Working Title

by

Justin Raymond

A thesis submitted to the
faculty of Wesleyan University
in partial fulfillment of the requirements for the
Degree of Bachelor of Arts
with Departmental Honors in Mathematics and Computer Science

Abstract

Contents

CHAPTER 1

Complexity Analysis

1. Introduction

The efficiency of programs is categorized by how the resource usage of a program increases with the input size in the limit. This is often called the asymptotic efficiency or complexity of a program. Asymptotic efficiency abstracts away the details of efficiency, allowing programs to be compared without knowledge of specific hardware architecture or the size and shape of the programs input (Cormen et al. [2001]). However, traditional complexity analysis is only first-order; the asymptotic efficiency of a program can only be expressed in terms of its input. For example it is not possible to describe the asymptotic efficiency of the function `map:(a -> b) -> [b] -> [a]`, which applies a function to every element in a list, in terms of the complexity of the function mapped over the list. Traditional complexity analysis can only show the asymptotic efficiency of `map` is bounded by the length of the list.

This thesis will build on work by Danner, in which the complexity of an expression is composed of a cost and a potential.

Previous Work.

Danner and Royer [2007], building on the work of others, introduced the idea that the complexity of an expression consists of a cost, representing an upper bound on the time it takes to evaluate the expression, and a potential, representing the cost of future uses of the expression. The notion of a potential is key because it allows the analysis of higher-order expressions. The complexity of a higher order function such as `map` depends on the potential of its argument function. They developed a type system for ATR, a call-by-value version of PCF, that consists of a part restricting the sizes of values of expressions and a part restricting the cost of evaluating a expression. Programs written

in ATR are constrained by the type system as to run in less than type-2 polynomial time. Danner and Royer [2009] extended this work to express more forms of recursion, in particular those required by insertion sort and selection sort.

Danner et al. [2013] utilized the notion of thinking of the complexity of an expression as a pair of a cost and a potential to statically analyze the complexity of a higher-order functional language with structural list recursion. The expressions in the higher-order functional language with structural list recursion, referred to as the source language, are mapped to expressions in a complexity language by a translation function. The translated expression describes an upper bound on the complexity of the original programs.

Danner et al. [2015] built on this work to formalize the extraction of recurrences from a higher-order functional language with structural recursion on arbitrary inductive data types. Arbitrary inductive data types are handled semantically using programmer-specified sizes of data types. Sizes must be programmer-specified because the structure of a data type does not always determine the interpretation of the size of a data type. Also, there exist different reasonable interpretations of size, and some may be preferable to others depending on what is being analyzed. For example, if the size of a list is interpreted as the length of the list, then the complexity `map`$\langle$`f,xs`$\rangle$ will be order length of `xs`. If the size of a list is interpreted as a pair of the length and its largest element, then the complexity will depend on the length of `xs`, the potential of `f`, and the largest element.

Proposed Work.

This thesis will focus on building a catalog of examples of the extraction of recurrences from functional programs using the approach by Danner et al. [2015]. The examples will help compare this approach with other methods, such as those by Avanzini et al. [2015] and Hoffmann and Hofmann [2010], highlighting the strengths and weakness of this approach compared with others. The examples include fold, reversing a list and parametric insertion sort.

We will categorize the recurrences extracted from the example programs to develop a sense of the forms of recurrences produced by the method.

We will develop techniques to massage recurrences into usable forms in the size-based denotational semantics and in the syntax of the complexity language. The recurrences produced by the complexity translation are difficult to understand. For example, a program to reverse a list may be written in the source language as

```
rev(xs) =
  rec(xs
     , Nil ↦ Nil
     , Cons ↦⟨x,⟨xs',r⟩⟩.
               rec(force(r), Nil ↦ Cons(⟨x, Nil⟩)
                           , Cons ↦⟨y,⟨ys,rys⟩⟩.
                                     Cons(⟨y,force(rys)⟩)))
```

and the translated program in the complexity language is

```
‖xs‖_c +_c
   rec(‖xs‖_p
      , Nil ↦ ⟨1, Nil⟩
      , Cons ↦⟨x,⟨xs',r⟩⟩.1 + r_c +_c
              rec(r_p
                 , Nil↦ ⟨1,Nil⟩
                 , Cons↦ ⟨y,⟨ys,rys⟩⟩.
                      ⟨1 + rys_c,Cons(⟨y_p,rys_p⟩)⟩)))
```

After interpreting this in a size based semantics that abstracts lists to lengths, the recurrence may be written as the much more manageable

$$g(n) = 1 + n + g(n-1)$$

Developing a catalog of examples will familiarize us with techniques of simplifying the recurrences and we may be able to formalize the process.

A more distant goal is to develop techniques to produce closed form solutions for the extracted recurrences. There are two ways to approach this. The first is to develop a library of tactics in Coq for transforming the recurrences. The second approach is use

the techniques of Albert et al. [2011], who has developed a framework for generating closed-form upper bounds on recurrences.

CHAPTER 2

Examples

Fast Reverse

Fast reverse is a linear time implementation of a function that reverses a list. A naive implementation of reverse append the head of the list to recursively reversed tail of the list. Fast reverse instead As this is the first example, we will walk through the translation and interpretation in gory detail. In following examples we will relegate the walk-through of the translation to the appendices, where the reader can peruse them, perhaps over a glass of carbenet sauvignon, as a relaxing end to a stressful day.

```
datatype list = Nil of unit | Cons of int × list
```

The following function reverses a list on $\Theta(n)$ time.

```
rev xs = λxs.rec(xs, Nil ↦ λa.a,
                 Cons ↦ ⟨x,⟨xs,r⟩⟩.λa.force(r) Cons⟨x,a⟩) Nil
```

If we write out the match explicitly using splits:

```
rev xs = λxs.rec(xs,
                 Nil ↦ λa.a,
                 Cons↦b.split(b,x.c.split(c,xs'.r.
                          λa.force(r) Cons⟨x,a⟩))) Nil
```

The specification of `rev` is `rev` $[x_0, \ldots, x_{n-1}]$ = $[x_{n-1}, \ldots, x_0]$. The specification of the auxilary function `rec(xs,...)` is `rec(`$[x_0, \ldots, x_{n-1}], \ldots$`)` $[y_0, \ldots, y_{m-1}]$ = $[x_{n-1}, \ldots, x_0, y_0, \ldots, y_{m-1}]$.

Complexity Language.

Translation of ***rev*** *using matching.* The translation into the complexity language proceeds as follows. First we apply the rule $\|\lambda x.e\| = \langle 0, \lambda x.\|e\|\rangle$

$$\|\texttt{rev}\| = \langle 0, \lambda \texttt{xs}.\|\texttt{rec(xs, Nil} \mapsto \lambda \texttt{a.a}$$
$$, \texttt{Cons} \mapsto \langle \texttt{x}, \langle \texttt{xs}, \texttt{r} \rangle \rangle . \lambda \texttt{a.force(r) Cons} \langle \texttt{x}, \texttt{a} \rangle) \texttt{ Nil}\| \rangle$$

The we apply the rule for function application, $\|\texttt{e}_0\ \texttt{e}_1\| = 1 + \|\texttt{e}_0\|_c + \|\texttt{e}_1\|_c +_c \|\texttt{e}_0\|_p \|\texttt{e}_1\|_p$.

$$\|\texttt{rev}\| = \langle 0, \lambda \texttt{xs}.(1 + \|\texttt{xs}\|_c + \|\texttt{rec(...)}\|_c + \|\texttt{Nil}\|_c) +_c \|\texttt{rec(...)}\|_p \|\texttt{Nil}\|_p \rangle$$

We will focus on the translation of the `rec` construct. We apply the rule $\|\texttt{rec(xs,}\overline{C \mapsto x.e_C}\texttt{)}\|$

$$= \|xs\|_c +_c \texttt{ rec}(\|\texttt{xs}\|_p, \overline{C \mapsto x.1 +_c \|e_C\|})$$

$$\|\texttt{rec(xs,...)}\| = \|\texttt{xs}\|_c +_c \texttt{ rec}(\|\texttt{xs}\|_p,$$
$$\texttt{Nil} \mapsto 1 +_c \|\lambda \texttt{a.a}\|,$$
$$\texttt{Cons} \mapsto \langle \texttt{x}, \langle \texttt{xs}, \texttt{r} \rangle \rangle . 1 +_c \|\lambda \texttt{a.force(r) Cons} \langle \texttt{x}, \texttt{a} \rangle\|)$$

$$= \langle 0, \texttt{xs} \rangle_c +_c \texttt{ rec}(\langle 0, \texttt{xs} \rangle_p,$$
$$\texttt{Nil} \mapsto 1 +_c \|\lambda \texttt{a.a}\|,$$
$$\texttt{Cons} \mapsto \langle \texttt{x}, \langle \texttt{xs}, \texttt{r} \rangle \rangle . 1 +_c \|\lambda \texttt{a.force(r) Cons} \langle \texttt{x}, \texttt{a} \rangle\|)$$

$$= \texttt{rec(xs,}$$
$$\texttt{Nil} \mapsto 1 +_c \|\lambda \texttt{a.a}\|,$$
$$\texttt{Cons} \mapsto \langle \texttt{x}, \langle \texttt{xs}, \texttt{r} \rangle \rangle . 1 +_c \|\lambda \texttt{a.force(r) Cons} \langle \texttt{x}, \texttt{a} \rangle\|)$$

The translation of the `Nil` branch is simple.

$$= 1 +_c \|\lambda \texttt{a.a}\|$$
$$= 1 +_c \langle 0, \lambda \texttt{a.}\|\texttt{a}\| \rangle$$
$$= 1 +_c \langle 0, \lambda \texttt{a.} \langle 0, \texttt{a} \rangle \rangle$$
$$= \langle 1, \lambda \texttt{a.} \langle 0, \texttt{a} \rangle \rangle$$

The translation of the `Cons` branch is a slightly more involved.

$$= 1 +_c \|\lambda \texttt{a.force(r) Cons} \langle \texttt{x}, \texttt{a} \rangle\|)$$

$$= 1 +_c \langle 0, \|\lambda \texttt{a.force(r) Cons} \langle \texttt{x}, \texttt{a} \rangle\| \rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, \|\mathtt{force(r)}\ \ \mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \|\mathtt{force(r)}\|_c + \|\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|_c) +_c \|\mathtt{force(r)}\|_p\ \ \|\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|_p\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + (\|\mathtt{r}\|_c +_c \|\mathtt{r}\|_p)_c + \|\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|_c) +_c (\|\mathtt{r}\|_c +_c \|\mathtt{r}\|_p)_p\ \ \|\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|_p\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + \|\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|_c) +_c\ \mathtt{r}_p\ \ \|\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\|_p\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + (\langle\|\langle \mathtt{x}\,,\mathtt{a}\rangle\|_c, \mathtt{Cons}\|\langle \mathtt{x}\,,\mathtt{a}\rangle\|_p\rangle)_c) +_c\ \mathtt{r}_p(\langle\|\langle \mathtt{x}\,,\mathtt{a}\rangle\|_c, \mathtt{Cons}\|\langle \mathtt{x}\,,\mathtt{a}\rangle\|_p\rangle)_p\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + \|\langle \mathtt{x}\,,\mathtt{a}\rangle\|_c) +_c\ \mathtt{r}_p\mathtt{Cons}\|\langle \mathtt{x}\,,\mathtt{a}\rangle\|_p\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + \langle\|\mathtt{x}\|_c + \|\mathtt{a}\|_c, \langle\|\mathtt{x}\|_p, \|\mathtt{a}\|_p\rangle\rangle_c) +_c\ \mathtt{r}_p\mathtt{Cons}\langle\|\mathtt{x}\|_c + \|\mathtt{a}\|_c, \langle\|\mathtt{x}\|_p, \|\mathtt{a}\|_p\rangle\rangle_p\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + \|\mathtt{x}\|_c + \|\mathtt{a}\|_c) +_c\ \mathtt{r}_p\mathtt{Cons}\langle\|\mathtt{x}\|_p, \|\mathtt{a}\|_p\rangle\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + \langle \mathtt{0}\,,\mathtt{x}\rangle_c + \langle \mathtt{0}\,,\mathtt{a}\rangle_c) +_c\ \mathtt{r}_p\mathtt{Cons}\langle\langle \mathtt{0}\,,\mathtt{x}\rangle_p, \langle \mathtt{0}\,,\mathtt{a}\rangle_p\rangle\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c + 0 + 0) +_c\ \mathtt{r}_p\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\rangle$$

$$= \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c) +_c\ \mathtt{r}_p\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\rangle$$

So the translation of the whole `rec` is:

$$\mathtt{rec(xs,\ Nil} \mapsto \langle \mathtt{1}\,, \lambda \mathtt{a} \,.\, \langle \mathtt{0}\,,\mathtt{a}\rangle\rangle,\ \mathtt{Cons} \mapsto \langle \mathtt{x}\,, \langle \mathtt{xs'}\,, \mathtt{r}\rangle\rangle \,.\, \langle 1, \lambda \mathtt{a} \,.\, (1 + \mathtt{r}_c) +_c\ \mathtt{r}_p\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\rangle)$$

We observe that in both cases, the cost of `rec` is 1, so we can simplify $\mathtt{r}_c$ to 1.

$$\mathtt{rec(xs,\ Nil} \mapsto \langle \mathtt{1}\,, \lambda \mathtt{a} \,.\, \langle \mathtt{0}\,,\mathtt{a}\rangle\rangle,\ \mathtt{Cons} \mapsto \langle \mathtt{x}\,, \langle \mathtt{xs'}\,, \mathtt{r}\rangle\rangle \,.\, \langle 1, \lambda \mathtt{a} \,.\, 2 +_c\ \mathtt{r}_p\mathtt{Cons}\langle \mathtt{x}\,,\mathtt{a}\rangle\rangle)$$

We will pick up where we left off with out translation of `rev`.

$$\|\texttt{rev}\| \;=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.\,(1 + \|\texttt{xs}\|_c + \|\texttt{rec(...)}\|_c + \|\texttt{Nil}\|_c) +_c \|\texttt{rec(...)}\|_p \|\texttt{Nil}\|_p \rangle$$

First we will translate the variables.

$$\begin{aligned}\|\texttt{rev}\| \;&=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.(1 + \langle \texttt{0}\,, \texttt{xs}\rangle_c + \|\texttt{rec(...)}\|_c + \langle \texttt{0}\,, \texttt{Nil}\rangle_c) +_c \|\texttt{rec(...)}\|_p \langle \texttt{0}\,, \texttt{Nil}\rangle_p \rangle \\ &=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.(1 + 0 + \|\texttt{rec(...)}\|_c + 0) +_c \|\texttt{rec(...)}\|_p \texttt{Nil} \rangle\end{aligned}$$

We use our translation of `rec(xs,...)` and the fact that the cost of every call to `rec` is 1 to get:

$$\begin{aligned}\|\texttt{rev}\| \;&=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.(1 + 0 + 1 + 0) +_c \|\texttt{rec(...)}\|_p \texttt{Nil} \rangle \\ &=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.2 +_c \texttt{rec(xs}\,,\ \texttt{Nil} \mapsto \langle \texttt{1}\,, \lambda \texttt{a}\,.\langle \texttt{0}\,, \texttt{a}\rangle\rangle\,, \\ &\qquad\qquad \texttt{Cons} \mapsto \langle \texttt{x}\,, \langle \texttt{xs'}\,, \texttt{r}\rangle\rangle\,.\langle 1, \lambda \texttt{a}\,.2 +_c \ \texttt{r}_p \texttt{Cons}\langle \texttt{x}\,, \texttt{a}\rangle\rangle)_p \texttt{Nil} \rangle\end{aligned}$$

So our complete translation of the linear time reversal function is

$$\begin{aligned}\|\texttt{rev}\| \;&=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.(1 + 0 + 1 + 0) +_c \|\texttt{rec(...)}\|_p \texttt{Nil} \rangle \\ &=\; \langle \texttt{0}\,, \lambda \texttt{xs}\,.2 +_c \texttt{rec(xs}\,,\ \texttt{Nil} \mapsto \langle \texttt{1}\,, \lambda \texttt{a}\,.\langle \texttt{0}\,, \texttt{a}\rangle\rangle\,, \\ &\qquad\qquad \texttt{Cons} \mapsto \langle \texttt{x}\,, \langle \texttt{xs'}\,, \texttt{r}\rangle\rangle\,.\langle 1, \lambda \texttt{a}\,.2 +_c \ \texttt{r}_p \texttt{Cons}\langle \texttt{x}\,, \texttt{a}\rangle\rangle)_p \texttt{Nil} \rangle\end{aligned}$$

The interpretation of `rev` is rather dull as the cost of `rev` is always null. Instead of interpreting `rev`, we will interpret `rev xs`. In preparation we will translate `rev xs`.

$$\begin{aligned}\|\texttt{rev xs}\| \;&=\; (1 + \|\texttt{rev}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{rev}\|_p \|\texttt{xs}\|_p \\ &=\; (1 + 0 + \langle \texttt{0}\,, \texttt{xs}\rangle_c) +_c \|\texttt{rev}\|_p \langle \texttt{0}\,, \texttt{xs}\rangle_p \\ &=\; (1 + 0 + 0) +_c \|\texttt{rev}\|_p \texttt{xs} \\ &=\; 1 +_c \|\texttt{rev}\|_p \texttt{xs} \\ &=\; 1 +_c (\lambda \texttt{xs}\,.\texttt{rec(xs}\,,\ \texttt{Nil} \mapsto \langle \texttt{1}\,, \lambda \texttt{a}\,.\langle \texttt{0}\,, \texttt{a}\rangle\rangle\,, \\ &\qquad\qquad \texttt{Cons} \mapsto \langle \texttt{x}\,, \langle \texttt{xs'}\,, \texttt{r}\rangle\rangle\,.\langle 1, \lambda \texttt{a}\,.2 +_c \ \texttt{r}_p \texttt{Cons}\langle \texttt{x}\,, \texttt{a}\rangle\rangle)_p \texttt{Nil})\ \texttt{xs} \\ &=\; 1 +_c \texttt{rec(xs}\,,\ \texttt{Nil} \mapsto \langle \texttt{1}\,, \lambda \texttt{a}\,.\langle \texttt{0}\,, \texttt{a}\rangle\rangle\,, \\ &\qquad\qquad \texttt{Cons} \mapsto \langle \texttt{x}\,, \langle \texttt{xs'}\,, \texttt{r}\rangle\rangle\,.\langle 1, \lambda \texttt{a}\,.2 +_c \ \texttt{r}_p \texttt{Cons}\langle \texttt{x}\,, \texttt{a}\rangle\rangle)_p \texttt{Nil}) \\ &=\; 1 +_c \texttt{rec(xs}\,,\ \texttt{Nil} \mapsto \langle \texttt{1}\,, \lambda \texttt{a}\,.\langle \texttt{0}\,, \texttt{a}\rangle\rangle\,,\end{aligned}$$

$$\texttt{Cons} \mapsto \langle \texttt{x}, \langle \texttt{xs'}, \texttt{r} \rangle \rangle . \langle 1, \lambda \texttt{a} . 2 +_c \ \texttt{r}_p \texttt{Cons} \langle \texttt{x}, \texttt{a} \rangle \rangle)_p \texttt{Nil}$$

$$\|\texttt{rev xs}\| = 1 +_c \texttt{rec(xs, Nil} \mapsto \langle 1, \lambda \texttt{a} . \langle 0, \texttt{a} \rangle \rangle,$$
$$\texttt{Cons} \mapsto \langle \texttt{x}, \langle \texttt{xs'}, \texttt{r} \rangle \rangle . \langle 1, \lambda \texttt{a} . 2 +_c \ \texttt{r}_p \texttt{Cons} \langle \texttt{x}, \texttt{a} \rangle \rangle)_p \texttt{Nil}$$

Interpretation. We intepret the size of an `list` to be the number of list constructors.

$$[\![\texttt{list}]\!] = \mathbb{N}^\infty$$
$$D^{list} = \{*\} + \{1\} \times \mathbb{N}^\infty$$
$$size_{list}(\texttt{Nil}) = 1$$
$$size_{list}(\texttt{Cons(1,n)}) = 1 + n$$

The interpretation of `rev xs` proceeds as follows.

$$[\![\|\texttt{rev xs}\|]\!] = [\![1 +_c \texttt{rec(xs,...)}_p \ \texttt{Nil}]\!]$$
$$= [\![\langle 1 + (\texttt{rec(xs,...)}_p \ \texttt{Nil})_c, (\texttt{rec(xs,...)}_p \ \texttt{Nil})_p \rangle]\!]$$
$$= \langle 1 + [\![(\texttt{rec(xs,...)}_p \ \texttt{Nil})_c]\!], [\![(\texttt{rec(xs,...)}_p \ \texttt{Nil})_p]\!] \rangle$$

We will focus on the interpretation of the auxilary function `rec(xs,...)`.

Let $g(n) = [\![\texttt{rec(xs,...)}]\!]\{xs \mapsto n\}$

$$g(n) = \bigvee_{size \ ys \leq n} case(ys, Nil \mapsto \langle 1, \lambda a . \langle 0, a \rangle \rangle, Cons \mapsto \langle 1, m \rangle . \langle 1, \lambda a . 2 +_c \pi_1 g(m)(a+1) \rangle)$$

For $n = 0$, $g(0) = \langle 1, \lambda a . \langle 0, a \rangle \rangle$.

