3.3 Small-Step Structural Operational Semantics (Small-Step SOS)

Known also under the names *transition semantics*, *reduction semantics*, *one-step operational semantics*, and *computational semantics*, small-step structural operational semantics, or small-step SOS for short, formally captures the intuitive notion of one atomic computational step. Unlike in big-step SOS where one defines all computation steps in one transition, in a small-step SOS definition a transition encodes only one step of computation. To distinguish small-step from big-step transitions, we use a plain arrow $\to$ instead of $\Downarrow$. To execute a small-step SOS definition, or to relate it to a big-step definition, we need to transitively close the small-step transition relation. Indeed, the conceptual relationship between big-step SOS and small-step SOS is that for any configuration C and any result configuration R, $C \Downarrow R$ if and only if $C \to^\star R$. Small-step semantics is typically preferred over big-step semantics when defining languages with a high-degree of non-determinism, such as, for example, concurrent languages, because in a small-step semantics one has a direct control over what can execute and when.

Like big-step SOS, a small-step SOS of a programming language or calculus is also given as a formal proof system (see Section 2.2). The *small-step SOS sequents* are also relations of configurations like in big-step SOS, but in small-step SOS they are written $C \to C'$ and have the meaning that C' is a configuration obtained from C after *one step* of computation. A *small-step SOS rule* therefore has the form

$$\frac{C_1 \to C_1' \qquad C_2 \to C_2' \qquad \ldots \qquad C_n \to C_n'}{C \to C'} \text{ [if } \textit{condition}\text{]}$$

where C, C', C_1, C_1', C_2, C_2', $\ldots$, C_n, C_n' are configurations holding fragments of program together with all the needed semantic components, like in big-step SOS, and *condition* is an optional side condition. Unlike in big-step SOS, the result configurations do not need to be explicitly defined because in small-step they are implicit: they are precisely those configurations which cannot be reduced anymore using the one-step relation.

Given a configuration holding a fragment of program, a small-step of computation typically takes place in some subpart of the fragment of program. However, when each of the subparts is already reduced, then the small-step can apply on the part itself. A small-step SOS is therefore finer-grain than big-step SOS, and thus more verbose, because one has to cover all the cases where a computation step can take place. For example, the small-step SOS of addition in IMP is

$$\frac{\langle a_1, \sigma\rangle \to \langle a_1', \sigma\rangle}{\langle a_1 + a_2, \sigma\rangle \to \langle a_1' + a_2, \sigma\rangle}$$

$$\frac{\langle a_2, \sigma\rangle \to \langle a_2', \sigma\rangle}{\langle a_1 + a_2, \sigma\rangle \to \langle a_1 + a_2', \sigma\rangle}$$

$$\langle i_1 + i_2, \sigma\rangle \to \langle i_1 +_{Int} i_2, \sigma\rangle$$

Here, the meaning of a relation $\langle a, \sigma\rangle \to \langle a', \sigma\rangle$ is that arithmetic expression a in state σ is reduced, in one small-step, to arithmetic expression a' and the state stays unchanged. Like for big-step SOS, one can encounter various other notations for small-step SOS configurations in the literature, e.g., $[a, \sigma]$, or (a, σ), or $\{a, \sigma\}$, or $\langle a \mid \sigma\rangle$, etc. Like for big-step SOS, we prefer to uniformly use the angle-bracket-and-comma notation $\langle a, \sigma\rangle$. Also, like for big-step SOS, one can encounter various decorations on the transition arrow $\to$, a notable situation being when the transition is

labeled. Again like for big-step SOS, we assume that such transition decorations are incorporated in the (source and/or target) configurations. How this can be effectively achieved is discussed in detail in Section 3.6 in the context of modular SOS (which allows rather complex transition labels).

The rules above rely on the fact that expression evaluation in IMP has no side effects. If there were side effects, like in the IMP extension in Section 3.5, then the σ's in the right-hand side configurations above would need to change to a different symbol, say σ', to account for the possibility that the small-step in the condition of the rules, and implicitly in their conclusion, may change the state as well. While in big-step SOS it is more common to derive sequents of the form $\langle a, \sigma \rangle \Downarrow \langle i \rangle$ than of the form $\langle a, \sigma \rangle \Downarrow \langle i, \sigma \rangle$, in small-step SOS the opposite tends to be norm, that is, it is more common to derive sequents of the form $\langle a, \sigma \rangle \to \langle a', \sigma \rangle$ than of the form $\langle a, \sigma \rangle \to \langle a' \rangle$. Nevertheless, the latter sequent type also works when defining languages like IMP whose expressions are side-effect-free (see Exercise 52). Some language designers may prefer this latter style, to keep sequents minimal. However, even if one prefers these simpler sequents, we prefer to keep the angle brackets on the right-hand sides of the transition relations (for the same reason like in big-step SOS—to maintain a uniform notation); in other words, we write $\langle a, \sigma \rangle \to \langle a' \rangle$ and not $\langle a, \sigma \rangle \to a'$.

In addition to rules, a small-step SOS may also include *structural identities*. For example, we can state that sequential composition is associative using the following structural identity:

$$(s_1 \; ; \; s_2) \; ; \; s_3 \equiv s_1 \; ; \; (s_2 \; ; \; s_3)$$

The small-step SOS rules apply *modulo* structural identities. In other words, the structural identities can be used anywhere in any configuration and at any moment during the derivation process, without counting as computational steps. In practice, they are typically used to rearrange the syntactic term so that some small-step SOS rule can apply. In particular, the structural rule above allows the designer of the small-step SOS to rely on the fact the first statement in a sequential composition is *not* a sequential composition, which may simplify the actual SOS rules; this is indeed the case in Section 3.5.4, where we extend IMP with dynamic threads (we do not need structural identities in the small-step SOS definition of the simple IMP language in this section). Structural identities are not easy to execute and/or implement in their full generality, because they can quickly yield an exponential explosion in the number of terms that need to be matched by rules. Their role in SOS semantics is the same as the role of equations in rewriting logic definitions; in fact, we effectively turn them into equations when we embed small-step SOS into rewriting logic (see Section 3.3.3).

3.3.1 IMP Configurations for Small-Step SOS

The configurations needed for the small-step semantics of IMP are a subset of those needed for its big-step semantics discussed in Section 3.2.1. Indeed, we still need all the two-component configurations containing a fragment of program and a state, but, for the particular small-step SOS style that we follow in this section, we do not need those result configurations of big-step SOS containing only a value or only a state. If one prefers to instead follow the minimalist style as in Exercise 52, then one would also need the other configuration types. We enumerate all the configuration types needed for the small-step SOS of IMP as given in the reminder of this section:

- $\langle a, \sigma \rangle$ grouping arithmetic expressions a and states σ;
- $\langle b, \sigma \rangle$ grouping Boolean expressions b and states σ;
- $\langle s, \sigma \rangle$ grouping statements s and states σ;

sorts:
Configuration
operations:
$\langle \_, \_ \rangle$: $AExp \times State \to Configuration$
$\langle \_, \_ \rangle$: $BExp \times State \to Configuration$
$\langle \_, \_ \rangle$: $Stmt \times State \to Configuration$
$\langle \_ \rangle$: $Pgm \to Configuration$

Figure 3.13: IMP small-step SOS configurations as an algebraic signature.

- $\langle p \rangle$ holding programs p.

We still need the one-component configuration holding only a program, because we still want to reduce a program in the default initial state (empty) without having to mention the empty state.

IMP Small-Step SOS Configurations as an Algebraic Signature

Figure 3.13 shows an algebraic signature defining the IMP configurations above, which is needed for the subsequent small-step operational semantics. We defined this algebraic signature in the same style and following the same assumptions as those for big-step SOS in Section 3.2.1.

3.3.2 The Small-Step SOS Rules of IMP

Figures 3.14 and 3.15 show all the rules in our IMP small-step SOS proof system, the former showing the rules corresponding to expressions, both arithmetic and Boolean, and the latter showing those rules corresponding to statements. The sequents that this proof system derives have the forms $\langle a, \sigma \rangle \to \langle a', \sigma \rangle$, $\langle b, \sigma \rangle \to \langle b', \sigma \rangle$, $\langle s, \sigma \rangle \to \langle s', \sigma' \rangle$, and $\langle p \rangle \to \langle s, \sigma \rangle$, where a ranges over *AExp*, b over *BExp*, s over *Stmt*, p over *Pgm*, and σ and σ' over *State*.

