CS 430/530
Formal Semantics

Zhong Shao

Yale University
Department of Computer Science

Lazy Evaluation; Parallelism

April 17, 2025

LPCF: PCF By-Need

Lazy variant of PCF with functions being called-by-need and the successor operator evaluated lazily;

Variables are bound to unevaluated expressions;

By-need evaluation uses memoization (or thunk) to share all such copies of an argument so it is only evaluated at most once

The syntax of **PCF** is given by the following grammar:

Typ	τ	$::=$	$\mathtt{nat}$	$\mathtt{nat}$	naturals
			$\mathtt{parr}(\tau_1;\tau_2)$	$\tau_1 \rightharpoonup \tau_2$	partial function
Exp	e	$::=$	x	x	variable
			$\mathtt{z}$	$\mathtt{z}$	zero
			$\mathtt{s}(e)$	$\mathtt{s}(e)$	successor
			$\mathtt{ifz}\{e_0; x.e_1\}(e)$	$\mathtt{ifz}\, e\, \{\mathtt{z} \hookrightarrow e_0 \mid \mathtt{s}(x) \hookrightarrow e_1\}$	zero test
			$\mathtt{lam}\{\tau\}(x.e)$	$\lambda\,(x:\tau)\,e$	abstraction
			$\mathtt{ap}(e_1;e_2)$	$e_1(e_2)$	application
			$\mathtt{fix}\{\tau\}(x.e)$	$\mathtt{fix}\, x:\tau\, \mathtt{is}\, e$	recursion

LPCF Statics (Same as PCF)

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \tag{19.1a}$$

$$\frac{}{\Gamma \vdash \mathtt{z} : \mathtt{nat}} \tag{19.1b}$$

$$\frac{\Gamma \vdash e : \mathtt{nat}}{\Gamma \vdash \mathtt{s}(e) : \mathtt{nat}} \tag{19.1c}$$

$$\frac{\Gamma \vdash e : \mathtt{nat} \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x : \mathtt{nat} \vdash e_1 : \tau}{\Gamma \vdash \mathtt{ifz}\{e_0; x.e_1\}(e) : \tau} \tag{19.1d}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathtt{lam}\{\tau_1\}(x.e) : \mathtt{parr}(\tau_1; \tau_2)} \tag{19.1e}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{parr}(\tau_2; \tau) \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{ap}(e_1; e_2) : \tau} \tag{19.1f}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau}{\Gamma \vdash \mathtt{fix}\{\tau\}(x.e) : \tau} \tag{19.1g}$$

LPCF Dynamics

The dynamics of **LPCF** is based on a transition system with states of the form $\nu\ \Sigma\ \{\, e \parallel \mu \,\}$, where Σ is a finite set of hypotheses $a_1 \sim \tau_1, \ldots, a_n \sim \tau_n$ associating types to symbols, e is an expression that can involve the symbols in Σ, and μ maps each symbol declared in Σ to either an expression or a special symbol, $\bullet$, called the *black hole*. (The role of the black hole is explained below.) As a notational convenience, we use a bit of legerdemain with the concrete syntax similar to that used in Chapter 34. Specifically, the concrete syntax for the expression $\texttt{via}(a)$, which fetches the contents of the assignable a, is $@\ a$.

The dynamics of **LPCF** is given by he following two forms of judgment:

1. $e\ \mathsf{val}_\Sigma$, stating that e is a value that can involve the symbols in Σ.
2. $\nu\ \Sigma\ \{\, e \parallel \mu \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \parallel \mu' \,\}$, stating that one step of evaluation of the expression e relative to memo table μ with the symbols declared in Σ results in the expression e' relative to the memo table μ' with symbols declared in Σ'.

LPCF Dynamics

The judgment $e\ \mathsf{val}_\Sigma$ expressing that e is a closed value is defined by the following rules:

$$\frac{}{\mathtt{z}\ \mathsf{val}_\Sigma} \tag{36.1a}$$

$$\frac{}{\mathtt{s}(@\,a)\ \mathsf{val}_{\Sigma,a\sim\mathtt{nat}}} \tag{36.1b}$$

$$\frac{}{\lambda\,(x:\tau)\,e\ \mathsf{val}_\Sigma} \tag{36.1c}$$

The initial and final states of evaluation are defined as follows:

$$\frac{}{\nu\,\emptyset\,\{\,e \parallel \emptyset\,\}\ \mathsf{initial}} \tag{36.2a}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\nu\,\Sigma\,\{\,e \parallel \mu\,\}\ \mathsf{final}} \tag{36.2b}$$

LPCF Dynamics

$$\frac{e\ \mathsf{val}_{\Sigma, a \sim \tau}}{\nu\ \Sigma, a \sim \tau\ \{\ @\ a \parallel \mu \otimes a \hookrightarrow e\ \} \longmapsto \nu\ \Sigma, a \sim \tau\ \{\ e \parallel \mu \otimes a \hookrightarrow e\ \}} \tag{36.3a}$$

$$\frac{\nu\ \Sigma, a \sim \tau\ \{\ e \parallel \mu \otimes a \hookrightarrow \bullet\ \} \longmapsto \nu\ \Sigma', a \sim \tau\ \{\ e' \parallel \mu' \otimes a \hookrightarrow \bullet\ \}}{\nu\ \Sigma, a \sim \tau\ \{\ @\ a \parallel \mu \otimes a \hookrightarrow e\ \} \longmapsto \nu\ \Sigma', a \sim \tau\ \{\ @\ a \parallel \mu' \otimes a \hookrightarrow e'\ \}} \tag{36.3b}$$

$$\frac{}{\nu\ \Sigma\ \{\ \mathtt{s}(e) \parallel \mu\ \} \longmapsto \nu\ \Sigma, a \sim \mathtt{nat}\ \{\ \mathtt{s}(@\ a) \parallel \mu \otimes a \hookrightarrow e\ \}} \tag{36.3c}$$

$$\frac{\nu\ \Sigma\ \{\ e \parallel \mu\ \} \longmapsto \nu\ \Sigma'\ \{\ e' \parallel \mu'\ \}}{\nu\ \Sigma\ \{\ \mathtt{ifz}\ e\ \{\mathtt{z} \hookrightarrow e_0 \mid \mathtt{s}(x) \hookrightarrow e_1\} \parallel \mu\ \} \longmapsto \nu\ \Sigma'\ \{\ \mathtt{ifz}\ e'\ \{\mathtt{z} \hookrightarrow e_0 \mid \mathtt{s}(x) \hookrightarrow e_1\} \parallel \mu'\ \}} \tag{36.3d}$$

$$\frac{}{\nu\ \Sigma\ \{\ \mathtt{ifz}\ \mathtt{z}\ \{\mathtt{z} \hookrightarrow e_0 \mid \mathtt{s}(x) \hookrightarrow e_1\} \parallel \mu\ \} \longmapsto \nu\ \Sigma\ \{\ e_0 \parallel \mu\ \}} \tag{36.3e}$$