For $n > 0$,

$$g(n+1) = \bigvee_{size \ ys \leq n+1} case(ys, Nil \mapsto \langle 1, \lambda a . \langle 0, a \rangle \rangle, Cons \mapsto \langle 1, m \rangle . \langle 1, \lambda a . 2 +_c \pi_1 g(m)(a+1) \rangle)$$

$$g(n+1) = \bigvee_{size \ ys \leq n} case(ys, Nil \mapsto \langle 1, \lambda a . \langle 0, a \rangle \rangle, Cons \mapsto \langle 1, m \rangle . \langle 1, \lambda a . 2 +_c \pi_1 g(m)(a+1) \rangle)$$

$$\vee \bigvee_{size}\ \bigvee_{ys=n+1} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 +_c \pi_1 g(m)(a+1)\rangle)$$

$$g(n+1) = g(n) \vee \bigvee_{size}\ \bigvee_{ys=n+1} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 +_c \pi_1 g(m)(a+1)\rangle)$$

$$g(n+1) = g(n) \vee \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle)$$

We want to show that g is monotonically increasing; $\forall n.g(n) \leq g(n+1)$. By definition of $\leq$, $g(n) \leq g(n+1) \Leftrightarrow \pi_0 g(n) \leq \pi_0 g(n+1) \wedge \pi_1 g(n) \leq \pi_1 g(n+1)$. First we will show $\forall n.\pi_0 g(n) = 1$, the immediate corollary of which is $\forall n.\pi_0 g(n) \leq \pi_0 g(n+1)$.

Proof. We prove this by induction on n.

Base case: $n = 0$**:**

By definition, $\pi_0 g(0) = 1$.

Induction step: $n > 0$**:**

By definition $\pi_0 g(n+1) = \pi_0(g(n) \vee \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle)$. We distribute the projection over the max: $\pi_0 g(n+1) = \pi_0 g(n) \vee 1$. By the induction hypothesis, $\pi_0 g(n) = 1$, so $\pi_0 g(n+1) = 1$.

□

Now we argue that $\pi_1 g(n) \leq \pi_1 g(n+1)$. First we prove the lemma $\forall n.\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$.

Proof. We prove this by induction on n.

$n = 0$**:**

$\pi_1 g(0)a = \langle 0, a\rangle \leq \pi_1 g(0)(a+1) = \langle 0, a+1\rangle$.

$n > 0$**:**

We assume $\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$.

$$\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$$

$$\pi_1 g(n)a \vee 2 +_c g(n)a \leq \pi_1 g(n)(a+1) \vee 2 +_c g(n)(a+1)$$

$$\pi_1 g(n+1)a \leq \pi_1 g(n+1)(a+1)$$

□

Now we show $\pi_1 g(n) \leq \pi_1 g(n+1)$.

Proof. By reflexivity, $\pi_1 g(n) \leq \pi_1 g(n)$. By the lemma we just proved:

$$\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$$

$$\pi_1 g(n)a \leq 2 +_c \pi_1 g(n)(a+1)$$

$$\lambda a.\pi_1 g(n)a \leq \lambda a.2 +_c \pi_1 g(n)(a+1)$$

□

So since for all n, $\pi_0 g(n) = 1$ and $\pi_1 g(n) \leq \lambda a.2 +_c \pi_1 g(n)(a+1)$, we can say

$$g(n) \leq \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle)$$

So

$$g(n+1) = \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle$$

To extract a recurrence from g, we apply g to the interpretation of a list a.

Let $h(n,a) = \pi_1 g(n)a$

For $n = 0$

$$\begin{aligned} h(0,a) &= \pi_1 g(0)a \\ &= (\lambda a.\langle 0, a\rangle)a \\ &= \langle 0, a\rangle \end{aligned}$$

For $n > 0$

$$\begin{aligned} h(n,a) &= \pi_1 g(n)a \\ &= (\lambda a.2 +_c \pi_1 g(n-1)(a+1))a \\ &= 2 +_c \pi_1 g(n-1)(a+1)) \\ &= 2 +_c h(n-1, a+1) \\ &= \langle 2 + \pi_0 h(n-1, a+1), \pi_1 h(n-1, a+1)\rangle \end{aligned}$$

From this recurrence, we can extract a recurrence for the cost. Let $h_c = \pi_0 \circ h$.

For $n = 0$

$$\begin{aligned} h_c(0, a) &= \pi_0 h(0, a) \\ &= \pi_0 \langle 0, a \rangle \\ &= 0 \end{aligned}$$

For $n > 0$

$$\begin{aligned} h_c(n, a) &= \pi_0 \langle 2 + \pi_0 h(n-1, a+1), \pi_1 h(n-1, a+1) \rangle \\ &= 2 + \pi_0 h(n-1, a+1) \\ &= 2 + h_c(n-1, a+1) \end{aligned}$$

We now have a recurrence for the cost of the auxilary function `rec(xs,...)`:

(1)
$$h_c(n, a) = \begin{cases} 0 & n = 0 \\ 2 + h_c(n-1, a+1) & n > 0 \end{cases}$$

Theroem: $h_c(n, a) = 2n$

Proof. We prove this by induction on n.

: Base case: $n = 0$

$$h_c(0, a) = 0 = 2 \cdot 0$$

: Induction case:

We assume $h_c(n, a+1) = 2n$.

$$h_c(n+1, a) = 2 + h_c(n, a+1) = 2 + 2n = 2(n+1)$$

□

The solution to the recurrence for the cost of the auxilary function `rec(xs,...)` is:

$$h_c(n, a) = 2n$$

We can also extract a recurrence for the potential. Let $h_p = \pi_1 \circ h$.

For $n = 0$

$$\begin{aligned} h_p(0, a) &= \pi_1 h(0, a) \\ &= \pi_1 \langle 0, a \rangle \\ &= a \end{aligned}$$

For $n > 0$

$$\begin{aligned} h_p(n, a) &= \pi_1 \langle 2 + \pi_0 h(n-1, a+1), \pi_1 h(n-1, a+1) \rangle \\ &= \pi_1 h(n-1, a+1) \\ &= h_p(n-1, a+1) \end{aligned}$$

We now have a recurrence for the potential of the auxilary function in `rev xs`:

$$h_p(n, a) = \begin{cases} a & n = 0 \\ h_p(n-1, a+1) & n > 0 \end{cases} \tag{2}$$

Theroem: $h_p(n, a) = n + a$

Proof. We prove this by induction on n.

: Base case: $n = 0$

$$h_p(0, a) = a$$

: Induction case:

$$h_p(n, a) = h_p(n-1, a+1) = n - 1 + a + 1 = n + a$$

□

So the solution to the recurrence for the potential of the auxilary function.

$$h_p(n, a) = n + a$$

We return to our interpretation of `rev xs`.

$$\begin{aligned}
&[\![\|\texttt{rev xs}\|]\!] = \langle 1 + [\![(\texttt{rec(xs,...)}_p\ \texttt{Nil})_c]\!], [\![(\texttt{rec(xs,...)}_p\ \texttt{Nil})_p]\!]\rangle \\
&= \langle 1 + \pi_0([\![(\texttt{rec(xs,...)}_p]\!]0), \pi_1([\![\texttt{rec(xs,...)}_p]\!]0)\rangle \\
&= \langle 1 + \pi_0(\pi_1 g(n)\ 0), \pi_1(\pi_1 g(n)\ 0)\rangle \ \texttt{where } n \texttt{ is the length of xs} \\
&= \langle 1 + \pi_0 h(n, 0), \pi_1 h(n, 0)\rangle \\
&= \langle 1 + h_c(n, 0), h_p(n, 0)\rangle \\
&= \langle 1 + 2n, n\rangle
\end{aligned}$$

This result tells us the cost of applying `rev` to a list `xs` of length n is $1 + 2n$, and the resulting list has size n. So `rev` $= \Theta(n)$.

Translation of `rev` using `split`. The linear time reversal function using splits instead of the matching syntactic sugar is written as follows:

```
rev xs = λxs.rec(xs,
               Nil ↦ λa.a,
               Cons↦b.split(b,x.c.split(c,xs'.r.
                         λa.force(r) Cons⟨x,a⟩))) Nil
```

Like before, we will begin by translating the `rec` construct.

```
‖rec(...)‖ = ‖rec(xs, Nil ↦ λa.a,
             Cons↦b.split(b,x.c.split(c,xs'.r.
                       λa.force(r) Cons⟨x,a⟩)))‖

 = ⟨0,xs⟩c+c rec(⟨0,xs⟩p, Nil ↦ 1+c ‖λa.a‖,
                 Cons↦b.1+c ‖split(b,x.c.split(c,xs'.r.
                           λa.force(r) Cons⟨x,a⟩))‖)
```

$$= \texttt{rec(xs, Nil} \mapsto 1 +_c \|\lambda \texttt{a.a}\|,$$
$$\texttt{Cons} \mapsto \texttt{b}.1 +_c \|\texttt{split(b,x.c.split(c,xs'.r.}$$
$$\lambda \texttt{a.force(r) Cons}\langle \texttt{x,a} \rangle))\|)$$

The translation of the `Nil` branch is simple.

$$= \texttt{1} +_c \|\lambda \texttt{a.a}\|$$

$$= \texttt{1} +_c \langle \texttt{0}, \lambda \texttt{a.}\|\texttt{a}\| \rangle$$

$$= \texttt{1} +_c \langle \texttt{0}, \lambda \texttt{a.}\langle \texttt{0,a} \rangle \rangle$$

$$= \langle \texttt{1}, \lambda \texttt{a.}\langle \texttt{0,a} \rangle \rangle$$

The translation of the `Cons` branch is a slightly more involved.

$$= \texttt{Cons} \mapsto \texttt{b}.1 +_c \|\texttt{split(b,x.c.split(c,xs'.r.}$$
$$\lambda \texttt{a.force(r) Cons}\langle \texttt{x,a} \rangle))\|)$$

$$= \texttt{Cons} \mapsto \texttt{b}.1 +_c \|\texttt{b}\|_c +_c \|\texttt{split(c,xs'.r.}$$
$$\lambda \texttt{a.force(r) Cons}\langle \texttt{x,a} \rangle)\|[\pi_0\|\texttt{b}\|_p/x, \pi_1\|\texttt{b}\|_p/c]$$

The translation of the type of `b` will illuminate the translation of the `split`. The type of `b` is $\texttt{b::int} \times \langle \texttt{list} \times \langle \texttt{list} \to \texttt{list} \rangle \rangle$.
The type of $\|\texttt{b}\|$ is $\langle \texttt{C} \times \langle \texttt{int} \times \langle \texttt{list} \times \langle \texttt{C} \times \texttt{list} \to \langle \texttt{C} \times \texttt{list} \rangle \rangle \rangle \rangle \rangle$. We can say that $\pi_0\|\texttt{b}\|_p$ is the head of the list `xs`, $\pi_0\pi_1\|\texttt{b}\|_p$ is the tail of the list `xs`, and $\pi_1\pi_1\|\texttt{b}\|_p$ is the result of the recursive call.

$$= \texttt{Cons} \mapsto \texttt{b}.1 +_c \langle \texttt{0}, \|\texttt{b}\|_p \rangle_c +_c \|\texttt{split(c,xs'.r.}$$
$$\lambda \texttt{a.force(r) Cons}\langle \texttt{x,a} \rangle\|[\pi_0\|\texttt{b}\|_p/x, \pi_1\|\texttt{b}\|_p/c]$$

$$= \texttt{Cons} \mapsto \texttt{b}\,.\,1 +_c 0 +_c (\|\texttt{c}\|_c +_c (\|\lambda \texttt{a}\,.\,\texttt{force(r)}\ \ \texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|)[\pi_0\|\texttt{c}\|_p/xs', \pi_1\|\texttt{c}\|_p/r])[\pi_0\|\texttt{b}\|_p/x, \pi_1\|\texttt{b}\|_p/c]$$

$$= \texttt{Cons} \mapsto \texttt{b}\,.\,1 +_c (\|\texttt{c}\|_c +_c \langle 0, \lambda \texttt{a}\,.\,\|\texttt{force(r)}\ \ \texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|\rangle[\pi_0\|\texttt{c}\|_p/xs', \pi_1\|\texttt{c}\|_p/r])[\pi_0\|\texttt{b}\|_p/x, \pi_1\|\texttt{b}\|_p/c]$$

Let us focus on the translation of $\|\texttt{force(r)}\ \ \texttt{Cons}\langle\texttt{x,a}\rangle\|$.

$$= (1 + \|\texttt{force(r)}\|_c + \|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\|_c) +_c \|\texttt{force(r)}\|_p\|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_p$$

$$= (1 + (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_c + \|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_c) +_c (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_p\|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_p$$

$$= (1 + (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_c + \|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_c) +_c (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_p\|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_p$$

$$= (1 + \|\texttt{r}\|_c + \|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_c) +_c \|\texttt{r}\|_p\|\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle\|_p$$

$$= (1 + \|\texttt{r}\|_c + (\langle\|\langle \texttt{x}\,,\texttt{a}\rangle\|_c\,,\texttt{Cons}\|\langle \texttt{x}\,,\texttt{a}\rangle\|_p\rangle)_c) +_c \|\texttt{r}\|_p\langle\|\langle \texttt{x}\,,\texttt{a}\rangle\|_c\,,\texttt{Cons}\|\langle \texttt{x}\,,\texttt{a}\rangle\|_p\rangle_p$$

$$= (1 + \|\texttt{r}\|_c + \|\langle \texttt{x}\,,\texttt{a}\rangle\|_c) +_c \|\texttt{r}\|_p\texttt{Cons}\|\langle \texttt{x}\,,\texttt{a}\rangle\|_p$$

$$= (1 + \|\texttt{r}\|_c + \langle\|\texttt{x}\|_c + \|\texttt{a}\|_c\,,\langle\|\texttt{x}\|_p\,,\|\texttt{a}\|_p\rangle\rangle_c) +_c \|\texttt{r}\|_p\texttt{Cons}\langle\|\texttt{x}\|_c + \|\texttt{a}\|_c\,,\langle\|\texttt{x}\|_p\,,\|\texttt{a}\|_p\rangle\rangle_p$$

$$= (1 + \|\texttt{r}\|_c + (\|\texttt{x}\|_c + \|\texttt{a}\|_c)) +_c \|\texttt{r}\|_p\texttt{Cons}\langle\|\texttt{x}\|_p\,,\|\texttt{a}\|_p\rangle$$

$$= (1+\texttt{r}_c + (\langle \texttt{0}\,,\texttt{x}\rangle_c + \langle \texttt{0}\,,\texttt{a}\rangle_c)) +_c \langle \texttt{0}\,,\texttt{r}\rangle_p\texttt{Cons}\langle\langle \texttt{0}\,,\texttt{x}\rangle_p\,,\langle \texttt{0}\,,\texttt{a}\rangle_p\rangle$$

$$= (1+\texttt{r}_c + 0 + 0)+_c\texttt{r}_p\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle$$

$$= (1+\texttt{r}_c)+_c\texttt{r}_p\texttt{Cons}\langle \texttt{x}\,,\texttt{a}\rangle$$

We can now use this in our translation of the `Cons` case.

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c (\|\texttt{c}\|_c +_c \langle 0, \lambda\texttt{a}\,.\,(1+\texttt{r}_c)+_c \texttt{r}_p\texttt{Cons}\langle\texttt{x}\,,\texttt{a}\rangle\rangle[\pi_0\|\texttt{c}\|_p/xs', \pi_1\|\texttt{c}\|_p/r])[\pi_0\|\texttt{b}\|_p/x, \pi_1\|\texttt{b}\|_p/c]$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c (\|\texttt{c}\|_c +_c \langle 0, \lambda\texttt{a}\,.\,(1 + (\pi_1\|\texttt{c}\|_p)_c) +_c (\pi_1\|\texttt{c}\|_p)_p\texttt{Cons}\langle\texttt{x}\,,\texttt{a}\rangle\rangle)[\pi_0\|\texttt{b}\|_p/x, \pi_1\|\texttt{b}\|_p/c]$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c (\|\pi_1\|\texttt{b}\|_p\|_c +_c \langle 0, \lambda\texttt{a}\,.\,(1 + (\pi_1\|\pi_1\|\texttt{b}\|_p\|_p)_c) +_c (\pi_1\|\pi_1\|\texttt{b}\|_p\|_p)_p\texttt{Cons}\langle\pi_1\|\texttt{b}\|_p\,,\texttt{a}\rangle\rangle)$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c (\|\pi_1\langle\texttt{0}\,,\texttt{b}\rangle_p\|_c +_c \langle 0, \lambda\texttt{a}\,.\,(1 + (\pi_1\|\pi_1\langle\texttt{0}\,,\texttt{b}\rangle_p\|_p)_c) +_c (\pi_1\|\pi_1\langle\texttt{0}\,,\texttt{b}\rangle_p\|_p)_p\texttt{Cons}\langle\pi_1\langle\texttt{0}\,,\texttt{b}\rangle_p\,,\texttt{a}\rangle\rangle)$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c (\|\pi_1\texttt{b}\|_c +_c \langle 0, \lambda\texttt{a}\,.\,(1 + (\pi_1\|\pi_1\texttt{b}\|_p)_c) +_c (\pi_1\|\pi_1\texttt{b}\|_p)_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle)$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c (\langle\texttt{0}\,,\pi_1\texttt{b}\rangle_c +_c \langle 0, \lambda\texttt{a}\,.\,(1 + (\pi_1\langle\texttt{0}\,,\pi_1\texttt{b}\rangle_p)_c) +_c (\pi_1\langle\texttt{0}\,,\pi_1\texttt{b}\rangle_p)_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle)$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,1 +_c \langle 0, \lambda\texttt{a}\,.\,(1 + (\pi_1\pi_1\texttt{b})_c) +_c (\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle$$

$$= \texttt{Cons}\mapsto\texttt{b}\,.\,\langle 1, \lambda\texttt{a}\,.\,(1 + (\pi_1\pi_1\texttt{b})_c) +_c (\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle$$

$$\|\texttt{rec(xs,...)}\| = \texttt{rec(xs,Nil}\mapsto\langle\texttt{1}\,,\lambda\texttt{a}\,.\,\langle\texttt{0}\,,\texttt{a}\rangle\rangle\,,\ \texttt{Cons}\mapsto\texttt{b}\,.\,\langle 1, \lambda\texttt{a}\,.\,(1 + (\pi_1\pi_1\texttt{b})_c) +_c (\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle\texttt{)}$$

So our translation of `rev` is

$$\|\texttt{rev}\| = \langle\texttt{0}\,,\lambda\texttt{xs}\,.\,\texttt{rec(xs,Nil}\mapsto\langle\texttt{1}\,,\lambda\texttt{a}\,.\,\langle\texttt{0}\,,\texttt{a}\rangle\rangle\,,$$

$$\texttt{Cons}\mapsto\texttt{b}\,.\,\langle 1, \lambda\texttt{a}\,.\,(1+(\pi_1\pi_1\texttt{b})_c)+_c(\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle)\rangle$$

We observe that in both cases of the `rec`, the cost of the recursive call is 1, so we can replace $\pi_1\pi_1\texttt{b}_c$ with 1.

$$\begin{aligned}\|\texttt{rev}\| \;=\; &\langle 0\,,\lambda\texttt{xs}\,.\,\texttt{rec(xs}\,,\texttt{Nil}\mapsto\langle 1\,,\lambda\texttt{a}\,.\,\langle 0\,,\texttt{a}\rangle\rangle\,,\\ &\qquad\texttt{Cons}\mapsto\texttt{b}\,.\,\langle 1,\lambda\texttt{a}\,.\,(2+_c(\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle)\rangle\end{aligned}$$

We are interested in the interpretation of `rev xs`.

$$\begin{aligned}\|\texttt{rev xs}\| \;&=\; (1+\|\texttt{rev}\|_c\|\texttt{xs}\|_c)+_c\|\texttt{rev}\|_p\|\texttt{xs}\|_p\\ &=\; (1+\langle 0\,,\lambda\texttt{xs}\,.\,\texttt{rec}(\ldots)\rangle_c+\langle 0\,,\texttt{xs}\rangle_c)+_c\|\texttt{rev}\|_p\langle 0\,,\texttt{xs}\rangle_p\\ &=\; (1+0+0)+_c(\lambda\texttt{xs}\,.\,\texttt{rec(xs}\,,\texttt{Nil}\mapsto\langle 1\,,\lambda\texttt{a}\,.\,\langle 0\,,\texttt{a}\rangle\rangle\,,\\ &\qquad\texttt{Cons}\mapsto\texttt{b}\,.\,\langle 1,\lambda\texttt{a}\,.\,(2+_c(\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle))_p\texttt{xs}\\ &=\; 1+_c\texttt{rec(xs}\,,\texttt{Nil}\mapsto\langle 1\,,\lambda\texttt{a}\,.\,\langle 0\,,\texttt{a}\rangle\rangle\,,\\ &\qquad\texttt{Cons}\mapsto\texttt{b}\,.\,\langle 1,\lambda\texttt{a}\,.\,(2+_c(\pi_1\pi_1\texttt{b})_p\texttt{Cons}\langle\pi_1\texttt{b}\,,\texttt{a}\rangle\rangle)\end{aligned}$$

Interpretation. We intepret the size of an `list` to be the number of list constructors.

$$\begin{aligned}&[\![\,\texttt{list}\,]\!] = \mathbb{N}^\infty\\ &D^{list} = \{*\}+\{1\}\times\mathbb{N}^\infty\\ &size_{list}(\texttt{Nil}) = 1\\ &size_{list}(\texttt{Cons(1,n)}) = 1+n\end{aligned}$$

The interpretation of `rev xs` proceeds as follows.

$$\begin{aligned}&[\![\|\texttt{rev xs}\|]\!] = [\![1+_c\texttt{rec(xs},\ldots)_p\;\;\texttt{Nil}]\!]\\ &=\; [\![\langle 1+(\texttt{rec(xs},\ldots)_p\;\;\texttt{Nil})_c,(\texttt{rec(xs},\ldots)_p\;\;\texttt{Nil})_p\rangle]\!]\end{aligned}$$

$= \langle 1 + [\![(\texttt{rec(xs,...)}_p \ \texttt{Nil})_c]\!], [\![(\texttt{rec(xs,...)}_p \ \texttt{Nil})_p]\!] \rangle$

We will focus on the interpretation of the auxilary function `rec(xs,...)`.