The meaning of derived triples of the form $\langle a, \sigma \rangle \to \langle a', \sigma \rangle$ is that given state σ, the arithmetic expression a reduces in one (small) step to the arithmetic expression a' and the state σ stays unchanged. The meaning of $\langle b, \sigma \rangle \to \langle b', \sigma \rangle$ is similar but with Boolean expressions instead of arithmetic expressions. The meaning of $\langle s, \sigma \rangle \to \langle s', \sigma' \rangle$ is that statement s in state σ reduces in one step to statement s' in a potentially modified state σ'. The meaning of $\langle p \rangle \to \langle s, \sigma \rangle$ is that program p reduces in one step to statement s in state σ (as expected, whenever such sequents can be derived, the statement s is the body of p and the state σ initializes to 0 the variables declared by p). The reason for which the state stays unchanged in the sequents corresponding to arithmetic and Boolean expressions is because, as discussed, IMP's expressions currently have no side effects; we will have to change these rules later on in Section 3.5 when we add a variable increment arithmetic expression construct to IMP. A small-step reduction of a statement may or may not change the state, so we use a different symbol in the right-hand side of statement transitions, σ', to cover both cases.

We next discuss each of the small-step SOS rules of IMP in Figures 3.14 and 3.15. Before we start, note that we have no small-step rules for reducing constant (integer or Boolean) expressions to their corresponding values as we had in big-step SOS (i.e., no rules corresponding to (BigStep-Int) and (BigStep-Bool) in Figure 3.7). Indeed, we do not want to have small-step SOS rules of the form $\langle v, \sigma \rangle \to \langle v, \sigma \rangle$ because no one-step reductions are further desired on values v: adding such rules would lead to undesired divergent SOS reductions later on when we consider the transitive closure

$$\langle x, \sigma\rangle \to \langle \sigma(x), \sigma\rangle \quad \text{if } \sigma(x) \neq \bot \qquad \text{(SmallStep-Lookup)}$$

$$\frac{\langle a_1, \sigma\rangle \to \langle a_1', \sigma\rangle}{\langle a_1 \texttt{+} a_2, \sigma\rangle \to \langle a_1' \texttt{+} a_2, \sigma\rangle} \qquad \text{(SmallStep-Add-Arg1)}$$

$$\frac{\langle a_2, \sigma\rangle \to \langle a_2', \sigma\rangle}{\langle a_1 \texttt{+} a_2, \sigma\rangle \to \langle a_1 \texttt{+} a_2', \sigma\rangle} \qquad \text{(SmallStep-Add-Arg2)}$$

$$\langle i_1 \texttt{+} i_2, \sigma\rangle \to \langle i_1 +_{Int} i_2, \sigma\rangle \qquad \text{(SmallStep-Add)}$$

$$\frac{\langle a_1, \sigma\rangle \to \langle a_1', \sigma\rangle}{\langle a_1 \texttt{/} a_2, \sigma\rangle \to \langle a_1' \texttt{/} a_2, \sigma\rangle} \qquad \text{(SmallStep-Div-Arg1)}$$

$$\frac{\langle a_2, \sigma\rangle \to \langle a_2', \sigma\rangle}{\langle a_1 \texttt{/} a_2, \sigma\rangle \to \langle a_1 \texttt{/} a_2', \sigma\rangle} \qquad \text{(SmallStep-Div-Arg2)}$$

$$\langle i_1 \texttt{/} i_2, \sigma\rangle \to \langle i_1 /_{Int} i_2, \sigma\rangle \quad \text{if } i_2 \neq 0 \qquad \text{(SmallStep-Div)}$$

$$\frac{\langle a_1, \sigma\rangle \to \langle a_1', \sigma\rangle}{\langle a_1 \texttt{<=} a_2, \sigma\rangle \to \langle a_1' \texttt{<=} a_2, \sigma\rangle} \qquad \text{(SmallStep-Leq-Arg1)}$$

$$\frac{\langle a_2, \sigma\rangle \to \langle a_2', \sigma\rangle}{\langle i_1 \texttt{<=} a_2, \sigma\rangle \to \langle i_1 \texttt{<=} a_2', \sigma\rangle} \qquad \text{(SmallStep-Leq-Arg2)}$$

$$\langle i_1 \texttt{<=} i_2, \sigma\rangle \to \langle i_1 \leq_{Int} i_2, \sigma\rangle \qquad \text{(SmallStep-Leq)}$$

$$\frac{\langle b, \sigma\rangle \to \langle b', \sigma\rangle}{\langle \texttt{not}\, b, \sigma\rangle \to \langle \texttt{not}\, b', \sigma\rangle} \qquad \text{(SmallStep-Not-Arg)}$$

$$\langle \texttt{not true}, \sigma\rangle \to \langle \texttt{false}, \sigma\rangle \qquad \text{(SmallStep-Not-True)}$$

$$\langle \texttt{not false}, \sigma\rangle \to \langle \texttt{true}, \sigma\rangle \qquad \text{(SmallStep-Not-False)}$$

$$\frac{\langle b_1, \sigma\rangle \to \langle b_1', \sigma\rangle}{\langle b_1 \texttt{ and } b_2, \sigma\rangle \to \langle b_1' \texttt{ and } b_2, \sigma\rangle} \qquad \text{(SmallStep-And-Arg1)}$$

$$\langle \texttt{false and}\, b_2, \sigma\rangle \to \langle \texttt{false}, \sigma\rangle \qquad \text{(SmallStep-And-False)}$$

$$\langle \texttt{true and}\, b_2, \sigma\rangle \to \langle b_2, \sigma\rangle \qquad \text{(SmallStep-And-True)}$$

Figure 3.14: SmallStep(IMP) — Small-step SOS of IMP expressions ($i_1, i_2 \in \mathit{Int}$; $x \in \mathit{Id}$; $a_1, a_1', a_2, a_2' \in \mathit{AExp}$; $b, b', b_1, b_1', b_2 \in \mathit{BExp}$; $\sigma \in \mathit{State}$).

of the one-step relation $\rightarrow$. Recall that the big-step SOS relation $\Downarrow$ captured all the reduction steps at once, including zero steps, and thus it did not need to be transitively closed like the small-step relation $\rightarrow$, so evaluating values to themselves was not problematic in big-step SOS.

The rule (SmallStep-Lookup) happens to be almost the same as in the big-step SOS; that's because variable lookup is an atomic-step operation both in big-step and in small-step SOS. The rules (SmallStep-Add-Arg1), (SmallStep-Add-Arg2), and (SmallStep-Add) give the small-step semantics of addition, and (SmallStep-Div-Arg1), (SmallStep-Div-Arg2), and (SmallStep-Div) give the small-step semantics of division, each covering all the three cases where a small-step reduction can take place. The first two cases in each group may apply non-deterministically. Recall from Section 3.2 that big-step SOS was inappropriate for defining the desired non-deterministic evaluation strategies for `+` and `/`. Fortunately, that was not a big problem for IMP, because its intended non-deterministic constructs are side-effect free. Therefore, the intended non-deterministic evaluation strategies of these particular language constructs did not affect the overall determinism of the IMP language, thus making its deterministic (see Exercise 42) big-step SOS definition in Section 3.2 acceptable. As expected, the non-deterministic evaluation strategies of `+` and `/`, which this time can be appropriately captured within the small-step SOS, will not affect the overall determinism of the IMP language (that is, the reflexive/transitive closure $\rightarrow^\star$ of $\rightarrow$; see Theorem 3).

The rules (SmallStep-Leq-Arg1), (SmallStep-Leq-Arg2), and (SmallStep-Leq) give the deterministic, sequential small-step SOS of `<=`. The first rule applies whenever a_1 is not an integer, then the second rule applies when a_1 is an integer but a_2 is not an integer, and finally, when both a_1 and a_2 are integers, the third rule applies. The rules (SmallStep-Not-Arg), (SmallStep-Not-True), and (SmallStep-Not-False) are self-explanatory, while the rules (SmallStep-And-Arg1), (SmallStep-And-False) and (SmallStep-And-True) give the short-circuited semantics of `and`: indeed, b_2 will not be reduced unless b_1 is first reduced to `true`.

