event from the process calculus and adding a new form of process that simply sends a message on a channel. The syntax of asynchronous send is as follows:

$$\text{Proc} \quad P \quad ::= \quad \texttt{asnd}[a](e) \quad !\,a(e) \quad \text{send}$$

The process $\texttt{asnd}[a](e)$ sends the message e on channel a and then terminates immediately. Without the synchronous send event, every event is, up to structural congruence, a choice of zero or more read events. The statics of asynchronous send is given by the following rule:

$$\frac{\Gamma \vdash_{\Sigma, a \sim \tau} e : \tau}{\Gamma \vdash_{\Sigma, a \sim \tau} !\,a(e)\ \mathsf{proc}} \tag{39.20}$$

The dynamics is given similarly:

$$\frac{e\ \mathsf{val}_\Sigma}{!\,a(e) \xmapsto[\Sigma]{a!e} \mathbf{1}} \tag{39.21}$$

The rule for communication remains unchanged. A pending asynchronous send is essentially a buffer holding the value to be sent once a receiver is available.

PiC Channel Passing

The syntax of channel reference types is given by the following grammar:

Typ	τ	$::=$	$\mathtt{chan}(\tau)$	$\tau\ \mathtt{chan}$	channel type
Exp	e	$::=$	$\mathtt{chref}[a]$	$\&\,a$	reference
Evt	E	$::=$	$\mathtt{sndref}(e_1; e_2; P)$	$!!\,(e_1\,;e_2\,;P)$	send
			$\mathtt{rcvref}(e; x.P)$	$??\,(e\,;x.P)$	receive

The events $\mathtt{sndref}(e_1; e_2; P)$ and $\mathtt{rcvref}(e; x.P)$ are dynamic versions of the events $\mathtt{snd}[a](e; P)$ and $\mathtt{rcv}[a](x.P)$ in which the channel reference is determined dynamically by evaluation of an expression.

The statics of channel references is given by the following rules:

$$\frac{}{\Gamma \vdash_{\Sigma, a\sim\tau} \&\,a : \tau\ \mathtt{chan}} \tag{39.22a}$$

$$\frac{\Gamma \vdash_\Sigma e_1 : \tau\ \mathtt{chan} \quad \Gamma \vdash_\Sigma e_2 : \tau \quad \Gamma \vdash_\Sigma P\ \mathsf{proc}}{\Gamma \vdash_\Sigma !!\,(e_1\,;e_2\,;P)\ \mathsf{event}} \tag{39.22b}$$

$$\frac{\Gamma \vdash_\Sigma e : \tau\ \mathtt{chan} \quad \Gamma, x : \tau \vdash_\Sigma P\ \mathsf{proc}}{\Gamma \vdash_\Sigma ??\,(e\,;x.P)\ \mathsf{event}} \tag{39.22c}$$

PiC Channel Passing

Because channel references are forms of expression, events must be evaluated to determine the channel to which they refer.

$$\frac{E \underset{\Sigma, a\sim\tau}{\longmapsto} E'}{\$(E) \underset{\Sigma, a\sim\tau}{\longmapsto} \$(E')} \tag{39.23a}$$

$$\frac{e \ \mathsf{val}_{\Sigma, a\sim\tau}}{\$(!!(\&a\,;e\,;P)+E) \underset{\Sigma, a\sim\tau}{\longmapsto} \$(!a(e\,;P)+E)} \tag{39.23b}$$

$$\frac{e \ \mathsf{val}_{\Sigma, a\sim\tau}}{\$(??(\&a\,;x.P)+E) \underset{\Sigma, a\sim\tau}{\longmapsto} \$(?a(x.P)+E)} \tag{39.23c}$$

Dynamic Classes

A dynamic class is a symbol generated at run-time.

A classified value consists of a symbol of type τ and a value of that type.

To compute with a classified value, it is compared with a known class.

