5 Dynamics

The *dynamics* of a language describes how programs are executed. The most important way to define the dynamics of a language is by the method of *structural dynamics*, which defines a *transition system* that inductively specifies the step-by-step process of executing a program. Another method for presenting dynamics, called *contextual dynamics*, is a variation of structural dynamics in which the transition rules are specified in a slightly different way. An *equational dynamics* presents the dynamics of a language by a collection of rules defining when one program is *definitionally equivalent* to another.

5.1 Transition Systems

A *transition system* is specified by the following four forms of judgment:

1. s state, asserting that s is a *state* of the transition system.
2. s final, where s state, asserting that s is a *final* state.
3. s initial, where s state, asserting that s is an *initial* state.
4. $s \longmapsto s'$, where s state and s' state, asserting that state s may transition to state s'.

In practice, we always arrange things so that no transition is possible from a final state: if s final, then there is no s' state such that $s \longmapsto s'$. A state from which no transition is possible is *stuck*. Whereas all final states are, by convention, stuck, there may be stuck states in a transition system that are not final. A transition system is *deterministic* iff for every state s there exists at most one state s' such that $s \longmapsto s'$; otherwise, it is *non-deterministic*.

A *transition sequence* is a sequence of states $s_0, \ldots, s_n$ such that s_0 initial, and $s_i \longmapsto s_{i+1}$ for every $0 \leq i < n$. A transition sequence is *maximal* iff there is no s such that $s_n \longmapsto s$, and it is *complete* iff it is maximal and s_n final. Thus, every complete transition sequence is maximal, but maximal sequences are not necessarily complete. The judgment $s \downarrow$ means that there is a complete transition sequence starting from s, which is to say that there exists s' final such that $s \longmapsto^* s'$.

The *iteration* of transition judgment $s \longmapsto^* s'$ is inductively defined by the following rules:

$$\frac{}{s \longmapsto^* s} \tag{5.1a}$$

$$\frac{s \longmapsto s' \quad s' \longmapsto^* s''}{s \longmapsto^* s''} \tag{5.1b}$$

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

When applied to the definition of iterated transition, the principle of rule induction states that to show that $P(s, s')$ holds when $s \longmapsto^* s'$, it is enough to show these two properties of P:

1. $P(s, s)$.
2. if $s \longmapsto s'$ and $P(s', s'')$, then $P(s, s'')$.

The first requirement is to show that P is reflexive. The second is to show that P is *closed under head expansion*, or *closed under inverse evaluation*. Using this principle, it is easy to prove that $\longmapsto^*$ is reflexive and transitive.

The *n-times iterated* transition judgment $s \longmapsto^n s'$, where $n \geq 0$, is inductively defined by the following rules:

$$\frac{}{s \longmapsto^0 s} \tag{5.2a}$$

$$\frac{s \longmapsto s' \quad s' \longmapsto^n s''}{s \longmapsto^{n+1} s''} \tag{5.2b}$$

Theorem 5.1. *For all states s and s', $s \longmapsto^* s'$ iff $s \longmapsto^k s'$ for some $k \geq 0$.*

Proof From left to right, by induction on the definition of multi-step transition. From right to left, by mathematical induction on $k \geq 0$. □

5.2 Structural Dynamics

A *structural dynamics* for the language **E** is given by a transition system whose states are closed expressions. All states are initial. The final states are the *(closed) values*, which represent the completed computations. The judgment e val, which states that e is a value, is inductively defined by the following rules:

$$\frac{}{\mathtt{num}[n]\ \mathsf{val}} \tag{5.3a}$$

$$\frac{}{\mathtt{str}[s]\ \mathsf{val}} \tag{5.3b}$$

The transition judgment $e \longmapsto e'$ between states is inductively defined by the following rules:

$$\frac{n_1 + n_2 = n}{\mathtt{plus}(\mathtt{num}[n_1]; \mathtt{num}[n_2]) \longmapsto \mathtt{num}[n]} \tag{5.4a}$$

$$\frac{e_1 \longmapsto e_1'}{\mathtt{plus}(e_1; e_2) \longmapsto \mathtt{plus}(e_1'; e_2)} \tag{5.4b}$$

$$\frac{e_1\ \mathsf{val} \quad e_2 \longmapsto e_2'}{\mathtt{plus}(e_1; e_2) \longmapsto \mathtt{plus}(e_1; e_2')} \tag{5.4c}$$