Before we continue with the remaining small-step SOS rules for statements, let us see an example of a small-step SOS reduction using the rules discussed so far; as in the case of the big-step SOS rules in Section 3.2, recall that the small-step SOS rules are also rule schemas, that is, they are parametric in the (meta-)variables a, a_1, b, s, σ, etc. The following is a correct derivation, where x and y are any variables and σ is any state with $\sigma(x) = 1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdot}{\langle x, \sigma\rangle \rightarrow \langle 1, \sigma\rangle}}{\langle y \,/\, x, \sigma\rangle \rightarrow \langle y \,/\, 1, \sigma\rangle}}{\langle x + (y \,/\, x), \sigma\rangle \rightarrow \langle x + (y \,/\, 1), \sigma\rangle}}{\langle (x + (y \,/\, x)) \mathrel{<=} x, \sigma\rangle \rightarrow \langle (x + (y \,/\, 1)) \mathrel{<=} x, \sigma\rangle}$$

The above can be regarded as a proof of the fact that replacing the second occurrence of x by 1 is a correct one-step computation of IMP, as defined using the small-step SOS rules discussed so far.

Let us now discuss the small-step SOS rules of statements in Figure 3.15. Unlike the reduction of expressions, a reduction step of a statement may also change the state. Rule (SmallStep-Asgn-Arg2) reduces the second argument—which is an arithmetic expression—of an assignment statement whenever possible, regardless of whether the assigned variable was declared or not. Exercise 46 proposes an alternative semantics where the arithmetic expression is only reduced when the assigned variable has been declared. When the second argument is already fully reduced (i.e., it is an integer value), the rule (SmallStep-Asgn) reduces the assignment statement to `skip`, at the same time updating the state accordingly. Therefore, two steps are needed in order to assign an already evaluated expression to a variable: one step to write the variable in the state and modify

$$\frac{\langle a, \sigma\rangle \to \langle a', \sigma\rangle}{\langle x := a, \sigma\rangle \to \langle x := a', \sigma\rangle} \qquad \text{(SmallStep-Asgn-Arg2)}$$

$$\langle x := i, \sigma\rangle \to \langle \texttt{skip}, \sigma[i/x]\rangle \quad \text{if } \sigma(x) \neq \bot \qquad \text{(SmallStep-Asgn)}$$

$$\frac{\langle s_1, \sigma\rangle \to \langle s_1', \sigma'\rangle}{\langle s_1 \texttt{ ; } s_2, \sigma\rangle \to \langle s_1' \texttt{ ; } s_2, \sigma'\rangle} \qquad \text{(SmallStep-Seq-Arg1)}$$

$$\langle \texttt{skip ; } s_2, \sigma\rangle \to \langle s_2, \sigma\rangle \qquad \text{(SmallStep-Seq-Skip)}$$

$$\frac{\langle b, \sigma\rangle \to \langle b', \sigma\rangle}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2, \sigma\rangle \to \langle \texttt{if } b' \texttt{ then } s_1 \texttt{ else } s_2, \sigma\rangle} \qquad \text{(SmallStep-If-Arg1)}$$

$$\langle \texttt{if true then } s_1 \texttt{ else } s_2, \sigma\rangle \to \langle s_1, \sigma\rangle \qquad \text{(SmallStep-If-True)}$$

$$\langle \texttt{if false then } s_1 \texttt{ else } s_2, \sigma\rangle \to \langle s_2, \sigma\rangle \qquad \text{(SmallStep-If-False)}$$

$$\langle \texttt{while } b \texttt{ do } s, \sigma\rangle \to \langle \texttt{if } b \texttt{ then } (s \texttt{ ; while } b \texttt{ do } s) \texttt{ else skip}, \sigma\rangle \qquad \text{(SmallStep-While)}$$

$$\langle \texttt{var } xl \texttt{ ; } s\rangle \to \langle s, (xl \mapsto 0)\rangle \qquad \text{(SmallStep-Var)}$$

Figure 3.15: SmallStep(IMP) — Small-step SOS of IMP statements ($i \in \mathit{Int}$; $x \in \mathit{Id}$; $xl \in \mathbf{List}\{\mathit{Id}\}$; $a, a' \in \mathit{AExp}$; $b, b' \in \mathit{BExp}$; $s, s_1, s_1', s_2 \in \mathit{Stmt}$; $\sigma, \sigma' \in \mathit{State}$).

$$C \to^* C \qquad \text{(SmallStep-Closure-Stop)}$$

$$\frac{C \to C'', \quad C'' \to^* C'}{C \to^* C'} \qquad \text{(SmallStep-Closure-More)}$$

Figure 3.16: SmallStep(IMP) — Reflexive/transitive closure of the small-step SOS relation, which is the same for any small-step SOS of any language or calculus ($C, C', C'' \in \mathit{Configuration}$).

the assignment to `skip`, and another step to dissolve the resulting `skip`. Exercise 47 proposes an alternative semantics where these operations can be done in one step.

The rules (SmallStep-Seq-Arg1) and (SmallStep-Seq-Skip) give the small-step SOS of sequential composition: if the first statement is reducible then reduce it, otherwise, if it is `skip`, move on in a small-step to the second statement. Another possibility (different from that in Exercise 47) to avoid wasting the computational step generated by reductions to `skip` like in the paragraph above, is to eliminate the rule (SmallStep-Seq-Skip) and instead to add a rule that allows the reduction of the second statement provided that the first one is terminated. This approach is proposed by Hennessy [36], where he introduces a new sequent for *terminated configurations*, say $C\surd$, and then includes a rule like the following (and no rule like our (SmallStep-Seq-Skip)):

$$\frac{\langle s_1, \sigma\rangle\surd \qquad \langle s_2, \sigma\rangle \to \langle s_2', \sigma'\rangle}{\langle s_1 \texttt{ ; } s_2, \sigma\rangle \to \langle s_2', \sigma'\rangle}$$

The three rules for the conditional, namely (SmallStep-If-Arg1), (SmallStep-If-True), and (SmallStep-If-False), are straightforward; note that the two branches are never reduced when the condition can still be reduced. Exercise 49 proposes an alternative semantics for the conditional which wastes no computational step on switching to one of the two branches once the condition is evaluated.

The small-step SOS of `while` unrolls the loop once; this unrolling semantics seems as natural as it can be, but one should notice that it actually also generates an artificial computational step. Exercise 50 proposes an alternative semantics for `while` which wastes no computational step.

Finally, (SmallStep-Var) gives the semantics of programs by reducing them to their body statement in the expected state formed by initializing all the declared variables to 0. Note, however, that this rule also wastes a computational step; indeed, one may not want the initialization of the state with default values for variables to count as a step. Exercise 51 addresses this problem.

It is worthwhile noting that one has some flexibility in how to give a small-step SOS semantics to a language. The same holds true for almost any language definitional style, not only for SOS.

On Proof Derivations, Evaluation, and Termination

To formally capture the notion of "sequence of transitions", in Figure 3.16 we define the relation of *reflexive/transitive closure* of the small-step SOS transition.

Definition 17. *Given appropriate* IMP *small-step SOS configurations C and C', the* IMP *small-step SOS sequent $C \to C'$ is* ***derivable****, written* SmallStep(IMP) $\vdash C \to C'$*, iff there is some proof tree rooted in $C \to C'$ which is derivable using the proof system* SmallStep(IMP) *in Figures 3.14 and 3.15. In this case, we also say that C* ***reduces in one step*** *to C'. Similarly, the many-step*

sequent $C \to^\star C'$ *is* ***derivable****, written* SmallStep(IMP) $\vdash C \to^\star C'$*, iff there is some proof tree rooted in* $C \to^\star C'$ *which is derivable using the proof system in Figures 3.14, 3.15, and 3.16. In this case, we also say that* C ***reduces*** *(in zero, one, or more steps) to* C'*. Configuration* R *is* ***irreducible*** *iff there is no configuration* C *such that* SmallStep(IMP) $\vdash R \to C$*, and is a* ***result*** *iff it has one of the forms* $\langle i, \sigma \rangle$*,* $\langle t, \sigma \rangle$*, or* $\langle \texttt{skip}, \sigma \rangle$*, where* $i \in Int$*,* $t \in Bool$*, and* $\sigma \in State$*. Finally, configuration* C ***terminates*** *under* SmallStep(IMP) *iff there is no infinite sequence of configurations* $C_0, C_1, \ldots$ *such that* $C_0 = C$ *and* C_i *reduces in one step to* C_{i+1} *for any natural number* i*.*

Result configurations are irreducible, but there are irreducible configurations which are not necessarily result configurations. For example, the configuration $\langle i \,/\, 0, \sigma \rangle$ is irreducible but it is not a result. Like for big-step SOS, to catch division-by-zero within the semantics we need to add special error values/states and propagate them through all the language constructs (see Exercise 54).