- **If the value is of this class, the underlying instance data is passed to the positive branch;**
- **Otherwise, the negative branch is taken**

The syntax of dynamic classification is given by the following grammar:

Typ	τ	::=	$\texttt{clsfd}$	$\texttt{clsfd}$	classified
Exp	e	::=	$\texttt{in}[a](e)$	$a \cdot e$	instance
			$\texttt{isin}[a](e; x.e_1; e_2)$	$\texttt{match}\, e\, \texttt{as}\, a \cdot x \hookrightarrow e_1\, \texttt{ow} \hookrightarrow e_2$	comparison

Dynamic Classes (Statics)

The statics of dynamic classification is defined by these rules:

$$\frac{\Gamma \vdash_{\Sigma,a\sim\tau} e : \tau}{\Gamma \vdash_{\Sigma,a\sim\tau} \mathtt{in}[a](e) : \mathtt{clsfd}} \tag{33.1a}$$

$$\frac{\Gamma \vdash_{\Sigma,a\sim\tau} e : \mathtt{clsfd} \quad \Gamma, x : \tau \vdash_{\Sigma,a\sim\tau} e_1 : \tau' \quad \Gamma \vdash_{\Sigma,a\sim\tau} e_2 : \tau'}{\Gamma \vdash_{\Sigma,a\sim\tau} \mathtt{isin}[a](e; x.e_1; e_2) : \tau'} \tag{33.1b}$$

Theorem 33.1 (Safety)**.**

1. *If* $\vdash_\Sigma e : \tau$ *and* $\nu\,\Sigma\,\{e\} \longmapsto \nu\,\Sigma'\,\{e'\}$, *then* $\Sigma' \supseteq \Sigma$ *and* $\vdash_{\Sigma'} e' : \tau$.
2. *If* $\vdash_\Sigma e : \tau$, *then either* $e\ \mathsf{val}_\Sigma$ *or* $\nu\,\Sigma\,\{e\} \longmapsto \nu\,\Sigma'\,\{e'\}$ *for some* e' *and* Σ'.

Dynamic Classes (Dynamics)

To maximize the flexibility in using dynamic classification, we will consider a free dynamics for symbol generation. Within this framework, the dynamics of classification is given by the following rules:

$$\frac{e\ \mathsf{val}_\Sigma}{\mathtt{in}[a](e)\ \mathsf{val}_\Sigma} \tag{33.2a}$$

$$\frac{\nu\ \Sigma\ \{\, e \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \,\}}{\nu\ \Sigma\ \{\, \mathtt{in}[a](e) \,\} \longmapsto \nu\ \Sigma'\ \{\, \mathtt{in}[a](e') \,\}} \tag{33.2b}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\nu\ \Sigma\ \{\, \mathtt{isin}[a](\mathtt{in}[a](e); x.e_1; e_2) \,\} \longmapsto \nu\ \Sigma\ \{\, [e/x]e_1 \,\}} \tag{33.2c}$$

$$\frac{e'\ \mathsf{val}_\Sigma \quad (a \neq a')}{\nu\ \Sigma\ \{\, \mathtt{isin}[a](\mathtt{in}[a'](e'); x.e_1; e_2) \,\} \longmapsto \nu\ \Sigma\ \{\, e_2 \,\}} \tag{33.2d}$$

$$\frac{\nu\ \Sigma\ \{\, e \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \,\}}{\nu\ \Sigma\ \{\, \mathtt{isin}[a](e; x.e_1; e_2) \,\} \longmapsto \nu\ \Sigma'\ \{\, \mathtt{isin}[a](e'; x.e_1; e_2) \,\}} \tag{33.2e}$$

Class References (Syntax & Statics)

The type $\texttt{cls}(\tau)$ has as values references to classes.

Typ	τ	$::=$	$\texttt{cls}(\tau)$	$\tau\ \texttt{cls}$	class reference
Exp	e	$::=$	$\texttt{cls}[a]$	$\&\, a$	reference
			$\texttt{mk}(e_1; e_2)$	$\texttt{mk}(e_1; e_2)$	instance
			$\texttt{isof}(e_0; e_1; x.e_2; e_3)$	$\texttt{isof}(e_0; e_1; x.e_2; e_3)$	dispatch