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

$$\frac{s_1 \hat{} s_2 = s \text{ str}}{\texttt{cat}(\texttt{str}[s_1];\texttt{str}[s_2]) \longmapsto \texttt{str}[s]} \tag{5.4d}$$

$$\frac{e_1 \longmapsto e_1'}{\texttt{cat}(e_1;e_2) \longmapsto \texttt{cat}(e_1';e_2)} \tag{5.4e}$$

$$\frac{e_1 \text{ val} \quad e_2 \longmapsto e_2'}{\texttt{cat}(e_1;e_2) \longmapsto \texttt{cat}(e_1;e_2')} \tag{5.4f}$$

$$\left[\frac{e_1 \longmapsto e_1'}{\texttt{let}(e_1;x.e_2) \longmapsto \texttt{let}(e_1';x.e_2)}\right] \tag{5.4g}$$

$$\frac{[e_1 \text{ val}]}{\texttt{let}(e_1;x.e_2) \longmapsto [e_1/x]e_2} \tag{5.4h}$$

We have omitted rules for multiplication and computing the length of a string, which follow a similar pattern. Rules (5.4a), (5.4d), and (5.4h) are *instruction transitions*, because they correspond to the primitive steps of evaluation. The remaining rules are *search transitions* that determine the order of execution of instructions.

The bracketed rule (5.4g) and bracketed premise on rule (5.4h) are included for a *by-value* interpretation of `let` and omitted for a *by-name* interpretation. The by-value interpretation evaluates an expression before binding it to the defined variable, whereas the by-name interpretation binds it in unevaluated form. The by-value interpretation saves work if the defined variable is used more than once but wastes work if it is not used at all. Conversely, the by-name interpretation saves work if the defined variable is not used and wastes work if it is used more than once.

A derivation sequence in a structural dynamics has a two-dimensional structure, with the number of steps in the sequence being its "width" and the derivation tree for each step being its "height." For example, consider the following evaluation sequence:

$$\begin{aligned}
&\texttt{let}(\texttt{plus}(\texttt{num}[1];\texttt{num}[2]);x.\texttt{plus}(\texttt{plus}(x;\texttt{num}[3]);\texttt{num}[4])) \\
&\quad\longmapsto\quad \texttt{let}(\texttt{num}[3];x.\texttt{plus}(\texttt{plus}(x;\texttt{num}[3]);\texttt{num}[4])) \\
&\quad\longmapsto\quad \texttt{plus}(\texttt{plus}(\texttt{num}[3];\texttt{num}[3]);\texttt{num}[4]) \\
&\quad\longmapsto\quad \texttt{plus}(\texttt{num}[6];\texttt{num}[4]) \\
&\quad\longmapsto\quad \texttt{num}[10]
\end{aligned}$$

Each step in this sequence of transitions is justified by a derivation according to rules (5.4). For example, the third transition in the preceding example is justified by the following derivation:

$$\frac{\dfrac{}{\texttt{plus}(\texttt{num}[3];\texttt{num}[3]) \longmapsto \texttt{num}[6]}\ (5.4a)}{\texttt{plus}(\texttt{plus}(\texttt{num}[3];\texttt{num}[3]);\texttt{num}[4]) \longmapsto \texttt{plus}(\texttt{num}[6];\texttt{num}[4])}\ (5.4b)$$

The other steps are similarly justified by composing rules.