Let $g(n) = [\![\texttt{rec(xs,...)}]\!]\{xs \mapsto n\}$

$$g(n) = \bigvee_{size\ ys \leq n} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 +_c \pi_1 g(m)(a+1)\rangle)$$

For $n = 0$, $g(0) = \langle 1, \lambda a.\langle 0, a\rangle\rangle$.

For $n > 0$,

$$g(n{+}1) = \bigvee_{size\ ys \leq n+1} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 {+_c} \pi_1 g(m)(a{+}1)\rangle)$$

$$g(n{+}1) = \bigvee_{size\ ys \leq n} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 {+_c} \pi_1 g(m)(a{+}1)\rangle)$$
$$\vee \bigvee_{size\ ys = n+1} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 +_c \pi_1 g(m)(a+1)\rangle)$$

$$g(n{+}1) = g(n) \vee \bigvee_{size\ ys = n+1} case(ys, Nil \mapsto \langle 1, \lambda a.\langle 0, a\rangle\rangle, Cons \mapsto \langle 1, m\rangle.\langle 1, \lambda a.2 {+_c} \pi_1 g(m)(a{+}1)\rangle)$$

$$g(n+1) = g(n) \vee \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle)$$

We want to show that g is monotonically increasing; $\forall n.g(n) \leq g(n+1)$. By definition of $\leq$, $g(n) \leq g(n+1) \Leftrightarrow \pi_0 g(n) \leq \pi_0 g(n+1) \wedge \pi_1 g(n) \leq \pi_1 g(n+1)$. First we will show $\forall n.\pi_0 g(n) = 1$, the immediate corollary of which is $\forall n.\pi_0 g(n) \leq \pi_0 g(n+1)$.

Proof. We prove this by induction on n.

Base case: $n = 0$**:**

By definition, $\pi_0 g(0) = 1$.

Induction step: $n > 0$**:**

By definition $\pi_0 g(n+1) = \pi_0(g(n) \vee \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle)$. We distribute the projection over the max: $\pi_0 g(n+1) = \pi_0 g(n) \vee 1$. By the induction hypothesis, $\pi_0 g(n) = 1$, so $\pi_0 g(n+1) = 1$.

□

Now we argue that $\pi_1 g(n) \leq \pi_1 g(n+1)$. First we prove the lemma $\forall n.\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$.

Proof. We prove this by induction on n.

$n = 0$**:**

$\pi_1 g(0)a = \langle 0, a\rangle \leq \pi_1 g(0)(a+1) = \langle 0, a+1\rangle$.

$n > 0$**:**

We assume $\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$.

$$\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$$

$$\pi_1 g(n)a \vee 2 +_c g(n)a \leq \pi_1 g(n)(a+1) \vee 2 +_c g(n)(a+1)$$

$$\pi_1 g(n+1)a \leq \pi_1 g(n+1)(a+1)$$

□

Now we show $\pi_1 g(n) \leq \pi_1 g(n+1)$.

Proof. By reflexivity, $\pi_1 g(n) \leq \pi_1 g(n)$. By the lemma we just proved:

$$\pi_1 g(n)a \leq \pi_1 g(n)(a+1)$$

$$\pi_1 g(n)a \leq 2 +_c \pi_1 g(n)(a+1)$$

$$\lambda a.\pi_1 g(n)a \leq \lambda a.2 +_c \pi_1 g(n)(a+1)$$

□

So since for all n, $\pi_0 g(n) = 1$ and $\pi_1 g(n) \leq \lambda a.2 +_c \pi_1 g(n)(a+1)$, we can say

$$g(n) \leq \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle)$$

So

$$g(n+1) = \langle 1, \lambda a.2 +_c \pi_1 g(n)(a+1)\rangle$$

To extract a recurrence from g, we apply g to the interpretation of a list a.

Let $h(n, a) = \pi_1 g(n)a$

For $n = 0$

$$\begin{aligned} h(0,a) &= \pi_1 g(0) a \\ &= (\lambda a.\langle 0, a\rangle) a \\ &= \langle 0, a\rangle \end{aligned}$$

For $n > 0$

$$\begin{aligned} h(n,a) &= \pi_1 g(n) a \\ &= (\lambda a.2 +_c \pi_1 g(n-1)(a+1)) a \\ &= 2 +_c \pi_1 g(n-1)(a+1)) \\ &= 2 +_c h(n-1, a+1) \\ &= \langle 2 + \pi_0 h(n-1, a+1), \pi_1 h(n-1, a+1)\rangle \end{aligned}$$

From this recurrence, we can extract a recurrence for the cost. Let $h_c = \pi_0 \circ h$.

For $n = 0$

$$\begin{aligned} h_c(0,a) &= \pi_0 h(0,a) \\ &= \pi_0 \langle 0, a\rangle \\ &= 0 \end{aligned}$$

For $n > 0$

$$\begin{aligned} h_c(n,a) &= \pi_0 \langle 2 + \pi_0 h(n-1, a+1), \pi_1 h(n-1, a+1)\rangle \\ &= 2 + \pi_0 h(n-1, a+1) \\ &= 2 + h_c(n-1, a+1) \end{aligned}$$

We now have a recurrence for the cost of the auxilary function `rec(xs,...)`:

(3)
$$h_c(n,a) = \begin{cases} 0 & n = 0 \\ 2 + h_c(n-1, a+1) & n > 0 \end{cases}$$

Theroem: $h_c(n,a) = 2n$

PROOF. We prove this by induction on n.

: Base case: $n = 0$

$$h_c(0, a) = 0 = 2 \cdot 0$$

: Induction case:

We assume $h_c(n, a + 1) = 2n$.

$$h_c(n + 1, a) = 2 + h_c(n, a + 1) = 2 + 2n = 2(n + 1)$$

□

The solution to the recurrence for the cost of the auxilary function `rec(xs,...)` is:

$$h_c(n, a) = 2n$$

We can also extract a recurrence for the potential. Let $h_p = \pi_1 \circ h$.

For $n = 0$

$$\begin{aligned} h_p(0, a) &= \pi_1 h(0, a) \\ &= \pi_1 \langle 0, a \rangle \\ &= a \end{aligned}$$

For $n > 0$

$$\begin{aligned} h_p(n, a) &= \pi_1 \langle 2 + \pi_0 h(n - 1, a + 1), \pi_1 h(n - 1, a + 1) \rangle \\ &= \pi_1 h(n - 1, a + 1) \\ &= h_p(n - 1, a + 1) \end{aligned}$$

We now have a recurrence for the potential of the auxilary function in `rev xs`:

(4)
$$h_p(n, a) = \begin{cases} a & n = 0 \\ h_p(n - 1, a + 1) & n > 0 \end{cases}$$

Theroem: $h_p(n, a) = n + a$

PROOF. We prove this by induction on n.

: Base case: $n = 0$

$$h_p(0, a) = a$$

: Induction case:

$$h_p(n, a) = h_p(n - 1, a + 1) = n - 1 + a + 1 = n + a$$

□

So the solution to the recurrence for the potential of the auxilary function.

$$h_p(n, a) = n + a$$

We return to our interpretation of `rev xs`.

$[\![\|\texttt{rev xs}\|]\!] = \langle 1 + [\![(\texttt{rec(xs,...)}_p\ \texttt{Nil})_c]\!], [\![(\texttt{rec(xs,...)}_p\ \texttt{Nil})_p]\!]\rangle$

$= \langle 1 + \pi_0([\![(\texttt{rec(xs,...)}_p]\!]0), \pi_1([\![\texttt{rec(xs,...)}_p]\!]0)\rangle$

$= \langle 1 + \pi_0(\pi_1 g(n)\ 0), \pi_1(\pi_1 g(n)\ 0)\rangle$ `where` n `is the length of xs`

$= \langle 1 + \pi_0 h(n, 0), \pi_1 h(n, 0)\rangle$

$= \langle 1 + h_c(n, 0), h_p(n, 0)\rangle$

$= \langle 1 + 2n, n\rangle$

This result tells us the cost of applying `rev` to a list `xs` of length n is $1 + 2n$, and the resulting list has size n. So `rev` $= \Theta(n)$.

1. Reverse

```
datatype intlist = Nil of unit | Cons of int × intlist
```

The following function reverses a list on $\Theta(n^2)$ time. It walks down a list, appending the head of the list to the end of the result of recursively calling itself on the tail of the list.

```
rev = λxs.rec(xs, Nil ↦ Nil
                , Cons ↦ ⟨x,⟨xs',r⟩⟩.
                      rec(force(r), Nil ↦ Cons⟨x, Nil⟩
                                  , Cons ↦ ⟨y,⟨ys,rys⟩⟩. Cons⟨y,force(rys)⟩
```

Complexity Language. The translation into the complexity language is

```
rev = ⟨0,λxs.rec(xs, Nil ↦ Nil
                   , Cons ↦ ⟨x,⟨xs',r⟩⟩.
                        rec(r, Nil ↦ Cons⟨x, Nil⟩
                             , Cons ↦ ⟨y,⟨ys,rys⟩⟩. Cons⟨y,rys⟩))⟩
```

It is more interesting if we consider the translation of `rev` applied to some `xs:intlist`. The translation of this function into the complexity language proceeds as follows. We begin with translating the outer `rec` construct.

```
rev xs  = 1 + ‖xs‖c +c rec(‖xs‖p
                         , Nil ↦ 1 +c ‖Nil‖
                         , Cons ↦ ⟨x,⟨xs',r⟩⟩. 1 +c ‖rec(...)‖)
```

The cost of the translation of an `intlist` is zero, and the potential of the translation of an `intlist` is the list itself.

```
rev xs = 1 +c rec(xs, Nil ↦ ⟨1,Nil⟩ , Cons ↦ ⟨x,⟨xs',r⟩⟩. 1 +c ‖rec(...)‖)
```

Next we translate the inner `rec`.

$$\|\texttt{rec(force(r), Nil} \mapsto \texttt{Cons}(\langle \texttt{x, Nil}\rangle)$$
$$\texttt{, Cons} \mapsto \langle \texttt{y,}\langle \texttt{ys,rys}\rangle\rangle.\ \texttt{Cons}\langle \texttt{y,force(rys)}\rangle))\|$$

Since `x`, `xs'`, `r` are terms in the complexity language, they do not need to be translated. First we apply the rules for `rec` and `force`.

$$\texttt{r}_c +_c \texttt{rec(r}_p\texttt{, Nil} \mapsto 1 +_c \|\texttt{Cons}(\langle \texttt{x, Nil}\rangle)\|$$
$$\texttt{, Cons} \mapsto \langle \texttt{y,}\langle \texttt{ys,rys}\rangle\rangle.\ 1 +_c \|\texttt{Cons}\langle \texttt{y,force(rys)}\rangle\|)$$

The cost of the `Cons` constructor is zero, and the translation of `force` is just the translation of its argument.

$$\texttt{r}_c +_c \texttt{rec(r}_p\texttt{, Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}_p\texttt{, Nil}\rangle\rangle$$
$$\texttt{, Cons} \mapsto \langle \texttt{y,}\langle \texttt{ys,rys}\rangle\rangle.\ \langle 1 + \texttt{rys}_c, \texttt{Cons}\langle \texttt{y}_p\texttt{,rys}_p\rangle\rangle)$$

Putting the pieces together, we get

$$\|\texttt{rev xs}\| = 1 +_c \texttt{rec(xs}_p$$
$$\texttt{, Nil} \mapsto \langle 1, \texttt{Nil}\rangle$$
$$\texttt{, Cons} \mapsto \langle \texttt{x,}\langle \texttt{xs',r}\rangle\rangle.\ 1 + \texttt{r}_c +_c$$
$$\texttt{rec(r}_p\texttt{, Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}_p\texttt{, Nil}\rangle\rangle$$
$$\texttt{, Cons} \mapsto \langle \texttt{y,}\langle \texttt{ys,rys}\rangle\rangle.\ \langle 1 + \texttt{rys}_c, \texttt{Cons}\langle \texttt{y}_p\texttt{,rys}_p\rangle\rangle)$$

Interpretation. We intepret the size of an `intlist` to be the number of constructors.

$[\![\ \texttt{intlist}\ ]\!] = \mathbb{N}^\infty$

$D^{intlist} = \{*\} + \{1\} \times \mathbb{N}^\infty$

$size_{intlist}(\texttt{Nil}) = 0$

$size_{intlist}(\texttt{Cons(1,n)}) = 1 + n$

Then $[\![\|\texttt{rev xs}\|_c]\!] = 1 + g(\|xs\|_p)$, where

$$g(n) = [\![rec(z, Nil \mapsto 1, Cons \mapsto \langle x, \langle xs', r\rangle\rangle.1 + r_c + h(r_p))]\!]\{z \mapsto n\}$$

$$h(n) = [\![rec(z, Nil \mapsto 1, Cons \mapsto \langle y, \langle ys', r\rangle\rangle.1 + r_c]\!]\{z \mapsto n\}$$

We calculate that $h(0) = 1$ and for $n > 0$, $h(n) = 1 + h(n-1)$. $g(0) = 1$ and for $n > 0$, $g(n) = 1 + g(n-1) + h(n-1)$

2. Parametric Insertion Sort

2.1. Parametric Insertion Sort. Parametric insertion sort is a higher order algorithm which sorts a list using a comparison function which is passed to it as an argument. The running time of insertion sort is $\mathcal{O}(n^2)$. This characterization of the complexity of parametric insertion sort does not capture role of the comparison function in the running time. When sorting a list of integers, where comparison between any two integers takes constant time, this does not matter. However, when sorting a list of strings, where the complexity of comparison is order the length of the string, the length of the strings may influence the running time more than the length of the list when sorting small lists of large strings.

2.1.1. *Insert.* The function `sort` relies on the function `insert` to insert the head of the list into the result of recursively sorted tail of the list. We will begin with a translation and interpretation of `insert`.

2.1.2. *Translation.* The translation of `insert` is broken into chunks to make it more manageable. Figure 19 steps through the translation of the comparison function `f` applied to variables `x` and `y`.

The translation `true` and `false` branches are given in figures 20 and 21 respectively.

Figure 22 uses the translation of `f x y` and the `true` and `false` branches to construct the translation of the inner `rec` construct.

The `Nil` and `Cons` branches of the outer `rec` construct are given in figures 23 and 24, respectively.

We put these together to give the translation of `insert`.

Finally we give a translation of `insert f x xs` in figure 26 because this is the term we will interpret in a size-based semantics.

The result is:

$$\begin{aligned}\|\texttt{insert f x xs}\| = (3 + \|\texttt{f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c)\\ +_c \texttt{rec}(\|\texttt{xs}\|_p,\\ \texttt{Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle\end{aligned}$$

$$\texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . (3 + ((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p \ \texttt{y})_c) +_c \texttt{rec}((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p \ \texttt{y})_p,$$
$$\texttt{True} \mapsto \langle 1, \texttt{Cons} \langle \|\texttt{x}\|_p, \texttt{Cons} \langle \texttt{y}, \texttt{y}$$
$$\texttt{False} \mapsto \langle 1 +_c \texttt{r}, \texttt{Cons} \langle \texttt{y}, \texttt{r}_p \rangle \rangle))$$

2.1.3. *Interpretation.* We well use an interpretation of lists as a pair of their greatest element and their length. Figure 27 formalizes this interpretation. We use the mutual ordering on pairs. That is, $(s, n) \leq (s', n')$ if $n \leq n'$ and $s < s'$ or $n < n'$ and $s \leq s'$.

First we interpret the `rec`, which drives of the cost of `insert`. As in the translation, we break the interpretation up to make it more manageable. We will write map, λ and $+_c$ in the semantics, which stand for the semantic equivalents of the syntactic `map`, λ and $+_c$. The definitions of these semantic functions mirror the definitions of their syntactic equivalents. Figures 28 and 29 walk through the interpretation.

The initial result is given in equation 10.

$$\begin{aligned} f_{Nil}(\langle\rangle) &= (1, (x, 1)) \\ f_{Cons}(j, (j, m)) &= (4 + \pi_0(\pi_1(f\ x)\ j) + \pi_0 g(j, m), \\ &\quad (max\{x, j, \pi_0\pi_1 g(j, m)\}, 2 + m \vee 1 + \pi_1\pi_1 g(j, m))) \\ g(i, n) &= \bigvee_{size(z) \leq (i,n)} case(z, (f_{Nil}, f_{Cons})) \end{aligned} \tag{5}$$

This recurrence is difficult to work with. Specifically, we cannot apply traditional methods of solving it. We will manipulate it into a more usable form by eliminating the arbitrary maximum. We will separate the recurrence into a recurrence for the cost and a recurrence for the potential, and solve those independently.

LEMMA 2.1. $g_c(i, n) \leq (4 + ((f\ x)_p\ i)_c n + 1$

PROOF. We prove this by induction on n. Recall we use the mutual ordering on pairs.

case $n = 0$**:**

$g_c(i, n) = (1, (x, 1))_c = 1$

case$n > 0$**:**

$$
\begin{aligned}
&= \bigvee_{size(z)\leq(i,n)} case(z,(f_{Nil},f_{Cons})) \\
&= \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} case((j,m),(f_{Nil},f_{Cons})) \\
&= \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} 4+((f\ x)_p\ j)_c + g_c(j,m') && \text{where } m' = m-1 \\
&= \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} 4+((f\ x)_p\ j)_c + (4+((f\ x)_p\ j)_c)m' + 1 && \text{by the induction hypothesis} \\
&= \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} (4+((f\ x)_p\ j)_c)(m'+1) + 1 \\
&= \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} (4+((f\ x)_p\ j)_c)m + 1 \\
&\leq \bigvee_{i<j,m\leq n \text{ or } i\leq j,m<n} (4+((f\ x)_p\ i)_c)n + 1 \\
&\leq (4+((f\ x)_p\ i)_c)n + 1
\end{aligned}
$$

□

As expected, we find the cost of insert is bounded by the length of the list and the largest element.

LEMMA 2.2. $g_p(i,n) \leq (max\{x,i\}, n+1)$

PROOF. We prove this by induction on n.

case $n = 0$**:**

$g_p(i,n) = (1,(x,1))_p = (x,1)$.

case $n > 0$**:**

$$
\begin{aligned}
&= \bigvee_{size(z)\leq(i,n)} case(z,(f_{Nil},f_{Cons})\\
&= \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} (max\{x,j,\pi_0\pi_1 g(j,m')\},2+m'\vee 1+\pi_1\pi_1 g(j,m')) \quad \text{where } m'=m-1\\
&\leq \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} (max\{x,j\},2+m') \quad \text{by the induction hypothesis}\\
&\leq \bigvee_{j<i,m\leq n \text{ or } j\leq i,m<n} (max\{x,i\},1+n)\\
&\leq (max\{x,i\},1+n)
\end{aligned}
$$

□

We find the length of the potential is bounded by one plus the length of the input, and the largest element in the output is bounded by the maximum of the element being inserted and the largest element in the input. This is somewhat unsatisfactory, since we would expect the relationship to be equality. What happens if we try to prove the equality?

LEMMA 2.3. $g_p(i,n) = (max\{x,i\},n+1)$

PROOF. We attempt to prove this by induction on n. The first steps proceed similarly to 3.2.

case $n = 0$**:**

$g_p(i,n) = (1,(x,1))_p = (x,1)$.

case $n > 0$**:**

$$
\begin{aligned}
&= \bigvee_{size(z)\leq(i,n)} case(z, (f_{Nil}, f_{Cons}) \\
&= \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} (max\{x, j, \pi_0\pi_1 g(j, m')\}, 2 + m' \vee 1 + \pi_1\pi_1 g(j, m')) \quad \text{where } m' = m - 1 \\
&= \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} (max\{x, j\}, 2 + m') \qquad \text{by the induction hypothesis} \\
&= \bigvee_{j<i, m\leq n} (max\{x, j\}, 1 + m) \vee \bigvee_{j\leq i, m<n} (max\{x, j\}, 1 + m)
\end{aligned}
$$

□

We see that we get stuck. Because of the mutual ordering on pairs, our big maximum is over all z such that $size(z) < (i, n)$. This includes (j, m) such that $j < i^m \leq n$. We have no way of reasoning about the potential of $g(i - 1, n) \vee g(i, n - 1)$. So we cannot prove equality for 3.2. This indicates we may not have the optimal ordering on pairs.

Using lemmas 3.1 and 3.2, we can express the cost and potential of `insert` in terms of its arguments.

$$insert\ f\ x\ xs \leq (4 + ((f\ x)_p\ i)_c n + 1, (max\{x, i\}, n + 1)) \tag{6}$$

2.1.4. *Sort.*

2.1.5. *Translation.* The translation of sort is shown in figure 32. The translation of the `Nil` and `Cons` branches in the `rec` are walked through in figures 30 and 31, respectively. The translation of `sort` applied to its arguments is given in figure 33.

2.1.6. *Interpretation.* The `rec` construct again drives the cost and potential of `sort`. The walk through of the interpretation of the `rec` is given in figure 34.