LPCF Dynamics

$$\left\{\begin{array}{c} \nu\ \Sigma, a \sim \mathtt{nat}\,\{\,\mathtt{ifz}\ \mathtt{s}(@\ a)\ \{\mathtt{z} \hookrightarrow e_0 \mid \mathtt{s}(x) \hookrightarrow e_1\} \parallel \mu \otimes a \hookrightarrow e\,\} \\ \longmapsto \\ \nu\ \Sigma, a \sim \mathtt{nat}\,\{\,[@\ a/x]e_1 \parallel \mu \otimes a \hookrightarrow e\,\} \end{array}\right\} \tag{36.3f}$$

$$\frac{\nu\ \Sigma\,\{\,e_1 \parallel \mu\,\} \longmapsto \nu\ \Sigma'\,\{\,e_1' \parallel \mu'\,\}}{\nu\ \Sigma\,\{\,e_1(e_2) \parallel \mu\,\} \longmapsto \nu\ \Sigma'\,\{\,e_1'(e_2) \parallel \mu'\,\}} \tag{36.3g}$$

$$\left\{\begin{array}{c} \nu\ \Sigma\,\{\,(\lambda\,(x:\tau)\,e)(e_2) \parallel \mu\,\} \\ \longmapsto \\ \nu\ \Sigma, a \sim \tau\,\{\,[@\ a/x]e \parallel \mu \otimes a \hookrightarrow e_2\,\} \end{array}\right\} \tag{36.3h}$$

$$\nu\ \Sigma\,\{\,\mathtt{fix}\,x:\tau\,\mathtt{is}\,e \parallel \mu\,\} \longmapsto \nu\ \Sigma, a \sim \tau\,\{\,@\ a \parallel \mu \otimes a \hookrightarrow [@\ a/x]e\,\} \tag{36.3i}$$

LPCF Type Safety

We write $\Gamma \vdash_\Sigma e : \tau$ to mean that e has type τ under the assumptions Γ, treating symbols declared in Σ as expressions of their associated type. The rules are as in Chapter 19, extended with the following rule for symbols:

$$\frac{}{\Gamma \vdash_{\Sigma, a \sim \tau} @\, a : \tau} \tag{36.4}$$

This rule states that the demand for the binding of a symbol, $@\, a$, is a form of expression. It is a "delayed substitution" that lazily replaces a demand for a by its binding.

The judgment $\nu\ \Sigma\ \{ e \parallel \mu \}\ \mathsf{ok}$ is defined by the following rules:

$$\frac{\vdash_\Sigma e : \tau \quad \vdash_\Sigma \mu : \Sigma}{\nu\ \Sigma\ \{ e \parallel \mu \}\ \mathsf{ok}} \tag{36.5a}$$

$$\frac{\forall a \sim \tau \in \Sigma \quad \mu(a) = e \neq \bullet \Longrightarrow \vdash_{\Sigma'} e : \tau}{\vdash_{\Sigma'} \mu : \Sigma} \tag{36.5b}$$

LPCF Type Safety

Theorem 36.1 (Preservation). *If* $\nu\ \Sigma\ \{\, e \parallel \mu \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \parallel \mu' \,\}$ *and* $\nu\ \Sigma\ \{\, e \parallel \mu \,\}$ *ok, then* $\nu\ \Sigma'\ \{\, e' \parallel \mu' \,\}$ *ok.*

Proof We prove by induction on rules (36.3) that if $\nu\ \Sigma\ \{\, e \parallel \mu \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \parallel \mu' \,\}$ and $\vdash_\Sigma \mu : \Sigma$ and $\vdash_\Sigma e : \tau$, then $\Sigma' \supseteq \Sigma$ and $\vdash_{\Sigma'} \mu' : \Sigma'$ and $\vdash_{\Sigma'} e' : \tau$.

Consider rule (36.3b), for which we have $e = e' = @\,a$, $\mu = \mu_0 \otimes a \hookrightarrow e_0$, $\mu' = \mu_0' \otimes a \hookrightarrow e_0'$, and

$$\nu\ \Sigma, a \sim \tau\ \{\, e_0 \parallel \mu_0 \otimes a \hookrightarrow \bullet \,\} \longmapsto \nu\ \Sigma', a \sim \tau\ \{\, e_0' \parallel \mu_0' \otimes a \hookrightarrow \bullet \,\}.$$

Assume that $\vdash_{\Sigma, a \sim \tau} \mu : \Sigma, a \sim \tau$. It follows that $\vdash_{\Sigma, a \sim \tau} e_0 : \tau$ and $\vdash_{\Sigma, a \sim \tau} \mu_0 : \Sigma$, and hence that

$$\vdash_{\Sigma, a \sim \tau} \mu_0 \otimes a \hookrightarrow \bullet : \Sigma, a \sim \tau.$$

We have by induction that $\Sigma' \supseteq \Sigma$ and $\vdash_{\Sigma', a \sim \tau} e_0' : \tau'$ and

$$\vdash_{\Sigma', a \sim \tau} \mu_0 \otimes a \hookrightarrow \bullet : \Sigma, a \sim \tau.$$

But then

$$\vdash_{\Sigma', a \sim \tau} \mu' : \Sigma', a \sim \tau,$$

which suffices for the result.

Consider rule (36.3g), so that e is the application $e_1(e_2)$ and

$$\nu\ \Sigma\ \{\, e_1 \parallel \mu \,\} \longmapsto \nu\ \Sigma'\ \{\, e_1' \parallel \mu' \,\}.$$

Suppose that $\vdash_\Sigma \mu : \Sigma$ and $\vdash_\Sigma e : \tau$. By inversion of typing $\vdash_\Sigma e_1 : \tau_2 \rightharpoonup \tau$ for some type τ_2 such that $\vdash_\Sigma e_2 : \tau_2$. By induction $\Sigma' \supseteq \Sigma$ and $\vdash_{\Sigma'} \mu' : \Sigma'$ and $\vdash_{\Sigma'} e_1' : \tau_2 \rightharpoonup \tau$. By weakening we have $\vdash_{\Sigma'} e_2 : \tau_2$, so that $\vdash_{\Sigma'} e_1'(e_2) : \tau$, which is enough for the result. □

LPCF Type Safety

The statement of the progress theorem allows for the occurrence of a black hole, representing a checkable form of non-termination. The judgment $\nu\ \Sigma\ \{\ e \parallel \mu\ \}\ \mathsf{loops}$, stating that e diverges by virtue of encountering the black hole, is defined by the following rules:

$$\frac{}{\nu\ \Sigma, a \sim \tau\ \{\ @\,a \parallel \mu \otimes a \hookrightarrow \bullet\ \}\ \mathsf{loops}} \tag{36.6a}$$

$$\frac{\nu\ \Sigma, a \sim \tau\ \{\ e \parallel \mu \otimes a \hookrightarrow \bullet\ \}\ \mathsf{loops}}{\nu\ \Sigma, a \sim \tau\ \{\ @\,a \parallel \mu \otimes a \hookrightarrow e\ \}\ \mathsf{loops}} \tag{36.6b}$$

$$\frac{\nu\ \Sigma\ \{\ e \parallel \mu\ \}\ \mathsf{loops}}{\nu\ \Sigma\ \{\ \mathtt{ifz}\,e\,\{\mathtt{z} \hookrightarrow e_0 \mid \mathtt{s}(x) \hookrightarrow e_1\} \parallel \mu\ \}\ \mathsf{loops}} \tag{36.6c}$$

$$\frac{\nu\ \Sigma\ \{\ e_1 \parallel \mu\ \}\ \mathsf{loops}}{\nu\ \Sigma\ \{\ e_1(e_2) \parallel \mu\ \}\ \mathsf{loops}} \tag{36.6d}$$

