

Universidad ORT Uruguay
Facultad de Ingeniería

Foundations for Mathematical Methodology

Entregado como requisito para la obtención del título de
Máster en Ingeniería

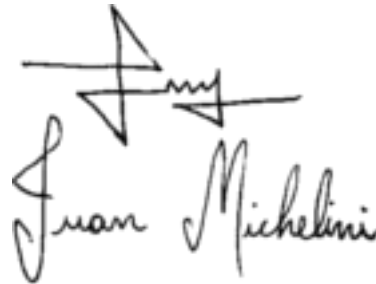
Juan Michelini - 163408

Tutor: Álvaro Tasistro

2016

Yo, Juan Michelini, declaro que el trabajo que se presenta en esta obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba el Proyecto;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

A handwritten signature in black ink. The signature consists of a stylized, large initial 'J' followed by the name 'uan Michelini' in a cursive script.

*To Tato,
for an education that Alexander the Great would envy.*

*A Papá, Mamá, Clari, Tati, Tincho & Pancha,
¡libertad!*

Resumen

Presentamos, para la teoría ecuacional del cálculo- λ , un método para encontrar conjeturas interesantes y diseñar sus demostraciones. Al igual que con los métodos enseñados en aritmética básica, durante la mayor parte del tiempo los practicantes, que imaginamos como estudiantes universitarios o hasta de secundaria, pueden limitarse a seguir las reglas pero, si lo desean, pueden realizar saltos intuitivos en cualquier momento sin comprometer el resultado final. Más específicamente, presentamos una serie de reglas que permiten al estudiante descubrir, por ejemplo, el tipo de las listas, descubrir que la función que invierte el orden de una lista es interesante, descubrir que dicha función es su propio inverso y demostrarlo, de manera similar a la que resuelve una ecuación de segundo grado. Hemos formalizado el método como un programa informático, lo que nos permite establecer propiedades del mismo con certeza, primero al realizar experimentos y, en algún trabajo futuro, al realizar Matemática sobre el método en sí.

Palabras Clave

Metodología Matemática, Programación Automática, Tácticas, Estrategia, Creatividad, Desarrollo Automático de Teorías, Educación Matemática, Matemática Calculacional, Cálculo Lambda

Abstract

We present, for the equational theory of a variant of typed λ -calculus, a method to find interesting conjectures and to design their proofs. Like the methods taught in basic arithmetic, for the most part the practitioners, which we imagine as undergraduate or even high-school students, can just follow the rules but, if they want to, they can apply an intuitive jump at any point without compromising the outcome. More specifically, we present a series of rules which allow the student to discover, say, the type of lists, discover that the function that reverses a list is interesting, discover that the reverse function is its own inverse and prove so, in much the same way in which they solve a second degree equation. We maintain a firm conceptual grasp of the method by formalizing it as a computer program. Such formalization allows us to establish properties of the method with certainty, first by doing experiments and, in some future work, by doing Mathematics over the method itself.

Key Words

Mathematical Methodology, Automatic Programming, Tactics, Strategy, Creativity, Automatic Theory Development, Mathematics Education, Computational Mathematics, Lambda Calculus

Contents

1	Introduction	7
2	Vocabulary	8
3	Survey	10
3.1	External	10
3.2	Internal	11
4	A Variant of Typed λ-calculus	12
5	The Method	12
6	Operations Permit	13
7	Tactics Prove	13
7.1	<i>Trivialize</i>	15
7.2	<i>Compute</i>	15
7.3	<i>Fundamentalize</i>	15
7.4	<i>Reorder</i>	17
7.5	<i>Induction</i>	18
7.6	Carrying out a proof	18
8	Strategies Propose	19
8.1	Proposing Data Types	20
8.2	Proposing Functions	21
8.3	Proposing Conjectures	22
9	Implementation as a Computer Program	22
10	Results	23
11	Concerns Against Method	25
12	Current Challenges and Future Work	26
13	Bibliography	27
	Appendices	29
A	Formalization of the Method in Haskell	30
B	Results	81

1 – Introduction

Mathematics is an often misunderstood art. The misunderstanding arises from the difference between what mathematicians do when they do Mathematics and what most students do while they follow a course on Mathematics. This abyss between the experience of the professional performer and that of the layman is one of the largest among the arts. Every student has been told to come up with an original story for a class assignment, but who has been told to come up with an original mathematical theory, or even an original conjecture? We ask children to draw particular things as exercise, but we also ask them to draw “what they feel like”. The same happens in Music class, where, along with all the *solfeggio*, we also ask them to choose what they want to play. No such equivalent happens with Mathematics, at least not in the experience of the author. If the readers did have any experience with creative Mathematics, please take a moment to cherish that memory, and consider themselves lucky.