Equation 14 shows the initial recurrence extracted.

$$g(i, n) = \bigvee_{size(z)\leq(i,n)} case(z, (\lambda(\langle\rangle).(1, (\neg\infty, 0), \lambda(j, m).4 + \pi_0 g(j, m)) +_c (insert\ f\ j\ \pi_1 g(j, m))) \tag{7}$$

Observe that in equation 14, the cost is depends on the potential of the recursive call. Therefore we must solve the recurrence for the potential first.

LEMMA 2.4. $\pi_1 g(n) \leq (j, n)$

PROOF. We prove this by induction on n. We use equation 6 to determine the potential of the *insert* function.

case $n = 0$**:** $\pi_1 g(i, n) = (i, 0)$

case $n > 0$**:**

$$\begin{aligned}
\pi_1 g(i,n) &= \pi_1 \bigvee_{size(z)\leq n} case(z, (\lambda(\langle\rangle).(1, (\neg\infty, 0)), \lambda(j,m).4 + \pi_0 g(j,m)) +_c (insert\ f\ j\ \pi_1 g(j,m))) \\
&= \bigvee_{j\leq i, m<n \text{ or } j<i, m\leq n} \pi_1(insert\ f\ j\ \pi_1 g(j', m')) \qquad j' \leq j, m' = m-1 \\
&\leq \bigvee_{j\leq i, m<n \text{ or } j<i, m\leq n} \pi_1(insert\ f\ j\ (j', m')) \\
&\leq \bigvee_{j\leq i, m<n \text{ or } j<i, m\leq n} (max\{j, j'\}, m' + 1 \\
&\leq \bigvee_{j\leq i, m<n \text{ or } j<i, m\leq n} (j, m) \\
&\leq \bigvee_{j\leq i, m<n \text{ or } j<i, m\leq n} (i, n) \\
&\leq (i, n)
\end{aligned}$$

□

As in the interpretation of `insert` we are left with a less than satisfactory bound on the potential of `sort`. It would grievous mistake to write a sorting function whose output was smaller than its input. Under the current interpretation of lists, this would mean either the length of the list decreased or the size of the largest element in the list decreased. Unfortunately we are stuck with an upper bound on the size of the output because or interpretation of `insert` only provides an upper bound on the potential of its output.

We may solve the recurrence for the cost of `sort`.

LEMMA 2.5. $\pi_0 g(n) \leq (4 + \pi_0(\pi_1(f\ x)\ i)n^2 + 5n + 1$

PROOF. We prove this by induction on n.

case $n = 0$**:** $\pi_0 g(i, n) = 1$

case $n > 0$**:**

$$\begin{aligned}
\pi_0 g(i,n) &= \pi_0 \bigvee_{size(z)\leq(i,n)} case(z, (\lambda(\langle\rangle).(1, (\neg\infty, 0)), \lambda(j,m).4 + \pi_0 g(j,m)) +_c (insert\ f\ j\ \pi_1 g(j,m))) \\
&= \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} 4 + \pi_0 g(j, m-1) + \pi_0(insert\ f\ j\ \pi_1 g(j, m-1)) \\
&\leq \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} 4 + \pi_0 g(j, m-1) + \pi_0(insert\ f\ j\ (j, m-1)) \\
&\leq \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} 4 + \pi_0 g(j, m-1) + (4 + \pi_0(\pi_1(f\ j)\ j))(m-1) + 1 \\
&\text{let } c_1 = (4 + \pi_0(\pi_1(f\ j)\ j)) \\
&\leq \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} 4 + c_1(m-1)^2 + 5(m-1) + 1 + c_1(m-1) + 1 \\
&\leq \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} 4 + c_1 m^2 - 2c_1 m + c_1 + 5m - 5 + 1 + c_1 m - c_1 + 1 \\
&\leq \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} c_1 m^2 - c_1 m + 5m + 1 \\
&\leq \bigvee_{j<i, m\leq n \text{ or } j\leq i, m<n} (4 + \pi_0(\pi_1(f\ i)\ i))n^2 + 5n + 1 \\
&\leq (4 + \pi_0(\pi_1(f\ i)\ i))n^2 + 5n + 1
\end{aligned}$$

□

As expected the cost of `sort` is $\mathcal{O}(n^2)$ where n is the length of the list. It is clear from the analysis how the cost of the comparison function determines the running time of `sort`. We can see that the comparison function is called order n^2 times.

FIGURE 1. Parametric insertion sort in the source language

```
data list = Nil of unit | Cons of int × list

insert = λf.λx.λxs.rec(xs, Nil↦ Cons⟨x,Nil⟩,
                          Cons↦⟨y,⟨ys,r⟩⟩.rec(f x y, True ↦ Cons⟨x,Cons⟨y,ys⟩⟩,
                                                 False ↦ Cons⟨y,force(r)⟩)

sort = λf.λxs.rec(xs, Nil↦ Nil, Cons↦⟨y,⟨ys,r⟩⟩.insert f y force(r))
```

FIGURE 2. Translation of `(f:Int→Int→Bool) (x:Int) (y:Int)`. We assume `f`, `x` and `y` are variables, so the cost of their translation is 0. Since `f` is a function of two arguments, the cost of `f` is 0 unless `f` is fully applied. Notice that `f` in $\|\mathtt{f}\|$ is a source variable while plain `f` is a potential variable.

$$\|\mathtt{f\ \ x\ \ y}\| = (1 + \|\mathtt{f\ \ x}\|_c + \|\mathtt{y}\|_c) +_c \|\mathtt{f\ \ x}\|_p \|\mathtt{y}\|_p$$

$$= (1 + 1 + \|\mathtt{f}\|_c + \|\mathtt{x}\|_c + (\|\mathtt{f}\|_p \|\mathtt{x}\|_p)_c + \|\mathtt{y}\|_c) +_c \|\mathtt{f}\|_p \|\mathtt{x}\|_p \|\mathtt{y}\|_p$$

$$= (2 + \langle \mathtt{0}, \mathtt{f} \rangle_c + \langle \mathtt{0}, \mathtt{x} \rangle_c + \langle \mathtt{0}, \mathtt{y} \rangle_c) +_c \langle \mathtt{0}, \mathtt{f} \rangle_p \langle \mathtt{0}, \mathtt{x} \rangle_p \langle \mathtt{0}, \mathtt{y} \rangle_p$$

$$= (2 + 0 + 0 + 0) +_c (\mathtt{f\ \ x\ \ y})$$

$$= \langle 2 + ((\mathtt{f\ \ x})_p\ \ \mathtt{y})_c, ((\mathtt{f\ \ x})_p\ \ \mathtt{y})_p \rangle$$

FIGURE 3. Translation of $\texttt{True}{\to}\texttt{Cons}\langle\texttt{x},\texttt{Cons}\langle\texttt{y},\texttt{ys}\rangle\rangle$ in the inner `rec` of `insert`. In this case the element we are inserting into the list comes before the head of the list under the ordering given by `f`.

$$
\begin{aligned}
&\|\texttt{True}{\mapsto}\texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\rangle\| \\
&= \texttt{True}{\mapsto}\, 1 +_c \|\texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\rangle\| \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\|\langle\texttt{x}\,,\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\rangle\|_c,\texttt{Cons}\|\langle\texttt{x}\,,\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\rangle\|_p\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\langle\|\texttt{x}\|_c + \|\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\|_c, \langle\|\texttt{x}\|_p, \|\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle_c\,, \\
&\qquad\qquad \texttt{Cons}\langle\|\texttt{x}\|_c + \|\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\|_c, \langle\|\texttt{x}\|_p, \|\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle_p\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\|\texttt{x}\|_c + \|\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\|_c,\texttt{Cons}\langle\|\texttt{x}\|_p, \|\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\langle\texttt{0}\,,\texttt{x}\rangle_c + \langle\|\langle\texttt{y}\,,\texttt{ys}\rangle\|_c,\texttt{Cons}\|\langle\texttt{y}\,,\texttt{ys}\rangle\|_p\rangle_c\,, \\
&\qquad\qquad \texttt{Cons}\langle\langle\texttt{0}\,,\texttt{x}\rangle_p, \langle\|\langle\texttt{y}\,,\texttt{ys}\rangle\|_c,\texttt{Cons}\|\langle\texttt{y}\,,\texttt{ys}\rangle\|_p\rangle_p\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle 0 + \|\langle\texttt{y}\,,\texttt{ys}\rangle\|_c\,, \texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\|\langle\texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\langle\|\texttt{y}\|_c + \|\texttt{ys}\|_c, \langle\|\texttt{y}\|_p, \|\texttt{ys}\|_p\rangle\rangle_c\,, \texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\|\texttt{y}\|_c + \|\texttt{ys}\|_c, \langle\|\texttt{y}\|_p, \|\texttt{ys}\|_p\rangle\rangle_p\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\|\texttt{y}\|_c + \|\texttt{ys}\|_c\,, \texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\|\texttt{y}\|_p, \|\texttt{ys}\|_p\rangle\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle\langle\texttt{0}\,,\texttt{y}\rangle_c + \langle\texttt{0}\,,\texttt{ys}\rangle_c\,, \texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\langle\texttt{0}\,,\texttt{y}\rangle_p, \langle\texttt{0}\,,\texttt{ys}\rangle_p\rangle\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle 0,\texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\rangle\rangle \\
&= \texttt{True}{\mapsto}\, 1 +_c \langle 1,\texttt{Cons}\langle\texttt{x}\,,\texttt{Cons}\langle\texttt{y}\,,\texttt{ys}\rangle\rangle\rangle
\end{aligned}
$$

FIGURE 4. Translation of the `False` branch of the inner `rec` of `insert`. `r` stands for the recursive call, and has type `susp list`. In this case the element we are inserting into the list comes after the head of the list under the ordering given by `f`.

$$
\begin{aligned}
&\|\texttt{False}\mapsto\texttt{Cons}\langle \texttt{y}\,,\texttt{force(r)}\rangle\| \\
&\quad= \texttt{False}\mapsto 1 +_c \|\texttt{Cons}\langle \texttt{y}\,,\texttt{force(r)}\rangle\| \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\|\langle \texttt{y}\,,\texttt{force(r)}\rangle\|_c\,,\texttt{Cons}\|\langle \texttt{y}\,,\texttt{force(r)}\rangle\|_p\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\langle\|\texttt{y}\|_c + \|\texttt{force(r)}\,\|_c, \langle\|\texttt{y}\|_p, \|\texttt{force(r)}\,\|_p\rangle\rangle_c\,, \\
&\qquad\qquad \texttt{Cons}\langle\|\texttt{y}\|_c + \|\texttt{force(r)}\,\|_c, \langle\|\texttt{y}\|_p, \|\texttt{force(r)}\,\|_p\rangle\rangle_p\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\|\texttt{y}\|_c + \|\texttt{force(r)}\,\|_c\,,\texttt{Cons}\langle\|\texttt{y}\|_p, \|\texttt{force(r)}\,\|_p\rangle\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\langle \texttt{0}\,,\texttt{y}\rangle_c + (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_c,\texttt{Cons}\langle\langle \texttt{0}\,,\texttt{y}\rangle_p, (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)\rangle\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle 0{+}\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle \\
&\quad= \texttt{False}\mapsto \langle 1{+}\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle
\end{aligned}
$$

FIGURE 5. Translation of the inner `rec` in `insert`. In the `True` case, we have found the place of `x` in the list and we so stop. In the `False` case, `x` comes after the head of list under the ordering given by `f` and we must recurse on the tail of the list.

$$\|\texttt{rec(f x y,True}\mapsto\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\texttt{,False}\mapsto\texttt{Cons}\langle\texttt{y,force(r)}\rangle\rangle\|$$

$$= \|\texttt{f x y}\|_c +_c \texttt{rec}(\|\texttt{f x y}\|_p, \texttt{True} \mapsto 1 +_c \|\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\|, \texttt{False} \mapsto 1 +_c \|\texttt{Cons}\langle\texttt{y,force(r)}\rangle\|)$$

$$= 2 + ((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_c +_c \texttt{rec}(((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_p, \texttt{True} \mapsto 1 +_c \|\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\|, \texttt{False} \mapsto 1 +_c \|\texttt{Cons}\langle\texttt{y,force(r)}\rangle\|)$$

$$= 2 + ((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_c +_c \texttt{rec}(((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_p, \texttt{True} \mapsto \langle 1,\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\rangle \texttt{False} \mapsto 1 +_c \|\texttt{Cons}\langle\texttt{y,force(r)}\rangle\|)$$

$$= 2 + ((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_c +_c \texttt{rec}(((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_p, \texttt{True} \mapsto \langle 1,\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\rangle \texttt{False} \mapsto \langle 1{+}\texttt{r}_c,\texttt{Cons}\langle\texttt{y,r}_p\rangle\rangle)$$

FIGURE 6. Translation of the `Nil` branch of the outer `rec` in `insert`. The insertion of an element into an empty list results in a singleton list containing only the element. This branch is also reached when the ordering given by `f` dictates `x` comes after than everything in the list, and should be placed at the back of the list.

$$
\begin{aligned}
&\|\texttt{Nil} \mapsto \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\| \\
&\quad = \texttt{Nil} \mapsto 1 +_c \|\texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\| \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \|\langle \texttt{x}, \texttt{Nil}\rangle\|_c, \texttt{Cons}\|\langle \texttt{x}, \texttt{Nil}\rangle\|_p\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \langle \|\texttt{x}\|_c + \|\texttt{Nil}\|_c, \langle \|\texttt{x}\|_p, \|\texttt{Nil}\|_p\rangle\rangle_c, \texttt{Cons}\langle \|\texttt{x}\|_c + \|\texttt{Nil}\|_c, \langle \|\texttt{x}\|_p, \|\texttt{Nil}\|_p\rangle\rangle_p\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \|\texttt{x}\|_c + \|\texttt{Nil}\|_c, \texttt{Cons}\langle \|\texttt{x}\|_p, \|\texttt{Nil}\|_p\rangle\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \langle \texttt{0}, \texttt{x}\rangle_c + \langle \texttt{0}, \texttt{Nil}\rangle_c, \texttt{Cons}\langle \langle \texttt{0}, \texttt{x}\rangle_p, \langle \texttt{0}, \texttt{Nil}\rangle_p\rangle\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle 0 + 0, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle \\
&\quad = \texttt{Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle
\end{aligned}
$$

FIGURE 7. Translation of the `Cons` branch of the outer `rec` in `insert`. In this branch we recurse on a nonempty list. We check if `x` is comes before the head of the list under the ordering given by `f`, in which case we are done, otherwise we recurse on the tail of the list.

$$\|\texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.\texttt{rec(f x y, True} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle, \texttt{False} \mapsto \texttt{Cons}\langle \texttt{y},\texttt{force(r)}\rangle)\|$$

$$= \texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.1 +_c \|\texttt{rec(f x y, True} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle, \texttt{False} \mapsto \texttt{Cons}\langle \texttt{y},\texttt{force(r)}\rangle)\|$$

$$= \texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.1 +_c (2 + ((\texttt{f x})_p\ \texttt{y})_c) +_c \texttt{rec}(((\texttt{f x})_p\ \texttt{y})_p, \texttt{True}\mapsto\langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle\rangle \texttt{False}\mapsto\langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle))$$

$$= \texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(3 + ((\texttt{f x})_p\ \texttt{y})_c) +_c \texttt{rec}(((\texttt{f x})_p\ \texttt{y})_p, \texttt{True}\mapsto\langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle\rangle \texttt{False}\mapsto\langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle))$$

FIGURE 8. Translation of `insert`

$$\begin{aligned}
\|\texttt{insert}\| \;=\; & \|\lambda \texttt{f}.\lambda \texttt{x}.\lambda \texttt{xs}.\texttt{rec}(\texttt{xs},\ \texttt{Nil} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Nil}\rangle, \\
& \qquad \texttt{Cons} \mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.\texttt{rec}(\texttt{f}\ \texttt{x}\ \texttt{y},\ \texttt{True} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle, \\
& \qquad\qquad \texttt{False} \mapsto \texttt{Cons}\langle \texttt{y},\texttt{force}(\texttt{r})\rangle\rangle \\
\\
= \; & \langle \texttt{0},\lambda \texttt{f}.\langle \texttt{0},\lambda \texttt{x}.\langle 0,\lambda \texttt{xs}.\|\texttt{rec}(\texttt{xs},\ \texttt{Nil} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Nil}\rangle, \\
& \qquad \texttt{Cons} \mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.\texttt{rec}(\texttt{f}\ \texttt{x}\ \texttt{y},\ \texttt{True} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y}, \\
& \qquad\qquad \texttt{False} \mapsto \texttt{Cons}\langle \texttt{y},\texttt{force} \\
\\
= \; & \langle \texttt{0},\lambda \texttt{f}.\langle \texttt{0},\lambda \texttt{x}.\langle 0,\lambda \texttt{xs}.\langle \texttt{0},\texttt{xs}\rangle_c +_c \\
& \qquad \texttt{rec}(\langle \texttt{0},\texttt{xs}\rangle_p, \\
& \qquad\quad \texttt{Nil} \mapsto \langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Nil}\rangle\rangle \\
& \qquad\quad \texttt{Cons} \mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(3+((\langle \texttt{0},\texttt{f}\rangle_p\ \texttt{x})_p\ \texttt{y})_c) +_c \texttt{rec}((((\langle \texttt{0},\texttt{f}\rangle_p\ \texttt{x})_p\ \texttt{y})_p, \\
& \qquad\qquad\qquad \texttt{True} \mapsto \langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{y} \\
& \qquad\qquad\qquad \texttt{False} \mapsto \langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle)) \\
\\
= \; & \langle \texttt{0},\lambda \texttt{f}.\langle \texttt{0},\lambda \texttt{x}.\langle 0,\lambda \texttt{xs}. \\
& \qquad \texttt{rec}(\texttt{xs}, \\
& \qquad\quad \texttt{Nil} \mapsto \langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Nil}\rangle\rangle \\
& \qquad\quad \texttt{Cons} \mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(3+((\texttt{f}\ \texttt{x})_p\ \texttt{y})_c) +_c \texttt{rec}((\texttt{f}\ \texttt{x})_p\ \texttt{y})_p, \\
& \qquad\qquad\qquad \texttt{True} \mapsto \langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle \\
& \qquad\qquad\qquad \texttt{False} \mapsto \langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle))
\end{aligned}$$

FIGURE 9. The translation of `insert f x xs`. Unlike before, we do not assume that `f`, `x`, `xs` are variables. They may be expressions with non-zero costs.

$$\|\texttt{insert f x xs}\| = (1 + \|\texttt{insert f x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f x}\|_p \|\texttt{xs}\|_p$$

$$= (1 + \|\texttt{insert f x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f x}\|_p \|\texttt{xs}\|_p$$

$$= (2 + \|\texttt{insert f}\|_c + \|\texttt{x}\|_c + (\|\texttt{insert f}\|_p \|\texttt{x}\|_p)_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (2 + \|\texttt{insert f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (3 + \|\texttt{insert}\|_c + \|\texttt{f}\|_c + (\|\texttt{insert}\|_p \|\texttt{f}\|_p \|\texttt{x}\|_p)_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert}\|_p \|\texttt{f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (3 + \|\texttt{f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert}\|_p \|\texttt{f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$\begin{aligned} = (3 + \|\texttt{f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c & \texttt{rec}(\|\texttt{xs}\|_p, \\ & \texttt{Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle \\ & \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle . (3 + ((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p\ \texttt{y})_c) +_c \texttt{rec}((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p\ \texttt{y})_p, \\ & \qquad \texttt{True} \mapsto \langle 1, \texttt{Cons}\langle \|\texttt{x}\|_p, \texttt{Cons}\langle \texttt{y}, \texttt{y} \\ & \qquad \texttt{False} \mapsto \langle 1 +_c \texttt{r}_c, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle)) \end{aligned}$$

FIGURE 10. Interpretation of lists as lengths

$$\begin{aligned}
[\![list]\!] &= \mathbb{Z} \times \mathbb{N}^\infty \\
D^{list} &= \{*\} + \{\mathbb{Z}\} \times \mathbb{N}^\infty \\
size_{list}(Nil) &= (-\infty, 0) \\
size_{list}(Cons(i,(j,n))) &= (max\{i,j\}, 1+n)
\end{aligned}$$

FIGURE 11. Interpretation of the inner `rec` of `insert` with lists abstracted to sizes

$[\![\texttt{rec}\,((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_p\,,$

$\quad \texttt{True} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}\,, \texttt{Cons}\langle \texttt{y}\,, \texttt{ys}\rangle\rangle$

$\quad \texttt{False} \mapsto \langle 1+\texttt{r}_c, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle)\ \ ]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto x, \texttt{y} \mapsto y, \texttt{ys} \mapsto (i,n), \texttt{r} \mapsto r\}$

$$f_{True}(\langle\rangle) = [\![\langle \texttt{1}\,, \texttt{Cons}\langle \texttt{x}\,, \texttt{Cons}\langle \texttt{y}\,, \texttt{ys}\rangle\rangle\rangle]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto x, \texttt{y} \mapsto y, \texttt{ys} \mapsto (i,n), \texttt{r} \mapsto r\}$$

$$= (1, (max\{x,y,i\}, 2+n))$$

$$f_{False}(\langle\rangle) = [\![\langle 1+\texttt{r}_c, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle)\ \ ]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto x, \texttt{y} \mapsto y, \texttt{ys} \mapsto (i,n), \texttt{r} \mapsto r\}$$

$$= (1 + \pi_0 r, (max\{y, \pi_0\pi_1 r\}, 1 + \pi_1\pi_1 r))$$

$$= \bigvee_{size(w) \leq \pi_1(\pi_1(f\ \ x)\ \ y)} case(w, (f_{True}, f_{False}))$$

$$= \bigvee_{size(w) \leq \pi_1(\pi_1(f\ \ x)\ \ y)} case(w, (\lambda\langle\rangle.(1, (max\{x,y,i\}, 2+n)), \lambda\langle\rangle.(1 + \pi_0 r, (max\{y, \pi_0\pi_1 r\}, 1 + \pi_1\pi_1$$

$$= (1, (max\{x,y,i\}, 2+n)) \vee (1 + \pi_0 r, (max\{y, \pi_0\pi_1 r\}, 1 + \pi_1\pi_1 r))$$

