15-819: Resource Aware Programming Languages

Lectures 3-5: Cost Semantics

Jan Hoffmann

September 29, 2020

1 Introduction

In the first lectures, we discussed that we would like to prove that a derived recurrence relation actually corresponds to the resource usage of a given program. However, we defined our cost models only informally, for example, as the number of multiplications executed. Such informal cost models are common in the analysis of algorithms and often convenient to focus on the main points of the analysis. Nevertheless, informal models of time are sometimes problematic when comparing different algorithms with similar resource usage or when there are different possibilities to implement certain operations. In any case, it is not possible to formally prove the correctness of a resource bound without precisely defining a cost model.

In this course, our cost models are defined by an *operational cost semantics* and (optionally) *resource metrics*. There are many ways of how to define a cost semantics and we explore only the most common ones in this lecture. Which cost semantics to pick depends on the resource of interest (time, stack space, clock-cycles, etc.) and the purpose of the cost semantics. One approach can work well for proving the soundness of a resource bound and another for implementing an interpreter that measures the resource usage according to the model or relating a high-level cost model to the compiled code in a compiler. Fortunately, there are connections between different cost semantics and the cost models defined by most variants are equivalent or at least compatible. This makes it possible to easily use different cost semantics for different purposes.

In this lecture, we only study a simple sequential evaluation model. However, there are cost semantics for modeling other evaluation strategies like parallel evaluation [Har12] and complex runtime features like garbage collection [NH18]. Such more advanced dynamic behavior is not beyond the scope of automatic resource analysis but not discussed in this course.

2 A Simple Language and its Static Semantics

Syntax We study cost semantics using a simple language that is sufficient to discuss the most interesting points: the simply-typed lambda calculus with fixed points and unit. Since we consider strict evaluation, we restrict fixed points to function types instead of general fixed points. The main properties that we later discuss carry over to other language features.

The types of the language are given by the following grammar. The role of unit in the language is just to provide a base type so that the set of types is not empty. Like in PFPL [Har12], we define a abstract syntax (using abstract binding trees) and a concrete syntax for each syntactic form. The abstract syntax is the actual definition but we sometimes use concrete syntax if it is convenient.

$$\begin{array}{lcll} \tau & ::= & \mathrm{arr}(\tau_1;\tau_2) & \tau_1 \to \tau_2 \\ & & \mathrm{unit} & \mathbf{1} \end{array}$$

Expressions are defined as follows. We have variables x, function applications $\mathrm{app}(e_1;e_2)$, recursive function abstraction $\mathrm{fun}\{\tau,\tau'\}(f,x.e)$, and the unit value triv. The reason for including recursive functions (instead of lambda abstraction) is that we can discuss issues that arise for diverging computations.

$\boxed{\Gamma \vdash e : \tau \qquad \text{“expression } e \text{ has type } \tau \text{ in context } \Gamma\text{”}}$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ (\text{T:Var}) \qquad \frac{}{\Gamma \vdash \mathrm{triv} : \mathrm{unit}}\ (\text{T:Unit})$$

$$\frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathrm{app}(e_1; e_2) : \tau'}\ (\text{T:App}) \qquad \frac{\Gamma, f : \tau \to \tau', x : \tau \vdash e : \tau'}{\Gamma \vdash \mathrm{fun}\{\tau, \tau'\}(f, x.e) : \tau \to \tau'}\ (\text{T:Fun})$$

Figure 1: Type rules.

The types τ and τ' in the form for recursive function abstraction ensure that every expression has a unique type (under a given type context). This is not a crucial property and we could drop the type annotations. However, it is a desirable property to have a canonical type (in our case even a unique type) for each expression that can be assigned a type.

$$\begin{array}{lcll} e & ::= & x & x \\ & & \mathrm{app}(e_1; e_2) & e_1(e_2) \\ & & \mathrm{fun}\{\tau, \tau'\}(f, x.e) & \mathsf{fun}\ f\, x = e \\ & & \mathrm{triv} & \langle\rangle \end{array}$$

Static semantics We now define a standard type system, the static semantics of our language. The benefit of working with a typed language in this lecture is that we can avoid dealing with failure in the dynamic semantics. Moreover, it is a good warm up for the more complex type systems that we will study in the following lectures.

The rules in Figure 1 inductively define a type judgement

$$\Gamma \vdash e : \tau$$

that reads expression e has type τ in type context Γ. As usual, a context Γ is a finite mapping from variables to types.

$$\Gamma \quad ::= \quad \cdot \mid \Gamma, x : \tau$$

The order in which variables appear in a context is irrelevant. A well-formed context contains each variable only once. In particular, if we write $\Gamma' = \Gamma, x : \tau$ then $\Gamma'(x) = \tau$.

We can show by induction on the type derivation that every expression has at most one type under a given context.

Theorem 1. *If* $\Gamma \vdash e : \tau_1$ *and* $\Gamma \vdash e : \tau_2$ *then* $\tau_1 = \tau_2$.

Abbreviations We can now define lambda abstraction $\mathrm{lam}\{\tau\}(x.e)$, or $\lambda(x : \tau)\, e$ in concrete syntax, as syntactic sugar. We would like to define something like

$$\mathrm{lam}\{\tau\}(x.e) \triangleq \mathrm{fun}\{\tau, \tau'\}(f, x.e)$$

for a fitting τ'. However, the right choice of τ' depends on the context, more precisely, the type context. So the abbreviation $\mathrm{lam}\{\tau\}(x.e)$ stands for (potentially infinitely) many expressions. However, we are only interested in well-typed expressions, that is, expressions e that come with a typing derivation $\Gamma \vdash e : \tau$.

For a given type context Γ and an expression e_0 that contains an abbreviated from $e \triangleq \mathrm{lam}\{\tau\}(x.e')$, we can find at most one type τ' such that $\Gamma \vdash e_0' : \tau_0$ for some type τ_0, where the expression e_0’ is e_0 in which e' is replaced with $\mathrm{fun}\{\tau, \tau'\}(f, x.e')$. It is a good exercise to convince yourself that at most one such τ' exists. If such a τ' does not exist then $\mathrm{lam}\{\tau\}(x.e)$ stands for $\mathrm{fun}\{\tau, \mathbf{1}\}(f, x.e')$ (or some other arbitrary pick for τ').