The syntax of IMP (Section 3.1.1, Figure 3.1) was deliberately ambiguous with regards to sequential composition, and that was motivated by the fact that the semantics of the language will be given in such a way that the syntactic ambiguity will become irrelevant. We can now rigorously prove that is indeed the case, that is, we can prove properties of the like "SmallStep(IMP) $\vdash \langle (s_1 \,;\, s_2) \,;\, s_3, \sigma \rangle \to \langle (s_1' \,;\, s_2) \,;\, s_3, \sigma' \rangle$ if and only if SmallStep(IMP) $\vdash \langle s_1 \,;\, (s_2 \,;\, s_3), \sigma \rangle \to \langle s_1' \,;\, (s_2 \,;\, s_3), \sigma' \rangle$", etc. Exercise 55 discusses several such properties which, together with the fact that the semantics of no language construct is structurally defined in terms of sequential composition, also says that adding the associativity of sequential composition as a structural identity to the small-step SOS of IMP does not change the set of global behaviors of any IMP program (though we do not add it). However, that will not be the case anymore when we extend IMP with dynamic threads in Section 3.5.4, because the semantics of thread spawning will be given making use of sequential composition in such a way that adding this structural identity will be necessary in order to capture the desired set of behaviors.

Note that there are non-terminating sequences which repeat configurations, as well as non-terminating sequences in which all the configurations are distinct; an example of the former is a sequence generated by reducing the statement "`while true do skip`", while an example of the latter is a sequence generated by reducing "`while (n > 0) do n := n + 1`". Nevertheless, in the case of IMP, a configuration terminates if and only if it reduces to some irreducible configuration (see Exercise 56). This is not necessarily the case for non-deterministic languages, such as the IMP++ extension in Section 3.5, because reductions of configurations in such language semantics may non-deterministically choose steps that lead to termination, as well as steps that may not lead to termination. In the case of IMP though, the local non-determinism given by rules like (SmallStep-Add-Arg1) and (SmallStep-Add-Arg2) does not affect the overall determinism of the IMP language (Exercise 57).

Relating Big-Step and Small-Step SOS

As expected, the reflexive/transitive closure $\to^\star$ of the small-step SOS relation captures the same complete evaluation meaning as the $\Downarrow$ relation in big-step SOS (Section 3.2). Since for demonstrations reasons we deliberately worked with different result configurations in big-step and in small-step SOS, our theorem below looks slightly more involved than usual; if we had the same configurations in both semantics, then the theorem below would simply state "for any configuration C and any result configuration R, SmallStep(IMP) $\vdash C \to^\star R$ if and only if BigStep(IMP) $\vdash C \Downarrow R$":

Theorem 3. *The following equivalences hold for any $a \in AExp$, $i \in Int$, $b \in BExp$, $t \in Bool$, $s \in Stmt$, $p \in Pgm$, and $\sigma, \sigma' \in State$:*

- $\textsc{SmallStep}(\text{IMP}) \vdash \langle a, \sigma \rangle \rightarrow^\star \langle i, \sigma' \rangle$ *for a state* σ' *iff* $\textsc{BigStep}(\text{IMP}) \vdash \langle a, \sigma \rangle \Downarrow \langle i \rangle$;
- $\textsc{SmallStep}(\text{IMP}) \vdash \langle b, \sigma \rangle \rightarrow^\star \langle t, \sigma' \rangle$ *for a state* σ' *iff* $\textsc{BigStep}(\text{IMP}) \vdash \langle b, \sigma \rangle \Downarrow \langle t \rangle$;
- $\textsc{SmallStep}(\text{IMP}) \vdash \langle s, \sigma \rangle \rightarrow^\star \langle\, \texttt{skip}\, , \sigma' \rangle$ *for a state* σ' *iff* $\textsc{BigStep}(\text{IMP}) \vdash \langle s, \sigma \rangle \Downarrow \langle\, \texttt{skip}\, \rangle$;
- $\textsc{SmallStep}(\text{IMP}) \vdash \langle p \rangle \rightarrow^\star \langle\, \texttt{skip}\, , \sigma \rangle$ *for a state* σ *iff* $\textsc{BigStep}(\text{IMP}) \vdash \langle p \rangle \Downarrow \langle \sigma \rangle$.

Note that the small-step SOS relation for IMP is a *recursive*, or *decidable* problem: indeed, given configurations C and C', one can use the small-step proof system in Figures 3.14 and 3.15 to exhaustively check whether indeed $C \rightarrow C'$ is derivable or not. Moreover, since the rules for the reflexive/transitive closure relation $\rightarrow^\star$ in Figure 3.16 can be used to systematically generate any sequence of reductions, we conclude that the relation $\rightarrow^\star$ is *recursively enumerable*. Theorem 3 together with the discussion at the end of Section 3.2.2 and Exercise 43, tell us that $\rightarrow^\star$ is properly recursively enumerable, that is, it cannot be recursive. This tells us, in particular, that non-termination of a program p is equivalent to saying that, no matter what the state σ is, $\textsc{SmallStep}(\text{IMP}) \vdash \langle p \rangle \rightarrow^\star \langle\, \texttt{skip}\, , \sigma \rangle$ cannot be derived. However, unlike for big-step SOS where nothing else can be said about non-terminating programs, in the case of small-step SOS definitions one can use the small-step relation, $\rightarrow$, to observe program executions for any number of steps.

3.3.3 Small-Step SOS in Rewriting Logic

Like for big-step SOS, we can also associate a conditional rewrite rule to each small-step SOS rule and hereby obtain a rewriting logic theory that faithfully (i.e., step-for-step) captures the small-step SOS definition. Additionally, we can associate a rewriting logic equation to each SOS structural identity, because in both cases rules are applied modulo structural identities or equations. An important technical aspect needs to be resolved, though. The rewriting relation of rewriting logic is by its own nature reflexively and transitively closed. On the other hand, the small-step SOS relation is not reflexively and transitively closed by default (its reflexive/transitive closure is typically defined a posteriori, as we did in Figure 3.13). Therefore, we need to devise mechanisms to inhibit rewriting logic's reflexive, transitive and uncontrolled application of rules.

We first show that any small-step SOS, say SmallStep, can be mechanically translated into a rewriting logic theory, say $\mathcal{R}_{\textsc{SmallStep}}$, in such a way that the corresponding derivation relations are step-for-step equivalent, that is, $\textsc{SmallStep} \vdash C \rightarrow C'$ if and only if $\mathcal{R}_{\textsc{SmallStep}} \vdash \mathcal{R}_{C \rightarrow C'}$, where $\mathcal{R}_{C \rightarrow C'}$ is the corresponding syntactic translation of the small-step SOS sequent $C \rightarrow C'$ into a rewriting logic sequent. Second, we apply our generic translation technique on the small-step SOS formal system $\textsc{SmallStep}(\text{IMP})$ defined in Section 3.3.2 and obtain a rewriting logic semantics of IMP that is step-for-step equivalent to the original small-step SOS of IMP. Finally, we show how $\mathcal{R}_{\textsc{SmallStep}(\text{IMP})}$ can be seamlessly defined in Maude, thus yielding another interpreter for IMP (in addition to the one similarly obtained from the big-step SOS of IMP in Section 3.2.3).