FIGURE 12. Interpretation of `rec` in `insert`.

$$g(i,n) = [\![\texttt{rec(xs,}$$
$$\texttt{Nil}\mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle$$
$$\texttt{Cons}\mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle.(3+((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_c)+_c \texttt{rec}((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_p,$$
$$\texttt{True}\mapsto \langle 1,\texttt{Cons}\langle \texttt{x}, \texttt{Cons}\langle \texttt{y}, \texttt{ys}\rangle\rangle\rangle$$
$$\texttt{False}\mapsto \langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle))]\!]\xi\{\texttt{f}\mapsto f,\texttt{x}\mapsto x$$

$$f_{Nil}(\langle\rangle) = [\![\langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle]\!]\xi\{\texttt{f}\mapsto f,\texttt{x}\mapsto x,\texttt{xs}\mapsto (i,n)\}$$

$$f_{Nil}(\langle\rangle) = (1,(x,1))$$

$$f_{Cons}((j,(j,m))) = [\![(3+((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_c)+_c \texttt{rec}(\ldots)]\!]\xi$$
$$\{\texttt{f}\mapsto f,\texttt{x}\mapsto x,\texttt{xs}\mapsto (i,n), \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (map^{\mathbb{Z}\times\mathbb{N}^\infty}(\lambda a.(a, [\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto a\}), (j,$$

$$= [\![\ldots]\!]\xi\{\ldots \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (j, map^{\mathbb{N}^\infty}(\lambda a.(a, [\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto a\}), (j,m)))\}$$

$$= [\![\ldots]\!]\xi\{\ldots \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (j, ((j,m), [\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto (j,m)\}))\}$$

$$= [\![\ldots]\!]\xi\{\ldots \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (j, ((j,m), g(j,m)))\}$$

$$= (3+\pi_0(\pi_1(f\ \ x)\ \ j))+_c$$
$$((1,(max\{x,j\},2+m))) \vee (1+\pi_0 g(j,m), (max\{j,\pi_0\pi_1 g(j,m)\}, 1+\pi_1\pi_1 g(j,m)))$$

$$= (3+\pi_0(\pi_1(f\ \ x)\ \ j))+_c$$
$$(1\vee(1+\pi_0 g(j,m)), (max\{x,j,\pi_0\pi_1 g(j,m)\}, 2+m\vee 1+\pi_1\pi_1 g(j,m)))$$

$$= (4+\pi_0(\pi_1(f\ \ x)\ \ j)+\pi_0 g(j,m), (max\{x,j\pi_0\pi_1 g(j,m)\}, 2+m\vee 1+\pi_1\pi_1 g(j,m)))$$

$$g(i,n) = \bigvee_{size(z)\leq(i,n)} case(z,(f_{Nil}, f_{Cons}))$$

FIGURE 13. Translation of `Nil` branch of `sort`.

$$\|\texttt{Nil} \mapsto \texttt{Nil}\|$$

$$= \texttt{Nil} \mapsto 1 +_c \|\texttt{Nil}\|$$

$$= \texttt{Nil} \mapsto 1 +_c \langle \texttt{0}, \texttt{Nil} \rangle$$

$$= \texttt{Nil} \mapsto \langle \texttt{1}, \texttt{Nil} \rangle$$

FIGURE 14. Translation of `Cons` branch of `sort`.

$$\|\texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . \texttt{insert f y force(r)}$$

$$= \texttt{Cons} \mapsto 1 +_c \|\texttt{insert f y force(r)}\|$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c (\|\texttt{force(r)}\|_c) +_c \|\texttt{insert f y}\|_p \|\texttt{force(r)}\|_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c ((\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_c) +_c \|\texttt{insert f y}\|_p (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c \texttt{r}_c +_c \|\texttt{insert f y}\|_p \texttt{r}_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c \texttt{r}_c +_c 3 +_c \|\texttt{insert}\|_p \ \texttt{f y r}_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . (4 + \texttt{r}_c) +_c \|\texttt{insert}\|_p \ \texttt{f y r}_p$$

FIGURE 15. Translation of `sort`

$$\|\texttt{sort}\| = \langle 0, \lambda \texttt{f} . \langle 0, \lambda \texttt{xs} . \|\texttt{rec(xs, Nil}\mapsto \texttt{Nil,}$$
$$\texttt{Cons}\mapsto \langle \texttt{y,}\langle \texttt{ys,r}\rangle\rangle . \texttt{insert f y force(r))}\|\rangle\rangle$$

$$= \langle 0, \lambda \texttt{f} . \langle 0, \lambda \texttt{xs} . \texttt{rec(xs, Nil}\mapsto \langle \texttt{1,Nil}\rangle$$
$$\texttt{Cons}\mapsto \langle \texttt{y,}\langle \texttt{ys,r}\rangle\rangle . 4 +_c \texttt{r}_c +_c \|\texttt{insert}\|_p \ \texttt{f y r}_p$$

FIGURE 16. Translation of `sort` applied to variables `f` and `xs`

$$\|\texttt{sort f xs}\| = (1 + \|\texttt{sort f}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{sort f}\|_p \|\texttt{xs}\|_p$$

$$= (1 + (1 + \|\texttt{sort}\|_c + \|\texttt{f}\|_c + \|\texttt{xs}\|_c)) +_c \|\texttt{sort}\|_p \|\texttt{f}\|_p \|\texttt{xs}\|_p$$

$$= (1 + (1 + 0 + 0 + 0)) +_c \|\texttt{sort}\|_p \|\texttt{f}\|_p \|\texttt{xs}\|_p$$

$$= 2 +_c \|\texttt{sort}\|_p \texttt{f}\|_p \|\texttt{xs}\|_p$$

$$= 2 +_c \texttt{rec(}\|\texttt{xs}\|_p\texttt{, Nil}\mapsto \langle \texttt{1,Nil}\rangle$$
$$\texttt{Cons}\mapsto \langle \texttt{y,}\langle \texttt{ys,r}\rangle\rangle . (4 +_c \texttt{r}_c) +_c \|\texttt{insert}\|_p \ \texttt{f y r}_p$$

FIGURE 17. Interpretation of `rec` in `sort`.TO DO FIX THIS

$$g(i,n) = [\![\texttt{rec}(\|\texttt{xs}\|_p,\ \texttt{Nil}\mapsto \langle \texttt{1},\texttt{Nil}\rangle \\ \texttt{Cons}\mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(4{+}\texttt{r}_c) +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p)]\!]\xi\{xs\mapsto n\}$$

$$= [\![\texttt{rec}(\|\texttt{xs}\|_p,\ \texttt{Nil}\mapsto \langle \texttt{1},\texttt{Nil}\rangle \\ \texttt{Cons}\mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(4{+}\texttt{r}_c) +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p)]\!]\xi\{xs\mapsto n\}$$

$$= [\![\texttt{rec}(\|\texttt{xs}\|_p,\ \texttt{Nil}\mapsto \langle \texttt{1},\texttt{Nil}\rangle \\ \texttt{Cons}\mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(4{+}\texttt{r}_c) +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p)]\!]\xi\{xs\mapsto n\}$$

$$= \bigvee_{size(z)\le n} case(z,(f_{Nil}, f_{Cons}))$$

$$f_{Nil}(\langle\rangle) = [\![\langle \texttt{1},\texttt{Nil}\rangle]\!]\xi$$

$$= f_{Nil}(\langle\rangle) = (1,(\neg\infty,0))$$

$$f_{Cons}((j,m)) = [\![\ldots]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto map^{j\times\mathbb{N}^\infty}(\lambda a.(a,[\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto a\}),(j,m))\}$$

$$= [\![\ldots]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto (map^{int}(\lambda a.(\ldots),j), map^{\mathbb{N}^\infty}(\lambda a.(a,[\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto a\}),m))\}$$

$$= [\![\ldots]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto (j,(m,[\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto m\}))\}$$

$$= [\![(4{+}\texttt{r}_c) +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto (j,(m,g(j,m))\}$$

$$= (4+\pi_0 g(j,m)) +_c insert\ \ f\ \ j\ \ \pi_1 g(j,m)$$

$$g(i,n) = \bigvee_{size(z)\le n} case(z,(\lambda(\langle\rangle).(1,(\neg\infty,0)), \lambda(j,m).(4+\pi_0 g(j,m)) +_c insert\ \ f\ \ j\ \ \pi_1 g(j,m)))$$

3. Insertion Sort

Insertion sort is a quadratic time sorting algorithm which sorts a list by inserting an element from an unsorted segment of a container into a sorted segment of the container. Although the asymptotic complexity of insertion sort is less than the optimal $\mathcal{O}(nlog_2n)$, insertion sort does have redeeming attributes. Insertion sort has small constant factors, making it more efficient on small datasets. The standard Python sorting algorithm, timsort, is a hybrid sorting algorithm that uses mergesort and switches to insertion sort for small datasets (**?**). Insertion sort may be done in-place (Cormen et al. [2001]). The running time of insertion sort is $\mathcal{O}(n^2)$.

3.1. Insert. `sort` relies on the function `insert` to insert the head of the list into the result of recursively sorted tail of the list. We will begin with a translation and interpretation of `insert`.

3.1.1. *Translation.* The translation of `insert` is broken into chunks to make it more managable. Figure 19 steps through the translation of the comparison function `<=` applied to variables `x` and `y`.

The translation `true` and `false` branches are given in figures 20 and 21 respectively.

Figure 22 uses the translation of `f x y` and the `true` and `false` branches to construct the translation of the inner `rec` construct.

The `Nil` and `Cons` branches of the outer `rec` construct are given in figures 23 and 24, respectively.

We put these together to give the translation of `insert`.

Finally we give a translation of `insert f x xs` in figure 26 because this is the term we will interpret in a size-based semantics.

The result is:

$$\begin{aligned}\|\texttt{insert f x xs}\| = (3 + \|\texttt{f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c)\\ +_c \texttt{rec}(\|\texttt{xs}\|_p,\\ \texttt{Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle\end{aligned}$$

$$\texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . (3 + ((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p \ \texttt{y})_c) +_c \texttt{rec}((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p \ \texttt{y})_p,$$
$$\texttt{True} \mapsto \langle 1, \texttt{Cons} \langle \|\texttt{x}\|_p, \texttt{Cons} \langle \texttt{y}, \texttt{y}$$
$$\texttt{False} \mapsto \langle 1 +_c \texttt{r}_c, \texttt{Cons} \langle \texttt{y}, \texttt{r}_p \rangle \rangle \rangle)$$

3.1.2. *Interpretation.* We well use an interpretation of lists as their lengths. Figure 27 gives formalizes this interpretation.

First we interpret the `rec`, which drives of the cost of `insert`. As in the translation, we break the interpretation up to make it more managable. We will write map, λ and $+_c$ in the semantics, which stand for the semantic equivalents of `map`, λ and $+_c$ in the syntax. The definitions of these semantic functions mirror the definitions of their syntactic equivalents. Figures 28 and 29 walk through the interpretation.

The initial result is given in equation 10.

$$f_{Nil}(\langle\rangle) = (1, 1) \tag{8}$$

$$f_{Cons}((1, m)) = (4 + \pi_0(\pi_1(f\ 1)\ 1) + \pi_0 g(m), (2 + m) \vee (1 + \pi_1 g(m))) \tag{9}$$

$$g(n) = \bigvee_{size(z) \leq n} case(z, (f_{Nil}, f_{Cons})) \tag{10}$$

This recurrence is difficult to work with. Specifically, we cannot apply traditional methods of solving it. We will manipulate it into a more usable form by eliminating the arbitrary maximum. Observe that for $n = 0$, $g(n) = f_{Nil}(\langle\rangle) = (1, 1)$. For $n > 0$,

$$\begin{aligned}
g(n) &= \bigvee_{size(z) \leq n} case(z, (f_{Nil}, f_{Cons})) \\
&= g(n-1) \vee \bigvee_{size(z) = n} case(z, (f_{Nil}, f_{Cons})) \\
&= g(n-1) \vee f_{Cons}(n) \\
&= g(n-1) \vee (4 + \pi_0(\pi_1(f\ 1)\ 1) + \pi_0 g(n-1), (1+n) \vee (1 + \pi_1 g(n-1))) && m = n - 1 \\
&= (4 + \pi_0(\pi_1(f\ 1)\ 1) + \pi_0 g(n-1), (1+n) \vee (1 + \pi_1 g(n-1))) && \text{lemma 4.1} \\
&= (4 + \pi_0(\pi_1(f\ 1)\ 1) + \pi_0 g(n-1), 1 + \pi_1 g(n-1)) && \text{lemma 4.2}
\end{aligned}$$

LEMMA 3.1. $g(n) > g(n-1)$

PROOF. TODO □

LEMMA 3.2. $\pi_1 g(n) > n$

PROOF. We prove this by induction on n.

case $n = 0$**::** $\pi_1 g(0) = 1$

case $n > 0$**::**

$$\begin{aligned}\pi_1 g(n) &= \pi_1(g(n-1) \vee (4 + \pi_0(\pi_1(f\ 1)\ 1) + \pi_0 g(n-1), (1+n) \vee (1 + \pi_1 g(n-1))))\\ &= \pi_1 g(n-1) \vee (1+n) \vee (1 + \pi_1 g(n-1))\\ &\geq n - 1 \vee (1+n) \vee (1+n-1)\\ &\geq 1 + n\\ &> n\end{aligned}$$

□

Equation 11 shows the extracted recurrence. Without the arbitrary maximum, it is much more obvious how to find a solution to the recurrence. The recurrence is from a potential to a complexity, consequently we can extract a recurrence for the cost, equation 12, and a recurrence for the potential, equation 13, simply by taking the projections of equation 11. The extracted recurrences for the cost and potential can then be solved by the substitution method.

$$g(n) = \begin{cases}(1,1) & n = 0\\ (4 + \pi_0(\pi_1(f\ 1)\ 1) + \pi_0 g(n-1), 1 + \pi_1 g(n-1)) & n > 0\end{cases} \tag{11}$$

The cost recurrence is given by $\pi_0 \circ g$.

$$c(n) = \begin{cases}1 & n = 0\\ 4 + \pi_0(\pi_1(f\ 1)\ 1) + c(n-1) & n > 0\end{cases} \tag{12}$$

This recurrence is quite simple to solve. The solution and proof of the solution are given in theorem 4.3.

THEOREM 3.3. $c(n) = (4 + \pi_0(\pi_1(f\ 1)\ 1))n + 1$

PROOF. We prove this by induction on n.

case $n = 0$**:** $c(0) = \pi_0 g(0) = 1$

case $n > 0$**:**

$$\begin{aligned} c(n) &= 4 + \pi_0(\pi_1(f\ 1)\ 1) + c(n-1) \\ &= 4 + \pi_0(\pi_1(f\ 1)\ 1) + (4 + \pi_0(\pi_1(f\ 1)\ 1))(n-1) + 1 \\ &= (4 + \pi_0(\pi_1(f\ 1)\ 1))n + 1 \end{aligned}$$

□

The solution tells us the cost of the `rec` construct in insert is linear in the size of the list. The constant factor cannot be determined because we do not know the cost of applying f to its arguments.

The potential recurrence is given by $\pi_1 \circ g$.

$$p(n) = \begin{cases} 1 & n = 0 \\ 1 + p(n-1) & n > 0 \end{cases} \tag{13}$$

THEOREM 3.4. $p(n) = 1 + n$

PROOF. We prove this by induction on n.

case $n = 0$**:** $p(0) = 1$

case $n > 0$**:** $p(n) = 1 + p(n-1) = 1 + n$

□

The solution of this recurrence tells us the size of the output in terms of the size of the input. As one would expect of `insert`, the size of the output is one larger than the size of the input.

3.2. Sort.

3.2.1. *Translation.* The translation of sort is shown in figure 32. The translation of the `Nil` and `Cons` branches in the `rec` are walked through in figures 30 and 31, respectively. The translation of `sort` applied to its arguments is given in figure 33.

3.2.2. *Interpretation.* The `rec` construct again drives the cost and potential of `sort`. The walkthrough of the interpretation of the `rec` is given in figure 34. Equation 14 shows the initial recurrence extracted.

$$g(n) = \bigvee_{size(z)\leq n} case(z, (\lambda(\langle\rangle).(1,0), \lambda(1,m).4 + \pi_0 g(m)) +_c insert\ f\ 1\ \pi_1 g(m)) \tag{14}$$

We work the recurrence into a more recognisable form using some manipulation of the big max operator and some facts about *insert*. Observe for $n = 0$, $g(n) = (1,0)$ and for $n > 0$

$$\begin{aligned}
g(n) &= \bigvee_{size(z)\leq n} case(z, (\lambda(\langle\rangle).(1,0), \lambda(1,m).4 + \pi_0 g(m)) +_c insert\ f\ 1\ \pi_1 g(m)) \\
&= g(n-1) \vee \bigvee_{size(z)=n} case(z, (\lambda(\langle\rangle).(1,0), \lambda(1,m).4 + \pi_0 g(m)) +_c insert\ f\ 1\ \pi_1 g(m)) \\
&= g(n-1) \vee (4 + \pi_0 g(n-1)) +_c insert\ f\ 1\ \pi_1 g(n-1) \\
&= g(n-1) \vee (4 + \pi_0 g(n-1) + \pi_0(insert\ f\ 1\ \pi_1 g(n-1)), \pi_1(insert\ f\ 1\ \pi_1 g(n-1))) \\
&\text{since } \pi_0(insert\ f\ 1\ m) > 0 \text{ and } \pi_1(insert\ f\ 1\ m) = 1 + m \\
&= (4 + \pi_0 g(n-1) + \pi_0(insert\ f\ 1\ \pi_1 g(n-1)), \pi_1(insert\ f\ 1\ \pi_1 g(n-1)))
\end{aligned}$$

So our simplified recurrence is

$$g(n) = \begin{cases} (1,0) & n = 0 \\ (4 + \pi_0 g(n-1) + \pi_0(insert\ f\ 1\ \pi_1 g(n-1)), \pi_1(insert\ f\ 1\ \pi_1 g(n-1))) & n > 0 \end{cases} \tag{15}$$

From this we can extract recurrences for the cost and the potential simply by taking projections from g. We begin with the potential because we will require the solution to the potential recurrence to solve the cost recurrence.

Let $p = \pi_1 \circ g$.

$$p(n) = \begin{cases} 0 & n = 0 \\ \pi_1(insert\ f\ 1\ \pi_1 g(n-1))) & n > 0 \end{cases} \tag{16}$$

We prove the size of the potential of the output is same as the size of the input. In other words, `sort` does not change the size of the list.

THEOREM 3.5. $p(n) = n$

PROOF. We prove this by straightforward induction on n.

case $n = 0$**:** $p(0) = 0$

case $n > 0$**:** $p(n) = \pi_1(insert\ f\ 1\ pi_g(n-1)) = \pi_1(insert\ f\ 1\ (n-1)) = n$

□

Let $c = \pi_0 \circ g$.

$$c(n) = \begin{cases} 1 & n = 0 \\ 4 + \pi_0 g(n-1) + \pi_0(insert\ f\ 1\ \pi_1 g(n-1) & n > 0 \end{cases} \tag{17}$$

THEOREM 3.6.

PROOF. We prove this by straightforward induction on n.

case $n = 0$**::** $c(0) = 1$

case $n > 0$**::**

$$\begin{aligned} c(n) &= 4 + \pi_0 g(n-1) + \pi_0(insert\ f\ 1\ \pi_1 g(n-1)) \\ &= 4 + \pi_0 g(n-1) + \pi_0(insert\ f\ 1\ (n-1) \\ &= 4 + (4 + \pi_0(\pi_1(f\ 1)\ 1))(n-1) + 1) + \pi_0 g(n-1) \end{aligned}$$

TODO COMPLETE

□

OTHER THINGS TODO: FIX FREE VARIABLES IN APPLICATION SMOOTH OUT THE INSERT F X XS SITUATION THE INTERPRETATION OF INSERT IS NOT QUITE CORRECT.