$\boxed{e \text{ val} \qquad \text{“expression } e \text{ is a value”}}$

$$\frac{}{\text{fun}\{\tau,\tau'\}(f,x.e)\ \text{val}}\ (\text{V:FUN}) \qquad\qquad \frac{}{\text{triv val}}\ (\text{V:UNIT})$$

Figure 2: Values.

Another useful abbreviation is a let binding, which we define as follows.

$$\text{let}(e_1;x.e_2) \triangleq \text{app}(\text{lam}\{\tau\}(x.e_2);e_1)$$

As before, the type τ has to be picked correctly to match the type of e_1 in the current context. The concrete syntax for let bindings is let $x = e_1$ in e_2.

3 Structural Cost Semantics

The first cost semantics that we study is a structural dynamic semantics, sometimes also called small-step semantics. As the latter name suggests, a structural dynamics defines a notion of steps and these steps can used to define the cost of an evaluation. Before we can define the semantics, we need to introduce substitutions.

Free Variables For an expression e, the set of free variables $FV(e)$ is the set of variables that appear unbound in e. Formally

$$\begin{array}{lcl} FV(x) & = & \{x\} \\ FV(t(\vec{x}_1.e_1,\ldots,\vec{x}_n.e_n)) & = & \bigcup_{1\le i\le n} FV(e_i) \setminus \{x_{i1},\ldots,x_{im_i}\} \end{array}$$

Note that a variable can appear both free and bound in an expression. Consider for example the expression $e \triangleq (\lambda(x:\tau)\,x)(x)$. The first appearance of x is bound but second appearance is not bound. Thus we have $x \in FV(e)$.

An important concept is that the names of bound variables are not relevant as long as there are no name clashes. As a result, we identify expressions that only defer in the names of bound variables. Details can be found in PFPL [Har12].

Substitutions Let x be a variable and let e and e' be expressions. Intuitively, a substitution $[e'/x]e$ replaces every free occurrence of the variable x in e with e'. We define

$$\begin{array}{lcll} [e'/x]x & = & e' & \\ {[e'/x]y} & = & y & \text{if } y \neq x \\ {[e'/x]t(\vec{x}_1.e_1,\ldots,\vec{x}_n.e_n)} & = & t(\vec{x}_1.e'_1,\ldots,\vec{x}_n.e'_n) & \text{where } x_{ij} \notin FV(e') \text{ for all i,j} \\ & & & \text{and } e'_i = \begin{cases} e_i & \text{if } \exists j\,.\,x = x_{ij} \\ [e'/x]e_i & \text{otherwise} \end{cases} \end{array}$$

The condition $x_{ij} \notin FV(e')$ in the third case prevents name-capturing substitutions which would have an unintended behavior. As a result, the substitution $[e'/x]e$ is not defined for some expressions. However, in the following we still assume that substitution is always defined. This is justified since bound variables can be renamed without changing the meaning of an expression.

Similarly, we define the simultaneous substitution of multiple expressions, written as

$$[e_1,\ldots,e_n/x_1,\ldots,x_n]e\,.$$

Here, the free variable x_i in the expression e are replaced by the expressions e_i for all $1 \le i \le n$. Simultaneously, means that there substitution within the expressions e_i are not performed. This might happen if we would perform one substitution after another.

$\boxed{e \longmapsto e' \qquad \text{“expression } e \text{ steps to expression } e'\text{”}}$

$$\frac{e_1 \longmapsto e_1'}{\mathrm{app}(e_1; e_2) \longmapsto \mathrm{app}(e_1'; e_2)}\ (\text{S:App1}) \qquad \frac{e_1\ \mathrm{val} \qquad e_2 \longmapsto e_2'}{\mathrm{app}(e_1; e_2) \longmapsto \mathrm{app}(e_1; e_2')}\ (\text{S:App2})$$

$$\frac{e_2\ \mathrm{val}}{\mathrm{app}(\mathrm{fun}\{\tau,\tau'\}(f,x.e); e_2) \longmapsto [\mathrm{fun}\{\tau,\tau'\}(f,x.e), e_2/f,x]e}\ (\text{S:AppFun})$$

$\boxed{e \longmapsto^n e' \qquad \text{“expression } e \text{ steps to expression } e' \text{ in } n \text{ steps”}}$

$$\frac{e \longmapsto e'' \qquad e'' \longmapsto^n e'}{e \longmapsto^{n+1} e'}\ (\text{N:Step}) \qquad \frac{}{e \longmapsto^0 e}\ (\text{N:Base})$$

Figure 3: Vanilla structural dynamic semantics.

Values A *value* is an expression that cannot be evaluated further. It is a result of a terminating evaluation that does not go wrong. As we will see, the type system ensures that evaluations do not go wrong. The values of our simple language are lambda abstractions and the unit value. They are formally defined in Figure 2.

3.1 Vanilla Structural Dynamics

A structural dynamic semantics is a transition system. The states of the transition system are expressions and transitions are inductively defined by transition rules. A transition corresponds to a step in the evaluation of the expression. The evaluation of an expression consists of a sequence of evaluation steps that is either infinite or ends with a value or a stuck state (i.e., a malformed expression).

We denote a step from expression e to e' by $e \longmapsto e'$. Figure 3 defines the evaluation rules. The only expressions that can take a step are function applications $\mathrm{app}(e_1; e_2)$. There are no rules for values or variables. We never need to evaluate a variable because we only evaluated closed expressions, that is, expressions that do not contain free variables. The rule S:App1 states that the transition for the state (or expression) $\mathrm{app}(e_1; e_2)$ is determined by the next transition of the state e_1. If e_1 is a value and thus does not have transitions then rule S:App2 states that the next transition is determined by expression e_2. If both e_1 and e_2 are values and e_1 is a function then rule S:AppFun is applicable and the next step is the substitution that is described in the rule.