The abyss is even deeper. All arts are taught in part by experience: we read and write lots of stories, we listen and play lots of songs, we read and write lots of proofs, but in Writing and Music, there is also an emphasis on discussion about styles. Rivers of ink have flowed trying to explain why that particular writer developed that story in that particular way, there are oceans of work about how to design stories, but only a few puddles here and there about how to design a proof, much less how to design a mathematical theory.

We do not stand alone pointing at this injustice: “A Mathematician’s Lament” [1] is a brilliant exposition of the problem, the books of Pólya (*How to Solve it* [2] and its sequels) have been widely praised, and a collection of works has sprouted around the concept of *Mathematical Methodology* following the lead of Dijkstra [3].

This defect of current mathematical education becomes even more perplexing when we consider its virtue at teaching basic arithmetic and algebra. That is to say, the few techniques that are taught, are taught with great results. Students learn rules which allow them to just compute the answer of a considerable class of problems, and, if given enough experience, they also develop an intuition of why those rules work and where to take shortcuts.

At this moment the question arises: could we fill the gap in Mathematics’s education and furthermore do it with the grace we have at teaching basic arithmetic? That is, could we teach how to design proofs and mathematical theories with the same success we have at teaching basic arithmetic? This work answers with a meek “yes”.

Namely, for the equational theory of a variant of typed λ -calculus, a method to find interesting conjectures and to design their proofs is proposed. Like the methods taught in basic arithmetic, for the most part the practitioners, which we imagine as undergraduate or even high-school students, can just follow the rules but, if they want to, they can apply an intuitive jump at any point without compromising the outcome. More specifically, we present a series of rules which allows the student to discover, say, the type of lists, discover that the function that reverses a list is interesting, discover that the reverse function is its own

inverse and prove so, in much the same way in which they solve a second degree equation.

Our answer is “meek” for two reasons: 1) because the variant of λ -calculus considered leaves lots of interesting things beyond the reach of the theory 2) because we lack a complete experience teaching the method. Still, we must remark that this is the only method that we know of that considers this much and that the experience so far, both tested by hand and simulated by computer, has been satisfactory.

This work will be structured as follows: a) we will first define some vocabulary to describe what the method is supposed to do, b) we will walk through a brief survey of related works, c) we will describe our variant of λ -calculus on which the method will work upon, d) we will describe the method -informally- mainly through examples, e) we will present an implementation of the method as a computer program which also serves as a formal specification, f) we will discuss the results found by said computer program, g) we will remark on the immediate improvements that could be carried out, as well as what the future might hold for Mathematics’s education.

2 – Vocabulary

We should first come out in the open by stating that the author is not a classically trained mathematician.¹ He can be considered as a somewhat rusty programmer who, failing a math class, decided to read a book on how to do what mathematicians do, and failing to find such matter, became obsessed with writing it. Lots of books on Mathematics have been written, so allow us to clarify by way of analogy: if we wanted to know what decisions were taken by statesmen in Florence in the Renaissance we could read a book such as *The Economy of Renaissance Florence*, but if instead we wanted to know how such decisions could be taken we could do no better than to read Machiavelli’s *Prince* [4][5]. The former says “In 1546 Cosimo I, barely ten years into his reign, took initiatives to set up a silk industry in Pisa”. While the latter says “The first method for estimating the intelligence of a ruler is to look at the men he has around him” [5]. The first is a book on facts, the second is a book on methodology. This work is one of methodology, mathematical methodology.