Faithful Embedding of Small-Step SOS into Rewriting Logic

Like for big-step SOS (Section 3.2.3), to define our translation from small-step SOS to rewriting logic generically, we assume that each parametric configuration C admits an equivalent algebraic

variant $\overline{C}$ as a term of sort *Configuration* over an appropriate signature of configurations like the one that we defined for IMP in Section 3.2.1 (see Figure 3.6); moreover, each *parameter* in C (e.g., arithmetic expression $a \in AExp$) gets replaced by a *variable* of corresponding sort in $\overline{C}$ (e.g., variable A of sort $AExp$). Consider now a general-purpose small-step SOS rule of the form

$$\frac{C_1 \to C_1' \qquad C_2 \to C_2' \qquad \dots \qquad C_n \to C_n'}{C \to C'} \ [\text{if } \mathit{condition}]$$

where C, C', C_1, C_1', C_2, C_2', ..., C_n, C_n' are configurations holding fragments of program together with all the needed semantic components, and *condition* is an optional side condition. Before we introduce our transformation, let us first discuss why the same straightforward transformation that we used in the case of big-step SOS,

$$\overline{C} \to \overline{C'} \ \textbf{ if } \ \overline{C_1} \to \overline{C_1'} \ \wedge \ \overline{C_2} \to \overline{C_2'} \ \wedge \ \dots \ \wedge \ \overline{C_n} \to \overline{C_n'} \ [\wedge \ \overline{\mathit{condition}}],$$

does not work in the case of small-step SOS. For example, with that transformation, the rewrite rules corresponding to the small-step SOS rules of IMP for assignment (SmallStep-Asgn-Arg2) and (SmallStep-Asgn) in Figure 3.15 would be

$$\begin{array}{rll} \langle X := A, \sigma \rangle \to \langle X := A', \sigma \rangle & \textbf{if} & \langle A, \sigma \rangle \to \langle A', \sigma \rangle \\ \langle X := I, \sigma \rangle \to \langle \, \texttt{skip} \, , \sigma[I/X] \rangle & \textbf{if} & \sigma(X) \neq \bot \end{array}$$

The problem with these rules is that the rewrite of a configuration of the form $\langle x := i, \sigma \rangle$ for some $x \in Id$, $i \in Int$ and $\sigma \in State$ may not terminate, applying forever the first rule: in rewriting logic, $\langle i, \sigma \rangle \to \langle i, \sigma \rangle$ because $\to$ is closed under reflexivity. Even if we may somehow solve this reflexivity aspect by defining and then including an additional condition $A \neq A'$, such rules still fail to capture the intended small-step transition, because $\to$ is also closed transitively in rewriting logic, so there could be many small-steps taking place in the condition of the first rule before the rule is applied.

To capture *exactly one step* of reduction, thus avoiding the inherent automatic reflexive and transitive closure of the rewrite relation which is desirable in rewriting logic but not in reduction semantics, we can mark the left-hand side (or, alternatively, the right-hand side) configuration in each rewrite sequent to always be distinct from the other one; then each rewrite sequent comprises precisely one step, from a marked to an unmarked configuration (or vice versa). For example, let us place a $\circ$ in front of all the left-hand side configurations and keep the right-hand side configurations unchanged. Then the generic small-step SOS rule above translates into the rewriting logic rule

$$\circ\overline{C} \to \overline{C'} \ \textbf{ if } \ \circ\overline{C_1} \to \overline{C_1'} \ \wedge \ \circ\overline{C_2} \to \overline{C_2'} \ \wedge \ \dots \ \wedge \ \circ\overline{C_n} \to \overline{C_n'} \ [\wedge \ \overline{\mathit{condition}}].$$

One can metaphorically think of a marked configuration $\circ\overline{C}$ as a hot configuration that needs to be cooled down in one step, while of an unmarked configuration $\overline{C}$ as a cool one. Theorem 4 below shows as expected that a small-step SOS sequent $C \to C'$ is derivable if and only if the term $\circ\overline{C}$ rewrites in the corresponding rewrite theory to $\overline{C'}$ (which is a normal form). Thus, to enable the resulting rewrite system on a given configuration, one needs to first mark the configuration to be reduced (by placing a $\circ$ in front of it) and then to let it rewrite to its normal form. Since the one-step reduction always terminates, the corresponding rewrite task also terminates.

If the original small-step SOS had structural identities, then we translate them into equations in a straightforward manner: each identity $t \equiv t'$ is translated into an equation $\overline{t} = \overline{t'}$. The only difference between the original structural identity and the resulting equation is that the meta-variables of

sorts:
ExtendedConfiguration
subsorts:
Configuration $<$ *ExtendedConfiguration*
operations:
$\circ\_$: *Configuration* $\rightarrow$ *ExtendedConfiguration* // reduce one step
$\star\_$: *Configuration* $\rightarrow$ *ExtendedConfiguration* // reduce all steps
rule:
$\star\, Cfg \rightarrow \star\, Cfg'$ **if** $\circ\, Cfg \rightarrow Cfg'$ // where Cfg, Cfg' are variables of sort *Configuration*

Figure 3.17: Representing small-step and many-step SOS reductions in rewriting logic.

the former become variables in the latter. The role of the two is the same in their corresponding frameworks and whatever we can do with one in one framework we can equivalently do with the other in the other framework; consequently, to simplify the notation and the presentation, we will make abstraction of structural identities and equations in our theoretical developments in the reminder of this chapter.

To obtain the reflexive and transitive many-step closure of the small-step SOS relation in the resulting rewrite setting and thus to be able to obtain an interpreter for the defined language when executing the rewrite system, we need to devise some mechanism to iteratively apply the one-step reduction step captured by rewriting as explained above. There could be many ways to do that, but one simple and uniform way is to add a new configuration marker, say $\star\overline{C}$, with the meaning that $\overline{C}$ must be iteratively reduced, small-step after small-step, either forever or until an irreducible configuration is reached. Figure 3.17 shows how one can define both configuration markers algebraically (assuming some existing *Configuration* sort, e.g., the one in Figure 3.13). To distinguish the marked configurations from the usual configurations and to also possibly allow several one-step markers at the same time, e.g., $\circ\circ\circ\,\overline{C}$, which could be useful for debugging/tracing reasons, we preferred to define the sort of marked configurations as a supersort of *Configuration*. Note that the rule in Figure 3.17 indeed gives $\star$ its desired reflexive and transitive closure property (the reflexivity follows from the fact that the rewrite relation in rewriting logic is reflexive, so $\star C \rightarrow \star C$ for any configuration term C).

Theorem 4. ***(Faithful embedding of small-step SOS into rewriting logic)*** *For any small-step SOS* SmallStep*, and any* SmallStep *appropriate configurations C and C', the following equivalences hold:*

$$\text{SmallStep} \vdash C \rightarrow C' \iff \mathcal{R}_{\text{SmallStep}} \vdash \circ\,\overline{C} \rightarrow^1 \overline{C'} \iff \mathcal{R}_{\text{SmallStep}} \vdash \circ\,\overline{C} \rightarrow \overline{C'}$$

$$\text{SmallStep} \vdash C \rightarrow^\star C' \iff \mathcal{R}_{\text{SmallStep}} \vdash \star\,\overline{C} \rightarrow \star\,\overline{C'}$$

where $\mathcal{R}_{\text{SmallStep}}$ is the rewriting logic semantic definition obtained from SmallStep *by translating each rule in* SmallStep *as above. (Recall from Section 2.7 that $\rightarrow^1$ is the one-step rewriting relation obtained by dropping the reflexivity and transitivity rules of rewriting logic. Also, as C and C' are parameter-free—parameters only appear in rules—, $\overline{C}$ and $\overline{C'}$ are ground terms.)*

Except for transforming parameters into variables, the only apparent difference between SmallStep and $\mathcal{R}_{\text{SmallStep}}$ is that the latter marks (using $\circ$) all the left-hand side configura-

tions and, naturally, uses conditional rewrite rules instead of conditional deduction rules. As Theorem 4 shows, there is a step-for-step correspondence between their corresponding computations (or executions, or derivations). Therefore, similarly to the big-step SOS representation in rewriting logic, the rewrite theory $\mathcal{R}_{\text{SmallStep}}$ *is* the small-step SOS SmallStep, and *not* an encoding of it.