FIGURE 18. Insertion sort in the source language

```
data list = Nil of unit | Cons of int × list

insert = λx.λxs.rec(xs, Nil↦ Cons⟨x,Nil⟩,
                        Cons↦⟨y,⟨ys,r⟩⟩.rec(x <= y, True ↦ Cons⟨x,Cons⟨y,ys⟩⟩
                                                     False ↦ Cons⟨y,force(r)⟩⟩

sort = λxs.rec(xs, Nil↦ Nil, Cons↦⟨y,⟨ys,r⟩⟩.insert y force(r))
```

FIGURE 19. Translation of `x <= y`. We assume `x` and `y` are variables. Consequently the cost of the translation of both is 0. We assume `<=` is a function with a constant cost of 2. Since `<=` is a function of two arguments, the cost of `<=` is 0 unless `<=` is fully applied.

$$\begin{aligned}\|\texttt{x <= y}\| &= (2 + \|\texttt{x}\|_c + \|\texttt{y}\|_c) +_c \|\texttt{x}\|_p \ \texttt{<=}\ \|\texttt{y}\|_p \\ &= \langle 2 +_c (\texttt{x <= y}) \rangle \\ &= \langle \langle 4, (\texttt{x <= y})_p \rangle\end{aligned}$$

FIGURE 20. Translation of $\texttt{True}{\to}\texttt{Cons}\langle\texttt{x},\texttt{Cons}\langle\texttt{y},\texttt{ys}\rangle\rangle$ in the inner `rec` of `insert`. In this case the element we are inserting into the list comes before the head of the list under the ordering given by `f`.

$$= \texttt{True}\mapsto 1 +_c \|\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\rangle\|$$

$$= \texttt{True}\mapsto 1 +_c \langle\|\langle \texttt{x}\,,\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\rangle\|_c,\texttt{Cons}\|\langle \texttt{x}\,,\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\rangle\|_p\rangle$$

$$\begin{aligned} = \texttt{True}\mapsto 1 +_c \langle\langle\|\texttt{x}\|_c + \|\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\|_c, \langle\|\texttt{x}\|_p, \|\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle_c\,,\\ \texttt{Cons}\langle\|\texttt{x}\|_c + \|\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\|_c, \langle\|\texttt{x}\|_p, \|\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle_p\rangle \end{aligned}$$

$$= \texttt{True}\mapsto 1 +_c \langle\|\texttt{x}\|_c + \|\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\|_c,\texttt{Cons}\langle\|\texttt{x}\|_p, \|\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle$$

$$\begin{aligned} = \texttt{True}\mapsto 1 +_c \langle\langle \texttt{0}\,,\texttt{x}\rangle_c + \langle\|\langle \texttt{y}\,,\texttt{ys}\rangle\|_c,\texttt{Cons}\|\langle \texttt{y}\,,\texttt{ys}\rangle\|_p\rangle_c\,,\\ \texttt{Cons}\langle\langle \texttt{0}\,,\texttt{x}\rangle_p, \langle\|\langle \texttt{y}\,,\texttt{ys}\rangle\|_c,\texttt{Cons}\|\langle \texttt{y}\,,\texttt{ys}\rangle\|_p\rangle_p\rangle\rangle \end{aligned}$$

$$= \texttt{True}\mapsto 1 +_c \langle 0 + \|\langle \texttt{y}\,,\texttt{ys}\rangle\|_c\,,\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\|\langle \texttt{y}\,,\texttt{ys}\rangle\|_p\rangle\rangle$$

$$= \texttt{True}\mapsto 1 +_c \langle\langle\|\texttt{y}\|_c + \|\texttt{ys}\|_c, \langle\|\texttt{y}\|_p, \|\texttt{ys}\|_p\rangle\rangle_c\,,\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\langle\|\texttt{y}\|_c + \|\texttt{ys}\|_c, \langle\|\texttt{y}\|_p, \|\texttt{ys}\|_p\rangle\rangle_p\rangle\rangle$$

$$= \texttt{True}\mapsto 1 +_c \langle\|\texttt{y}\|_c + \|\texttt{ys}\|_c\,,\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\langle\|\texttt{y}\|_p, \|\texttt{ys}\|_p\rangle\rangle\rangle$$

$$= \texttt{True}\mapsto 1 +_c \langle\langle \texttt{0}\,,\texttt{y}\rangle_c + \langle \texttt{0}\,,\texttt{ys}\rangle_c\,,\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\langle\langle \texttt{0}\,,\texttt{y}\rangle_p, \langle \texttt{0}\,,\texttt{ys}\rangle_p\rangle\rangle\rangle$$

$$= \texttt{True}\mapsto 1 +_c \langle 0,\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\rangle\rangle$$

$$= \texttt{True}\mapsto \langle 1,\texttt{Cons}\langle \texttt{x}\,,\texttt{Cons}\langle \texttt{y}\,,\texttt{ys}\rangle\rangle\rangle$$

FIGURE 21. Translation of the `False` branch of the inner `rec` of `insert`. `r` stands for the recursive call, and has type `susp list`. In this case the element we are inserting into the list comes after the head of the list under the ordering given by `f`.

$$
\begin{aligned}
&\|\texttt{False}\mapsto\texttt{Cons}\langle \texttt{y}\,,\texttt{force(r)}\rangle\| \\
&\quad= \texttt{False}\mapsto 1 +_c \|\texttt{Cons}\langle \texttt{y}\,,\texttt{force(r)}\rangle\| \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\|\langle \texttt{y}\,,\texttt{force(r)}\rangle\|_c\,,\texttt{Cons}\|\langle \texttt{y}\,,\texttt{force(r)}\rangle\|_p\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\langle\|\texttt{y}\|_c + \|\texttt{force(r)}\,\|_c, \langle\|\texttt{y}\|_p, \|\texttt{force(r)}\,\|_p\rangle\rangle_c\,, \\
&\qquad\qquad \texttt{Cons}\langle\|\texttt{y}\|_c + \|\texttt{force(r)}\,\|_c, \langle\|\texttt{y}\|_p, \|\texttt{force(r)}\,\|_p\rangle\rangle_p\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\|\texttt{y}\|_c + \|\texttt{force(r)}\,\|_c\,,\texttt{Cons}\langle\|\texttt{y}\|_p, \|\texttt{force(r)}\,\|_p\rangle\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle\langle \texttt{0}\,,\texttt{y}\rangle_c + (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_c,\texttt{Cons}\langle\langle \texttt{0}\,,\texttt{y}\rangle_p, (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)\rangle\rangle \\
&\quad= \texttt{False}\mapsto 1 +_c \langle 0+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle \\
&\quad= \texttt{False}\mapsto \langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle
\end{aligned}
$$

FIGURE 22. Translation of the inner `rec` in `insert`. In the `True` case, we have found the place of `x` in the list and we so stop. In the `False` case, `x` comes after the head of list under the ordering given by `f` and we must recurse on the tail of the list.

$$\begin{aligned}
&\|\texttt{rec(f x y,True}\mapsto\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\texttt{,False}\mapsto\texttt{Cons}\langle\texttt{y,force(r)}\rangle\rangle\| \\
&\quad = \|\texttt{f x y}\|_c +_c \texttt{rec}(\|\texttt{f x y}\|_p\texttt{,True}\mapsto 1 +_c \|\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\|\texttt{,} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{False}\mapsto 1 +_c \|\texttt{Cons}\langle\texttt{y,force(r)}\rangle\|\texttt{)} \\
&\quad = 2 + ((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_c +_c \texttt{rec}(((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_p\texttt{,} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{True}\mapsto 1 +_c \|\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\|\texttt{,} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{False}\mapsto 1 +_c \|\texttt{Cons}\langle\texttt{y,force(r)}\rangle\|\texttt{)} \\
&\quad = 2 + ((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_c +_c \texttt{rec}(((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_p\texttt{,} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{True}\mapsto \langle 1,\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{False}\mapsto 1 +_c \|\texttt{Cons}\langle\texttt{y,force(r)}\rangle\|\texttt{)} \\
&\quad = 2 + ((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_c +_c \texttt{rec}(((\|\texttt{f}\|_p\ \texttt{x})_p\ \texttt{y})_p\texttt{,} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{True}\mapsto \langle 1,\texttt{Cons}\langle\texttt{x,Cons}\langle\texttt{y,ys}\rangle\rangle\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{False}\mapsto \langle 1+\texttt{r}_c,\texttt{Cons}\langle\texttt{y,r}_p\rangle\rangle\texttt{)}
\end{aligned}$$

FIGURE 23. Translation of the `Nil` branch of the outer `rec` in `insert`. The insertion of an element into an empty list results in a singleton list containing only the element. This branch is also reached when the ordering given by `f` dictates `x` comes after than everything in the list, and should be placed at the back of the list.

$$
\begin{aligned}
&\|\texttt{Nil} \mapsto \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\| \\
&\quad = \texttt{Nil} \mapsto 1 +_c \|\texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\| \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \|\langle \texttt{x}, \texttt{Nil}\rangle\|_c, \texttt{Cons}\|\langle \texttt{x}, \texttt{Nil}\rangle\|_p\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \langle \|\texttt{x}\|_c + \|\texttt{Nil}\|_c, \langle \|\texttt{x}\|_p, \|\texttt{Nil}\|_p\rangle\rangle_c, \texttt{Cons}\langle \|\texttt{x}\|_c + \|\texttt{Nil}\|_c, \langle \|\texttt{x}\|_p, \|\texttt{Nil}\|_p\rangle\rangle_p\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \|\texttt{x}\|_c + \|\texttt{Nil}\|_c, \texttt{Cons}\langle \|\texttt{x}\|_p, \|\texttt{Nil}\|_p\rangle\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle \langle \texttt{0}, \texttt{x}\rangle_c + \langle \texttt{0}, \texttt{Nil}\rangle_c, \texttt{Cons}\langle \langle \texttt{0}, \texttt{x}\rangle_p, \langle \texttt{0}, \texttt{Nil}\rangle_p\rangle\rangle \\
&\quad = \texttt{Nil} \mapsto 1 +_c \langle 0 + 0, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle \\
&\quad = \texttt{Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle
\end{aligned}
$$

FIGURE 24. Translation of the `Cons` branch of the outer `rec` in `insert`. In this branch we are recursing on a nonempty list. We check if `x` is comes before the head of the list under the ordering given by `f`, in which case we are done, otherwise we recurse on the tail of the list.

$$\|\texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.\texttt{rec(f x y, True} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle, \texttt{False} \mapsto \texttt{Cons}\langle \texttt{y},\texttt{force(r)}\rangle)\|$$

$$= \texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.1 +_c \|\texttt{rec(f x y, True} \mapsto \texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle, \texttt{False} \mapsto \texttt{Cons}\langle \texttt{y},\texttt{force(r)}\rangle)\|$$

$$= \texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.1 +_c (2 + ((\texttt{f x})_p\ \texttt{y})_c) +_c \texttt{rec}(((\texttt{f x})_p\ \texttt{y})_p, \texttt{True}\mapsto\langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle\rangle \texttt{False}\mapsto\langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle))$$

$$= \texttt{Cons}\mapsto\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.(3 + ((\texttt{f x})_p\ \texttt{y})_c) +_c \texttt{rec}(((\texttt{f x})_p\ \texttt{y})_p, \texttt{True}\mapsto\langle 1,\texttt{Cons}\langle \texttt{x},\texttt{Cons}\langle \texttt{y},\texttt{ys}\rangle\rangle\rangle \texttt{False}\mapsto\langle 1+\texttt{r}_c,\texttt{Cons}\langle \texttt{y},\texttt{r}_p\rangle\rangle))$$

FIGURE 25. Translation of `insert`

$$\begin{aligned}
\|\texttt{insert}\| \;=\; \|&\lambda\texttt{f}.\lambda\texttt{x}.\lambda\texttt{xs}.\texttt{rec}(\texttt{xs},\ \texttt{Nil}\mapsto\ \texttt{Cons}\langle\texttt{x},\texttt{Nil}\rangle,\\
&\qquad\texttt{Cons}\mapsto\langle\texttt{y},\langle\texttt{ys},\texttt{r}\rangle\rangle.\texttt{rec}(\texttt{f}\ \texttt{x}\ \texttt{y},\ \texttt{True}\ \mapsto\ \texttt{Cons}\langle\texttt{x},\texttt{Cons}\langle\texttt{y},\texttt{ys}\rangle\rangle,\\
&\qquad\qquad\texttt{False}\ \mapsto\ \texttt{Cons}\langle\texttt{y},\texttt{force}(\texttt{r})\rangle)\\
\\
=\ \langle\texttt{0},\lambda\texttt{f}.\langle\texttt{0},\lambda\texttt{x}.\langle 0,\lambda\texttt{xs}.\|&\texttt{rec}(\texttt{xs},\ \texttt{Nil}\mapsto\ \texttt{Cons}\langle\texttt{x},\texttt{Nil}\rangle,\\
&\qquad\texttt{Cons}\mapsto\langle\texttt{y},\langle\texttt{ys},\texttt{r}\rangle\rangle.\texttt{rec}(\texttt{f}\ \texttt{x}\ \texttt{y},\ \texttt{True}\ \mapsto\ \texttt{Cons}\langle\texttt{x},\texttt{Cons}\langle\texttt{y},\\
&\qquad\qquad\texttt{False}\ \mapsto\ \texttt{Cons}\langle\texttt{y},\texttt{force}\\
\\
=\ \langle\texttt{0},\lambda\texttt{f}.\langle\texttt{0},\lambda\texttt{x}.\langle 0,\lambda\texttt{xs}.\langle\texttt{0},\texttt{xs}\rangle_c+_c&\\
\texttt{rec}(\langle\texttt{0},\texttt{xs}\rangle_p,&\\
\texttt{Nil}\mapsto\langle 1,\texttt{Cons}\langle\texttt{x},\texttt{Nil}\rangle\rangle&\\
\texttt{Cons}\mapsto\langle\texttt{y},\langle\texttt{ys},\texttt{r}\rangle\rangle.(3+((\langle\texttt{0},\texttt{f}\rangle_p\ \texttt{x})_p\ \texttt{y})_c)+_c\texttt{rec}(((\langle\texttt{0},\texttt{f}\rangle_p\ \texttt{x})_p\ \texttt{y})_p,&\\
\texttt{True}\mapsto\langle 1,\texttt{Cons}\langle\texttt{x},\texttt{Cons}\langle\texttt{y},\texttt{y}&\\
\texttt{False}\mapsto\langle 1+\texttt{r}_c,\texttt{Cons}\langle\texttt{y},\texttt{r}_p\rangle\rangle))&\\
\\
=\ \langle\texttt{0},\lambda\texttt{f}.\langle\texttt{0},\lambda\texttt{x}.\langle 0,\lambda\texttt{xs}.&\\
\texttt{rec}(\texttt{xs},&\\
\texttt{Nil}\mapsto\langle 1,\texttt{Cons}\langle\texttt{x},\texttt{Nil}\rangle\rangle&\\
\texttt{Cons}\mapsto\langle\texttt{y},\langle\texttt{ys},\texttt{r}\rangle\rangle.(3+((\texttt{f}\ \texttt{x})_p\ \texttt{y})_c)+_c\texttt{rec}((\texttt{f}\ \texttt{x})_p\ \texttt{y})_p,&\\
\texttt{True}\mapsto\langle 1,\texttt{Cons}\langle\texttt{x},\texttt{Cons}\langle\texttt{y},\texttt{ys}\rangle\rangle&\\
\texttt{False}\mapsto\langle 1+\texttt{r}_c,\texttt{Cons}\langle\texttt{y},\texttt{r}_p\rangle\rangle))&
\end{aligned}$$

FIGURE 26. The translation of `insert f x xs`. Unlike before, we do not assume that `f`, `x`, `xs` are variables. They may be expressions with non-zero costs.

$$\|\texttt{insert f x xs}\| = (1 + \|\texttt{insert f x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f x}\|_p \|\texttt{xs}\|_p$$

$$= (1 + \|\texttt{insert f x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f x}\|_p \|\texttt{xs}\|_p$$

$$= (2 + \|\texttt{insert f}\|_c + \|\texttt{x}\|_c + (\|\texttt{insert f}\|_p \|\texttt{x}\|_p)_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (2 + \|\texttt{insert f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (3 + \|\texttt{insert}\|_c + \|\texttt{f}\|_c + (\|\texttt{insert}\|_p \|\texttt{f}\|_p \|\texttt{x}\|_p)_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert}\|_p \|\texttt{f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (3 + \|\texttt{f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{insert}\|_p \|\texttt{f}\|_p \|\texttt{x}\|_p \|\texttt{xs}\|_p$$

$$= (3 + \|\texttt{f}\|_c + \|\texttt{x}\|_c + \|\texttt{xs}\|_c) +_c \texttt{rec}\,(\|\texttt{xs}\|_p\,,$$
$$\texttt{Nil} \mapsto \langle 1\,, \texttt{Cons}\langle \texttt{x}\,, \texttt{Nil}\rangle\rangle$$
$$\texttt{Cons} \mapsto \langle \texttt{y}\,, \langle \texttt{ys}\,, \texttt{r}\rangle\rangle\,.\,(3 + ((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p\ \ \texttt{y})_c) +_c \texttt{rec}\,((\|\texttt{f}\|_p \|\texttt{x}\|_p)_p\ \ \texttt{y})_p\,,$$
$$\texttt{True} \mapsto \langle 1, \texttt{Cons}\langle \|\texttt{x}\|_p\,, \texttt{Cons}\langle \texttt{y}\,, \texttt{y}$$
$$\texttt{False} \mapsto \langle 1 +_c \texttt{r}_c, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle\rangle)$$

FIGURE 27. Interpretation of lists as lengths

$$\begin{aligned}
[\![list]\!] &= \mathbb{N}^\infty \\
D^{list} &= \{*\} + \{1\} \times \mathbb{N}^\infty \\
size_{list}(Nil) &= 0 \\
size_{list}(Cons(1,n)) &= 1+n
\end{aligned}$$

FIGURE 28. Interpretation of the inner `rec` of `insert` with lists abstracted to sizes

$[\![\texttt{rec}\,(\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_p\,,$

$\quad \texttt{True} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}\,, \texttt{Cons}\langle \texttt{y}\,, \texttt{ys}\rangle\rangle\rangle$

$\quad \texttt{False} \mapsto \langle 1+\texttt{r}_c, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle\rangle\ \ ]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto 1, \texttt{y} \mapsto 1, \texttt{ys} \mapsto n, \texttt{r} \mapsto r\}$

$$\begin{aligned}
f_{True}(\langle\rangle) &= [\![\langle \texttt{1}\,, \texttt{Cons}\langle \texttt{x}\,, \texttt{Cons}\langle \texttt{y}\,, \texttt{ys}\rangle\rangle\rangle]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto 1, \texttt{y} \mapsto 1, \texttt{ys} \mapsto n, \texttt{r} \mapsto r\} \\
&= (1, 2+n)
\end{aligned}$$

$$\begin{aligned}
f_{False}(\langle\rangle) &= [\![\langle 1+\texttt{r}_c, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle\rangle\ \ ]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto 1, \texttt{y} \mapsto 1, \texttt{ys} \mapsto n, \texttt{r} \mapsto r\} \\
&= (1+\pi_0 r, 1+\pi_1 r)
\end{aligned}$$

$$= \bigvee_{size(w) \leq \pi_1(\pi_1(f\ \ 1)\ \ 1)} case(w, (f_{True}, f_{False}))$$

$$= \bigvee_{size(w) \leq \pi_1(\pi_1(f\ \ 1)\ \ 1)} case(w, (\lambda\langle\rangle.(1, 2+n), \lambda\langle\rangle.(1+\pi_0 r, 1+\pi_1 r))$$

$$= (1, 2+n) \vee (1+\pi_0 r, 1+\pi_1 r)$$

FIGURE 29. Interpretation of `rec` in `insert`.

$$g(n) = [\![\texttt{rec(xs,}$$
$$\texttt{Nil} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle$$
$$\texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle . (3 + ((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_c) +_c \texttt{rec}((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_p,$$
$$\texttt{True} \mapsto \langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Cons}\langle \texttt{y}, \texttt{ys}\rangle\rangle\rangle$$
$$\texttt{False} \mapsto \langle 1 +_c \texttt{r}, \texttt{Cons}\langle \texttt{y}, \texttt{r}_p\rangle\rangle))[\![\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto$$

$$f_{Nil}(\langle\rangle) = [\![\langle 1, \texttt{Cons}\langle \texttt{x}, \texttt{Nil}\rangle\rangle]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto 1, \texttt{xs} \mapsto n\}$$

$$f_{Nil}(\langle\rangle) = (1, 1)$$

$$f_{Cons}((1, m)) = [\![(3 + ((\texttt{f}\ \ \texttt{x})_p\ \ \texttt{y})_c) +_c \texttt{rec}(\ldots)]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto 1, \texttt{xs} \mapsto n\}$$

$$= (3 + \pi_0(\pi_1(f\ \ 1)\ \ 1)) +_c$$
$$[\![\texttt{rec}(\ldots)]\!]\xi\{\texttt{f} \mapsto f, \ldots, \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto map^{1\times\mathbb{N}^\infty}(\lambda a.(a, [\![\texttt{rec}(w, \ldots)]\!]\xi\{w \mapsto a\})$$

$$= (3 + \pi_0(\pi_1(f\ \ 1)\ \ 1)) +_c$$
$$[\![\texttt{rec}(\ldots)]\!]\xi\{\texttt{f} \mapsto f, \ldots, \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (1, map^{\mathbb{N}^\infty}(\lambda a.(a, [\![\texttt{rec}(w, \ldots)]\!]\xi\{w \mapsto a\})$$

$$= (3 + \pi_0(\pi_1(f\ \ 1)\ \ 1)) +_c$$
$$[\![\texttt{rec}(\ldots)]\!]\xi\{\texttt{f} \mapsto f, \ldots, \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (1, (m, [\![\texttt{rec}(w, \ldots)]\!]\xi\{w \mapsto m\}))\}$$

$$= (3 + \pi_0(\pi_1(f\ \ 1)\ \ 1)) +_c$$
$$[\![\texttt{rec}(\ldots)]\!]\xi\{\texttt{f} \mapsto f, \texttt{x} \mapsto 1, \texttt{xs} \mapsto n, \langle \texttt{y}, \langle \texttt{ys}, \texttt{r}\rangle\rangle \mapsto (1, (m, g(m)))\}$$

$$= (3 + \pi_0(\pi_1(f\ \ 1)\ \ 1)) +_c ((1, 2 + m) \vee (1 + \pi_0 g(m)), 1 + \pi_1 g(m)))$$

$$= (3 + \pi_0(\pi_1(f\ \ 1)\ \ 1) + (1 \vee (1 + \pi_0 g(m))), (2 + m) \vee (1 + \pi_1 g(m))$$

$$= (4 + \pi_0(\pi_1(f\ \ 1)\ \ 1) + \pi_0 g(m), (2 + m) \vee (1 + \pi_1 g(m)))$$