We are interested in teaching the doing of Mathematics, that is, teaching mathematical reasoning and the faculties on which it stands. We cannot think of anyone better than Turing [6] to illuminate us:

Mathematical reasoning may be regarded rather schematically as the exercise of a combination of two faculties, which we may call intuition and ingenuity. The activity of the intuition consists in making spontaneous judgments which are not the result of conscious trains of reasoning. [...] The exercise of ingenuity in Mathematics consists in aiding the intuition through suitable arrangements of propositions, and perhaps geometrical figures or drawings. It is intended that when these are really well arranged the validity of the intuitive steps which are required cannot seriously be doubted. The parts played by these two faculties differ of course from occasion to occasion, and from mathematician to math-

¹This is subject to change without prior notice.

ematician. This arbitrariness can be removed by the introduction of a formal logic. The necessity for using the intuition is then greatly reduced by setting down formal rules for carrying out inferences which are always intuitively valid.[...] Ingenuity will then determine which steps are the more profitable for the purpose of proving a particular proposition. [...] We then imagine that all proofs take the form of a search through this enumeration for the theorem for which a proof is desired. In this way ingenuity is replaced by patience. [...] We are leaving out of account that most important faculty which distinguishes topics of interest from others.

Intuition, ingenuity, patience and aesthetic judgement: these are the faculties on which Mathematics stands. To do Mathematics, to become mathematicians, we must master all four; but we must master patience first. It is through patience that we gather experience to eventually master the rest of the faculties. Of course, life is brief and it so often ends before we get to understand *how to wait*.² A good method does just that, it reduces everything to patience; it tells us exactly how to wait (and what to do while waiting) so that intuition, ingenuity and aesthetic judgement can bloom.

We consider the work of mathematicians to be mathematical theories, that is, collections of proofs. Mathematicians must 1) decide what to attempt to prove 2) attempt to prove it 3) stay on top of all the chores that follow from the other two. This division of work is of course arbitrary, we often decide what to prove while attempting to prove something else, blurring the line between (1) and (2). Nevertheless, it is a useful distinction, because that is how the traditional Mathematics's course is structured. It is expected of teachers to carry out (1) -that is handout theorems to prove- and it is expected of students to carry out (2) -prove them-. The curricula also tends to oscillate between (1) and (2). We practice proving (2) in order to tackle interesting problems (1), conversely, we find certain theorems interesting (1) because of how challenging they are (2). Interest, like many of the concepts we study is inductive: two interesting things that are connected, become, once we notice the connection, much more interesting. Like all well founded inductive concepts, interest has base cases: 0, 1, = and ∞ are interesting regardless of anything else.

A similar distinction between deciding what to do, doing it and the chores surrounding both can be done for the other arts. Writers first decide what deserves to be said (1), they then arrange in a coherent sequence of words (2), along the way they type and correct typos (3). Photographers decide on a subject (1), they then decide on more concrete details like lenses, light, and exposure (2), along the way they must measure and keep a steady hand (3). We will name each of this divisions of work as (1) strategical, (2) tactical and (3) operational. Notice that we can excel at one of those levels while having foundational problems at the others. The president of a business corporation delegates tactical thinking to middle management; simultaneously, middle management is delegating strategical thinking to the president.

The four faculties of Mathematics are needed at all levels. For instance, a theorem can be interesting but have an ugly proof and a uninteresting theorem can have a beautiful proof. We want a method that allows us to perform at all three levels, requiring nothing but patience, and that allows the other three

²In the words of Pólya: "The first rule of discovery is to have brains and good luck. The second rule of discovery is to sit tight and wait till you get a bright idea" [2]

faculties to take root. There is no impossibility here, in much the same way in which we use a dead stick to help a tree grow, we use method to help a mind grow. Beware, great care has to be taken when choosing a method; the right one will do wonders, the wrong one will be far worse than no method at all.

3 – Survey

By the time we had defined the problem well enough to know where to look for known solutions, we had already worked out a possible solution ourselves. So please read the following section not so much as a representative sample of the state of the art, but as a representative sample of what the author has, however briefly, considered.

We consider any work that discusses how Mathematics could be done as related to ours. We consider those works to be either *external* or *internal*. By *external* we mean those that stand on some external source of trust, like a compiler, a proof assistant, or a colleague. By *internal*, we mean those that ask us to put our trust in our minds, which depending on our previous experience, could be a blessing or a curse.