LPCF Type Safety

Theorem 36.2 (Progress). *If $\nu\,\Sigma\,\{\,e \parallel \mu\,\}$ ok, then either $\nu\,\Sigma\,\{\,e \parallel \mu\,\}$ final, or $\nu\,\Sigma\,\{\,e \parallel \mu\,\}$ loops, or there exists μ' and e' such that $\nu\,\Sigma\,\{\,e \parallel \mu\,\} \longmapsto \nu\,\Sigma'\,\{\,e' \parallel \mu'\,\}$.*

Proof We proceed by induction on the derivations of $\vdash_\Sigma e : \tau$ and $\vdash_\Sigma \mu : \Sigma$ implicit in the derivation of $\nu\,\Sigma\,\{\,e \parallel \mu\,\}$ ok.

Consider rule (19.1a), where the symbol a is declared in Σ. Thus, $\Sigma = \Sigma_0, a \sim \tau$ and $\vdash_\Sigma \mu : \Sigma$. It follows that $\mu = \mu_0 \otimes a \hookrightarrow e_0$ with $\vdash_\Sigma \mu_0 : \Sigma_0$ and $\vdash_\Sigma e_0 : \tau$. Note that $\vdash_\Sigma \mu_0 \otimes a \hookrightarrow \bullet : \Sigma$. Applying induction to the derivation of $\vdash_\Sigma e_0 : \tau$, we consider three cases:

1. $\nu\,\Sigma\,\{\,e_0 \parallel \mu \otimes a \hookrightarrow \bullet\,\}$ final. By inversion of rule (36.2b) we have e_0 val_Σ, and hence by rule (36.3a) we obtain $\nu\,\Sigma\,\{\,@\,a \parallel \mu\,\} \longmapsto \nu\,\Sigma\,\{\,e_0 \parallel \mu\,\}$.
2. $\nu\,\Sigma\,\{\,e_0 \parallel \mu_0 \otimes a \hookrightarrow \bullet\,\}$ loops. By applying rule (36.6b) we obtain $\nu\,\Sigma\,\{\,@\,a \parallel \mu\,\}$ loops.
3. $\nu\,\Sigma\,\{\,e_0 \parallel \mu_0 \otimes a \hookrightarrow \bullet\,\} \longmapsto \nu\,\Sigma'\,\{\,e_0' \parallel \mu_0' \otimes a \hookrightarrow \bullet\,\}$. By applying rule (36.3b) we obtain

$$\nu\,\Sigma\,\{\,@\,a \parallel \mu \otimes a \hookrightarrow e_0\,\} \longmapsto \nu\,\Sigma'\,\{\,@\,a \parallel \mu' \otimes a \hookrightarrow e_0'\,\}.$$

□

LFPC = FPC By-Need

The language **LFPC** is **FPC** but with a by-need dynamics. For example, the dynamics of product types in **LFPC** is given by the following rules:

$$\frac{}{\langle @\, a_1,\, @\, a_2 \rangle\ \mathsf{val}_{\Sigma, a_1 \sim \tau_1, a_2 \sim \tau_2}} \tag{36.7a}$$

$$\left\{ \begin{array}{c} \nu\, \Sigma\, \{\, \langle e_1, e_2 \rangle \parallel \mu \,\} \\ \longmapsto \\ \nu\, \Sigma, a_1 \sim \tau_1, a_2 \sim \tau_2\, \{\, \langle @\, a_1, @\, a_2 \rangle \parallel \mu \otimes a_1 \hookrightarrow e_1 \otimes a_2 \hookrightarrow e_2 \,\} \end{array} \right\} \tag{36.7b}$$

$$\frac{\nu\, \Sigma\, \{\, e \parallel \mu \,\} \longmapsto \nu\, \Sigma'\, \{\, e' \parallel \mu' \,\}}{\nu\, \Sigma\, \{\, e \cdot \mathtt{l} \parallel \mu \,\} \longmapsto \nu\, \Sigma'\, \{\, e' \cdot \mathtt{l} \parallel \mu' \,\}} \tag{36.7c}$$

LFPC = FPC By-Need

$$\frac{\nu\ \Sigma\ \{\, e \parallel \mu \,\}\ \mathsf{loops}}{\nu\ \Sigma\ \{\, e \cdot \mathtt{l} \parallel \mu \,\}\ \mathsf{loops}} \tag{36.7d}$$

$$\left\{\begin{array}{c} \nu\ \Sigma, a_1 \sim \tau_1, a_2 \sim \tau_2\ \{\, \langle @\, a_1, @\, a_2 \rangle \cdot \mathtt{l} \parallel \mu \,\} \\ \longmapsto \\ \nu\ \Sigma, a_1 \sim \tau_1, a_2 \sim \tau_2\ \{\, @\, a_1 \parallel \mu \,\} \end{array}\right\} \tag{36.7e}$$

$$\frac{\nu\ \Sigma\ \{\, e \parallel \mu \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \parallel \mu' \,\}}{\nu\ \Sigma\ \{\, e \cdot \mathtt{r} \parallel \mu \,\} \longmapsto \nu\ \Sigma'\ \{\, e' \cdot \mathtt{r} \parallel \mu' \,\}} \tag{36.7f}$$

$$\frac{\nu\ \Sigma\ \{\, e \parallel \mu \,\}\ \mathsf{loops}}{\nu\ \Sigma\ \{\, e \cdot \mathtt{r} \parallel \mu \,\}\ \mathsf{loops}} \tag{36.7g}$$

$$\left\{\begin{array}{c} \nu\ \Sigma, a_1 \sim \tau_1, a_2 \sim \tau_2\ \{\, \langle @\, a_1, @\, a_2 \rangle \cdot \mathtt{r} \parallel \mu \,\} \\ \longmapsto \\ \nu\ \Sigma, a_1 \sim \tau_1, a_2 \sim \tau_2\ \{\, @\, a_2 \parallel \mu \,\} \end{array}\right\} \tag{36.7h}$$