The judgement $e \longmapsto^n e'$, defined in Figure 3, denotes that expression e steps to e' in n steps. We write $e \longmapsto^* e'$ if there exists an n such that $e \longmapsto^n e'$. The number of steps n it takes for an evaluation to step to a value is a natural measure of the time cost of the evaluation. It is important to note that this cost (and the resulting value) is uniquely defined.

Lemma 1. *If* $e \longmapsto^{n_1} v_1$, $e \longmapsto^{n_2} v_2$, v_1 val*, and* v_2 val *then* $v_1 = v_2$ *and* $n_1 = n_2$.

Type Safety We have mentioned that well-typed expressions do not get stuck. This fact is known as type safety. A well-typed expression is an expression e for which we can find a context Γ and a type τ such that $\Gamma \vdash e : \tau$. When evaluating expressions, we are interested in well-typed, closed expressions, that is, expressions $\cdot \vdash e : \tau$ that can be typed with an empty context $\cdot$. We usually just write $e : \tau$ instead of $\cdot \vdash e : \tau$.

Structural dynamic semantics is a particularly convenient semantics for formulating and proving type safety using *progress* and *preservation*.

Theorem 2 (Progress). *If $e : \tau$ then either e val or there exists an expression e' such that $e \longmapsto e'$.*

Theorem 3 (Preservation). *If $e : \tau$ and $e \longmapsto e'$ then $e' : \tau$.*

We can prove progress by induction on the type judgment $e : \tau$ and preservation by induction on the evaluation judgment $e \longmapsto e'$.

Now we can formally define the evaluation cost of well-typed closed expressions.

Definition 1. *Let $e : \tau$ be a closed expression. The evaluation cost of e is n if $e \longmapsto^n e'$ for some e' and ∞ otherwise.*

3.2 Resource Metrics

Our vanilla structural dynamics provides an adequate cost model for evaluation time. However, we could criticize that it is not quite realistic to account the same cost for each evaluation step. In particular, substitution and finding the right spot in an expression for preforming the next step seem like they could be quite costly. Moreover, it seems questionable to assign cost 0 to the evaluation of values. Don't we have to at least look at the complete expression to decide if it is a value? These issues could potentially be fixed by switching to a different transition system (an abstract machine) that is more in line with an actual implementation. However, we are not only interested in time but also in other resources like memory usage or specific metrics like the number of function calls performed. Switching to a different transition system would not directly address this.

To make the resource accounting more general, we introduce a resource metric that assigns a cost to each evaluation step. This cost can be negative to model that resources become available (like deallocation of memory) and depend on the current expression. However, in view of our goal of performing a resource analysis, we require that a metric only depends on statically available information such as the size of a tuple and not on dynamic information such as the length of a list. However, you could also include dynamic information like number of substituted variables in the metric if your main goal is to have a cost model that matches your evaluation strategy.

To keep things simple, we work with a metric M that associates a single constant with each syntactic form of the language.

$$M : \{\mathsf{var}, \mathsf{app}, \mathsf{fun}, \mathsf{trv}\} \to \mathbb{Q}$$

There are two ways of integrating the resource metric with the evaluation dynamics: resource effects and resource safety.

3.3 Resource Effects

The idea of resource effects is to augment the dynamic semantics with additional information about resource usage that is recorded during the evaluation but does not influence the transitions that are taken.

Figure 4 defines the step transition of the form $M \vdash e \longmapsto e' \mid q$, where e and e' are expressions and $q \in \mathbb{Q}$. The intended meaning is that e steps to e' incurring cost q under metric M. If q is negative then $-q$ resources become available. Like for the vanilla structural dynamics, we define a many-steps judgement $M \vdash e \longmapsto^* e' \mid q$ where q is the sum of the cost of the individual steps.

The following lemma shows that the resource metric does influence evaluation.

Lemma 2. *Let M and M' be two metrics. If $M \vdash e \longmapsto e' \mid q$ then $M' \vdash e \longmapsto e' \mid q'$ for some q'.*

We can prove the following relations to the vanilla structural dynamics.

Lemma 3. *There exists a q such that $M \vdash e \longmapsto^* e' \mid q$ if and only if $e \longmapsto^* e'$.*

From Lemma 3, it follows that type soundness directly carries over if we replace the vanilla dynamics with the version with resource effects.

$\boxed{M \vdash e \longmapsto e' \mid q \qquad \text{"expression } e \text{ steps to expression } e' \text{ with cost } q\text{"}}$

$$\frac{M \vdash e_1 \longmapsto e_1' \mid q}{M \vdash \mathrm{app}(e_1; e_2) \longmapsto \mathrm{app}(e_1'; e_2) \mid q}\ (\textsc{Se:App1}) \qquad \frac{e_1 \text{ val} \qquad M \vdash e_2 \longmapsto e_2' \mid q}{M \vdash \mathrm{app}(e_1; e_2) \longmapsto \mathrm{app}(e_1; e_2') \mid q}\ (\textsc{Se:App2})$$

$$\frac{e_2 \text{ val}}{M \vdash \mathrm{app}(\mathrm{fun}\{\tau,\tau'\}(f,x.e); e_2) \longmapsto [\mathrm{fun}\{\tau,\tau'\}(f,x.e), e_2/f,x]e \mid M(\mathsf{app})}\ (\textsc{Se:AppLam})$$

$\boxed{M \vdash e \longmapsto^* e' \mid q \qquad \text{"expression } e \text{ steps to expression } e' \text{ with net cost } q\text{"}}$

$$\frac{M \vdash e \longmapsto e'' \mid q_1 \qquad M \vdash e'' \longmapsto^* e' \mid q_2}{M \vdash e \longmapsto^* e' \mid q_1 + q_2}\ (\textsc{Ne:Step}) \qquad \frac{}{M \vdash e \longmapsto^* e \mid 0}\ (\textsc{Ne:Base})$$

Figure 4: Structural dynamic semantics with resource effects.

Theorem 4. *Let $M(\mathsf{app}) = 1$. Then $e \longmapsto^n e'$ iff $M \vdash e \longmapsto^* e' \mid n$.*

We point out that the cost constants fun, trv, and var have no influence on the cost of the evaluation and it is not immediately clear how to change this. However, as we will see, these constants are relevant cost semantics for evaluation dynamics.