The statics of these constructs is given by the following rules:

$$\frac{}{\Gamma \vdash_{\Sigma, a \sim \tau} \texttt{cls}[a] : \texttt{cls}(\tau)} \tag{33.3a}$$

$$\frac{\Gamma \vdash_\Sigma e_1 : \texttt{cls}(\tau) \quad \Gamma \vdash_\Sigma e_2 : \tau}{\Gamma \vdash_\Sigma \texttt{mk}(e_1; e_2) : \texttt{clsfd}} \tag{33.3b}$$

$$\frac{\Gamma \vdash_\Sigma e_0 : \texttt{cls}(\tau) \quad \Gamma \vdash_\Sigma e_1 : \texttt{clsfd} \quad \Gamma, x : \tau \vdash_\Sigma e_2 : \tau' \quad \Gamma \vdash_\Sigma e_3 : \tau'}{\Gamma \vdash_\Sigma \texttt{isof}(e_0; e_1; x.e_2; e_3) : \tau'} \tag{33.3c}$$

Class References (Dynamics)

The corresponding dynamics is given by these rules:

$$\frac{\nu\ \Sigma\ \{\, e_1 \,\} \longmapsto \nu\ \Sigma'\ \{\, e_1' \,\}}{\nu\ \Sigma\ \{\, \mathtt{mk}(e_1; e_2) \,\} \longmapsto \nu\ \Sigma'\ \{\, \mathtt{mk}(e_1'; e_2) \,\}} \tag{33.4a}$$

$$\frac{e_1\ \mathsf{val}_\Sigma \quad \nu\ \Sigma\ \{\, e_2 \,\} \longmapsto \nu\ \Sigma'\ \{\, e_2' \,\}}{\nu\ \Sigma\ \{\, \mathtt{mk}(e_1; e_2) \,\} \longmapsto \nu\ \Sigma'\ \{\, \mathtt{mk}(e_1; e_2') \,\}} \tag{33.4b}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\nu\ \Sigma\ \{\, \mathtt{mk}(\mathtt{cls}[a]; e) \,\} \longmapsto \nu\ \Sigma\ \{\, \mathtt{in}[a](e) \,\}} \tag{33.4c}$$

$$\frac{\nu\ \Sigma\ \{\, e_0 \,\} \longmapsto \nu\ \Sigma'\ \{\, e_0' \,\}}{\nu\ \Sigma\ \{\, \mathtt{isof}(e_0; e_1; x.e_2; e_3) \,\} \longmapsto \nu\ \Sigma'\ \{\, \mathtt{isof}(e_0'; e_1; x.e_2; e_3) \,\}} \tag{33.4d}$$

$$\frac{}{\nu\ \Sigma\ \{\, \mathtt{isof}(\mathtt{cls}[a]; e_1; x.e_2; e_3) \,\} \longmapsto \nu\ \Sigma\ \{\, \mathtt{isin}[a](e_1; x.e_2; e_3) \,\}} \tag{33.4e}$$

Classifying Secrets

Dynamic classification can be used to enforce *confidentiality* and *integrity* of data values in a program. A value of type clsfd may only be constructed by *sealing* it with some class a and may only be deconstructed by a case analysis that includes a branch for a. By controlling which parties have access to the classifier a we may control how classified values are created (ensuring their *integrity*) and how they are inspected (ensuring their *confidentiality*). Any party that lacks access to a cannot decipher a value classified by a, nor may it create a classified value with this class. Because classes are dynamically generated symbols, they offer a confidentiality guarantee among parties in a computation

Consider the following simple protocol for controlling the integrity and confidentiality of data in a program. A fresh symbol a is introduced, and we return a pair of functions of type

$$(\tau \rightharpoonup \texttt{clsfd}) \times (\texttt{clsfd} \rightharpoonup \tau\ \texttt{opt}),$$

called the *constructor* and *destructor* functions for that class, which is accomplished by writing

$$\begin{array}{l}\texttt{new}\ a \sim \tau\ \texttt{in} \\ \quad \langle\ \lambda(x:\tau)\, a \cdot x\,, \\ \qquad \lambda(x:\texttt{clsfd})\,\texttt{match}\,x\,\texttt{as}\,a \cdot y \hookrightarrow \texttt{just}(y)\,\texttt{ow} \hookrightarrow \texttt{null}\ \rangle.\end{array}$$