Entirely external methods go against the very meaning of Mathematics. As Voevodsky says: *“The computer-generated proofs are the proofs which teach us very little. And there is a correct perception that if we go towards computer-generated proofs then we lose all the good that there is in Mathematics — Mathematics as a spiritual discipline, Mathematics as something which helps to form a pure mind”* [7]. Notice how he implies that all computer-generated proofs are, by necessity, non-internal. However, he should not be blamed for thinking this, since even John McCarthy, one of the founders of the field of Artificial Intelligence, claims that the focus of the field should firmly rest on the external: *“ [This] is or should be our main scientific activity — studying the structure of information and the structure of problem solving processes independently of applications and independently of its realization in animals or humans.”* [8].

Of course, internal tools alone are not sufficient: they tend to be compromised whenever we are tired or nervous, they are intangible, invisible and thus we tend to forget they are even there, referring to them as *just how things are*. They also require lots of introspection to learn, teach and master; which increases the student’s burden.

Our method is both internal and external: it allows us to shift the trust from internal to external sources, oscillating as we see fit. **This is, as far as we have dived in the literature, our main contribution.** Our method can be run on a computer and the resulting proofs will still be faithful to Mathematics as a spiritual discipline. It can be run entirely on our minds and still be exposed nakedly for all to see. If we relax this requirement of being both internal and external at the same time, any of the following works would serve us better.

3.1 External

One of the first successes in automatic theory development was The AM program[9]. It used the programming language Lisp as a way to encode mathematical concepts and used heuristics to discover interesting conjectures. The heuristics were

presented in English and as a computer program. In this way, AM is similar to our work in form. Yet it is subtly different in intention. As the successor of the AM program, Eurisko, makes it clear, the goal of the research was to make better external tools for the development of Mathematics [9].

Most computer programs for automatic development of Mathematics are inspired by how mathematicians appear to work but do not care about imitating it or improving directly upon it. They care about getting results through the tool. Simon Colton puts it best on *On Automated Theory Formation in Pure Mathematics*: he begins with “[...] suggested that there was some methodology behind theory formation, which in turn suggested using a computer to follow that methodology” but quickly goes on to “*It was not the aim of this project to study how mathematicians form theories. [...] there is no compelling reason why a computer program should form theories as humans do.*”. Colton’s work contains an extensive survey of external tools in a very accessible presentation [10].

Beyond discovery, there is also proving. Here we have the usual suspects: SMT-solvers like Z3, the Otter and the Mace program, plus various automatic tactics within the frameworks of Agda, Coq and Isabelle. All these provide us with tools to better frame our thoughts and to automatize a lot of menial work, but they are clearly external since they expect to guide the mathematician by providing results instead of initiation.

3.2 Internal

If revealing the secrets of a craft to the general public is a noble enterprise, then Pólya’s work is heroic. His work is very extensive, considers both discovery and proving, and has been highly praised [2][11]. Pólya discusses methods for doing Mathematics. He advises to restate the problem in more familiar terms, to look for related problems, to try our luck with a more general problem, to try our luck with particular instances, to consider the data in many different presentations, among many other courses of action. He also pursues rules for guessing such as *If A is analogous to B and B is true, then A is more credible* [12]. Still, his techniques require a strong foundation on *intuition* and *ingenuity* to be followed.

Dijkstra’s work went further, looking to abolish ingenuity wherever was possible [13]. He did so by doing Mathematics in the most explicit way possible, and by studying notation and how it affects the development of proof. Nevertheless, he fell short of providing a method; the method he followed, if any, has to be extrapolated from all his, very carefully explained, proofs. In this way, his work can be considered on equals parts style and methodology. Dijkstra’s contribution took root and was further developed by many others. Like in *A Logical Approach to Discrete Mathematics* by Gries and Schneider or the many contributions at the MathMeth.com community [14][3]. Ralph-Johan Back built upon that style and worked towards a formalization of a *Calculational Proof* [15].

All these works contain discussions about methodology, but they all seem to

resist exposing all the methodology at once.

4 – A Variant of Typed λ -calculus

Our chosen language is a variant of typed λ -calculus, it is also a subset of the Haskell programming language, which belongs to the ML family. It only allows recursive functions, finite types (like `Bool`), inductive types (like \mathbb{N}) and inductive parametric types (like `(List a)`, i.e. lists of any fixed type `a`). In our notation those would be:

```
data Bool = True | False
data N = Z | S N
data (List a) = Nil | Cons a (List a)
```

To manipulate values we may use the case construct. To simplify matters we will not allow pattern matching or classes. So we will write:

```
sum :: N -> N -> N
sum =  $\lambda x.\lambda y.$  case x of {
    Z -> y;
    S j -> S(sum j y)
}
```

to define the addition of two natural numbers. To simplify matters even further, we will only consider primitive recursion.