High-Water Mark If a metric defines non-negative cost for each step then the resource usage is monotonic in the number of steps. If some cost are negative then this is not the case and we are usually interested in the *high-water mark* resource usage, that is, the maximal amount of resources that are used at some point during the evaluation.

Definition 2. *Let $e : \tau$ be a closed expression. The high-water mark resource usage of e under metric M is defined as $\max\{q \mid M \vdash e \longmapsto^* e' \mid q\}$ if the maximum exists and ∞ otherwise.*

Note that the high-water mark can be finite for diverging computations that, for example, alternate between allocating and freeing resources.

3.4 Resource Safety

Another way to model resource usage is by *resource safety*. Instead of resource usage being an effect of the evaluation, we view resources as fuel that gets consumed during the evaluation. If the fuel is not sufficient to cover the cost then the evaluation gets stuck. So in contrast to resource effects, resources can influence the evaluation and type safety does only hold if we start with a sufficient amount of resources.

Figure 5 defines the judgment $M \vdash \langle e \mid q\rangle \longmapsto^* \langle e' \mid q'\rangle$, which states that, with $q \in \mathbb{Q}_{\geq 0}$ available resources, expression e evaluates to expression e' and $q' \in \mathbb{Q}_{\geq 0}$ available resources. The difference $q - q'$ of initial and remaining resources is the net cost of the evaluation. For an expression, there are now many possible initial resources we can start the evaluation with. However, the net cost of the evaluation is invariant. Note that Lemma 5 does not hold for $c < 0$.

Lemma 4. *If $M \vdash \langle e \mid q\rangle \longmapsto^n \langle e' \mid q'\rangle$, $M \vdash \langle e \mid p\rangle \longmapsto^n \langle e'' \mid p'\rangle$. Then $e' = e''$ and $q - q' = p - p'$.*

Lemma 5. *If $M \vdash \langle e \mid q\rangle \longmapsto^* \langle e' \mid q'\rangle$ and $c \geq 0$ then $M \vdash \langle e \mid q + c\rangle \longmapsto^* \langle e' \mid q' + c\rangle$.*

Let us examine how the resource safety dynamics relates to the version with resource effects.

Theorem 5. *Let $e : \tau$ be a closed expression. If $M \vdash \langle e \mid q\rangle \longmapsto^* \langle e' \mid q'\rangle$ then $M \vdash e \longmapsto^* e' \mid q - q'$.*

$$\boxed{M \vdash \langle e \mid q \rangle \longmapsto \langle e' \mid q' \rangle \qquad \text{“with } q \geq 0 \text{ available resources, } e \text{ steps to } e' \text{ and } q' \geq 0 \text{ resources”}}$$

$$\frac{M \vdash \langle e_1 \mid q \rangle \longmapsto \langle e_1' \mid q' \rangle}{M \vdash \langle \mathsf{app}(e_1; e_2) \mid q \rangle \longmapsto \langle \mathsf{app}(e_1'; e_2) \mid q \rangle} \text{(Ss:App1)}$$

$$\frac{e_1 \text{ val} \qquad M \vdash \langle e_2 \mid q \rangle \longmapsto \langle e_2' \mid q' \rangle}{M \vdash \langle \mathsf{app}(e_1; e_2) \mid q \rangle \longmapsto \langle \mathsf{app}(e_1; e_2') \mid q' \rangle} \text{(Ss:App2)}$$

$$\frac{e_2 \text{ val} \qquad q' = q - M(\mathsf{app}) \geq 0}{M \vdash \langle \mathsf{app}(\mathsf{fun}\{\tau, \tau'\}(f, x.e); e_2) \mid q \rangle \longmapsto \langle [\mathsf{fun}\{\tau, \tau'\}(f, x.e), e_2 / f, x] e \mid q' \rangle} \text{(Ss:AppFun)}$$

$$\boxed{M \vdash \langle e \mid q \rangle \longmapsto^* \langle e' \mid q' \rangle \qquad \text{“with } q \geq 0 \text{ available resources, } e \text{ steps to } e' \text{ and } q' \geq 0 \text{ resources”}}$$

$$\frac{M \vdash \langle e \mid q \rangle \longmapsto \langle e'' \mid q'' \rangle \qquad M \vdash \langle e'' \mid q'' \rangle \longmapsto^* \langle e' \mid q' \rangle}{M \vdash \langle e \mid q \rangle \longmapsto^* \langle e' \mid q' \rangle} \text{(Ns:Step)}$$

$$\frac{}{M \vdash \langle e \mid q \rangle \longmapsto^* \langle e \mid q \rangle} \text{(Ns:Base)}$$

Figure 5: Structural dynamic semantics with resource safety.

Theorem 6. *Let $e : \tau$ be a closed expression. If $M \vdash e \longmapsto^* e' \mid q$ then there exist $p, p' \in \mathbb{Q}_{\geq 0}$ such that $p - p' = q$ and $M \vdash \langle e \mid p \rangle \longmapsto^* \langle e' \mid p' \rangle$.*

You may have expected a stronger version of Theorem 6 in which we conclude $M \vdash \langle e \mid q \rangle \longmapsto^* \langle e' \mid 0 \rangle$ (for $q \geq 0$). However, this only holds for metrics M without negative cost. A more general version of this question is how two evaluations $M \vdash \langle e_1 \mid q \rangle \longmapsto^* \langle e_2 \mid q' \rangle$ and $M \vdash \langle e_2 \mid p \rangle \longmapsto^* \langle e_3 \mid p' \rangle$ compose. We know that there exists $r, r' \in \mathbb{Q}_{\geq 0}$ such that $M \vdash \langle e_1 \mid r \rangle \longmapsto^* \langle e_3 \mid r' \rangle$ but it is not clear how we can express them in terms of q, q', p, and p'. We will revisit this question in Section 5.

As you can infer from the formulation of Theorem 6, type safety with the resource safety semantics is slightly different from the vanilla version: If you do not have enough resources then you can get stuck even if the expression is well typed. We could state progress as follows.

If $e : \tau$ then e val or there exists q, q', and e such that $M \vdash \langle e \mid q \rangle \longmapsto \langle e' \mid q' \rangle$.