The principle of rule induction for the structural dynamics of **E** states that to show $\mathcal{P}(e \longmapsto e')$ when $e \longmapsto e'$, it is enough to show that $\mathcal{P}$ is closed under rules (5.4). For example, we may show by rule induction that the structural dynamics of **E** is *determinate*,

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

which means that an expression may transition to at most one other expression. The proof requires a simple lemma relating transition to values.

Lemma 5.2 (Finality of Values)**.** *For no expression e do we have both e val, and $e \longmapsto e'$ for some e'.*

Proof By rule induction on rules (5.3) and (5.4). □

Lemma 5.3 (Determinacy)**.** *If $e \longmapsto e'$ and $e \longmapsto e''$, then e' and e'' are α-equivalent.*

Proof By rule induction on the premises $e \longmapsto e'$ and $e \longmapsto e''$, carried out either simultaneously or in either order. The primitive operators, such as addition, are assumed to have a unique value when applied to values. □

Rules (5.4) exemplify the *inversion principle* of language design, which states that the elimination forms are *inverse* to the introduction forms of a language. The search rules determine the *principal arguments* of each elimination form, and the instruction rules specify how to evaluate an elimination form when all of its principal arguments are in introduction form. For example, rules (5.4) specify that both arguments of addition are principal and specify how to evaluate an addition once its principal arguments are evaluated to numerals. The inversion principle is central to ensuring that a programming language is properly defined, the exact statement of which is given in Chapter 6.

5.3 Contextual Dynamics

A variant of structural dynamics, called *contextual dynamics*, is sometimes useful. There is no fundamental difference between contextual and structural dynamics, but rather one of style. The main idea is to isolate instruction steps as a special form of judgment, called *instruction transition*, and to formalize the process of locating the next instruction using a device called an *evaluation context*. The judgment $e\ \mathsf{val}$ defining whether an expression is a value, remains unchanged.

The instruction transition judgment $e_1 \to e_2$ for $\mathbf{E}$ is defined by the following rules, together with similar rules for multiplication of numbers and the length of a string.

$$\frac{m + n\ \mathsf{is}\ p\ \mathsf{nat}}{\mathtt{plus}(\mathtt{num}[m];\mathtt{num}[n]) \to \mathtt{num}[p]} \tag{5.5a}$$

$$\frac{s \hat{} t = u\ \mathsf{str}}{\mathtt{cat}(\mathtt{str}[s];\mathtt{str}[t]) \to \mathtt{str}[u]} \tag{5.5b}$$

$$\frac{}{\mathtt{let}(e_1; x.e_2) \to [e_1/x]e_2} \tag{5.5c}$$

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

The judgment $\mathcal{E}$ ectxt determines the location of the next instruction to execute in a larger expression. The position of the next instruction step is specified by a "hole," written $\circ$, into which the next instruction is placed, as we shall detail shortly. (The rules for multiplication and length are omitted for concision, as they are handled similarly.)

$$\frac{}{\circ \text{ ectxt}} \tag{5.6a}$$

$$\frac{\mathcal{E}_1 \text{ ectxt}}{\texttt{plus}(\mathcal{E}_1; e_2) \text{ ectxt}} \tag{5.6b}$$

$$\frac{e_1 \text{ val} \quad \mathcal{E}_2 \text{ ectxt}}{\texttt{plus}(e_1; \mathcal{E}_2) \text{ ectxt}} \tag{5.6c}$$

The first rule for evaluation contexts specifies that the next instruction may occur "here," at the occurrence of the hole. The remaining rules correspond one-for-one to the search rules of the structural dynamics. For example, rule (5.6c) states that in an expression $\texttt{plus}(e_1; e_2)$, if the first argument, e_1, is a value, then the next instruction step, if any, lies at or within the second argument, e_2.