Recall from Section 3.2.3 that in the case of big-step SOS there were some subtle differences between the one-step $\rightarrow^1$ (obtained by dropping the reflexivity and transitivity rules of rewriting logic) and the usual $\rightarrow$ relations in the rewrite theory corresponding to the big-step SOS. The approach followed in this section based on marking configurations, thus keeping the left-hand and the right-hand sides always distinct, eliminates all the differences between the two rewrite relations in the case of the one-step reduction (the two relations are identical on the terms of interest). The second equivalence in Theorem 4 tells us that we can turn the rewriting logic representation of the small-step SOS language definition into an interpreter by simply marking the configuration to be completely reduced with a $\star$ and then letting the rewrite engine do its job.

It is worthwhile noting that like in the case of the big-step SOS representation in rewriting logic, unfortunately, $\mathcal{R}_{\text{SmallStep}}$ lacks the main strengths of rewriting logic: in rewriting logic, rewrite rules can apply under any context and in parallel. Indeed, the rules of $\mathcal{R}_{\text{SmallStep}}$ can only apply at the top, sequentially. This should not surprise because, as stated, $\mathcal{R}_{\text{SmallStep}}$ *is* SmallStep, with all its strengths and limitations. By all means, both the $\mathcal{R}_{\text{SmallStep}}$ above and the $\mathcal{R}_{\text{BigStep}}$ in Section 3.2.3 are rather poor-style rewriting logic specifications. However, that is normal, because neither big-step SOS nor small-step SOS were meant to have the capabilities of rewriting logic w.r.t. context-insensitivity and parallelism; since their representations in rewriting logic are faithful, one should not expect that they inherit the additional capabilities of rewriting logic (if they did, then the representations would not be step-for-step faithful, so something would be wrong).

Small-Step SOS of IMP in Rewriting Logic

Figure 3.18 gives the rewriting logic theory $\mathcal{R}_{\text{SmallStep(IMP)}}$ that is obtained by applying the procedure above to the small-step SOS of IMP, namely the formal system SmallStep(IMP) presented in Figures 3.14 and 3.15. As usual, we used the rewriting logic convention that variables start with upper-case letters, and like in the rewrite theory corresponding to the big-step SOS of IMP in Figure 3.8, we used σ (a larger σ symbol) for variables of sort *State*. Besides the parameter vs. variable subtle (but not unexpected) aspect, the only perceivable difference between SmallStep(IMP) and $\mathcal{R}_{\text{SmallStep(IMP)}}$ is the different notational conventions they use. The following corollary of Theorem 4 establishes the faithfulness of the representation of the small-step SOS of IMP in rewriting logic:

Corollary 3. $\text{SmallStep(IMP)} \vdash C \rightarrow C' \iff \mathcal{R}_{\text{SmallStep(IMP)}} \vdash \circ\overline{C} \rightarrow \overline{C'}$.

Therefore, there is no perceivable computational difference between the IMP-specific proof system SmallStep(IMP) and generic rewriting logic deduction using the IMP-specific rewrite rules in $\mathcal{R}_{\text{SmallStep(IMP)}}$; the two are faithfully equivalent.

☆ Maude Definition of IMP Small-Step SOS

Figure 3.19 shows a straightforward Maude representation of the rewrite theory $\mathcal{R}_{\text{SmallStep(IMP)}}$ in Figure 3.18, including representations of the algebraic signatures of small-step SOS configurations in Figure 3.13 and of their extensions in Figure 3.17, which are needed to capture small-step SOS

$$\circ\langle X,\sigma\rangle \to \langle \sigma(X),\sigma\rangle \ \textbf{if}\ \sigma(X) \neq \bot$$

$$\begin{aligned}
\circ\langle A_1 + A_2,\sigma\rangle &\to \langle A_1' + A_2,\sigma\rangle \ \textbf{if}\ \circ\langle A_1,\sigma\rangle \to \langle A_1',\sigma\rangle\\
\circ\langle A_1 + A_2,\sigma\rangle &\to \langle A_1 + A_2',\sigma\rangle \ \textbf{if}\ \circ\langle A_2,\sigma\rangle \to \langle A_2',\sigma\rangle\\
\circ\langle I_1 + I_2,\sigma\rangle &\to \langle I_1\ +_{Int}\ I_2,\sigma\rangle
\end{aligned}$$

$$\begin{aligned}
\circ\langle A_1 \,/\, A_2,\sigma\rangle &\to \langle A_1' \,/\, A_2,\sigma\rangle \ \textbf{if}\ \circ\langle A_1,\sigma\rangle \to \langle A_1',\sigma\rangle\\
\circ\langle A_1 \,/\, A_2,\sigma\rangle &\to \langle A_1 \,/\, A_2',\sigma\rangle \ \textbf{if}\ \circ\langle A_2,\sigma\rangle \to \langle A_2',\sigma\rangle\\
\circ\langle I_1 \,/\, I_2,\sigma\rangle &\to \langle I_1\ /_{Int}\ I_2,\sigma\rangle \ \textbf{if}\ I_2 \neq 0
\end{aligned}$$

$$\begin{aligned}
\circ\langle A_1 \texttt{<=} A_2,\sigma\rangle &\to \langle A_1' \texttt{<=} A_2,\sigma\rangle \ \textbf{if}\ \circ\langle A_1,\sigma\rangle \to \langle A_1',\sigma\rangle\\
\circ\langle I_1 \texttt{<=} A_2,\sigma\rangle &\to \langle I_1 \texttt{<=} A_2',\sigma\rangle \ \textbf{if}\ \circ\langle A_2,\sigma\rangle \to \langle A_2',\sigma\rangle\\
\circ\langle I_1 \texttt{<=} I_2,\sigma\rangle &\to \langle I_1\ \leq_{Int}\ I_2,\sigma\rangle
\end{aligned}$$

$$\begin{aligned}
\circ\langle\, \texttt{not}\ B,\sigma\rangle &\to \langle\, \texttt{not}\ B',\sigma\rangle \ \textbf{if}\ \circ\langle B,\sigma\rangle \to \langle B',\sigma\rangle\\
\circ\langle\, \texttt{not true},\sigma\rangle &\to \langle\, \texttt{false},\sigma\rangle\\
\circ\langle\, \texttt{not false},\sigma\rangle &\to \langle\, \texttt{true},\sigma\rangle
\end{aligned}$$

$$\begin{aligned}
\circ\langle B_1 \,\texttt{and}\, B_2,\sigma\rangle &\to \langle B_1' \,\texttt{and}\, B_2,\sigma\rangle \ \textbf{if}\ \circ\langle B_1,\sigma\rangle \to \langle B_1',\sigma\rangle\\
\circ\langle\, \texttt{false and}\, B_2,\sigma\rangle &\to \langle\, \texttt{false},\sigma\rangle\\
\circ\langle\, \texttt{true and}\, B_2,\sigma\rangle &\to \langle B_2,\sigma\rangle
\end{aligned}$$

$$\begin{aligned}
\circ\langle X \texttt{:=} A,\sigma\rangle &\to \langle X \texttt{:=} A',\sigma\rangle \ \textbf{if}\ \circ\langle A,\sigma\rangle \to \langle A',\sigma\rangle\\
\circ\langle X \texttt{:=} I,\sigma\rangle &\to \langle\, \texttt{skip},\sigma[I/X]\rangle \ \textbf{if}\ \sigma(X) \neq \bot
\end{aligned}$$

$$\begin{aligned}
\circ\langle S_1\ \texttt{;}\ S_2,\sigma\rangle &\to \langle S_1'\ \texttt{;}\ S_2,\sigma'\rangle \ \textbf{if}\ \circ\langle S_1,\sigma\rangle \to \langle S_1',\sigma'\rangle\\
\circ\langle\, \texttt{skip}\ \texttt{;}\ S_2,\sigma\rangle &\to \langle S_2,\sigma\rangle
\end{aligned}$$

$$\begin{aligned}
\circ\langle\, \texttt{if}\, B\, \texttt{then}\, S_1\, \texttt{else}\, S_2,\sigma\rangle &\to \langle\, \texttt{if}\, B'\, \texttt{then}\, S_1\, \texttt{else}\, S_2,\sigma\rangle \ \textbf{if}\ \circ\langle B,\sigma\rangle \to \langle B',\sigma\rangle\\
\circ\langle\, \texttt{if true then}\, S_1\, \texttt{else}\, S_2,\sigma\rangle &\to \langle S_1,\sigma\rangle\\
\circ\langle\, \texttt{if false then}\, S_1\, \texttt{else}\, S_2,\sigma\rangle &\to \langle S_2,\sigma\rangle
\end{aligned}$$

$$\circ\langle\, \texttt{while}\, B\, \texttt{do}\, S,\sigma\rangle \to \langle\, \texttt{if}\, B\, \texttt{then}\, (S\ \texttt{;}\ \texttt{while}\, B\, \texttt{do}\, S)\, \texttt{else skip},\sigma\rangle$$

$$\circ\langle\, \texttt{var}\, Xl\ \texttt{;}\ S\rangle \to \langle S, (Xl \mapsto 0)\rangle$$