We regret that this section is not self-contained, but many great works have been written explaining languages like the one we are using. Any introductory text to Haskell that includes inductive proofs should be sufficient.

We will only consider propositions of the form $\dots = \dots$. That is, we will only consider equalities among expressions of one and the same type where all the variables are quantified with a \forall . We will consider theorems like $x + y = y + x$, but we will leave out others like $(\exists x : \mathbb{N})(\forall y : \mathbb{N})(x + y = y)$ and $(\forall x : \mathbb{N})(x \geq Z)$ out of scope. The set of theorems involving just $(\forall \dots)(\dots = \dots)$ was significant enough to be attractive while being small enough for the author not to shiver in terror.

The language and the choice of theorems follows a course at Universidad ORT Uruguay, Foundations of Computing, which -as we have said elsewhere- turned out to be very fulfilling for all parties involved [16].

5 – The Method

Our method arises from the author's experience while teaching Foundations of Computing. The careful pen of the course designer, Álvaro Tasistro, built an environment where, being so formal and rule oriented, ingenuity could thrive. He also insisted in following method wherever possible, which became the foundations for our method. I made three contributions: 1) that it is possible -and desirable!- for students to invent their own exercises, 2) that we can also explain how to judge a lemma and when it is best to apply it, 3) that with the previous additions we can package everything in one single method. Those additions, coupled with motivation and other niceties form the present work.

As we said before, our method is both *internal* and *external*. It can fit entirely in the students mind and it can fit entirely outside of it. We have to prove both of this properties. We will prove that it is internal by explaining it in somewhat informal mathematical English. The readers can then become convinced by applying the informal definition with their own hands and eyes. We will prove that it is external by providing an implementation as a computer program in Haskell. A glaring omission from our work is proving the correspondence between the informal internal definition and the formal external one. We apologize. Anyone who understands Haskell and is brave enough to dive read the code, should, after many profanities, be convinced that this is so.

As we have mentioned, we will divide the method into three levels: the *strategical*, the *tactical* and the *operational*. That is: “*What to prove*”, “*How to prove*” and “*How to do everything else needed*”. Like in the traditional mathematical education we will start with the *operational* (how to check for equality, how to substitute), then go into the *tactical* (when to apply a lemma, when to carry induction) and finally into the *strategical* (which theorems are interesting).

We now present the *internal* version of the method.

6 – Operations Permit

The operational level defines what it is possible. It defines all the possible actions we could take. What we can and cannot do. It permits certain actions and denies others. To our purposes this level contains everything needed for the proper care of syntax. This level includes for example recognizing when there is a type error (i.e. whether an expression compiles), writing a primitive schema of recursion from a type definition, applying a β -rule, recognizing where a lemma can be applied regardless of whether it is desirable to do so.

Much has been written elsewhere about this operational concerns, so we can safely skip them. This does not mean they are not important, *au contraire*. A mathematician that does not understand substitution is like a singer who does not breath properly or a dancer with posture problems. Special care should be taken so that all students overcome the hurdles of parsing and proper eye-hand coordination. Teaching anything else before fixing that is like trying to fill a bathtub without putting the stopper first. Nevertheless, most of the things in this level are well known in Computing Science, and the rest seem to belong to the field of Didactics.

7 – Tactics Prove

The tactical level defines how we attempt things. In the operational level we just *do*. In the tactical level we *do in order to*. Choice, desire and the risk of failure come into play. Tactics are operations tagged with a purpose that might fail. On the writing of fiction, Raymond Chandler proposes the following tactic “*When in doubt, have a man come through a door with a gun in his hand*” [17]. The purpose is to generate drama, and it comes with the suggestion to only use it when we cannot think of anything else. On drawing, Betty Edwards