High-Water Mark With resource safety, the high-water mark is the minimal amount of resources that we need to not run out of resources. The question is how to come up with a definition that captures this idea. A first idea would be something like this

$$\min\{q \mid \forall p, p', e'. M \vdash \langle e \mid p \rangle \longmapsto^* \langle e' \mid p' \rangle \implies \exists q'. M \vdash \langle e \mid q \rangle \longmapsto^* \langle e' \mid q' \rangle\}$$

However, this is not what we want since it would result in the high-water mark 0 for expressions such as $\mathsf{fun}(f, x.f(x))(\mathsf{triv})$, which steps to itself. We could fix this by adding the number of steps to the judgment.

$$\min\{q \mid \forall p, p', e', n. M \vdash \langle e \mid p \rangle \longmapsto^n \langle e' \mid p' \rangle \implies \exists q'. M \vdash \langle e \mid q \rangle \longmapsto^n \langle e' \mid q' \rangle\}$$

This works but is unsatisfactory since we have talk about the resource usage and the number of steps, which is itself a notion of time usage. Another possibility would be to focus on terminating evaluations only.

$$\min\{q \mid \exists v, q'. v \text{ val and } M \vdash \langle e \mid q \rangle \longmapsto^* \langle v \mid q' \rangle\}$$

This is nice and simple but the downside is that we do not define a high-water mark for diverging evaluations. The definition we pick directly captures our initial intuition about not running out of resources.

Definition 3. *Let $e : \tau$ be a closed expression. The high-water mark resource usage $hwm_M(e)$ of e under metric M is defined as the smallest q such that the following holds (or ∞ if no such q exists). For all e', q', if $\langle e \mid q \rangle \longmapsto^* \langle e' \mid q' \rangle$ and not e' val then $\langle e' \mid q' \rangle \longmapsto \langle e'' \mid q'' \rangle$ for some e'', q''.*

Note that the definition is a generalization of the previous definition for values. Of course, we expect the following theorem. However, it is not directly clear if the statement is indeed correct and how we can prove it. So you need to prove it on the next homework assignment.

Theorem 7. *Let $e : \tau$ be a closed expression and M a metric. The maximum $m = \max\{q \mid M \vdash e \longmapsto^* e' \mid q\}$ exists if and only if $hwm_M(e)$ is finite. In this case $hwm_M(e) = m$.*

4 Cost Annotations

The simple lambda calculus that we studied in this lecture so far is a bit too simple to illustrate some of the issues with high-water marks. The problem is that we only have one relevant cost constant, which cannot be positive and negative at the same time. However, the previous discussion and theorems still hold for more interesting language extensions.

A simple extension that is flexible enough to demonstrate most interesting resource behaviors are cost annotations that allow the use to define the cost of certain parts of the program. To add cost annotations, we extend the expression with an additional syntactic form tick$\{q\}$ where $q \in \mathbb{Q}$.

$$\begin{array}{lll} e & ::= & \ldots \\ & & \text{tick}\{q\} \quad \mathsf{tick}\,(q) \end{array}$$

Ticks only influence the resource usage and do not have a computational effect. The type rule for ticks is defined as follows.

$$\frac{}{\Gamma \vdash \text{tick}\{q\} : \text{unit}} \;(\text{T:TICK})$$

We extend resource metrics to be functions

$$M : (\{\mathsf{var}, \mathsf{app}, \mathsf{fun}, \mathsf{trv}\} \cup \{\mathsf{tick}_q \mid q \in Q\}) \to \mathbb{Q}$$

The rules of the dynamics semantics can be extended in a straightforward way. For example, the structural dynamics with presource effects is extended with the following rule.

$$\frac{}{M \vdash \text{tick}\{q\} \longmapsto \text{triv} \mid M(q)} \;(\text{SE:TICK})$$

The rule for the vanilla structural semantics is simply

$$\frac{}{\text{tick}\{q\} \longmapsto \text{triv}} \;(\text{S:TICK})$$

and the structural dynamics with resource safety is extended with the following rule.

$$\frac{p' = p - M(\mathsf{tick}_q) \geq 0}{M \vdash \langle \text{tick}\{q\} \mid p \rangle \longmapsto \langle \text{triv} \mid p' \rangle} \;(\text{SS:TICK})$$

All previously stated theorems and lemmas are still true with this extension.

Tick Metric The *tick metric M_T* is the metric with $M_T(\mathsf{tick}_q) = q$ and $M_T(K) = 0$ for $K \neq \mathsf{tick}_q$. It only accounts for the cost annotations that have been inserted in the program by a user.

$$\boxed{M \vdash e \Downarrow^{q}_{q'} v \qquad \text{“with } q \geq 0 \text{ available resources, } e \text{ valuates } v \text{ and } q' \geq 0 \text{ resources remain”}}$$

$$\frac{q' = q - M(\mathsf{trv}) \geq 0}{M \vdash \text{triv} \Downarrow^{q}_{q'} \text{triv}}\ (\textsc{Es:Unit}) \qquad \frac{q' = q - M(\mathsf{fun}) \geq 0}{M \vdash \text{fun}\{\tau,\tau'\}(f,x.e) \Downarrow^{q}_{q'} \text{fun}\{\tau,\tau'\}(f,x.e)}\ (\textsc{Es:Fun})$$

$$\frac{M \vdash e_2 \Downarrow^{p}_{p'} v_2 \qquad \begin{array}{c} M \vdash e_1 \Downarrow^{q}_{p} \text{fun}\{\tau,\tau'\}(f,x.e) \\ M \vdash [\text{fun}\{\tau,\tau'\}(f,x.e), v_2/f,x]e \Downarrow^{r}_{q'} v \end{array} \qquad r = p' - M(\mathsf{app}) \geq 0}{M \vdash \text{app}(e_1;e_2) \Downarrow^{q}_{q'} v}\ (\textsc{Es:App})$$

Figure 6: Evaluation dynamics with resource safety.