SFPC = FPC with Suspension Type

Informally, the type τ `susp` has as introduction form `susp` $x : \tau$ `is` e representing a suspended, self-referential, computation, e, of type τ. It has as elimination form the operation `force`(e) that evaluates the suspended computation presented by e, records the value in a memo table, and returns that value as result. Using suspension types, we can construct lazy types at will. For example, the type of lazy pairs with components of type τ_1 and τ_2 is expressible as the type

$$\tau_1 \,\texttt{susp} \times \tau_2 \,\texttt{susp}$$

and the type of by-need functions with domain τ_1 and range τ_2 is expressible as the type

$$\tau_1 \,\texttt{susp} \rightharpoonup \tau_2.$$

We may also express more complex combinations of eagerness and laziness, such as the type of "lazy lists" consisting of computations that, when forced, evaluate either to the empty list, or a non-empty list consisting of a natural number and another lazy list:

$$\texttt{rec}\, t \,\texttt{is}\, (\texttt{unit} + (\texttt{nat} \times t))\,\texttt{susp}.$$

Contrast this preceding type with this one:

$$\texttt{rec}\, t \,\texttt{is}\, (\texttt{unit} + (\texttt{nat} \times t\,\texttt{susp})).$$

SFPC Syntax & Statics

The language **SFPC** extends **FPC** with a type of suspensions:

Typ	τ	$::=$	$\mathtt{susp}(\tau)$	$\tau\ \mathtt{susp}$	suspension
Exp	e	$::=$	$\mathtt{susp}\{\tau\}(x.e)$	$\mathtt{susp}\, x : \tau\ \mathtt{is}\, e$	delay
			$\mathtt{force}(e)$	$\mathtt{force}(e)$	force
			$\mathtt{lcell}[a]$	$\mathtt{lcell}[a]$	indirection

The statics of **SFPC** is given using a judgment of the form $\Gamma \vdash_\Sigma e : \tau$, where Σ assigns types to the names of suspensions. It is defined by the following rules:

$$\frac{\Gamma, x : \mathtt{susp}(\tau) \vdash_\Sigma e : \tau}{\Gamma \vdash_\Sigma \mathtt{susp}\{\tau\}(x.e) : \mathtt{susp}(\tau)} \tag{36.8a}$$

$$\frac{\Gamma \vdash_\Sigma e : \mathtt{susp}(\tau)}{\Gamma \vdash_\Sigma \mathtt{force}(e) : \tau} \tag{36.8b}$$

$$\frac{}{\Gamma \vdash_{\Sigma, a \sim \tau} \mathtt{lcell}[a] : \mathtt{susp}(\tau)} \tag{36.8c}$$

SFPC Dynamics

The dynamics of **SFPC** is eager, with memoization confined to the suspension type as described by the following rules:

$$\frac{}{\texttt{lcell}[a]\ \mathsf{val}_{\Sigma, a \sim \tau}} \tag{36.9a}$$

$$\left\{ \begin{array}{c} \nu\ \Sigma\ \{\ \texttt{susp}\{\tau\}(x.e) \parallel \mu\ \} \\ \longmapsto \\ \nu\ \Sigma, a \sim \tau\ \{\ \texttt{lcell}[a] \parallel \mu \otimes a \hookrightarrow [\texttt{lcell}[a]/x]e\ \} \end{array} \right\} \tag{36.9b}$$

$$\frac{\nu\ \Sigma\ \{\ e \parallel \mu\ \} \longmapsto \nu\ \Sigma'\ \{\ e' \parallel \mu'\ \}}{\nu\ \Sigma\ \{\ \texttt{force}(e) \parallel \mu\ \} \longmapsto \nu\ \Sigma'\ \{\ \texttt{force}(e') \parallel \mu'\ \}} \tag{36.9c}$$

SFPC Dynamics

$$\frac{e\ \mathsf{val}_{\Sigma,a\sim\tau}}{\left\{\begin{array}{c}\nu\ \Sigma, a\sim\tau\,\{\,\texttt{force(lcell[}a\texttt{])}\parallel\mu\otimes a\hookrightarrow e\,\}\\ \longmapsto\\ \nu\ \Sigma, a\sim\tau\,\{\,e\parallel\mu\otimes a\hookrightarrow e\,\}\end{array}\right\}} \tag{36.9d}$$

$$\frac{\begin{array}{c}\nu\ \Sigma, a\sim\tau\,\{\,e\parallel\mu\otimes a\hookrightarrow\bullet\,\}\\ \longmapsto\\ \nu\ \Sigma', a\sim\tau\,\{\,e'\parallel\mu'\otimes a\hookrightarrow\bullet\,\}\end{array}}{\left\{\begin{array}{c}\nu\ \Sigma, a\sim\tau\,\{\,\texttt{force(lcell[}a\texttt{])}\parallel\mu\otimes a\hookrightarrow e\,\}\\ \longmapsto\\ \nu\ \Sigma', a\sim\tau\,\{\,\texttt{force(lcell[}a\texttt{])}\parallel\mu'\otimes a\hookrightarrow e'\,\}\end{array}\right\}} \tag{36.9e}$$

Nested Parallelism

PPCF extends PCF with *nested parallelism*.

Nested parallelism has a hierarchical structure arising from *forking* two (or more) parallel computations, then *joining* these computations to combine their results before proceeding.

Nested parallelism is also known as *fork-join parallelism*.

Two forms of dynamics for nested parallelism.

- **The first is a structural dynamics in which a single transition on a compound expression may involve multiple transitions on its constituent expressions.**
- **The second is a cost dynamics (introduced in Chapter 7) that focuses attention on the sequential and parallel complexity (also known as the *work* and the *depth*, or *span*) of a parallel program by associating a *series-parallel graph* with each computation.**

PPCF with Binary Fork-Join

The syntax of **PPCF** extends that of **PCF** with the following construct:

$$\text{Exp} \quad e \quad ::= \quad \texttt{par}(e_1; e_2; x_1.x_2.e) \quad \texttt{par}\, x_1 = e_1 \,\texttt{and}\, x_2 = e_2 \,\texttt{in}\, e \quad \text{parallel let}$$

The variables x_1 and x_2 are bound only within e, and not within e_1 or e_2, which ensures that they are not mutually dependent and hence can be evaluated simultaneously. The variable bindings represent a fork of two parallel computations e_1 and e_2, and the body e represents their join.