advices “*To draw the iris (the colored part of the eye)—don’t draw it. Draw the shape of the white. The white can be regarded as negative space, sharing edges with the iris. By drawing the (negative) shape of the white part, you’ll get the iris right because you’ll bypass your memorized symbol for iris. Note that this bypassing technique works for everything that you might find ‘hard to draw’. The technique is to shift to the next adjacent shape or space and draw that instead*” [18]. The purpose of this tactic is to make realistic drawings and it comes with the suggestion of using it for ‘hard things’ and in particular for irises. In the development of a proof, Pólya advises “*If you have to prove a theorem, do not rush. First of all, understand fully what the theorem says, try to see clearly what it means. Then check the theorem; it could be false. Examine the consequences, verify as many particular instances as are needed to convince yourself of the truth. When you have satisfied yourself that the theorem is true, you can start proving it.*” [2]. Its purpose is to prove a theorem and the suggestion is to first make use of all the tactics that can disprove it before trying to prove it.

Of course, tactics can fail and all artists make extensive use of drafts to compensate for this. If we do hide the drafts, we are hiding the method, but asking students to keep an archive of drafts and the reason for their dismissal is yet another hurdle they might carry. To solve this we structured our main tactic, composed of five minor tactics, in such a way as to eliminate, as much as possible, the need for backtracking. By only undoing the current step, and never previous ones, we can proceed without the need for drafts. This idea originated as way of presenting proofs during lectures -where backtracking tends to be a source of confusion- but we also found that it greatly reduces the anxiety when the students perform the proofs themselves. At every step they know that what they have already done was worthwhile, and thus can put their whole attention on the next step.

Our method will receive a conjecture to prove, a series of lemmas and hypotheses of the same form and the environment defining the datatypes and the variables. It will then either fail or succeed. If it succeeds it might produce a list of other conjectures to prove, each with its own list of new hypotheses. In that case we will apply the method again for each new conjecture taking into account its hypotheses and so on. Each time the tactic succeeds we will write a new line of the proof for each new conjecture. If the tactic succeeds and produces no new conjectures, it means we have proved the conjecture successfully and nothing else is left to be done.

Notice that we have not considered how to disprove theorems. If the tactic fails it merely means that we have not found a proof, not that no proof exists. We will leave tactics for attempting to disprove (like instantiating values on boundary cases) for another work.

At all times we are talking about *attempts*. The theory is of course undecidable, so we have to settle for methods that bring useful results in many, but not all, of the situations in which we are interested.

Our main tactic is composed of five tactics which are tried in turn. If one succeeds, we reapply the main tactic to all the new conjectures that arise, otherwise we try with the next one. If all the five tactics fail, the attempt fails. The tactics are *Trivialize*, *Compute*, *Fundamentalize*, *Reorder* and *Induction*.

7.1 *Trivialize*

Trivialize is the easiest tactic and the only one that can succeed without generating more work. Given a theorem of the form $\alpha = \beta$ it returns success with the empty list if α and β differ only in the names of the bound variables.¹ Otherwise, it fails.

7.2 *Compute*

Computing, i.e. solving redexes, has the benefit of reducing the formulas to a normal form. This increases the chance of succeeding with *Trivialize*. We should be careful not to complicate matters for the other tactics by making the resulting formulas more complex. To fix this, we will only solve redexes when we can do so without ending in a case construct or a lambda expression. That is, given: $sum\ Z\ x = sum\ x\ Z$ with the definition of *sum* already given, we will compute the left side to x but we will not compute the right side. So we will end up with $x = sum\ x\ Z$ instead of $x = case\ x\ of\ \{Z \rightarrow Z; S\ j \rightarrow S\ (sum\ j\ Z)\}$. We do this because the former is much more ergonomic (when performing induction) than the latter.

7.3 *Fundamentalize*

This is the first tactic that considers lemmas and hypotheses. The decision of whether to apply a given lemma is the same as whether to apply a given hypothesis, so we will only mention lemmas from now on. We want to, whenever possible, use lemmas as rewriting rules. To do so we have to choose the direction in which to apply the lemma. That is, given $\alpha = \beta$, we want to transform it to either $\alpha \rightarrow \beta$ or $\beta \rightarrow \alpha$. We noticed through experience and intuition that some expressions are more desirable than others, making the decision of which direction to use the lemma an obvious one. When it is obvious that the proper way to apply the lemma is $\alpha \rightarrow \beta$ we say that β is more *fundamental* than α .