Concurrent Algol

The syntax of **CA** is obtained by removing assignables from **MA**, and adding a syntactic level of *processes* to represent the global state of a program:

Typ	τ	::=	$\mathtt{cmd}(\tau)$	$\tau\ \mathtt{cmd}$	commands
Exp	e	::=	$\mathtt{cmd}(m)$	$\mathtt{cmd}\, m$	command
Cmd	m	::=	$\mathtt{ret}\, e$	$\mathtt{ret}\, e$	return
			$\mathtt{bnd}(e; x.m)$	$\mathtt{bnd}\, x \leftarrow e\ ;\, m$	sequence
Proc	p	::=	$\mathtt{stop}$	$\mathbf{1}$	idle
			$\mathtt{run}(m)$	$\mathtt{run}(m)$	atomic
			$\mathtt{conc}(p_1; p_2)$	$p_1 \otimes p_2$	concurrent
			$\mathtt{new}[\tau](a.p)$	$\nu\, a \sim \tau.p$	new channel

Concurrent Algol

The process `run`(m) is an atomic process executing the command m. The other forms of process are adapted from Chapter 39. If Σ has the form $a_1 \sim \tau_1, \ldots, a_n \sim \tau_n$, then we sometimes write $\nu\,\Sigma\{p\}$ for the iterated form $\nu\,a_1 \sim \tau_1 \ldots \nu\,a_n \sim \tau_n.p$.

The statics of **CA** is given by these judgments:

$\Gamma \vdash_\Sigma e : \tau$	expression typing
$\Gamma \vdash_\Sigma m \div \tau$	command typing
$\Gamma \vdash_\Sigma p\ \mathsf{proc}$	process formation
$\Gamma \vdash_\Sigma \alpha\ \mathsf{action}$	action formation

Concurrent Algol

Process formation is defined by the following rules:

$$\frac{}{\vdash_\Sigma \mathbf{1}\ \mathsf{proc}} \tag{40.1a}$$

$$\frac{\vdash_\Sigma m \div \tau}{\vdash_\Sigma \mathtt{run}(m)\ \mathsf{proc}} \tag{40.1b}$$

$$\frac{\vdash_\Sigma p_1\ \mathsf{proc} \quad \vdash_\Sigma p_2\ \mathsf{proc}}{\vdash_\Sigma p_1 \otimes p_2\ \mathsf{proc}} \tag{40.1c}$$

$$\frac{\vdash_{\Sigma, a \sim \tau} p\ \mathsf{proc}}{\vdash_\Sigma \nu\, a \sim \tau\, p\ \mathsf{proc}} \tag{40.1d}$$

Processes are identified up to structural congruence, as described in Chapter 39.

Action formation is defined by the following rules:

$$\frac{}{\vdash_\Sigma \varepsilon\ \mathsf{action}} \tag{40.2a}$$

$$\frac{\vdash_\Sigma e : \mathtt{clsfd} \quad e\ \mathsf{val}_\Sigma}{\vdash_\Sigma e\,!\ \mathsf{action}} \tag{40.2b}$$

$$\frac{\vdash_\Sigma e : \mathtt{clsfd} \quad e\ \mathsf{val}_\Sigma}{\vdash_\Sigma e\,?\ \mathsf{action}} \tag{40.2c}$$

Concurrent Algol

The dynamics of **CA** is defined by transitions between processes, which represent the state of the computation. More precisely, the judgment $p \overset{\alpha}{\underset{\Sigma}{\mapsto}} p'$ states that the process p evolves in one step to the process p' while undertaking action α.