An evaluation context is a template that is instantiated by replacing the hole with an instruction to be executed. The judgment $e' = \mathcal{E}\{e\}$ states that the expression e' is the result of filling the hole in the evaluation context $\mathcal{E}$ with the expression e. It is inductively defined by the following rules:

$$\frac{}{e = \circ\{e\}} \tag{5.7a}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{plus}(e_1; e_2) = \texttt{plus}(\mathcal{E}_1; e_2)\{e\}} \tag{5.7b}$$

$$\frac{e_1 \text{ val} \quad e_2 = \mathcal{E}_2\{e\}}{\texttt{plus}(e_1; e_2) = \texttt{plus}(e_1; \mathcal{E}_2)\{e\}} \tag{5.7c}$$

There is one rule for each form of evaluation context. Filling the hole with e results in e; otherwise, we proceed inductively over the structure of the evaluation context.

Finally, the contextual dynamics for **E** is defined by a single rule:

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \to e_0' \quad e' = \mathcal{E}\{e_0'\}}{e \longmapsto e'} \tag{5.8}$$

Thus, a transition from e to e' consists of (1) decomposing e into an evaluation context and an instruction, (2) execution of that instruction, and (3) replacing the instruction by the result of its execution in the same spot within e to obtain e'.

The structural and contextual dynamics define the same transition relation. For the sake of the proof, let us write $e \longmapsto_{\mathsf{s}} e'$ for the transition relation defined by the structural dynamics (rules (5.4)), and $e \longmapsto_{\mathsf{c}} e'$ for the transition relation defined by the contextual dynamics (rules (5.8)).

Theorem 5.4. *$e \longmapsto_{\mathsf{s}} e'$ if, and only if, $e \longmapsto_{\mathsf{c}} e'$.*

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

Proof From left to right, proceed by rule induction on rules (5.4). It is enough in each case to exhibit an evaluation context $\mathcal{E}$ such that $e = \mathcal{E}\{e_0\}$, $e' = \mathcal{E}\{e_0'\}$, and $e_0 \to e_0'$. For example, for rule (5.4a), take $\mathcal{E} = \circ$, and note that $e \to e'$. For rule (5.4b), we have by induction that there exists an evaluation context $\mathcal{E}_1$ such that $e_1 = \mathcal{E}_1\{e_0\}$, $e_1' = \mathcal{E}_1\{e_0'\}$, and $e_0 \to e_0'$. Take $\mathcal{E} = \mathtt{plus}(\mathcal{E}_1; e_2)$, and note that $e = \mathtt{plus}(\mathcal{E}_1; e_2)\{e_0\}$ and $e' = \mathtt{plus}(\mathcal{E}_1; e_2)\{e_0'\}$ with $e_0 \to e_0'$.

From right to left, note that if $e \longmapsto_{\mathsf{c}} e'$, then there exists an evaluation context $\mathcal{E}$ such that $e = \mathcal{E}\{e_0\}$, $e' = \mathcal{E}\{e_0'\}$, and $e_0 \to e_0'$. We prove by induction on rules (5.7) that $e \longmapsto_{\mathsf{s}} e'$. For example, for rule (5.7a), e_0 is e, e_0' is e', and $e \to e'$. Hence, $e \longmapsto_{\mathsf{s}} e'$. For rule (5.7b), we have that $\mathcal{E} = \mathtt{plus}(\mathcal{E}_1; e_2)$, $e_1 = \mathcal{E}_1\{e_0\}$, $e_1' = \mathcal{E}_1\{e_0'\}$, and $e_1 \longmapsto_{\mathsf{s}} e_1'$. Therefore, e is $\mathtt{plus}(e_1; e_2)$, e' is $\mathtt{plus}(e_1'; e_2)$, and therefore by rule (5.4b), $e \longmapsto_{\mathsf{s}} e'$. □

Because the two transition judgments coincide, contextual dynamics can be considered an alternative presentation of a structural dynamics. It has two advantages over structural dynamics, one relatively superficial, one rather less so. The superficial advantage stems from writing rule (5.8) in the simpler form

$$\frac{e_0 \to e_0'}{\mathcal{E}\{e_0\} \longmapsto \mathcal{E}\{e_0'\}}\ . \tag{5.9}$$

This formulation is superficially simpler in that it does not make explicit how an expression is decomposed into an evaluation context and a reducible expression. The deeper advantage of contextual dynamics is that all transitions are between complete programs. One need never consider a transition between expressions of any type other than the observable type, which simplifies certain arguments, such as the proof of Lemma 47.16.