FIGURE 30. Translation of `Nil` branch of `sort`.

$$\|\texttt{Nil} \mapsto \texttt{Nil}\|$$

$$= \texttt{Nil} \mapsto 1 +_c \|\texttt{Nil}\|$$

$$= \texttt{Nil} \mapsto 1 +_c \langle \texttt{0}, \texttt{NiL} \rangle$$

$$= \texttt{Nil} \mapsto \langle \texttt{1}, \texttt{NiL} \rangle$$

FIGURE 31. Translation of `Nil` branch of `sort`.

$$\|\texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . \texttt{insert f y force(r)}$$

$$= \texttt{Cons} \mapsto 1 +_c \|\texttt{insert f y force(r)}\|$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c (\|\texttt{force(r)}\|_c) +_c \|\texttt{insert f y}\|_p \|\texttt{force(r)}\|_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c ((\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_c) +_c \|\texttt{insert f y}\|_p (\|\texttt{r}\|_c +_c \|\texttt{r}\|_p)_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c \texttt{r}_c +_c \|\texttt{insert f y}\|_p \texttt{r}_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 1 +_c \texttt{r}_c +_c 3 +_c \|\texttt{insert}\|_p \texttt{ f y r}_p$$

$$= \texttt{Cons} \mapsto \langle \texttt{y}, \langle \texttt{ys}, \texttt{r} \rangle \rangle . 4 +_c \texttt{r}_c +_c \|\texttt{insert}\|_p \texttt{ f y r}_p$$

FIGURE 32. Translation of `sort`

$$\|\texttt{sort}\| = \langle 0, \lambda \texttt{f} . \langle 0, \lambda \texttt{xs} . \|\texttt{rec(xs, Nil}\mapsto \texttt{Nil,}$$
$$\texttt{Cons}\mapsto \langle \texttt{y}, \langle \texttt{ys,r} \rangle \rangle . \texttt{insert f y force(r))} \| \rangle \rangle$$

$$= \langle 0, \lambda \texttt{f} . \langle 0, \lambda \texttt{xs} . \texttt{rec(xs, Nil}\mapsto \langle \texttt{1,Nil} \rangle$$
$$\texttt{Cons}\mapsto \langle \texttt{y}, \langle \texttt{ys,r} \rangle \rangle . 4 +_c \texttt{r}_c +_c \|\texttt{insert}\|_p \ \texttt{f y r}_p$$

FIGURE 33. Translation of `sort`

$$\|\texttt{sort f xs}\| = (1 + \|\texttt{sort f}\|_c + \|\texttt{xs}\|_c) +_c \|\texttt{sort f}\|_p \|\texttt{xs}\|_p$$

$$= (1 + (1 + \|\texttt{sort}\|_c + \|\texttt{f}\|_c + \|\texttt{xs}\|_c)) +_c \|\texttt{sort}\|_p \|\texttt{f}\|_p \|\texttt{xs}\|_p$$

$$= (1 + (1 + 0 + 0 + 0)) +_c \|\texttt{sort}\|_p \|\texttt{f}\|_p \|\texttt{xs}\|_p$$

$$= 2 +_c \|\texttt{sort}\|_p \texttt{f}\|_p \|\texttt{xs}\|_p$$

$$= 2 +_c \texttt{rec}(\|\texttt{xs}\|_p, \ \texttt{Nil}\mapsto \langle \texttt{1,Nil} \rangle$$
$$\texttt{Cons}\mapsto \langle \texttt{y}, \langle \texttt{ys,r} \rangle \rangle . 4 +_c \texttt{r}_c +_c \|\texttt{insert}\|_p \ \texttt{f y r}_p$$

Figure 34. Interpretation of `rec` in `sort`.

$$g(n) = [\![\texttt{rec}(\|\texttt{xs}\|_p,\ \texttt{Nil}\mapsto \langle \texttt{1},\texttt{Nil}\rangle \quad \texttt{Cons}\mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.4{+_c}\texttt{r}_c +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p)]\!]\xi\{xs \mapsto n\}$$

$$= [\![\texttt{rec}(\|\texttt{xs}\|_p,\ \texttt{Nil}\mapsto \langle \texttt{1},\texttt{Nil}\rangle \quad \texttt{Cons}\mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.4{+_c}\texttt{r}_c +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p)]\!]\xi\{xs \mapsto n\}$$

$$= [\![\texttt{rec}(\|\texttt{xs}\|_p,\ \texttt{Nil}\mapsto \langle \texttt{1},\texttt{Nil}\rangle \quad \texttt{Cons}\mapsto \langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle.4{+_c}\texttt{r}_c +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p)]\!]\xi\{xs \mapsto n\}$$

$$= \bigvee\nolimits_{size(z)\le n} case(z, (f_{Nil}, f_{Cons}))$$

$$f_{Nil}(\langle\rangle) = [\![\langle \texttt{1},\texttt{Nil}\rangle]\!]\xi$$

$$= f_{Nil}(\langle\rangle) = (1, 0)$$

$$f_{Cons}((1,m)) = [\![\ldots]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto map^{1\times\mathbb{N}^\infty}(\lambda a.(a, [\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto a\}), (1,m))\}$$

$$= [\![\ldots]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto (map^{1}(\lambda a.(\ldots), 1), map^{\mathbb{N}^\infty}(\lambda a.(a, [\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto a\}), m))\}$$

$$= [\![\ldots]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto (1, (m, [\![\texttt{rec}(w,\ldots)]\!]\xi\{w\mapsto m\}))\}$$

$$= [\![(4{+}\texttt{r}_c) +_c \|\texttt{insert}\|_p\ \texttt{f}\ \texttt{y}\ \texttt{r}_p]\!]\xi\{\langle \texttt{y},\langle \texttt{ys},\texttt{r}\rangle\rangle \mapsto (1, (m, g(m))\}$$

$$= (4 + \pi_0 g(m)) +_c insert\ \ f\ \ 1\ \ \pi_1 g(m)$$

$$g(n) = \bigvee\nolimits_{size(z)\le n} case(z, (\lambda(\langle\rangle).(1,0), \lambda(1,m).4 + \pi_0 g(m)) +_c insert\ \ f\ \ 1\ \ \pi_1 g(m)))$$

CHAPTER 3

Work and Span

1. Work and span

Work and span is a method of calculating the cost of programs that may be run on multiple machines. The work of a program corresponds to the total number of steps needed to run. The span of a program is the steps in the critical path. The critical path is the largest number of steps that must be executed sequentially. The length of the critical path determines how much a program can be parallelized. If the span is equal to the work, than every step in the computation depends on the previous step, and the program cannot be parallelized.

Instead of calculating the cost of program, we will construct a cost graph. The cost graph represents dependencies between computations in a program. The work and span can be extracted from the call graph. The call graph is used to determine execution strategies.

A call graph is defined as follows.

$$\mathcal{C} ::= 0|1|\mathcal{C} \oplus \mathcal{C}|\mathcal{C} \otimes \mathcal{C}$$

The operator $\oplus$ connects to cost graphs who must be computed sequentially. The operator $\otimes$ connects cost graphs which may be computed in independently in parallel.

We alter the operational semantics of the source language slightly to reflect that the cost of evaluating an expression is a cost graph instead of an integer. Figure 1 shows the new operational semantics. For tuples, the subexpressions may be evaluated in parallel, so the cost of evaluating a tuple is the cost graphs of the subexpressions connected by $\otimes$. For `split`, the second subexpression depends on the result of the first subexpression,

FIGURE 1. Source language operational semantics

$$\frac{e_0 \downarrow^{n_0} v_0 \qquad e_1 \downarrow^{n_1} v_1}{\langle e_0, e_1 \rangle \downarrow^{n_0 \otimes n_1} \langle v_0, v_1 \rangle}$$

$$\frac{e_0 \downarrow^{n_0} \langle v_0, v_1 \rangle \qquad e_1[v_0/x_0, v_1/x_1] \downarrow^{n_1} v}{split(e_0, x_0.x_1.e_1) \downarrow^{n_0 \oplus n_1} v}$$

$$\frac{e_0 \downarrow^{n_0} \lambda x.e_0' \qquad e_1 \downarrow^{n_1} v_1 \qquad e_0'[v_1/x] \downarrow^{n} v}{e_0\ e_1 \downarrow^{(n_0 \otimes n_1) \oplus n \oplus 1} v}$$

$$\frac{}{delay(e) \downarrow^{0} delay(e)}$$

$$\frac{e \downarrow^{n_0} delay(e_0) \qquad e_0 \downarrow^{n_1} v}{force(e) \downarrow^{n_0 \oplus n_1} v}$$

$$\frac{e \downarrow^{n} v}{Ce \downarrow^{n} Cv}$$

$$\frac{e \downarrow^{n_0} Cv_0 \qquad map^{\phi_C}(y.\langle y, delay(rec(y, \overline{C \mapsto x.e_C}))\rangle, v_0) \downarrow^{n_1} v_1 \qquad e_C[v_1/x] \downarrow^{n_2} v}{rec(e, \overline{C \mapsto x.e_C}) \downarrow^{1 \oplus n_0 \oplus n_1 \oplus n_2} v}$$

$$\frac{}{map^{t}(x.v, v_0) \downarrow^{0} v[v_0/x]}$$

$$\frac{}{map^{\tau}(x.v, v_0) \downarrow^{0} v_0}$$

$$\frac{map^{\phi_0}(x.v, v_0) \downarrow^{n_0} v_0' \qquad map^{\phi_1}(x.v, v_1) \downarrow^{n_1} v_1'}{map^{\phi_0 \times \phi_1}(x.v, \langle v_0, v_1 \rangle) \downarrow^{n_0 \otimes n_1} \langle v_0', v_1' \rangle}$$

$$\frac{}{map^{\tau \to \phi}(x.v, \lambda y.e) \downarrow^{0} \lambda y.let(e, z.map^{\phi}(x.v, z))}$$

$$\frac{e_0 \downarrow^{n_0} v_0 \qquad e_1[v_0/x] \downarrow^{n_1} v}{let(e_0, x.e_1) \downarrow^{n_0 \oplus n_1} v}$$

so the cost of evaluating the `split` is the cost graphs of the subexpression connected by $\oplus$.

2. Bounding Relation

TODO

3. Parallel List Map

TODO

4. Parallel Tree Map

A program which is embarrasingly parallel is tree map. When a function f is mapped over a tree t, each application of f to the label at each node can be done independently. Furthermore, the tree data structure itself is dividable by construction.

Dividing the work requires only destruction of the node constructor to yield the left and right subtrees.

Source Language. Recall the definition of the tree data type and the function map.

CHAPTER 4

Mutual Recurrence

1. Motivation

The interpretation of a recursive function can be seperated into a recurrence for the cost and a recurrence for the potential. The recurrence for the cost depends on the recurrence for the potential. However, the recurrence for the potential does not depend on the cost. We prove this by designing a pure potential translation. The pure potential translation is identical to the complexity translation except that it does not keep track of the cost.

We then show by logical relations that the potential of the complexity translation is related to the pure potential relation.

2. Pure Potential Translation

Our pure potential translation is defined below. The translation of an expression is essientially the expression itself, without suspensions.

$$|x| = x$$

$$|\langle\rangle| = \langle\rangle$$

$$|\langle e_0, e_1\rangle| = \langle |e_0|, |e_1|\rangle$$

$$|\texttt{split}(e_0, x_0.x_1.e_1)| = |e_1|[\pi_0|e_0|/x_0, \pi_0|e_0|/x_1]$$

$$|\lambda x.e| = \lambda x.|e|$$

$$|e_0\ e_1| = |e_0|\ |e_1|$$

$$|delay(e)| = |e|$$

$$|force(e)| = |e|$$

$$|C_i^\delta e| = C_i^\delta |e|$$

$$|rec^\delta(e, \overline{C \mapsto x.e_C})| = rec^\delta(|e|, \overline{C \mapsto x.|e_C|})$$

$$|map^\phi(x.v_0, v_1)| = map^{|\phi|}(x.|v_0|, |v_1|)$$

$$|let(e_0, x.e_1)| = |e_1|[|e_0|/x]$$

3. Logical Relation

We define our logical relation below.

$$E \sim_{\texttt{Unit}} E' \text{always}$$

$$E \sim_{\tau_0 \times \tau_1} E' \Leftrightarrow \forall k.\langle k, \pi_0 E_p\rangle \sim_{\tau_0} \pi_0 E', \forall k.\langle k, \pi_1 E_p\rangle \sim_{\tau_1} \pi_1 E'$$

$$E \sim_{\texttt{susp}\ \tau} E' \Leftrightarrow E_p \sim_\tau E'$$

$$E \sim_{\sigma \to \tau} E' \Leftrightarrow \forall E_0 \sim_\sigma E_0'.E_p E_{0p} \sim_\tau E' E_0'$$

$$E \sim_\delta E' \Leftrightarrow \exists k, k', C, V, V'.V \sim_{\phi[\delta]} V', E \downarrow \langle k, CV_p\rangle, E' \downarrow CV'$$

The relation is defined on closed terms, but we extend it to open terms. Let Θ and Θ' be any substitutions such that $\forall x : \|\tau\|, \forall k, \langle k, \Theta(x)\rangle \sim_\tau \Theta'(x)$. If $E\ \Theta \sim_\tau E'\ \Theta'$, then $E \sim_\tau E'$.

4. Proof

We require some lemmas.

The first states we can always ignore the cost of related terms.

Lemma 4.1 (Ignore Cost).

$$E \sim_\tau E' \Leftrightarrow \forall k, \langle k, E_p\rangle \sim_\tau E'$$

Proof. We proceed by induction on type.

Case $E \sim_{\texttt{Unit}} E'$. Then $\forall k, \langle k, E_p\rangle \sim_{\texttt{Unit}} E'$ by definition.

Case $E \sim_{\tau_0\times\tau_1} E'$. By definition for $i \in 0, 1, \forall k_i, \langle k_i, \pi_i E_p\rangle \sim_{\tau_i} \pi_i E'$. Let k be some cost. Then $\langle k, E_p\rangle \sim_{\tau_0\times\tau_1} E'$ by definition.

Case $E \sim_{\texttt{susp}\ \tau} E'$. By definition $E_p \sim_\tau E'$. Let k be some cost. Then $\langle k, E_p\rangle \sim_{\texttt{susp}\ \tau} E'$.

Case $E \sim_{\sigma\to\tau} E'$. Let E_0, E_0' by some complexity language terms such that $E_0 \sim_\sigma E_0'$. Let k be some cost. Then, $E_p\ E_0 \sim_\tau E'\ E_0'$. So $\langle k, E_p\rangle \sim_{\sigma\to\tau} E'$.

Case $E \sim_\delta E'$. Then by definition there exists costs k and k', a constructor C, and complexity language values V and V' such that $V \sim_{\Phi[\delta]} V', E \downarrow \langle k, CV_p\rangle$, and $E' \downarrow CV'$. Since $E \downarrow \langle k, CV_p\rangle$, we know $\forall k_0, \exists k_0'.\langle k_0, E_p\rangle \downarrow \langle k_0', CV_p\rangle$. So by definition we have $\forall k_0, \langle k_0, E_p\rangle \sim_\Phi E'$. □

The next lemma states that if two terms step to related terms, then those terms are related.

Lemma 4.2 (Related Step Back).

$$E \to F, E' \to F', F \sim_\sigma F' \implies E \sim_\sigma E'$$

Proof. The proof proceeds by induction on type.

Case `Unit`. Trivial since $E \sim_{\texttt{Unit}} E'$ always.

Case δ. By definition $\exists C, U, U', k, k'$ such that $F \downarrow \langle k, CU_p\rangle, F' \downarrow CU', U \sim_{\phi[\delta]} U'$. Since $E \to F$ and $E' \to F'$, $E \downarrow \langle k, CU_p\rangle$ and $E' \downarrow CU'$. Therefore since $U \sim_{\phi[\delta]} U'$, we have $E \sim_\delta E'$.

Case $\sigma \to \tau$. Let $E_0 \sim_\sigma E_0'$. By definition, $F\ E_0 \sim_\tau F'\ E_0'$. Since $E \to F$ and $E' \to F'$, $E\ E_0 \to F\ E_0$ and $E'\ E_0' \to F'\ E_0'$. So by the induction hypothesis, $E\ E_0 \sim_\tau E'\ E_0'$. So by definition, $E \sim_{\sigma\to\tau} E'$.

Case $\tau_0 \times \tau_1$. Since $F \sim_{\tau_0\times\tau_1} F'$, for $i \in \{0, 1\}$, $\forall k_i, \langle k_i, \pi_i F_p\rangle \sim_{\tau_i} \pi_i F'$, by definition. From $E \to F$, we get $\langle k_i, \pi_i E_p\rangle \to \langle k_i', \pi_i F_p\rangle$. From $E' \to F'$, we get $\pi_i E' \to \pi_i F'$. We can apply our induction hypothesis to get $\langle k_i, \pi_i E_p\rangle \sim_{\tau_i} \pi_i E'$. By 4.1, $\forall k_i, \langle k_i, \pi_i E_p\rangle \sim_{\tau_i} \pi_i E$. So by definition $E \sim_{\tau_0\times\tau_1} E'$.

Case `susp` τ. Since $F \sim_{\texttt{susp}\ \tau} F'$, by definition $F_p \sim_\tau F'$. Since $E \to F$, $E_p \to F_p$. So by the induction hypothesis, since $E_p \to F_p, E' \to F', F_p \sim_\tau F'$, $E_p \sim_\tau E'$. So by definition $E \sim_{\texttt{susp}\ \tau} E'$.

□

The next lemma states that related terms step to related terms

Lemma 4.3. *[Related Step]*

$$E \to F, E' \to F', E \sim_\sigma E' \implies F \sim_\sigma F'$$

Proof. The proof is by induction on type.

Case `Unit`. $F \sim_{\texttt{Unit}} F'$ always.

Case δ. By definition, $E \sim_\delta E'$ implies $\exists C, V, V', k$ such that $E \downarrow \langle k, CV_p\rangle, E' \downarrow CV', V \sim_{\phi[\delta]} V'$. Since $E \to F$, $F \downarrow \langle k, CV_p\rangle$; and since $E \to F'$, $F' \downarrow CV'$. By 4.1, $\langle k, V_p\rangle \sim_{\phi[\delta]} V'$. So because $F \downarrow \langle k, CV_p\rangle, F' \downarrow CV', \langle k, V_p\rangle \sim_{\phi[\delta]} V'$, we can apply our induction hypothesis to get $F \sim_\delta F'$.

Case $\tau_0 \times \tau_1$. By definition $E \sim_{\tau_0\times\tau_1} \implies \forall i \in \{0, 1\}, \forall k, \langle k_i, \pi_i E_p\rangle \sim_{\tau_i} \pi_i E'$. Fix some k_i. Since $E \to F$, $\langle k_i, \pi_i E_p\rangle \to \langle k_i, \pi_i F_p\rangle$. Since $E' \to F'$, $\pi_i E' \to \pi_i F'$.

From $\langle k_i, \pi_i E_p\rangle \to \langle k_i, \pi_i F_p\rangle, \langle k_i, \pi_i E_p\rangle \sim_{\tau_i} \pi_i E'$, the induction hypothesis tells us $\langle k_i, \pi_i F_p\rangle \sim_{\tau_i} \pi_i F'$. So by definition $F \sim_{\tau_0\times\tau_1} F'$.

Case $\texttt{susp}\ \tau$. By definition $E \sim_{\texttt{susp}\ \tau} E' \implies E_p \sim_\tau E'$. Since $E \to F$, $E_p \to F_p$. From $E_p \to F_p, E' \to F', E_p \sim_\tau E'$, the induction hypothesis gives us $F_p \sim_\tau F'$. So by definition $F \sim_{\texttt{susp}\ \tau} F'$.

Case $\sigma \to \tau$. Let $E_0 \sim_\sigma E_0'$. By definition, $E\ E_0 \sim_\tau E'\ E_0'$. Since $E \to F$, $E\ E_0 \to F\ E_0$. Since $E' \to F'$, $E'\ E_0' \to F'\ E_0'$. From $E\ E_0 \to F\ E_0, E'\ E_0' \to F'\ E_0', E\ E_0 \sim_\tau E'\ E_0'$, the induction hypothesis tells us $F\ E_0 \sim_\tau F'\ E_0'$. So by definition $F \sim_{\sigma\to\tau} F'$. □

The next lemma states that if the arguments to *map* are related, then *map* preserves the relatedness.

LEMMA 4.4. *[Related Map]*

$$E \sim_{\tau_1} E', E_0 \sim_{\tau_0} E_0' \implies \forall k.\langle k, map^\Phi(x, E_p, E_{0p})\rangle \sim_{\Phi[\tau_1]} map^\Phi(x, E', E_0')$$

PROOF. The proof proceeds by induction on type.