$$\frac{m \overset{\alpha}{\underset{\Sigma}{\Longrightarrow}} \nu\, \Sigma'\, \{\, m' \otimes p \,\}}{\mathtt{run}(m) \overset{\alpha}{\underset{\Sigma}{\mapsto}} \nu\, \Sigma' \{\mathtt{run}(m') \otimes p\}} \tag{40.3a}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\mathtt{run}(\mathtt{ret}\, e) \overset{\varepsilon}{\underset{\Sigma}{\mapsto}} \mathbf{1}} \tag{40.3b}$$

$$\frac{p_1 \overset{\alpha}{\underset{\Sigma}{\mapsto}} p_1'}{p_1 \otimes p_2 \overset{\alpha}{\underset{\Sigma}{\mapsto}} p_1' \otimes p_2} \tag{40.3c}$$

$$\frac{p_1 \overset{\alpha}{\underset{\Sigma}{\mapsto}} p_1' \quad p_2 \overset{\overline{\alpha}}{\underset{\Sigma}{\mapsto}} p_2'}{p_1 \otimes p_2 \overset{\varepsilon}{\underset{\Sigma}{\mapsto}} p_1' \otimes p_2'} \tag{40.3d}$$

$$\frac{p \overset{\alpha}{\underset{\Sigma, a \sim \tau}{\longmapsto}} p' \quad \vdash_\Sigma \alpha\ \mathsf{action}}{\nu\, a \sim \tau . p \overset{\alpha}{\underset{\Sigma}{\mapsto}} \nu\, a \sim \tau . p'} \tag{40.3e}$$

Concurrent Algol

Executing a command in **CA** may, in addition to evolving to another command, allocate a new channel or may spawn a new process. More precisely, the judgment[1]

$$m \overset{\alpha}{\underset{\Sigma}{\Longrightarrow}} \nu\, \Sigma' \{ m' \otimes p' \}$$

states that the command m transitions to the command m' while creating new channels Σ' and new processes p'. The action α specifies the interactions of which m is capable when executed. As a notational convenience, we drop mention of the new channels or processes when either are trivial.

Concurrent Algol

The following rules define the execution of the basic forms of command inherited from **MA**:

$$\frac{e \underset{\Sigma}{\mapsto} e'}{\mathtt{ret}\, e \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} \mathtt{ret}\, e'} \tag{40.4a}$$

$$\frac{m_1 \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} \nu\, \Sigma' \{ m_1' \otimes p' \}}{\mathtt{bnd}\, x \leftarrow \mathtt{cmd}\, m_1 \,;\, m_2 \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} \nu\, \Sigma' \{ \mathtt{bnd}\, x \leftarrow \mathtt{cmd}\, m_1' \,;\, m_2 \otimes p' \}} \tag{40.4b}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\mathtt{bnd}\, x \leftarrow \mathtt{cmd}\,(\mathtt{ret}\, e) \,;\, m_2 \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} [e/x]m_2} \tag{40.4c}$$

$$\frac{e_1 \underset{\Sigma}{\mapsto} e_1'}{\mathtt{bnd}\, x \leftarrow e_1 \,;\, m_2 \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} \mathtt{bnd}\, x \leftarrow e_1' \,;\, m_2} \tag{40.4d}$$

These rules are supplemented by rules governing communication and synchronization among processes in the next two sections.

CA with Broadcast Communication

The syntax of the commands pertinent to broadcast communication is given by the following grammar:

Cmd	m	$::=$	`spawn`(e)	`spawn`(e)	spawn
			`emit`(e)	`emit`(e)	emit message
			`acc`	`acc`	accept message
			`newch`$\{\tau\}$	`newch`	new channel

The command `spawn`(e) spawns a process that executes the encapsulated command given by e. The commands `emit`(e) and `acc` emit and accept messages, which are classified values whose class is the channel on which the message is sent. The command `newch`$[\tau]$ returns a reference to a fresh class carrying values of type τ.

CA with Broadcast Communication

The statics of broadcast communication is given by the following rules:

$$\frac{\Gamma \vdash_\Sigma e : \mathtt{cmd(unit)}}{\Gamma \vdash_\Sigma \mathtt{spawn}(e) \div \mathtt{unit}} \tag{40.5a}$$

$$\frac{\Gamma \vdash_\Sigma e : \mathtt{clsfd}}{\Gamma \vdash_\Sigma \mathtt{emit}(e) \div \mathtt{unit}} \tag{40.5b}$$

$$\frac{}{\Gamma \vdash_\Sigma \mathtt{acc} \div \mathtt{clsfd}} \tag{40.5c}$$

$$\frac{}{\Gamma \vdash_\Sigma \mathtt{newch}\{\tau\} \div \mathtt{cls}(\tau)} \tag{40.5d}$$