The static of **PPCF** enriches that of **PCF** with the following rule for parallel let:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e : \tau}{\Gamma \vdash \texttt{par}(e_1; e_2; x_1.x_2.e) : \tau} \tag{37.1}$$

PPCF Structural Dynamics

The *sequential structural dynamics* of **PPCF** is defined by a transition judgment of the form $e \mapsto_{\mathsf{seq}} e'$ defined by these rules:

$$\frac{e_1 \mapsto_{\mathsf{seq}} e_1'}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{seq}} \mathtt{par}(e_1'; e_2; x_1.x_2.e)} \tag{37.2a}$$

$$\frac{e_1 \ \mathsf{val} \quad e_2 \mapsto_{\mathsf{seq}} e_2'}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{seq}} \mathtt{par}(e_1; e_2'; x_1.x_2.e)} \tag{37.2b}$$

$$\frac{e_1 \ \mathsf{val} \quad e_2 \ \mathsf{val}}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{seq}} [e_1, e_2/x_1, x_2]e} \tag{37.2c}$$

PPCF Structural Dynamics

The *parallel structural dynamics* of **PPCF** is given by a transition judgment of the form $e \mapsto_{\mathsf{par}} e'$, defined as follows:

$$\frac{e_1 \mapsto_{\mathsf{par}} e_1' \quad e_2 \mapsto_{\mathsf{par}} e_2'}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{par}} \mathtt{par}(e_1'; e_2'; x_1.x_2.e)} \tag{37.3a}$$

$$\frac{e_1 \mapsto_{\mathsf{par}} e_1' \quad e_2\ \mathsf{val}}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{par}} \mathtt{par}(e_1'; e_2; x_1.x_2.e)} \tag{37.3b}$$

$$\frac{e_1\ \mathsf{val} \quad e_2 \mapsto_{\mathsf{par}} e_2'}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{par}} \mathtt{par}(e_1; e_2'; x_1.x_2.e)} \tag{37.3c}$$

$$\frac{e_1\ \mathsf{val} \quad e_2\ \mathsf{val}}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \mapsto_{\mathsf{par}} [e_1, e_2/x_1, x_2]e} \tag{37.3d}$$

PPCF Implicit Parallelism Theorem

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad [v_1, v_2/x_1, x_2]e \Downarrow v}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \Downarrow v} \tag{37.4}$$

Lemma 37.1. *For all v val, $e \mapsto^*_{seq} v$ if, and only if, $e \Downarrow v$.*

Proof It suffices to show that if $e \mapsto_{\mathsf{seq}} e'$ and $e' \Downarrow v$, then $e \Downarrow v$, and that if $e_1 \mapsto^*_{\mathsf{seq}} v_1$ and $e_2 \mapsto^*_{\mathsf{seq}} v_2$ and $[v_1, v_2/x_1, x_2]e \mapsto^*_{\mathsf{seq}} v$, then

$$\mathtt{par}\, x_1 = e_1 \,\mathtt{and}\, x_2 = e_2 \,\mathtt{in}\, e \mapsto^*_{\mathsf{seq}} v.$$

□

Lemma 37.2. *For all v val, $e \mapsto^*_{par} v$ if, and only if, $e \Downarrow v$.*

Proof It suffices to show that if $e \mapsto_{\mathsf{par}} e'$ and $e' \Downarrow v$, then $e \Downarrow v$, and that if $e_1 \mapsto^*_{\mathsf{par}} v_1$ and $e_2 \mapsto^*_{\mathsf{par}} v_2$ and $[v_1, v_2/x_1, x_2]e \mapsto^*_{\mathsf{par}} v$, then

$$\mathtt{par}\, x_1 = e_1 \,\mathtt{and}\, x_2 = e_2 \,\mathtt{in}\, e \mapsto^*_{\mathsf{par}} v.$$

The proof of the first is by induction on the parallel dynamics. The proof of the second proceeds by simultaneous induction on the derivations of $e_1 \mapsto^*_{\mathsf{par}} v_1$ and $e_2 \mapsto^*_{\mathsf{par}} v_2$. If $e_1 = v_1$ with v_1 val and $e_2 = v_2$ with v_2 val, then the result follows immediately from the third premise. If $e_2 = v_2$ but $e_1 \mapsto_{\mathsf{par}} e_1' \mapsto^*_{\mathsf{par}} v_1$, then by induction we have that $\mathtt{par}\, x_1 = e_1' \,\mathtt{and}\, x_2 = v_2 \,\mathtt{in}\, e \mapsto^*_{\mathsf{par}} v$, and hence the result follows by an application of rule (37.3b). The symmetric case follows similarly by an application of rule (37.3c), and in the case that both e_1 and e_2 transition, the result follows by induction and rule (37.3a). □

PPCF Cost Dynamics

In this section, we define a *parallel cost dynamics* that assigns a *cost graph* to the evaluation of a **PPCF** expression. Cost graphs are defined by the following grammar:

Cost	c	$::=$	
		$\mathbf{0}$	zero cost
		$\mathbf{1}$	unit cost
		$c_1 \otimes c_2$	parallel combination
		$c_1 \oplus c_2$	sequential combination

We associate with each cost graph two numeric measures, the *work*, $wk(c)$, and the *depth*, $dp(c)$. The work is defined by the following equations:

$$wk(c) = \begin{cases} 0 & \text{if } c = \mathbf{0} \\ 1 & \text{if } c = \mathbf{1} \\ wk(c_1) + wk(c_2) & \text{if } c = c_1 \otimes c_2 \\ wk(c_1) + wk(c_2) & \text{if } c = c_1 \oplus c_2 \end{cases} \tag{37.5}$$

The depth is defined by the following equations:

$$dp(c) = \begin{cases} 0 & \text{if } c = \mathbf{0} \\ 1 & \text{if } c = \mathbf{1} \\ \max(dp(c_1), dp(c_2)) & \text{if } c = c_1 \otimes c_2 \\ dp(c_1) + dp(c_2) & \text{if } c = c_1 \oplus c_2 \end{cases} \tag{37.6}$$

PPCF Cost Dynamics

The judgment $e \Downarrow^{c} v$, where e is a closed expression, v is a closed value, and c is a cost graph specifies the cost dynamics. By definition we arrange that $e \Downarrow^{\mathbf{0}} e$ when e val. The cost assignment for `let` is given by the following rule:

$$\frac{e_1 \Downarrow^{c_1} v_1 \quad e_2 \Downarrow^{c_2} v_2 \quad [v_1, v_2/x_1, x_2]e \Downarrow^{c} v}{\mathtt{par}(e_1; e_2; x_1.x_2.e) \Downarrow^{(c_1 \otimes c_2) \oplus \mathbf{1} \oplus c} v} \tag{37.7}$$

Theorem 37.6. *If $e \Downarrow^{c} v$, then $e \mapsto^{w}_{seq} v$ and $e \mapsto^{d}_{par} v$, where $w = wk(c)$ and $d = dp(c)$. Conversely, if $e \mapsto^{w}_{seq} v$, then there exists c such that $e \Downarrow^{c} v$ with $wk(c) = w$, and if $e \mapsto^{d}_{par} v'$, then there exists c' such that $e \Downarrow^{c'} v'$ with $dp(c') = d$.*

Multiple Fork-Join (Syntax)

We will consider here a simple language of sequence operations to illustrate the main ideas.