Figure 3.18: $\mathcal{R}_{\text{SmallStep(IMP)}}$: the small-step SOS of IMP in rewriting logic.

```
mod IMP-CONFIGURATIONS-SMALLSTEP is including IMP-SYNTAX + STATE .
  sorts Configuration ExtendedConfiguration .
  subsort Configuration < ExtendedConfiguration .
  op <_,_> : AExp State -> Configuration .
  op <_,_> : BExp State -> Configuration .
  op <_,_> : Stmt State -> Configuration .
  op <_> : Pgm -> Configuration .
  op o_ : Configuration -> ExtendedConfiguration [prec 80] .   --- one step
  op *_ : Configuration -> ExtendedConfiguration [prec 80] .   --- all steps
  var Cfg Cfg' : Configuration .
 crl * Cfg => * Cfg' if o Cfg => Cfg' .
endm

mod IMP-SEMANTICS-SMALLSTEP is including IMP-CONFIGURATIONS-SMALLSTEP .
  var X : Id .  var Sigma Sigma' : State .  var I I1 I2 : Int .  var Xl : List{Id} .
  var A A' A1 A1' A2 A2' : AExp .  var B B' B1 B1' B2 : BExp .  var S S1 S1' S2 : Stmt .

 crl o < X,Sigma > => < Sigma(X),Sigma > if Sigma(X) =/=Bool undefined .

 crl o < A1 + A2,Sigma > => < A1' + A2,Sigma > if o < A1,Sigma > => < A1',Sigma > .
 crl o < A1 + A2,Sigma > => < A1 + A2',Sigma > if o < A2,Sigma > => < A2',Sigma > .
  rl o < I1 + I2,Sigma > => < I1 +Int I2,Sigma > .

 crl o < A1 / A2,Sigma > => < A1' / A2,Sigma > if o < A1,Sigma > => < A1',Sigma > .
 crl o < A1 / A2,Sigma > => < A1 / A2',Sigma > if o < A2,Sigma > => < A2',Sigma > .
 crl o < I1 / I2,Sigma > => < I1 /Int I2,Sigma > if I2 =/=Bool 0 .

 crl o < A1 <= A2,Sigma > => < A1' <= A2,Sigma > if o < A1,Sigma > => < A1',Sigma > .
 crl o < I1 <= A2,Sigma > => < I1 <= A2',Sigma > if o < A2,Sigma > => < A2',Sigma > .
  rl o < I1 <= I2,Sigma > => < I1 <=Int I2,Sigma > .

 crl o < not B,Sigma > => < not B',Sigma > if o < B,Sigma > => < B',Sigma > .
  rl o < not  true,Sigma > => < false,Sigma > .
  rl o < not false,Sigma > => <  true,Sigma > .

 crl o < B1 and B2,Sigma > => < B1' and B2,Sigma > if o < B1,Sigma > => < B1',Sigma > .
  rl o < false and B2,Sigma > => < false,Sigma > .
  rl o <  true and B2,Sigma > => < B2,Sigma > .

 crl o < X := A,Sigma > => < X := A',Sigma > if o < A,Sigma > => < A',Sigma > .
 crl o < X := I,Sigma > => < skip,Sigma[I / X] > if Sigma(X) =/=Bool undefined .

 crl o < S1 ; S2,Sigma > => < S1' ; S2,Sigma' > if o < S1,Sigma > => < S1',Sigma' > .
  rl o < skip ; S2,Sigma > => < S2,Sigma > .

 crl o < if B then S1 else S2,Sigma > => < if B' then S1 else S2,Sigma >
  if o < B,Sigma > => < B',Sigma  > .
  rl o < if  true then S1 else S2,Sigma > => < S1,Sigma > .
  rl o < if false then S1 else S2,Sigma > => < S2,Sigma > .

  rl o < while B do S,Sigma > => < if B then (S ; while B do S) else skip,Sigma > .

  rl o < var Xl ; S > => < S,(Xl |-> 0) > .
endm
```

Figure 3.19: The small-step SOS of IMP in Maude, including the definition of configurations.

in rewriting logic. The Maude module `IMP-SEMANTICS-SMALLSTEP` in Figure 3.19 is executable, so Maude, through its rewriting capabilities, yields a small-step SOS interpreter for IMP the same way it yielded a big-step SOS interpreter in Section 3.2.3; for example, the command

```
rewrite * < sumPgm > .
```

where `sumPgm` is the first program defined in the module `IMP-PROGRAMS` in Figure 3.4, produces a result of the form (the exact statistics are also irrelevant, so they were replaced by "..."):

```
rewrites: 7132 in ... cpu (... real) (... rewrites/second)
result ExtendedConfiguration: * < skip,n |-> 0 , s |-> 5050 >
```

Like for the big-step SOS definition in Maude, one can also use any of the general-purpose tools provided by Maude on the small-step SOS definition above. For example, one can exhaustively search for all possible behaviors of a program using the `search` command:

```
search * < sumPgm > =>! Cfg:ExtendedConfiguration .
```

As expected, only one behavior will be discovered because our IMP language so far is deterministic. However, a relatively large number of states will be explored, 1509, due to the non-deterministic evaluation strategy of the various language constructs:

```
Solution 1 (state 1508)
states: 1509  rewrites: 8732 in ... cpu (... real) (0 rewrites/second)
Cfg:ExtendedConfiguration --> * < skip,n |-> 0 & s |-> 5050 >
```

3.3.4 Notes

Small-step structural operational semantics was introduced as just *structural operational semantics* (SOS; no "small-step" qualifier at that time) by Plotkin in a 1981 technical report (University of Aarhus Technical Report DAIMI FN-19, 1981) that included his lecture notes of a programming language course [70]. For more than 20 years this technical report was cited as the main SOS reference by hundreds of scientists who were looking for mathematical rigor in their programming language research. It was only in 2004 that Plotkin's SOS was finally published in a journal [71].

Small-step SOS is pedagogically discussed in several textbooks, two early notable ones being Hennessy [36] (1990) and Winskel [98] (1993). Hennessy [36] uses the same notation as Plotkin, but Winskel [98] prefers a different one to make it clear that it is a one step semantics: $\langle s, \sigma \rangle \rightarrow_1 \langle s', \sigma' \rangle$. Like for big-step SOS, there is no well-established notation for small-step SOS sequents. There is a plethora of research projects and papers that explicitly or implicitly take SOS as *the* formal language semantics framework. Also, SOS served as a source of inspiration, or of problems to be fixed, to other semantic framework designers, including the author. There is simply too much work on SOS, using it, or modifying it, to attempt to cover it here. We limit ourselves to directly related research focused on capturing SOS as a methodological fragment of rewriting logic.

The marked configuration style that we adopted in this section to faithfully represent small-step SOS in rewriting logic was borrowed from Șerbănuță *et al.* [85]; there, the configuration marker "$\circ$" was called a "configuration modifier". An alternative way to keep the left-hand and the right-hand side configurations distinct was proposed by Meseguer and Braga in [50, 16] in the context of representing MSOS into rewrite logic (see Section 3.6); the idea there was to use two different types of configuration wrappers, one for the left-hand side of the transitions and one for the right-hand side, yielding rewriting logic rules of the form:

$$\{C\} \rightarrow [C'] \;\; \text{if} \;\; \{C_1\} \rightarrow [C'_1] \;\wedge\; \{C_2\} \rightarrow [C'_2] \;\wedge \cdots \wedge \{C_n\} \rightarrow [C'_n].$$

The solution proposed by Meseguer and Braga in [50, 16] builds upon experience with a previous representation of MSOS in rewriting logic in [17] as well as with an implementation of it in Maude [15, 18], where the necessity of being able to inhibit the default reflexivity and transitivity of the rewrite relation took shape. We preferred to follow the configuration modifier approach proposed by Șerbănuță *et al.* [85] because it appears to be slightly less intrusive (we only tag the already existing left-hand terms of rules) and more general (the left-hands of rules can have any structure, not only configurations, including no structure at all, as it happens in most of the rules of reduction semantics with evaluation contexts—see Section 3.7, e.g., Figure 3.40).