CA with Broadcast Communication

Execution of these commands is defined as follows:

$$\frac{}{\texttt{spawn(cmd}(m)) \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} \texttt{ret}\,\langle\rangle \otimes \texttt{run}(m)} \tag{40.6a}$$

$$\frac{e \overset{}{\underset{\Sigma}{\mapsto}} e'}{\texttt{spawn}(e) \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} \texttt{spawn}(e')} \tag{40.6b}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\texttt{emit}(e) \overset{e!}{\underset{\Sigma}{\Rightarrow}} \texttt{ret}\,\langle\rangle} \tag{40.6c}$$

$$\frac{e \overset{}{\underset{\Sigma}{\mapsto}} e'}{\texttt{emit}(e) \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} \texttt{emit}(e')} \tag{40.6d}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\texttt{acc} \overset{e?}{\underset{\Sigma}{\Rightarrow}} \texttt{ret}\,e} \tag{40.6e}$$

$$\frac{}{\texttt{newch}\{\tau\} \overset{\varepsilon}{\underset{\Sigma}{\Rightarrow}} \nu\,\texttt{a} \sim \tau\,\{\texttt{ret}\,(\&\,\texttt{a})\}} \tag{40.6f}$$

CA with Broadcast Communication

Lemma 40.1. *If* $m \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} \nu\, \Sigma'\, \{m' \otimes p'\}$, $\vdash_\Sigma m \div \tau$, *then* $\vdash_\Sigma \alpha$ *action*, $\vdash_{\Sigma\,\Sigma'} m' \div \tau$, *and* $\vdash_{\Sigma\,\Sigma'} p'$ *proc.*

Proof By induction on rules (40.4). □

With this in hand, the proof of preservation goes along familiar lines.

Theorem 40.2 (Preservation). *If* $\vdash_\Sigma p$ *proc and* $p \overset{}{\underset{\Sigma}{\mapsto}} p'$, *then* $\vdash_\Sigma p'$ *proc.*

Proof By induction on transition, appealing to Lemma 40.1 for the crucial steps. □

Typing does not, however, guarantee progress with respect to unlabeled transition, for the simple reason that there may be no other process with which to communicate. By extending progress to labeled transitions, we may state that this is the *only* way for process execution to get stuck. But some care must be taken to account for allocating new channels.

Theorem 40.3 (Progress). *If* $\vdash_\Sigma p$ *proc, then either* $p \equiv \mathbf{1}$, *or* $p \equiv \nu\, \Sigma'\{p'\}$ *such that* $p' \overset{\alpha}{\underset{\Sigma\,\Sigma'}{\longmapsto}} p''$ *for some* $\vdash_{\Sigma\,\Sigma'} p''$ *and some* $\vdash_{\Sigma\,\Sigma'} \alpha$ *action.*

CA with Selective Communication

Typ	τ	$::=$	$\texttt{event}(\tau)$	$\tau\ \texttt{event}$	events
Exp	e	$::=$	$\texttt{rcv}[a]$	$?\,a$	selective read
			$\texttt{never}\{\tau\}$	$\texttt{never}$	null
			$\texttt{or}(e_1; e_2)$	$e_1\ \texttt{or}\ e_2$	choice
			$\texttt{wrap}(e_1; x.e_2)$	$e_1\ \texttt{as}\ x\ \texttt{in}\ e_2$	post-composition
Cmd	m	$::=$	$\texttt{sync}(e)$	$\texttt{sync}(e)$	synchronize