5 Evaluation Dynamics

The second class of cost semantics we study is based on evaluation dynamics (or big-step semantics). In contrast to structural dynamics, evaluation dynamics does not enjoy a notion of steps, so the term big-step is somewhat misleading. The vanilla evaluation dynamics defines a judgment $e \Downarrow v$ where e is an expression and v is a value. The judgment for values is same as for the structural dynamics and defined in Figure 2.

We skip the inductive definition of $e \Downarrow v$. You can obtain the rules by removing the resource annotations for the rules in Figure 6. We can prove the following theorem.

Theorem 8. *Let $e : \tau$ be a closed expression then $e \Downarrow v$ if and only if $e \longmapsto^* v$ and v* val.

Note that it is not directly possible to formulate type safety with an evaluation dynamics since there is no distinction between divergence and failure: there is no judgment for both.[1]

5.1 Resource Safety

We first study an evaluation dynamics that implements the idea of resources safety. Figure 6 inductively defines the judgment $M \vdash e \Downarrow^{q}_{q'} v$ with the following intended meaning. If $q \geq 0$ resources are available then expression e evaluation to value v and q' resources are available after the evaluation. Like for the structural dynamics, we can show that the net cost is invariant.

Lemma 6. *Let $e : \tau$ be a closed expression. If $M \vdash e \Downarrow^{q}_{q'} v$ and $M \vdash e \Downarrow^{p}_{p'} v'$ then $q - q' = p - p'$.*

Lemma 7. *Let $e : \tau$ and $M \vdash e \Downarrow^{q}_{q'} v$. Then $M \vdash e \Downarrow^{q+c}_{q'+c} v$.*

A particularly nice feature of an evaluation dynamics is that the shape of the rules matches to the shape of the type rules. This simplifies inductive proofs of statements that involve an evaluation judgment and a type judgment for the same expression.

With an evaluation dynamics we can also assign cost to values. In our simple language, we do not have composed values likes lists. As we will see later in the course, an evaluation dynamics naturally assigns non-constant cost to such values as the evaluation rules will evaluate the subexpressions of the value. So for a list of values, this would lead to a cost that is linear in the size of the value. In contrast, the structural dynamics assigns cost 0 to all values. This is not realistic as an implementation would have to traverse the list to check if all elements are indeed values.

When we relate the evaluation dynamics with resource safety to cost semantics based on a structural dynamics then we have to assume a metric M that does not assign cost to value evaluation.

[1] To distinguish divergence and failure, we could introduce another evaluation judgment for failures. However, this would drastically increase the number of rules and diminish some of the benefits of evaluation dynamics.

$$\boxed{M \vdash e \Downarrow v \mid (q, q') \qquad \text{“}e \text{ valuates } v \text{ with high-water mark } q \geq 0 \text{ and } q' \geq 0 \text{ remaining resources”}}$$

$$\frac{}{M \vdash \mathsf{triv} \Downarrow \mathsf{triv} \mid M(\mathsf{trv})}\ (\textsc{Ee:Unit}) \qquad \frac{}{M \vdash \mathsf{fun}\{\tau,\tau'\}(f, x.e) \Downarrow \mathsf{fun}\{\tau,\tau'\}(f, x.e) \mid M(\mathsf{fun})}\ (\textsc{Ee:Fun})$$

$$\frac{\begin{array}{c} M \vdash e_1 \Downarrow \mathsf{fun}\{\tau,\tau'\}(f, x.e) \mid (q_1, q_1') \\ M \vdash e_2 \Downarrow v_2 \mid (q_2, q_2') \qquad M \vdash [\mathsf{fun}\{\tau,\tau'\}(f, x.e), v_2/f, x]e \Downarrow v \mid (q, q') \end{array}}{M \vdash \mathsf{app}(e_1; e_2) \Downarrow v \mid (q_1, q_1') \cdot (q_2, q_2') \cdot M(\mathsf{app}) \cdot (q, q')}\ (\textsc{Ee:App})$$

Figure 7: Evaluation dynamics with resource effects.

Theorem 9. *Let M be a metric such that $M(\mathsf{trv}) = M(\mathsf{fun}) = 0$ and let $e : \tau$ be an expression. Then $M \vdash e \Downarrow^{q}_{q'} v$ if and only if $M \vdash \langle e \mid q \rangle \longmapsto^* \langle v \mid q' \rangle$ and v val.*

The *high-water mark resource* usage can only be defined for terminating evaluations. Let $e : \tau$ be a closed expression. The high-water mark resource cost is

$$\min\{q \mid \exists v, q'.\, v \text{ val and } M \vdash e \Downarrow^{q}_{q'} v\}$$

if the minimum exists and undefined otherwise. From Theorem 9, we derive the following corollary.

Corollary 1. *Let M be a metric such that $M(\mathsf{trv}) = M(\mathsf{fun}) = 0$ and let $e : \tau$ be an expression. If $m = \min\{q \mid \exists v, q'.\, v$ val and $M \vdash e \Downarrow^{q}_{q'} v\}$ exists then $hwm_M(e) = m$.*

Downsides of Resource Safety There are a number of issues with using resource safety to define the resource usage of programs. One issue is that the evaluation judgement is non-deterministic, that is, for a given expression e, we have different evaluation judgements $M \vdash e \Downarrow^{q}_{q'} v$ and $M \vdash e \Downarrow^{p}_{p'} v$ where $p \neq q$. This is partially mitigated by the fact that there exists a canonical judgement that has a minimal amount of initial resources. However, the evaluation rules in Figure 6 do not help us to find this canonical judgement. Therefore, if we would like to implement an interpreter based on the evaluation rules with resource safety then we cannot use it to *measure* the high-watermark resource usage of programs. We rather have to start with some fuel that we have to pick before we run the interpreter. Note that these issues equally apply to resource safety in the evaluation dynamics and the structural dynamics.