5.4 Equational Dynamics

Another formulation of the dynamics of a language regards computation as a form of equational deduction, much in the style of elementary algebra. For example, in algebra, we may show that the polynomials $x^2 + 2x + 1$ and $(x + 1)^2$ are equivalent by a simple process of calculation and re-organization using the familiar laws of addition and multiplication. The same laws are enough to determine the value of any polynomial, given the values of its variables. So, for example, we may plug in 2 for x in the polynomial $x^2 + 2x + 1$ and calculate that $2^2 + 2 \times 2 + 1 = 9$, which is indeed $(2 + 1)^2$. We thus obtain a model of computation in which the value of a polynomial for a given value of its variable is determined by substitution and simplification.

Very similar ideas give rise to the concept of *definitional*, or *computational*, *equivalence* of expressions in **E**, which we write as $\mathcal{X} \mid \Gamma \vdash e \equiv e' : \tau$, where Γ consists of one assumption of the form $x : \tau$ for each $x \in \mathcal{X}$. We only consider definitional equality of well-typed expressions, so that when considering the judgment $\Gamma \vdash e \equiv e' : \tau$, we tacitly

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

assume that $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$. Here, as usual, we omit explicit mention of the variables $\mathcal{X}$ when they can be determined from the forms of the assumptions Γ.

Definitional equality of expressions in **E** under the by-name interpretation of `let` is inductively defined by the following rules:

$$\frac{}{\Gamma \vdash e \equiv e : \tau} \tag{5.10a}$$

$$\frac{\Gamma \vdash e' \equiv e : \tau}{\Gamma \vdash e \equiv e' : \tau} \tag{5.10b}$$

$$\frac{\Gamma \vdash e \equiv e' : \tau \quad \Gamma \vdash e' \equiv e'' : \tau}{\Gamma \vdash e \equiv e'' : \tau} \tag{5.10c}$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \mathtt{num} \quad \Gamma \vdash e_2 \equiv e_2' : \mathtt{num}}{\Gamma \vdash \mathtt{plus}(e_1; e_2) \equiv \mathtt{plus}(e_1'; e_2') : \mathtt{num}} \tag{5.10d}$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \mathtt{str} \quad \Gamma \vdash e_2 \equiv e_2' : \mathtt{str}}{\Gamma \vdash \mathtt{cat}(e_1; e_2) \equiv \mathtt{cat}(e_1'; e_2') : \mathtt{str}} \tag{5.10e}$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 \equiv e_2' : \tau_2}{\Gamma \vdash \mathtt{let}(e_1; x.e_2) \equiv \mathtt{let}(e_1'; x.e_2') : \tau_2} \tag{5.10f}$$

$$\frac{n_1 + n_2 \text{ is } n \text{ nat}}{\Gamma \vdash \mathtt{plus}(\mathtt{num}[n_1]; \mathtt{num}[n_2]) \equiv \mathtt{num}[n] : \mathtt{num}} \tag{5.10g}$$

$$\frac{s_1 \hat{} s_2 = s \text{ str}}{\Gamma \vdash \mathtt{cat}(\mathtt{str}[s_1]; \mathtt{str}[s_2]) \equiv \mathtt{str}[s] : \mathtt{str}} \tag{5.10h}$$

$$\frac{}{\Gamma \vdash \mathtt{let}(e_1; x.e_2) \equiv [e_1/x]e_2 : \tau} \tag{5.10i}$$

Rules (5.10a) through (5.10c) state that definitional equality is an *equivalence relation*. Rules (5.10d) through (5.10f) state that it is a *congruence relation*, which means that it is compatible with all expression-forming constructs in the language. Rules (5.10g) through (5.10i) specify the meanings of the primitive constructs of **E**. We say that rules (5.10) define the *strongest congruence* closed under rules (5.10g), (5.10h), and (5.10i).