Recall the definition of the *map* macro.

$$\begin{aligned} map^t(x.E, E_0) &= E[E_0/x] \\ map^T(x.E, E_0) &= E_0 \\ map^{\Phi_0\times\Phi_1}(x.E, E_0) &= \langle map^{\Phi_0}(x.E, \pi_0 E_0), map^{\Phi_1}(x.E, \pi_1 E_0\rangle \\ map^{T\to\Phi}(x.E, E_0) &= \lambda y. map^\Phi(x.E, E_0\ y) \end{aligned}$$

Case $\Phi = t$. Then $map^t(x.E_p, E_{0p}) = E_p[E_{0p}/x]$ and $map^t(x.E', E_0') = E'[E_0'/x]$. Let k be some cost. By 4.1, $E \sim_{\tau_1} E'$ implies $\langle k, E_p\rangle \sim_{\tau_1} E'$. Since $\langle k, E_p\rangle \sim_{\tau_1} E'$ and $E_0 \sim_{\tau_0} E_0'$, $\langle k, E_p\rangle[E_{0p}/x] \sim_{\phi[\tau_0]} E'[E_0'/x]$. So $\forall k, \langle k, map^t(x.E_p, E_{0p})\rangle \sim_{\Phi[\tau_1]} map^t(x.E', E_0')$.

Case $\Phi = T$. Then $map^T(x.E_p, E_{0p}) = E_{0p}$ and $map^T(x.E', E_0') = E_0'$. By 4.1 $\forall k, \langle k, E_{0p}\rangle \sim_{\tau_0} E_0'$. So $\forall k, \langle k, map^T(x.E_p, E_{0p})\rangle \sim_{\Phi[\tau_1]} map^T(x.E', E_0')$.

Case $\Phi = \Phi_0 \times \Phi_1$. Then $map^{\Phi_0\times\Phi_1}(x.E_p, E_{0p}) = \langle map^{\Phi_0}(x.E_p, \pi_0 E_{0p}), map^{\Phi_1}(x.E_p, \pi_1 E_{0p})\rangle$. Similarly $map^{\Phi_0\times\Phi_1}(x.E', E'_0) = \langle map^{\Phi_0}(x.E', \pi_0 E'_0), map^{\Phi_1}(x.E', \pi_1 E'_0)\rangle$. By definition, $\forall k, \langle k, \pi_0 E_{0p}\rangle \sim_{\Phi_0[\tau_0]} \pi_0 E'_0$. By the induction hypothesis, $\forall k, \langle k, map^{\Phi_0}(x.E_p, \pi_0 E_{0p}) \sim_{\Phi_0[\tau_1]} map^{\Phi_0[\tau_1]}(x.E', E'_0)$. By definition, $\forall k, \langle k, \pi_1 E_{0p}\rangle \sim_{\Phi_1[\tau_0]} \pi_1 E'_0$. By the induction hypothesis, $\forall k, \langle k, map^{\Phi_1}(x.E_p, \pi_1 E_{0p}) \sim_{\Phi_1[\tau_1]} map^{\Phi_1[\tau_1]}(x.E', E'_0)$. So by definition, $\forall k, \langle k, \langle map^{\Phi_0}(x.E_p, \pi_0 E$ $\langle\langle map^{\Phi_0[\tau_1]}(x.E', E'_0), map^{\Phi_1[\tau_1]}(x.E', E'_0)\rangle\rangle$.

Case $T \to \Phi$. Then $map^{T\to\Phi}(x.E_p, E_{0p}) = \lambda y.map^{\Phi}(x.E_p, E_{0p}\ y)$ and $map^{T\to\Phi}(x.E', E'_0) = \lambda y.map^{\Phi}(x.E', E'_0\ y)$. Let $E_1 : T$. Then $\lambda y.map^{\Phi}(x.E_p, E_{0p}\ y)\ E_1 \to map^{\Phi}(x.E_p, E_{0p}\ E_1)$. Similarly, $\lambda y.map^{\Phi}(x.E', E'_0\ y)\ E'_1 \to map^{\Phi}(x.E', E'_0\ E'_1)$. Since $E_0 \sim E'_0$ and $E_1 \sim E'_1$, we have $E_{0p}\ E_1 \sim E'_0\ E'_1$. So by our induction hypothesis, $map^{\Phi}(x.E_p, E_{0p}\ E_1) \sim map^{\Phi}(x.E', E'_0\ E'_1)$. So by 4.2, $\lambda y.map^{\Phi}(x.E_p, E_{0p}\ y)\ E_1 \sim \lambda y.map^{\Phi}(x.E', E'_0\ y)\ E'_1$. So by definition, $\lambda y.map^{\Phi}(x.E_p, E_{0p}\ y) \sim \lambda y.map^{\Phi}(x.E', E'_0\ y)$. So $map^{T\to\Phi}(x.E_p, E_{0p}) \sim map^{T\to\Phi}(x.E', E'_0)$. □

Our last lemma is about the relatedness of *rec* terms.

Lemma 4.5 (Related Rec).

$$E \sim_\delta E', \forall C, E_C \sim_\tau E'_C \implies rec(E_p, \overline{C \mapsto x.E_c}) \sim_\tau rec(E', \overline{C \mapsto x.E'_c})$$

Proof. Recall the rule for evaluating *rec* in the complexity language:

$$\frac{E \downarrow CV_0 \qquad map^{\Phi}(y, \langle y, rec(y, \overline{C \mapsto x.E_C})\rangle, V_0) \downarrow V_1 \qquad E_C[V_1/x] \downarrow V}{rec(E, \overline{C \mapsto x.E_C}) \downarrow V}$$

By definition of $\sim_\delta$, $\exists k, C, V_0, V'_0$ such that $E \downarrow \langle k, CV_{0p}\rangle, E' \downarrow CV'_0$, and $V_0 \sim_\delta V'_0$. Our proof proceeds by induction on the number of constructors in CV_{0p}. If $\Phi = T$, then $map^{\Phi}(y, \langle y, rec(y, \overline{C \mapsto x.E_C})\rangle, V_{0p}) = \langle y, rec(y, \overline{C \mapsto x.E_C})\rangle[V_{0p}/y] = \langle V_{0p}, rec(V_{0p}, \overline{C \mapsto x.E_C})\rangle$. Similarly for the pure potential, $map^{\Phi}(y, \langle y, rec(y, \overline{C \mapsto x.E'_C})\rangle, V'_{0p}) = \langle y, rec(y, \overline{C \mapsto x.E'_C})\rangle[V'_0/y] = \langle V'_0, rec(V'_0, \overline{C \mapsto x.E'_C})\rangle$. By the induction hypothesis, $rec(V_{0p}, \overline{C \mapsto x.E_C}) \sim_\tau rec(V'_0, \overline{C \mapsto x.E'_C})$. By definition of $\sim_{\texttt{susp}\ \tau}$, for any k, $\langle k, rec(V_{0p}, \overline{C \mapsto x.E_C})\rangle \sim_{\texttt{susp}\ \tau} rec(V'_0, \overline{C \mapsto x.E'_C})$. So by definition of $\sim_{\tau_0\times\tau_1}$, $\langle 0, \langle V_{0p}, rec(V_{0p}, C \mapsto x.E_C)\rangle\rangle \sim_{\phi[\delta\times\texttt{susp}\ \tau]} \langle V'_0, rec(V'_0, \overline{C \mapsto x.E'_C})\rangle$. So by 4.4, $\forall k.\langle k, map^{\Phi}(y, \langle y, rec(y, \overline{C \mapsto x.E_C})\rangle, V_{0p}) \sim_{\phi[\delta\times\texttt{su}}$

$map^\Phi(y, \langle y, rec(y, \overline{C \mapsto x.E'_C})\rangle, V'_0)$. Let $\langle 0, map^\Phi(y, \langle y, rec(y, \overline{C \mapsto x.E_C})\rangle, V_{0p}) \downarrow V_1$. Let $map^\Phi(y, \langle y, rec(y, \overline{C \mapsto x.E'_C})\rangle, V'_0) \downarrow V'_1$. By 4.3, $V_1 \sim_{\phi[\delta \times \texttt{susp}\ \tau]} V'_1$.

If $\Phi = t$, then $map^\Phi(y, \langle y, rec(y, \overline{C \mapsto x.E_C})\rangle, V_{0p}) = V_{0p}$. Similarly, $map^\Phi(y, \langle y, rec(y, \overline{C \mapsto x.E'_C})\rangle V'_0) =$ V'_0. So in this case $V_0 = V_1$ and $V'_0 = V'_1$. We have already established $V_0 \sim_\tau V'_0$.

So in both cases $V_1 \sim_{\phi[\delta \times \texttt{susp}\ \tau]} V'_1$.

By definition of the relation $E_C[V_{1p}/x] \sim_\tau E'_C[V'_1/x]$. Let $E_C[V_{1p}/x] \downarrow V_2$ and $E'_C[V'_1/x] \downarrow V'_2$. By 4.3, $V_2 \sim_\tau V'_2$. So by 4.2, $rec(E_p, \overline{C \mapsto x.E_C}) \sim_\tau rec(E', \overline{C \mapsto x.E'_C})$.

□

Our theorem is that for all well-typed terms in the source language, the complexity translation of the term is related to the pure potential translation of that term.

THEOREM 4.6 (Distinct Recurrence).

$$\gamma \vdash e : \tau \implies \|e\| \sim_\tau |e|$$

PROOF. Our proof is by induction on the typing derivation $\gamma \vdash e : \tau$.

Case $\dfrac{}{\gamma, x : \sigma \vdash x : \sigma}$. Then by definition of the logical relation, $\forall k, \langle k, \Theta(x)\rangle \sim_\sigma \Theta'(x)$. Since $\|x\| = \langle 0, x\rangle$ and $|x| = x$, we have $\langle 0, x\rangle \sim_\sigma x$.

Case $\dfrac{}{\gamma \vdash e : Unit}$. By definition, $\|e\| \sim_{\texttt{Unit}} |e|$ always.

Case $\dfrac{\gamma \vdash e_0 : \tau_0 \qquad \gamma \vdash e_1 : \tau_1}{\gamma \vdash \langle e_0, e_1\rangle : \tau_0 \times \tau_1}$ By the induction hypothesis, $\|e_0\| \sim_{\tau_0} |e_0|$ and $\|e_1\| \sim_{\tau_1} |e_1|$. By 4.1, $\forall k, \langle k, \|e_0\|_p\rangle \sim_{\tau_0} |e_0|$ and $\forall k, \langle k, \|e_1\|_p\rangle \sim_{\tau_1} |e_1|$. So by definition, $\|\langle e_0, e_1\rangle\| \sim_{\tau_0 \times \tau_1} |\langle e_0, e_1\rangle|$.

Case $\dfrac{\gamma \vdash e_0 : \tau_0 \times \tau_1 \qquad \gamma, x_0 : \tau_0, x_1 : \tau_1 \vdash e_1 : \tau}{\gamma \vdash split(e_0, x_0.x_1.e_1) : \tau}$ By the induction hypothesis, $\|e_0\| \sim_{\tau_0 \times \tau_1} |e_0|$ and $\|e_1\| \sim_\tau |e_1|$. From $\|e_0\| \sim_{\tau_0 \times \tau_1} |e_0|$ it follows by definition that $\forall k, \langle k, \pi_0\|e_0\|_p\rangle \sim_{\tau_0} \pi_0|e_0|$ and $\forall k, \langle k, \pi_1\|e_1\|_p\rangle \sim_{\tau_1} \pi_1|e_1|$. The complixity translation is $\|split(e_0, x_0.x_1.e_1)\| = \|e_0\|_c +_c \|e_1\|[\pi_0\|e_0\|_p/x_0, \pi_1\|e_1\|_p/x_1]$. The pure potential translation is $|split(e_0, x_0.x_1.e_1)| = |e_1|[\pi_0|e_0|/x_0, \pi_1|e_0|/x_1]$. By 4.1, it suffices to show $\|e_1\|[\pi_0\|e_0\|_p/x_0, \pi_1\|e_1\|_p/x_1] \sim_\tau |e_1|[\pi_0|e_0|/x_0, \pi_1|e_0|/x_1]$ By definition of the relation, it suffices to show $\|e_1\| \sim_\tau |e_1|$, $\forall k, \langle k, \pi_0\|e_0\|_p\rangle \sim_{\tau_0} \pi_0|e_0|$, and $\forall k, \langle k, \pi_1\|e_0\|_p\rangle \sim_{\tau_1}$

$\pi_1|e_0|$. Since we have already establed all three conditions, we have $\|split(e_0, x_0.x_1.e_1)\| \sim_\tau$ $|split(e_0, x_0.x_1.e_1)|$.

Case $\dfrac{\gamma, x : \sigma \vdash e : \tau}{\gamma \vdash \lambda x.e : \sigma \to \tau}$ By the induction hypothesis $\|e\| \sim_\tau |e|$. The complexity translation is $\|\lambda x.e\| = \langle 0, \lambda x.\|e\|\rangle$. The pure potential translation is $|\lambda x.e| = \lambda x.|e|$. Let $E_0 : \|\sigma\|$ and $E_0' : |\sigma|$ be complexity language terms such that $E_0 \sim_\sigma E_0'$. Then $\langle 0, \lambda x.\|e\|\rangle\ E_0 \to \langle 0 + E_{0c}, \|e\|[x \mapsto E_0]\rangle$ and $\lambda x.|e|\ E_0' \to |e|[x \mapsto E_0']$. Since $\|e\| \sim_\tau |e|$ and $E_0 \sim_\sigma E_0'$, $\|e\|[x \mapsto E_0] \sim_\tau |e|[x \mapsto E_0']$. By 4.2, $\langle 0, \lambda x.\|e\|\rangle\ E_0 \sim_\tau (\lambda x.|e|)\ E_0'$. So by definition $\langle 0, \lambda x.\|e\|\rangle \sim_{\sigma\to\tau} \lambda x.|e|$. So $\|\lambda x.e\| \sim_{\sigma\to\tau} |\lambda x.e|$.

Case $\dfrac{\gamma \vdash e_0 : \sigma \to \tau \qquad \gamma \vdash e_1 : \sigma}{\gamma \vdash e_0\ e_1 : \tau}$ The complexity translation is $\|e_0\ e_1\| = (1 + \|e_0\|_c + \|e_1\|_c) +_c \|e_0\|_p\|e_1\|_p$. The pure potential translation is $|e_0\ e_1| = |e_0||e_1|$. By 4.1, it suffices to show $\|e_0\|_p\|e_1\|_p \sim_\tau |e_0||e_1|$. By the induction hypothesis, $\|e_0\| \sim_{\sigma\to\tau} |e_0|$ and $\|e_1\| \sim_\sigma |e_1|$. By definition, $\|e_0\|_p\|e_1\|_p \sim_\tau |e_0||e_1|$.

Case $\dfrac{\gamma \vdash e : \tau}{\gamma \vdash delay(e) : susp\ \tau}$ By the induction hypothesis $\|e\| \sim_\tau |e|$. So $\langle 0, \|e\|\rangle \sim_{\texttt{susp}\ \tau}$ $|e|$. The complexity translation is $\|delay(e)\| = \langle 0, \|e\|\rangle$. The pure potential translatio is $|delay(e)| = |e|$. So $\|delay(e)\| \sim_{\texttt{susp}\ \tau} |delay(e)|$.

Case $\dfrac{\gamma \vdash e : susp\ \tau}{\gamma \vdash force(e) : \tau}$ By the induction hypothesis $\|e\| \sim_{\texttt{susp}\ \tau} |e|$. So by definition of the relation at `susp` type, $\|e\|_p \sim_\tau |e|$. By 4.1, $\forall k, \langle k, \|e\|_{pp}\rangle \sim_\tau |e|$. The complexity translation is $\|force(e)\| = \|e\|_c +_c \|e\|_p$. The pure potential translation is $|force(e)| = |e|$. So $\|e\|_c +_c \|e\|_p \sim_\tau |e|$. So $\|force(e)\| \sim_\tau |force(e)|$.

Case $\dfrac{\gamma \vdash e_0 : \sigma \qquad \gamma, x : \sigma \vdash e_1 : \tau}{\gamma \vdash let(e_0, x.e_1) : \tau}$ By the induction hypothesis $\|e_0\| \sim_\sigma |e_0|$ and $\|e_1\| \sim_\tau |e_1|$. So $\|e_1\|[\|e_0\|_p/x] \sim_\tau |e_1|[|e_0|/x]$. By 4.1, $\forall k, \langle k, \|e_1\|_p[\|e_0\|_p/x]\rangle \sim_\tau |e_1|[|e_0|/x]$. The complexity translation is $\|let(e_0, x.e_1)\| = \|e_0\|_c +_c \|e_1\|[\|e_0\|_p/x]$. The pure potential translation is $|let(e_0, x.e_1)| = |e_1|[|e_0|/x]$. So $\|let(e_0, x.e_1)\| \sim_\tau |let(e_0, x.e_1)|$.

Case $\dfrac{\gamma, x : \tau_0 \vdash v_1 : \tau_1 \qquad \gamma \vdash v_0 : \phi[\tau_0]}{\gamma \vdash map^\phi(x.v_1, v_0) : \phi[\tau_1]}$ By the induction hypothesis $\|v_1\| \sim_{\tau_1} |v_1|$ and $\|v_0\| \sim_{\phi[\tau_0]} |v_0|$. By 4.4, $\forall k, \langle k, map^\Phi(x.\|v_1\|_p, \|v_0\|_p)\rangle \sim_{\phi[\tau_1]} map^\Phi(x.|v_1|, |v_0|)$. The

complexity translation is $\|map^{\phi}(x.v_1, v_0)\| = \langle 0, map^{\Phi}(x.\|v_0\|_p, \|v_1\|_p)\rangle$. The pure potential translation is $|map^{\phi}(x.v_1, v_0| = map^{\Phi}(x, |v_0|, |v_1|)$. So we have $\|map^{\phi}(x.v_1, v_0)\| \sim_{\phi[\tau_1]}$ $|map^{\phi}(x.v_1, v_0|$.

Case $\dfrac{\gamma \vdash e_0 : \delta \qquad \forall C(\gamma, x : \phi_C[\delta \times susp\ \tau] \vdash e_c : \tau}{\gamma \vdash rec^{\delta}(e_0, \overline{C \mapsto x.e_C}) : \tau}$ By the induction hypothesis $\|e_0\| \sim_{\delta} |e_0|$ and $\forall C, \|e_c\| \sim_{\tau} |e_c|$. By 4.1, $\forall k, \langle k \|e_C\| \sim_{\tau} |e_c$, so $1 +_c \|e_C\| \sim_{\tau} |e_c|$. So by 4.5, $rec(\|e_0\|_p, \overline{C \mapsto x.1 +_c \|e_C\|}) \sim_{\tau} rec(|e_0|, \overline{C \mapsto x.1 +_c |e_C|})$. So by 4.1, $\|e_0\|_c +_c$ $rec(\|e_0\|_p, \overline{C \mapsto x.1 +_c \|e_C\|}) \sim_{\tau} rec(|e_0|, \overline{C \mapsto x.1 +_c |e_C|})$

Case $\dfrac{\gamma \vdash e : \phi[\delta]}{\gamma \vdash Ce : \delta}$ By the induction hypothesis, $\|e\| \sim_{\phi[\delta]} |e|$. There exists V, V' such that $\|e\| \downarrow V$ and $|e| \downarrow V'$. By 4.3 $V \sim_{\phi[\delta]} V'$. Since $\|e\| \downarrow V$, $\langle k, C\ \|e\|\rangle \downarrow \langle k, C\ V_p$. Similarly, since $|e| \downarrow V'$, $C|e| \downarrow CV'$. So by definition we have $\langle k, C\ \|e\|\rangle \sim_{\delta} C|e|$. The complexity translation is $\|Ce\| = \langle \|e\|, C\|e\|_p\rangle$. The pure potential translation is $|Ce| = C|e|$. Therefore by 4.1, $\|Ce\| \sim_{\delta} |Ce|$.

□

Bibliography

Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. Closed-form upper bounds in static cost analysis. *J. Autom. Reason.*, 46(2):161–203, February 2011. ISSN 0168-7433. doi: 10.1007/s10817-010-9174-1. URL http://dx.doi.org/10.1007/s10817-010-9174-1.

Martin Avanzini, Ugo Dal Lago, and Georg Moser. Analysing the complexity of functional programs: Higher-order meets first-order (long version). In *In Proceedings of the International Conference on Functional Programming*, 2015. URL http://arxiv.org/abs/1506.05043.

Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

Norman Danner and James S. Royer. Adventures in time and space. *Logical Methods in Computer Science*, 3(1), 2007. doi: 10.2168/LMCS-3(1:9)2007.

Norman Danner and James S. Royer. Two algorithms in search of a type-system. *Theory of Computing Systems*, 45(4):787–821, 2009. doi: 10.1007/s00224-009-9181-y. URL http://dx.doi.org/10.1007/s00224-009-9181-y.

Norman Danner, Jennifer Paykin, and James S. Royer. A static cost analysis for a higher-order language. In *Proceedings of the 7th workshop on Programming languages meets program verification*, pages 25–34. ACM Press, 2013. doi: 10.1145/2428116.2428123. URL http://arxiv.org/abs/1206.3523.

Norman Danner, Daniel R. Licata, and Ramyaa Ramyaa. Denotational cost semantics for functional languages with inductive types. In *In Proceedings of the International Conference on Functional Programming*, volume abs/1506.01949, 2015. URL

http://arxiv.org/abs/1506.01949.

Jan Hoffmann and Martin Hofmann. Amortized Resource Analysis with Polymorphic Recursion and Partial Big-Step Operational Semantics. In *8th Asian Symp. on Prog. Langs. (APLAS'10)*, volume 6461 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2010.