5.2 Resource Effects

In this subsection, we develop an evaluation dynamics that uses resource effects. The first idea that comes to mind is to define a judgment of the form $M \vdash e \Downarrow^{q} v$, where $q \geq 0$ is the high-water mark resource usage. But how would we compose the cost of two evaluation $M \vdash e_1 \Downarrow^{q_1} v$ and $M \vdash e_2 \Downarrow^{q_2} v$ in the rule for function application? Both, taking the sum $q_1 + q_2$ of the cost and the maximum $\max(q_1, q_2)$ of the subexpressions is incorrect. The high-water mark depends on the resources that are available after evaluating e_1. If we have q_2 resources available after the first evaluation, that is, $M \vdash \langle e_1 \mid q_1 \rangle \longmapsto^* \langle v \mid q_2 \rangle$, then q_1 seems reasonable for the high-water mark. However, if we do not have any resources left, that is, $M \vdash \langle e_1 \mid q_1 \rangle \longmapsto^* \langle v \mid 0 \rangle$, then the high-water mark is $q_1 + q_2$ because the resources needed for evaluating e_2 are used in addition to the ones consumed by e_1.

To define a precise composition operation, we need to keep track of both the high-water mark and the remaining resource. Figure 7 defines the judgement $M \vdash e \Downarrow v \mid (q, q')$. The intended meaning is that under metric M, expression e evaluates to value v, the high-water mark resource usage is $q \in \mathbb{Q}_{\geq 0}$ and after the evaluation there are $q' \in \mathbb{Q}_{\geq 0}$ resource units available. The net resource consumption is then $q - q'$. This difference is negative if resources become available during the execution of e.

Resource Monoid It is handy to view the pairs (q,q') in the evaluation judgments as elements of a monoid $\mathcal{Q} = (\mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0}, \cdot)$. A monoid has an associative operation $\cdot$ and a neutral element. The neutral element is $(0,0)$, which means that resources are neither needed before the evaluation nor returned after the evaluation. The operation $(q,q') \cdot (p,p')$ defines how to account for an evaluation consisting of a sequence of two evaluations whose resource consumption are defined by (q,q') and (p,p'), respectively. We define

$$(q,q') \cdot (p,p') = \begin{cases} (q+p-q',\ p') & \text{if } q' \leq p \\ (q,\ p'+q'-p) & \text{if } q' > p \end{cases}$$

If resources are never returned (as with time) then we only have elements of the form $(q,0)$ and $(q,0) \cdot (p,0)$ is just $(q+p,0)$.

We identify a rational number $q \in \mathbb{Q}$ with an element of $\mathcal{Q}$ as follows: $q \geq 0$ denotes $(q,0)$ and $q < 0$ denotes $(0,-q)$. This notation avoids case distinctions in the evaluation rules in Figure 7.

Another way to define the resource monoid is given in the following lemma, which shoes that the resource monoid on natural numbers (instead of non-negative rationals) is the *bicyclic semigroup*.

Lemma 8. $(q,q') \cdot (p,p') = (q+p-\min(q',p),\ p'+q'-\min(q',p))$

Lemma 9. *Let $(q,q') = (r,r') \cdot (s,s')$. We can prove the following statements.*

1. $q \geq r$ *and* $q - q' = r - r' + s - s'$
2. *If $(p,p') = (\bar{r},r') \cdot (s,s')$ and $\bar{r} \geq r$ then $p \geq q$ and $p' = q'$*
3. *If $(p,p') = (r,r') \cdot (\bar{s},s')$ and $\bar{s} \geq s$ then $p \geq q$ and $p' \leq q'$*
4. $(r,r') \cdot ((s,s') \cdot (t,t')) = ((r,r') \cdot (s,s')) \cdot (t,t')$

Properties of the Dynamics A beneficial feature of the evaluation dynamics with resource effects is that the judgment is deterministic in the sense of the following lemma. So the rules are suitable to implement an interpreter that keeps track of the resource usage.

Lemma 10. *Let $e : \tau$ be a closed expression. If $M \vdash e \Downarrow v \mid (q,q')$ and $M \vdash e \Downarrow v' \mid (p,p')$ then $v = v'$ and $(q,q') = (p,p')$.*

We can show the following relations to the evaluation dynamics with resource safety.

Theorem 10. *Let $e : \tau$ be a closed expression. If $M \vdash e \Downarrow^{q}_{q'} v$ then $M \vdash e \Downarrow v \mid (p,p')$ for some p,p' with $p \leq q$ and $p - p' = q - q'$.*

Theorem 11. *Let $e : \tau$ be a closed expression. If $M \vdash e \Downarrow v \mid (q,q')$ then $M \vdash e \Downarrow^{q}_{q'} v$.*

Theorem 10 and Theorem 11 show that the high-water mark defined by resource monoid is identical to the high-water we defined using resource safety.

Corollary 2. *Let $e : \tau$ be a closed expression. We have $M \vdash e \Downarrow v \mid (p,p')$ for some p' if and only if $p = \min\{q \mid \exists v, q'.\ v \text{ val} \text{ and } M \vdash e \Downarrow^{q}_{q'} v\}$.*

5.3 Partial Evaluations

A remaining disadvantage of the evaluation dynamics (in both versions) is that the high-water mark resource usage is only defined for terminating evaluations. To extend the evaluation dynamics to diverging computation, we define the judgment $M \vdash e \Downarrow \circ \mid q$ in Figure 8. It states that under metric M, the *partial evaluation* of expression e leads to high-water mark resource usage q. The idea is that the rule EP:STOP can be used to derive the resource usage for a starting fragment of a (possibly diverging) evaluation. So we can show that the high-water mark of a terminating evaluation always exceeds the high-water mark of a partial evaluation of the same expression.

$\boxed{M \vdash e \Downarrow \circ \mid q \qquad \text{“}e \text{ needs } q \geq 0 \text{ resources at some point in the evaluation”}}$

$$\frac{}{M \vdash e \Downarrow \circ \mid 0}\ (\textsc{Ep:Stop}) \qquad \frac{M \vdash e_1 \Downarrow v \mid q}{M \vdash \mathrm{app}(e_1;e_2) \Downarrow \circ \mid q}\ (\textsc{Ep:App1})$$

$$\frac{M \vdash e_1 \Downarrow v \mid (q_1,q_1') \qquad M \vdash e_2 \Downarrow \circ \mid q_2 \qquad (p,p') = (q_1,q_1') \cdot q_2}{M \vdash \mathrm{app}(e_1;e_2) \Downarrow \circ \mid p}\ (\textsc{Ep:App2})$$

$$\frac{\begin{array}{c} M \vdash e_1 \Downarrow \mathrm{fun}\{\tau,\tau'\}(f,x.e) \mid (q_1,q_1') \qquad M \vdash e_2 \Downarrow v_2 \mid (q_2,q_2) \\ M \vdash [\mathrm{fun}\{\tau,\tau'\}(f,x.e), v_2/f,x]e \Downarrow \circ \mid q \qquad (p,p') = (q_1,q_1') \cdot (q_2,q_2') \cdot M(\mathsf{app}) \cdot q \end{array}}{M \vdash \mathrm{app}(e_1;e_2) \Downarrow \circ \mid p}\ (\textsc{Ep:App3})$$

Figure 8: Partial evaluation dynamics with resource effects.