Typ	τ	::=	$\texttt{seq}(\tau)$	$\tau\ \texttt{seq}$	sequence
Exp	e	::=	$\texttt{seq}(e_0, \ldots, e_{n-1})$	$[e_0, \ldots, e_{n-1}]$	sequence
			$\texttt{len}(e)$	$\lvert e \rvert$	size
			$\texttt{sub}(e_1; e_2)$	$e_1[e_2]$	element
			$\texttt{tab}(x.e_1; e_2)$	$\texttt{tab}(x.e_1; e_2)$	tabulate
			$\texttt{map}(x.e_1; e_2)$	$[e_1 \mid x \in e_2]$	map
			$\texttt{cat}(e_1; e_2)$	$\texttt{cat}(e_1; e_2)$	concatenate

Multiple Fork-Join (Statics)

The statics of these operations is given by the following typing rules:

$$\frac{\Gamma \vdash e_0 : \tau \quad \ldots \quad \Gamma \vdash e_{n-1} : \tau}{\Gamma \vdash \mathtt{seq}(e_0, \ldots, e_{n-1}) : \mathtt{seq}(\tau)} \tag{37.8a}$$

$$\frac{\Gamma \vdash e : \mathtt{seq}(\tau)}{\Gamma \vdash \mathtt{len}(e) : \mathtt{nat}} \tag{37.8b}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{seq}(\tau) \quad \Gamma \vdash e_2 : \mathtt{nat}}{\Gamma \vdash \mathtt{sub}(e_1; e_2) : \tau} \tag{37.8c}$$

$$\frac{\Gamma, x : \mathtt{nat} \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \mathtt{nat}}{\Gamma \vdash \mathtt{tab}(x.e_1; e_2) : \mathtt{seq}(\tau)} \tag{37.8d}$$

$$\frac{\Gamma \vdash e_2 : \mathtt{seq}(\tau) \quad \Gamma, x : \tau \vdash e_1 : \tau'}{\Gamma \vdash \mathtt{map}(x.e_1; e_2) : \mathtt{seq}(\tau')} \tag{37.8e}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{seq}(\tau) \quad \Gamma \vdash e_2 : \mathtt{seq}(\tau)}{\Gamma \vdash \mathtt{cat}(e_1; e_2) : \mathtt{seq}(\tau)} \tag{37.8f}$$

Multiple Fork-Join (Cost Dynamics)

The cost dynamics of these constructs is defined by the following rules:

$$\frac{e_0 \Downarrow^{c_0} v_0 \quad \ldots \quad e_{n-1} \Downarrow^{c_{n-1}} v_{n-1}}{\texttt{seq}(e_0, \ldots, e_{n-1}) \Downarrow^{\bigotimes_{i=0}^{n-1} c_i} \texttt{seq}(v_0, \ldots, v_{n-1})} \tag{37.9a}$$

$$\frac{e \Downarrow^{c} \texttt{seq}(v_0, \ldots, v_{n-1})}{\texttt{len}(e) \Downarrow^{c \oplus \mathbf{1}} \texttt{num}[n]} \tag{37.9b}$$

$$\frac{e_1 \Downarrow^{c_1} \texttt{seq}(v_0, \ldots, v_{n-1}) \quad e_2 \Downarrow^{c_2} \texttt{num}[i] \quad (0 \leq i < n)}{\texttt{sub}(e_1; e_2) \Downarrow^{c_1 \oplus c_2 \oplus \mathbf{1}} v_i} \tag{37.9c}$$

$$\frac{e_2 \Downarrow^{c} \texttt{num}[n] \quad [\texttt{num}[0]/x]e_1 \Downarrow^{c_0} v_0 \quad \ldots \quad [\texttt{num}[n-1]/x]e_1 \Downarrow^{c_{n-1}} v_{n-1}}{\texttt{tab}(x.e_1; e_2) \Downarrow^{c \oplus \bigotimes_{i=0}^{n-1} c_i} \texttt{seq}(v_0, \ldots, v_{n-1})} \tag{37.9d}$$

$$\frac{\begin{array}{c} e_2 \Downarrow^{c} \texttt{seq}(v_0, \ldots, v_{n-1}) \\ [v_0/x]e_1 \Downarrow^{c_0} v'_0 \quad \ldots \quad [v_{n-1}/x]e_1 \Downarrow^{c_{n-1}} v'_{n-1} \end{array}}{\texttt{map}(x.e_1; e_2) \Downarrow^{c \oplus \bigotimes_{i=0}^{n-1} c_i} \texttt{seq}(v'_0, \ldots, v'_{n-1})} \tag{37.9e}$$

$$\frac{e_1 \Downarrow^{c_1} \texttt{seq}(v_0, \ldots, v_{m-1}) \quad e_2 \Downarrow^{c_2} \texttt{seq}(v'_0, \ldots, v'_{n-1})}{\texttt{cat}(e_1; e_2) \Downarrow^{c_1 \oplus c_2 \oplus \bigotimes_{i=0}^{m+n} \mathbf{1}} \texttt{seq}(v_0, \ldots, v_{m-1}, v'_0, \ldots, v'_{n-1})} \tag{37.9f}$$

Bounded Implementation

Building a bounded implementation of parallelism involves two major tasks.

- **Show that the primitives of the language can be implemented efficiently on the abstract machine model.**
- **Show how to schedule the workload across the processors to minimize execution time by maximizing parallelism.**

We aim to give an asymptotic bound on the time complexity of the implementation that relates the abstract cost of the computation to cost of implementing the workload on a p-way multiprocessor. This leads to *Brent's Theorem*.

Theorem 37.8. *If $e \Downarrow^c v$ with $wk(c) = w$ and $dp(c) = d$, then e can be evaluated on a p-processor SMP in time $O(\max(w/p, d))$.*

The theorem tells us that we can never execute a program in fewer steps than its depth d and that, at best, we can divide the work up evenly into w/p rounds of execution by the p processors. Note that if $p = 1$ then the theorem establishes an upper bound of $O(w)$ steps, the sequential complexity of the computation. Moreover, if the work is proportional to the depth, then we are unable to exploit parallelism, and the overall time is proportional to the work alone.

P Machine

The *global state* of the **P** machine is a configuration of the form $\nu\, \Sigma\, \{\, \mu \,\}$, where Σ is degenerated to just a finite set of (pairwise distinct) *task names* and μ is a finite mapping the task names in Σ to *local states*, representing the state of an individual task. A *local state* is either a closed **PCF** expression, or one of two special *join points* that implement the sequential and parallel dependencies of a task on one or two ancestors, respectively.[2] Thus, when expanded out, a global state has the form

$$\nu\, a_1, \ldots, a_n\, \{\, a_1 \hookrightarrow s_1 \otimes \ldots \otimes a_n \hookrightarrow s_n \,\},$$

where $n \geq 1$, and each s_i is a local state. The ordering of the tasks in a state, like the order of declarations in the signature, is not significant.

A **P** machine state transition has the form $\nu\, \Sigma\, \{\, \mu \,\} \longmapsto \nu\, \Sigma'\, \{\, \mu' \,\}$. There are two forms of such transitions, the *global* and the *local*. A global step selects as many tasks as are available, up to a pre-specified parameter $p > 0$, which represents the number of processors available at each round.