CA with Selective Communication

The statics of event expressions is given by the following rules:

$$\frac{\Sigma \vdash a \sim \tau}{\Gamma \vdash_\Sigma \texttt{rcv}[a] : \texttt{event}(\tau)} \tag{40.7a}$$

$$\frac{}{\Gamma \vdash_\Sigma \texttt{never}\{\tau\} : \texttt{event}(\tau)} \tag{40.7b}$$

$$\frac{\Gamma \vdash_\Sigma e_1 : \texttt{event}(\tau) \quad \Gamma \vdash_\Sigma e_2 : \texttt{event}(\tau)}{\Gamma \vdash_\Sigma \texttt{or}(e_1; e_2) : \texttt{event}(\tau)} \tag{40.7c}$$

$$\frac{\Gamma \vdash_\Sigma e_1 : \texttt{event}(\tau_1) \quad \Gamma, x : \tau_1 \vdash_\Sigma e_2 : \tau_2}{\Gamma \vdash_\Sigma \texttt{wrap}(e_1; x.e_2) : \texttt{event}(\tau_2)} \tag{40.7d}$$

CA with Selective Communication

The corresponding dynamics is defined by these rules:

$$\frac{\Sigma \vdash a \sim \tau}{\mathtt{rcv}[a]\ \mathsf{val}_\Sigma} \tag{40.8a}$$

$$\frac{}{\mathtt{never}\{\tau\}\ \mathsf{val}_\Sigma} \tag{40.8b}$$

$$\frac{e_1\ \mathsf{val}_\Sigma \quad e_2\ \mathsf{val}_\Sigma}{\mathtt{or}(e_1; e_2)\ \mathsf{val}_\Sigma} \tag{40.8c}$$

$$\frac{e_1 \underset{\Sigma}{\mapsto} e_1'}{\mathtt{or}(e_1; e_2) \underset{\Sigma}{\mapsto} \mathtt{or}(e_1'; e_2)} \tag{40.8d}$$

$$\frac{e_1\ \mathsf{val}_\Sigma \quad e_2 \underset{\Sigma}{\mapsto} e_2'}{\mathtt{or}(e_1; e_2) \underset{\Sigma}{\mapsto} \mathtt{or}(e_1; e_2')} \tag{40.8e}$$

$$\frac{e_1 \underset{\Sigma}{\mapsto} e_1'}{\mathtt{wrap}(e_1; x.e_2) \underset{\Sigma}{\mapsto} \mathtt{wrap}(e_1'; x.e_2')} \tag{40.8f}$$

$$\frac{e_1\ \mathsf{val}_\Sigma}{\mathtt{wrap}(e_1; x.e_2)\ \mathsf{val}_\Sigma} \tag{40.8g}$$

CA with Selective Communication

The statics of the synchronization command is given by the following rule:

$$\frac{\Gamma \vdash_\Sigma e : \mathtt{event}(\tau)}{\Gamma \vdash_\Sigma \mathtt{sync}(e) \mathbin{\dot{\sim}} \tau} \tag{40.9a}$$

The type of the event determines the type of value returned by the synchronization command.

Execution of a synchronization command depends on the event.

$$\frac{e \underset{\Sigma}{\mapsto} e'}{\mathtt{sync}(e) \underset{\Sigma}{\overset{\varepsilon}{\Rightarrow}} \mathtt{sync}(e')} \tag{40.10a}$$

$$\frac{e\ \mathsf{val}_\Sigma \quad \vdash_\Sigma e : \tau \quad \Sigma \vdash a \sim \tau}{\mathtt{sync}(\mathtt{rcv}[a]) \underset{\Sigma}{\overset{a \cdot e\,?}{\Longrightarrow}} \mathtt{ret}(e)} \tag{40.10b}$$

$$\frac{\mathtt{sync}(e_1) \underset{\Sigma}{\overset{\alpha}{\Rightarrow}} m_1}{\mathtt{sync}(\mathtt{or}(e_1; e_2)) \underset{\Sigma}{\overset{\alpha}{\Rightarrow}} m_1} \tag{40.10c}$$

CA with Selective Communication

$$\frac{\texttt{sync}(e_2) \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} m_2}{\texttt{sync}(\texttt{or}(e_1; e_2)) \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} m_2} \tag{40.10d}$$

$$\frac{\texttt{sync}(e_1) \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} m_1}{\texttt{sync}(\texttt{wrap}(e_1; x.e_2)) \overset{\alpha}{\underset{\Sigma}{\Rightarrow}} \texttt{bnd}(\texttt{cmd}(m_1); x.\texttt{ret}(e_2))} \tag{40.10e}$$