Theorem 12. *Let $e : \tau$ be a closed expression. If $M \vdash e \Downarrow v \mid (q,q')$ and $M \vdash e \Downarrow \circ \mid p$ then $p \leq q$.*

But what can we say about diverging computations? Let us first define the high-water mark resource usage with partial evaluations.

Definition 4. *Let $e : \tau$ be a closed expression. The high-water mark resource usage of e under metric M is defined as*

- *q if $M \vdash e \Downarrow v \mid (q,q')$ for some v and q', or otherwise*
- $\max\{q \mid M \vdash e \Downarrow \circ \mid q\}$ *if the maximum exists, or otherwise*
- ∞

The following lemmas relates partial evaluation to the structural dynamics with resource effects.

Lemma 11. *Let $e : \tau$. If $M \vdash e \Downarrow \circ \mid q$ for some q then there exists e' such that $M \vdash e \longmapsto e' \mid q$.*

Lemma 12. *Let $e : \tau$ and $n \in \mathbb{N}$. If $q_m = \max\{q \mid \exists e', m \leq n. M \vdash e \longmapsto^m e' \mid q\}$ then $M \vdash e \Downarrow \circ \mid q_m$.*

From Lemma 11 and Lemma 12 we obtain the following theorem. Note that we do not have to require any restrictions on the metric. However, this is only the case because the values in our simple language are atomic. If we were to add lists and respective cost to the language then the theorem would only hold if the cost of evaluating values would always be 0.

Theorem 13. *Let $e : \tau$ be a closed expression. The $m = \max\{q \mid M \vdash e \Downarrow \circ \mid q\}$ exists if and only if $n = \max\{q \mid M \vdash e \longmapsto^* e' \mid q\}$ exists. In this case $n = m$.*

Evaluation-Step Metric Interestingly, we can use our partial evaluation dynamics, to formulate and prove type safety. To this end, let M_S be the step metric, that is, $M_S(\mathsf{trv}) = M_S(\mathsf{fun}) = 0$ and $M_S(\mathsf{app}) = 1$. Then we can show that $q' = 0$ for every judgment $M_S \vdash e \Downarrow v \mid (q,q')$. Additionally, we can prove the following corollary.

Corollary 3. *For $e : \tau$, we have $M_S \vdash e \Downarrow \circ \mid n$ if and only if $e \longmapsto^n e'$ for some n.*

If for a terminating evaluation of cost n we can derive partial evaluations for every $m \leq n$.

Lemma 13. *Let $e : \tau$. If $M_S \vdash e \Downarrow v \mid (n,0)$ and $m \leq n$ then $M_S \vdash e \Downarrow \circ \mid m$.*

We can formulate progress and preservation as follows.

Theorem 14 (Preservation)**.** *If $e : \tau$ and $M_S \vdash e \Downarrow v \mid (n,0)$ then $v : \tau$.*

Theorem 15 (Progress)**.** *If $e : \tau$ and $M_S \vdash e \Downarrow \circ \mid n$ then either $M_S \vdash e \Downarrow v \mid (n,0)$ for some v or $M_S \vdash e \Downarrow \circ \mid n+1$.*

6 Compiling Resource Metrics to Cost Annotations

Resource metrics are a flexible and general method to define the resource usage of a program. However, to study and implement resource analysis, it is simpler to rely on cost annotations only. For one thing, you can imagine that you would like to enable a programmer to define the cost of a program. For another thing, cost annotations can be added to an intermediate language to match the cost that is defined by a resource metric at the source level.

On the second homework assignment, you will define a translation from expressions in our simple language to expression in *let-normal form* with cost annotations, which can be seen as an intermediate language. For a fixed metric M, the resulting expression is equivalent to the source expression in the sense that it evaluates to the same value (if the source expression terminates) and the cost annotations reflect the resource consumption of the source expression under metric M.

In the remainder of the course, we will not consider resource metrics anymore. The translation you carry out in the homework shows that we can do so without loss of generality. For example, the resource effects evaluation rules for the language in this lecture can be defined as follows. Note that the resource metric M is missing from the rules.

$$\frac{}{\mathrm{triv} \Downarrow \mathrm{triv} \mid 0}\ (\text{E:Unit}) \qquad \frac{}{\mathrm{fun}\{\tau,\tau'\}(f,x.e) \Downarrow \mathrm{fun}\{\tau,\tau'\}(f,x.e) \mid 0}\ (\text{E:Fun})$$

$$\frac{\begin{array}{c} e_1 \Downarrow \mathrm{fun}\{\tau,\tau'\}(f,x.e) \mid (q_1,q_1') \\ e_2 \Downarrow v_2 \mid (q_2,q_2') \qquad [\mathrm{fun}\{\tau,\tau'\}(f,x.e), v_2/f,x]e \Downarrow v \mid (q,q') \end{array}}{\mathrm{app}(e_1;e_2) \Downarrow v \mid (q_1,q_1')\cdot(q_2,q_2')\cdot(q,q')}\ (\text{E:App}) \qquad \frac{}{\mathrm{tick}\{q\} \Downarrow \mathrm{triv} \mid q}\ (\text{E:Tick})$$

References

[Har12] Robert Harper. *Practical Foundations for Programming Languages.* Cambridge University Press, 2012.

[NH18] Yue Niu and Jan Hoffmann. Automatic space bound analysis for functional programs with garbage collection. In *22nd International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'18)*, 2018.