The local transitions of the **P** machine corresponding to the steps of **PCF** itself are illustrated by the following example rules for application; the others follow a similar pattern.[3]

P Machine Local Transitions

$$\frac{\neg(e_1\ \mathsf{val})}{\nu\, a\, \{\, a \hookrightarrow e_1(e_2)\,\} \longmapsto_{loc} \nu\, a\, a_1\, \{\, a \hookrightarrow \mathtt{join}[a_1](x_1.x_1(e_2)) \otimes a_1 \hookrightarrow e_1\,\}} \quad (37.10a)$$

$$\frac{e_1\ \mathsf{val}}{\nu\, a\, \{\, a \hookrightarrow e_1(e_2)\,\} \longmapsto_{loc} \nu\, a\, a_2\, \{\, a \hookrightarrow \mathtt{join}[a_2](x_2.e_1(x_2)) \otimes a_2 \hookrightarrow e_2\,\}} \quad (37.10b)$$

$$\frac{e_1\ \mathsf{val} \quad e_2\ \mathsf{val}}{\nu\, a_1\, a_2\, \{\, a_1 \hookrightarrow \mathtt{join}[a_2](x_2.e_1(x_2)) \otimes a_2 \hookrightarrow e_2\,\} \longmapsto_{loc} \nu\, a_1\, \{\, a_1 \hookrightarrow [e_2/x_2]e_1\,\}} \quad (37.10c)$$

$$\frac{e_2\ \mathsf{val}}{\nu\, a\, \{\, a \hookrightarrow (\lambda\,(x : \tau_2)\, e)(e_2)\,\} \longmapsto_{loc} \nu\, a\, \{\, a \hookrightarrow [e_2/x]e\,\}} \quad (37.10d)$$

P Machine Local Transitions

$$\left\{\begin{array}{c} \nu\, a\, \{\, a \hookrightarrow \mathtt{par}(e_1; e_2; x_1.x_2.e)\,\} \\ \longmapsto_{loc} \\ \nu\, a_1, a_2, a\, \{\, a_1 \hookrightarrow e_1 \otimes a_2 \hookrightarrow e_2 \otimes a \hookrightarrow \mathtt{join}[a_1; a_2](x_1; x_2.e)\,\} \end{array}\right\} \quad e_1\ \mathsf{val} \quad e_2\ \mathsf{val} \tag{37.11a}$$

$$\left\{\begin{array}{c} \nu\, a_1, a_2, a\, \{\, a_1 \hookrightarrow e_1 \otimes a_2 \hookrightarrow e_2 \otimes a \hookrightarrow \mathtt{join}[a_1; a_2](x_1; x_2.e)\,\} \\ \longmapsto_{loc} \\ \nu\, a\, \{\, a \hookrightarrow [e_1, e_2/x_1, x_2]e\,\} \end{array}\right\} \tag{37.11b}$$

P Machine Global Transitions

Each global transition is the simultaneous execution of one step of computation on as many as $p \geq 1$ processors.

$$\frac{\begin{array}{c} \nu\ \Sigma_1\ a_1\ \{\ \mu_1 \otimes a_1 \hookrightarrow s_1\ \} \longmapsto_{loc} \nu\ \Sigma'_1\ a_1\ \{\ \mu'_1 \otimes a_1 \hookrightarrow s'_1\ \} \\ \ldots \\ \nu\ \Sigma_n\ a_n\ \{\ \mu_n \otimes a_n \hookrightarrow s_n\ \} \longmapsto_{loc} \nu\ \Sigma'_n\ a_n\ \{\ \mu'_n \otimes a_n \hookrightarrow s'_n\ \} \end{array}}{\left\{\begin{array}{l} \nu\ \Sigma_0\ \Sigma_1\ a_1\ \ldots\ \Sigma_n\ a_n\ \{\ \mu_0 \otimes \mu_1 \otimes a_1 \hookrightarrow s_1 \otimes \ldots \otimes \mu_n \otimes a_n \hookrightarrow s_n\ \} \\ \qquad\qquad \longmapsto_{glo} \\ \nu\ \Sigma_0\ \Sigma'_1\ a_1\ \ldots\ \Sigma'_n\ a_n\ \{\ \mu_0 \otimes \mu'_1 \otimes a_1 \hookrightarrow s'_1 \otimes \ldots \otimes \mu'_n \otimes a_n \hookrightarrow s'_n\ \} \end{array}\right\}} \tag{37.12}$$

Futures and Speculations

A *future* is a computation that is performed before it is value is needed.

- **Like a suspension, a future represents a value that is to be determined later.**
- **Unlike a suspension, a future is always evaluated, regardless of whether its value is required.**
- **In a sequential setting, futures are of little interest; a future of type τ is just an expression of type τ . In a parallel setting, however, futures are of interest because they provide a means of initiating a parallel computation whose result is not needed until later, by which time it will have been completed.**

A *speculation* is a delayed computation whose result might be needed for the overall computation to finish. The dynamics for speculations executes suspended computations in parallel with the main thread of computation, *regardless whether the value of the speculation is needed by the main thread*. If the value of the speculation is needed, then such a dynamics pays off, but if not, the effort to compute it is wasted.

Futures: Syntax & Statics

The syntax of futures is given by the following grammar:

Typ	τ	$::=$	$\texttt{fut}(\tau)$	$\tau\,\texttt{fut}$	future
Exp	e	$::=$	$\texttt{fut}(e)$	$\texttt{fut}(e)$	future
			$\texttt{fsyn}(e)$	$\texttt{fsyn}(e)$	synchronize
			$\texttt{fcell}[a]$	$\texttt{fcell}[a]$	indirection

The statics of futures is given by the following rules:

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{fut}(e) : \texttt{fut}(\tau)} \tag{38.1a}$$

$$\frac{\Gamma \vdash e : \texttt{fut}(\tau)}{\Gamma \vdash \texttt{fsyn}(e) : \tau} \tag{38.1b}$$

Futures: Sequential Dynamics

The sequential dynamics of futures is easily defined. Futures are evaluated eagerly; synchronization returns the value of the future.

$$\frac{e\ \mathsf{val}}{\mathtt{fut}(e)\ \mathsf{val}} \tag{38.2a}$$

$$\frac{e \longmapsto e'}{\mathtt{fut}(e) \longmapsto \mathtt{fut}(e')} \tag{38.2b}$$

$$\frac{e \longmapsto e'}{\mathtt{fsyn}(e) \longmapsto \mathtt{fsyn}(e')} \tag{38.2c}$$

$$\frac{e\ \mathsf{val}}{\mathtt{fsyn}(\mathtt{fut}(e)) \longmapsto e} \tag{38.2d}$$

Under a sequential dynamics futures have little purpose: they introduce a pointless level of indirection.