Vardejo and Martì-Oliet [95] give a Maude implementation of a small-step SOS definition for a simple imperative language similar to our IMP (Hennessy's *WhileL* language [36]), in which they do not attempt to prevent the inherent transitivity of rewriting. While they indeed obtain an executable semantics that is reminiscent of the original small-step SOS of the language, they actually define directly the transitive closure of the small-step SOS relation; they explicitly disable the reflexive closure by checking $C \neq C'$ next to rewrites $C \to C'$ in rule conditions. A small-step SOS of a simple functional language (Hennessy's *Fpl* language [36]) is also given in [95], following a slightly different style, which avoids the problem above. They successfully inhibit rewriting's inherent transitivity in their definition by using a rather creative rewriting representation style for sequents. More precisely, they work with sequents which appear to the user as having the form $\sigma \vdash a \to a'$, where σ is a state and a, a' are arithmetic expressions, etc., but they actually are rewrite relations between terms $\sigma \vdash a$ and a' (an appropriate signature to allow that to parse is defined). Indeed, there is no problem with the automatic reflexive/transitive closure of rewriting here because the LHS and the RHS of each rewrite rule have different structures. The simple functional language in [95] was pure (no side effects), so there was no need to include a resulting state in the RHS of their rules; if the language had side effects, then this Vardejo and Martì-Oliet's representation of small-step SOS sequents in [95] would effectively be the same as the one by Meseguer and Braga in [50, 16] (possibly using different notations, which is irrelevant).

3.3.5 Exercises

Prove the following exercises, all referring to the IMP small-step SOS in Figures 3.14 and 3.15.

Exercise 44. *Change the small-step rules for* `/` *so that it short-circuits when* a_1 *evaluates to* 0.

Exercise 45. *Change the small-step SOS of the* IMP *conjunction so that it is not short-circuited.*

Exercise 46. *One can rightfully argue that the arithmetic expression in an assignment should not be reduced any step when the assigned variable is not declared. Change the small-step SOS of* IMP *to only reduce the arithmetic expression when the assigned variable is declared.*

Exercise 47. *A sophisticated language designer could argue that the reduction of the assignment statement to* `skip` *is an artifact of using small-step SOS, therefore an artificial and undesired step which affects the intended computational granularity of the language. Change the small-step SOS of* IMP *to eliminate this additional small-step.*
Hint: Follow the style in Exercise 52; note, however, that that style will require more rules and more types of configurations, so from that point of view is more complex.

Exercise 48. *Give a proof system for deriving "terminated configuration" sequents* $C\surd$.

Exercise 49. *One could argue that our small-step SOS rules for the conditional waste a computational step when switching to one of the two branches once the condition is evaluated.*

1. *Give an alternative small-step SOS for the conditional which does not require a computational step to switch to one of the two branches.*

2. *Can one do better than that? That is, can one save an additional step by reducing the corresponding branch one step at the same time with reducing the condition to true or false in one step? Hint: one may need terminated configurations, like in Exercise 48.*

Exercise 50. *Give an alternative small-step SOS definition of* `while` *which wastes no computational step. Hint: do a case analysis on b, like in the rules for the conditional.*

Exercise 51. *Give an alternative small-step SOS definition of variable declarations which wastes no computational steps. Hint: one may need terminated configurations, like in Exercise 48.*

Exercise 52. *Modify the small-step SOS definition of* IMP *such that the configurations in the right-hand sides of the transition sequents are minimal (they should contain both a fragment of program and a state only when absolutely needed). What are the drawbacks of this minimalistic approach, compared to the small-step SOS semantics that we chose to follow?*

Exercise 53. *Show that the small-step SOS resulting from Exercise 52 is equivalent to the one in Figure 3.14 on arithmetic and Boolean expressions, that is, $\langle a, \sigma\rangle \to \langle a', \sigma\rangle$ is derivable with the proof system in Figure 3.14 if and only if $\langle a, \sigma\rangle \to \langle a'\rangle$ is derivable with the proof system in Exercise 52, and similarly for Boolean expressions. However, show that the equivalence does not hold true for statements.*

Exercise 54. *Add* `error` *values and statements, and modify the small-step semantics in Figures 3.14 and 3.15 to allow derivations of sequents whose right-hand side configurations contain* `error` *as their syntactic component. Also, experiment with more meaningful error messages.*

Exercise 55. *For any* IMP *statements s_1, s_1', s_2, s_3 and any states σ, σ', the following hold:*

1. $\textsc{SmallStep}(\text{IMP}) \vdash \langle(\ \texttt{skip}\ ;\ s_2)\ ;\ s_3, \sigma\rangle \to \langle s_2\ ;\ s_3, \sigma\rangle$ *and*
   $\textsc{SmallStep}(\text{IMP}) \vdash \langle\ \texttt{skip}\ ;\ (s_2\ ;\ s_3), \sigma\rangle \to \langle s_2\ ;\ s_3, \sigma\rangle$*; and*

2. $\textsc{SmallStep}(\text{IMP}) \vdash \langle(s_1\ ;\ s_2)\ ;\ s_3, \sigma\rangle \to \langle(s_1'\ ;\ s_2)\ ;\ s_3, \sigma'\rangle$ *if and only if*
   $\textsc{SmallStep}(\text{IMP}) \vdash \langle s_1\ ;\ (s_2\ ;\ s_3), \sigma\rangle \to \langle s_1'\ ;\ (s_2\ ;\ s_3), \sigma'\rangle$.

Consequently, the following also hold (prove them by structural induction on s_1):
$\textsc{SmallStep}(\text{IMP}) \vdash \langle(s_1\ ;\ s_2)\ ;\ s_3, \sigma\rangle \to^\star \langle s_2\ ;\ s_3, \sigma'\rangle$ *if and only if*
$\textsc{SmallStep}(\text{IMP}) \vdash \langle s_1\ ;\ (s_2\ ;\ s_3), \sigma\rangle \to^\star \langle s_2\ ;\ s_3, \sigma'\rangle$ *if and only if*
$\textsc{SmallStep}(\text{IMP}) \vdash \langle s_1, \sigma\rangle \to^\star \langle\ \texttt{skip}, \sigma'\rangle$.

Exercise 56. *With the* $\textsc{SmallStep}(\text{IMP})$ *proof system in Figures 3.14, 3.15, and 3.16, configuration C terminates iff* $\textsc{SmallStep}(\text{IMP}) \vdash C \to^\star R$ *for some irreducible configuration R.*

Exercise 57. *The small-step SOS of* IMP *is globally deterministic: if* $\textsc{SmallStep}(\text{IMP}) \vdash C \to^\star R$ *and* $\textsc{SmallStep}(\text{IMP}) \vdash C \to^\star R'$ *for irreducible configurations R and R', then $R = R'$. Show the same result for the proof system detecting division-by-zero as in Exercise 54.*

Exercise 58. *Show that if* $\textsc{SmallStep}(\text{IMP}) \vdash C \to^\star \langle i, \sigma \rangle$ *for some configuration* C*, integer* i*, and state* σ*, then* C *must be of the form* $\langle a, \sigma \rangle$ *for some arithmetic expression* a*. Show a similar result for Boolean expressions. For statements, show that if* $\textsc{SmallStep}(\text{IMP}) \vdash C \to^\star \langle \texttt{skip}, \sigma \rangle$ *then* C *must be either of the form* $\langle s, \sigma' \rangle$ *for some statement* s *and some state* σ'*, or of the form* $\langle p \rangle$ *for some program* p*.*

Exercise 59. *Prove Theorem 3.*

Exercise 60. *State and show a result similar to Theorem 3 but for the small-step and big-step SOS proof systems in Exercises 54 and 41, respectively.*