Rules (5.10) suffice to calculate the value of an expression by a deduction similar to that used in high school algebra. For example, we may derive the equation

$$\mathtt{let}\, x\, \mathtt{be}\, 1 + 2\, \mathtt{in}\, x + 3 + 4 \equiv 10 : \mathtt{num}$$

by applying rules (5.10). Here, as in general, there may be many different ways to derive the same equation, but we need find only one derivation in order to carry out an evaluation.

Definitional equality is rather weak in that many equivalences that we might intuitively think are true are not derivable from rules (5.10). A prototypical example is the putative equivalence

$$x_1 : \mathtt{num}, x_2 : \mathtt{num} \vdash x_1 + x_2 \equiv x_2 + x_1 : \mathtt{num}, \tag{5.11}$$

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

which, intuitively, expresses the commutativity of addition. Although we shall not prove this here, this equivalence is *not* derivable from rules (5.10). And yet we *may* derive all of its closed instances,

$$n_1 + n_2 \equiv n_2 + n_1 : \mathtt{num}, \tag{5.12}$$

where n_1 nat and n_2 nat are particular numbers.

The "gap" between a general law, such as Equation (5.11), and all of its instances, given by Equation (5.12), may be filled by enriching the notion of equivalence to include a principle of proof by mathematical induction. Such a notion of equivalence is sometimes called *semantic equivalence*, because it expresses relationships that hold by virtue of the dynamics of the expressions involved. (Semantic equivalence is developed rigorously for a related language in Chapter 46.)

Theorem 5.5. *For the expression language* **E**, *the relation* $e \equiv e' : \tau$ *holds iff there exists* e_0 val *such that* $e \longmapsto^* e_0$ *and* $e' \longmapsto^* e_0$.

Proof The proof from right to left is direct, because every transition step is a valid equation. The converse follows from the following, more general, proposition, which is proved by induction on rules (5.10): if $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e \equiv e' : \tau$, then when $e_1 : \tau_1, e_1' : \tau_1, \ldots, e_n : \tau_n, e_n' : \tau_n$, if for each $1 \leq i \leq n$ the expressions e_i and e_i' evaluate to a common value v_i, then there exists e_0 val such that

$$[e_1, \ldots, e_n/x_1, \ldots, x_n]e \longmapsto^* e_0$$

and

$$[e_1', \ldots, e_n'/x_1, \ldots, x_n]e' \longmapsto^* e_0.$$

□

5.5 Notes

The use of transition systems to specify the behavior of programs goes back to the early work of Church and Turing on computability. Turing's approach emphasized the concept of an abstract machine consisting of a finite program together with unbounded memory. Computation proceeds by changing the memory in accordance with the instructions in the program. Much early work on the operational semantics of programming languages, such as the SECD machine (Landin, 1965), emphasized machine models. Church's approach emphasized the language for expressing computations and defined execution in terms of the programs themselves, rather than in terms of auxiliary concepts such as memories or tapes. Plotkin's elegant formulation of structural operational semantics (Plotkin, 1981), which we use heavily throughout this book, was inspired by Church's and Landin's ideas (Plotkin, 2004). Contextual semantics, which was introduced by Felleisen and Hieb (1992), may be seen as an alternative formulation of structural semantics in which "search rules" are replaced by "context matching." Computation viewed as equational deduction goes back to the early work of Herbrand, Gödel, and Church.

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press

Exercises

5.1. Prove that if $s \longmapsto^* s'$ and $s' \longmapsto^* s''$, then $s \longmapsto^* s''$.

5.2. Complete the proof of Theorem 5.1 along the lines suggested there.

5.3. Complete the proof of Theorem 5.5 along the lines suggested there.

https://doi.org/10.1017/CBO9781316576892.007 Published online by Cambridge University Press