Speculations: Syntax & Statics

The syntax of (non-recursive) speculations is given by the following grammar:

Typ	τ	$::=$	$\texttt{spec}(\tau)$	$\tau\ \texttt{spec}$	speculation
Exp	e	$::=$	$\texttt{spec}(e)$	$\texttt{spec}(e)$	speculate
			$\texttt{ssyn}(e)$	$\texttt{ssyn}(e)$	synchronize
			$\texttt{scell}[a]$	$\texttt{scell}[a]$	indirection

The statics of speculations is given by the following rules:

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{spec}(e) : \texttt{spec}(\tau)} \tag{38.3a}$$

$$\frac{\Gamma \vdash e : \texttt{spec}(\tau)}{\Gamma \vdash \texttt{ssyn}(e) : \tau} \tag{38.3b}$$

Speculations: Sequential Dynamics

The definition of the sequential dynamics of speculations is like that of futures, except that speculations are values.

$$\frac{}{\mathsf{spec}(e)\ \mathsf{val}} \tag{38.4a}$$

$$\frac{e \longmapsto e'}{\mathtt{ssyn}(e) \longmapsto \mathtt{ssyn}(e')} \tag{38.4b}$$

$$\frac{}{\mathtt{ssyn}(\mathtt{spec}(e)) \longmapsto e} \tag{38.4c}$$

Under a sequential dynamics speculations are simply a re-formulation of suspensions.

Futures: Parallel Dynamics

The parallel dynamics of futures relies on a modest extension to the language given in Section 38.1 to introduce *names* for tasks. Let Σ be a finite mapping assigning types to names. As mentioned earlier, the expression $\texttt{fcell}[a]$ is a value referring to the outcome of task a. The statics of this expression is given by the following rule:[2]

$$\frac{}{\Gamma \vdash_{\Sigma, a \sim \tau} \texttt{fcell}[a] : \texttt{fut}(\tau)} \tag{38.5}$$

Rules (38.1) carry over in the obvious way with Σ recording the types of the task names.

States of the parallel dynamics have the form $\nu\, \Sigma\, \{\, e \parallel \mu \,\}$, where e is the *focus* of evaluation, and μ records the active parallel futures (or speculations). Formally, μ is a finite mapping assigning expressions to the task names declared in Σ. A state is well-formed according to the following rule:

$$\frac{\vdash_\Sigma e : \tau \quad (\forall a \in dom(\Sigma))\ \vdash_\Sigma \mu(a) : \Sigma(a)}{\nu\, \Sigma\, \{\, e \parallel \mu \,\}\ \mathsf{ok}} \tag{38.6}$$

Futures: Parallel Dynamics

The parallel dynamics is divided into two phases, the *local* phase, which defines the basic steps of evaluation of an expression, and the *global* phase, which executes all possible local steps in parallel. The local dynamics of futures is defined by the following rules:[3]

$$\frac{}{\texttt{fcell}[a]\ \mathsf{val}_{\Sigma,a\sim\tau}} \tag{38.7a}$$

$$\frac{}{\nu\ \Sigma\ \{\ \texttt{fut}(e)\ \|\ \mu\ \} \longmapsto_{loc} \nu\ \Sigma, a\sim\tau\ \{\ \texttt{fcell}[a]\ \|\ \mu \otimes a \hookrightarrow e\ \}} \tag{38.7b}$$

$$\frac{\nu\ \Sigma\ \{\ e\ \|\ \mu\ \} \longmapsto_{loc} \nu\ \Sigma'\ \{\ e'\ \|\ \mu'\ \}}{\nu\ \Sigma\ \{\ \texttt{fsyn}(e)\ \|\ \mu\ \} \longmapsto_{loc} \nu\ \Sigma'\ \{\ \texttt{fsyn}(e')\ \|\ \mu'\ \}} \tag{38.7c}$$

$$\frac{e'\ \mathsf{val}_{\Sigma,a\sim\tau}}{\left\{\begin{array}{c}\nu\ \Sigma, a\sim\tau\ \{\ \texttt{fsyn}(\texttt{fcell}[a])\ \|\ \mu\otimes a\hookrightarrow e'\ \} \\ \longmapsto_{loc} \\ \nu\ \Sigma, a\sim\tau\ \{\ e'\ \|\ \mu\otimes a\hookrightarrow e'\ \}\end{array}\right\}} \tag{38.7d}$$

Rule (38.7b) activates a future named a executing the expression e and returns a reference to it. Rule (38.7d) synchronizes with a future whose value has been determined. Note that a local transition always has the form

$$\nu\ \Sigma\ \{\ e\ \|\ \mu\ \} \longmapsto_{loc} \nu\ \Sigma\ \Sigma'\ \{\ e'\ \|\ \mu\otimes\mu'\ \}$$

where Σ' is either empty or declares the type of a single symbol, and μ' is either empty or of the form $a \hookrightarrow e'$ for some expression e'.

Futures: Parallel Dynamics

A global step of the parallel dynamics consists of at most one local step for the focal expression and one local step for each of up to p futures, where $p > 0$ is a fixed parameter representing the number of processors.

$$\frac{\begin{array}{c}\mu = \mu_0 \otimes a_1 \hookrightarrow e_1 \otimes \ldots \otimes a_n \hookrightarrow e_n \\ \mu'' = \mu_0 \otimes a_1 \hookrightarrow e'_1 \otimes \ldots \otimes a_n \hookrightarrow e'_n \\ \nu\ \Sigma\ \{\, e \parallel \mu \,\} \longmapsto^{\mathbf{0,1}}_{loc} \nu\ \Sigma\ \Sigma'\ \{\, e' \parallel \mu \otimes \mu' \,\} \\ (\forall 1 \leq i \leq n \leq p) \quad \nu\ \Sigma\ \{\, e_i \parallel \mu \,\} \longmapsto_{loc} \nu\ \Sigma\ \Sigma'_i\ \{\, e'_i \parallel \mu \otimes \mu'_i \,\}\end{array}}{\left\{\begin{array}{c}\nu\ \Sigma\ \{\, e \parallel \mu \,\} \\ \longmapsto_{glo} \\ \nu\ \Sigma\ \Sigma'\ \Sigma'_1\ \ldots\ \Sigma'_n\ \{\, e' \parallel \mu'' \otimes \mu' \otimes \mu'_1 \otimes \ldots \otimes \mu'_n \,\}\end{array}\right\}} \tag{38.8a}$$

Futures: Parallel Dynamics

The initial state of a computation, for futures or speculations, is defined by the rule

$$\frac{}{\nu\,\emptyset\,\{\,e \parallel \emptyset\,\}\ \mathsf{initial}} \tag{38.9}$$

For futures, a state is final only if the focus and all parallel futures have completed evaluation:

$$\frac{e\ \mathsf{val}_\Sigma \quad \mu\ \mathsf{val}_\Sigma}{\nu\,\Sigma\,\{\,e \parallel \mu\,\}\ \mathsf{final}} \tag{38.10a}$$

$$\frac{(\forall a \in dom(\Sigma))\ \mu(a)\ \mathsf{val}_\Sigma}{\mu\ \mathsf{val}_\Sigma} \tag{38.10b}$$

For speculations, a state is final only if the focus is a value, regardless of whether any other speculations have completed:

$$\frac{e\ \mathsf{val}_\Sigma}{\nu\,\Sigma\,\{\,e \parallel \mu\,\}\ \mathsf{final}} \tag{38.11}$$

All futures must terminate to ensure that the work performed in parallel matches that performed sequentially; no future is created whose value is not needed according to the sequential semantics. In contrast, speculations can be abandoned when their values are not needed.