What do we mean by *fundamental*? Well, the expression x is more fundamental than $x + 0$. This is easy to see, because to understand $x + 0$ we must understand two concepts more (+ and 0). We also consider + to be more fundamental than *, because the former is used in the definition of the latter. Now let us say that, for some f , we have to choose between the expression $f(\alpha + \beta)$ and the expression $f(\gamma * \delta)$. Which one would we prefer? Without knowing anything else, we would prefer the former since + is more fundamental than *. We would prefer it even if α and β contained multiplications because it is the outermost function which acts as a bridge with f and by virtue of being more fundamental it is much more likely that we have lemmas involving f and + than involving f and *. We consider, then, that $f(\alpha + \beta)$ is more fundamental than $f(\gamma * \delta)$. Since in our language all functions are prefix, the outermost function always sits syntactically at the left of the expression². So the expression $f(g(x))$

¹That is, if they are α -equivalent.

²In our language all functions are prefix. We use infix notation only to simplify some explanations.

should be more fundamental than the expression $g(f(x))$ if and only if f is more fundamental than g .

Taking this into account, we propose a formalization of *fundamentality* as an index, in such a way that the lower the index, the more fundamental the expression. Constructors have a fundamentality of 1. Bound variables a fundamentality of 0. Variables names defined in the environment have the fundamentality of their definition. A case construct has the sum of the fundamentality of its expressions plus 1 for each bound variable. A lambda term adds one to the fundamentality of its body. An application $\alpha\beta$ will have, on one hand, the sum of the fundamentality of α and β if α is less fundamental than β . On the other hand, if β is more fundamental than α , the fundamentality of $\alpha\beta$ will be equal to the fundamentality of β . This last comparison is done to assure ourselves that *sum* ($mul\ x\ y$) ($mul\ x\ z$) is more fundamental than $mul\ x\ (sum\ y\ z)$ for the reasons mentioned in the previous paragraph. Equationally:

$$\begin{aligned}
fund\ env\ K &= 1 \text{ (where } K \text{ is a constructor, for example } Z \text{ or } Cons) \\
fund\ env\ x &= 0 \text{ (where } x \text{ is a variable not previously defined in } env) \\
fund\ env\ f &= fund\ env'\ def \text{ (where } f \text{ is defined in } env \text{ as } def \\
&\quad \text{and } env' \text{ is } env \text{ without } f) \\
fund\ env\ (M\ N) &= (fund\ env\ M) + (fund\ env\ N) \text{ (if } fund\ env\ M > \\
&\quad\quad\quad fund\ env\ N) \\
fund\ env\ (M\ N) &= (fund\ env\ N) \text{ (if } fund\ env\ M \leq fund\ env\ N) \\
fund\ env\ (\lambda x.N) &= 1 + (fund\ env\ N) \\
fund\ env\ (case\ \alpha\ of\ \{K\ v_1, v_2, \dots v_n \rightarrow \beta; \Gamma\}) &= n + fund\ env\ \beta + \\
&\quad\quad\quad fund\ env\ (case\ \alpha\ of\ \{\Gamma\}) \\
fund\ env\ case\ \alpha\ of\ \{\} &= fund\ env\ \alpha
\end{aligned}$$

As long as all recursion is primitive, this suffices. Once general recursion is allowed, the fundamentality of an expression is mostly weighted by how fundamental the schema of recursion used is. We leave such considerations to the sequel of this work.

We can now use *fund* to decide in which direction to apply a lemma. We can simply calculate the fundamentality index and pick the direction that lowers it. When having to choose between two possible application of lemmas, we can apply first the one which will give us the most fundamental expression.

Let us consider this list of lemmas:

1. $sum\ x\ Z = x$
2. $mul\ x\ Z = Z$
3. $sum\ x\ (S\ y) = S\ (sum\ x\ y)$
4. $mul\ x\ (sum\ y\ z) = sum\ (mul\ x\ y)(mul\ x\ z)$
5. $length\ (append\ xs\ ys) = sum\ (length\ xs)\ (length\ ys)$
6. $sum\ x\ y = sum\ y\ x$

where *sum* maintains its previous definition and the other functions involved are defined as:

$$\begin{aligned}
mul &:: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \\
mul &= \lambda x.\lambda y.\ case\ x\ of\ \{ \\
&\quad Z \rightarrow y;
\end{aligned}$$

$$\begin{aligned}
& \text{S } j \rightarrow \text{sum } y \text{ (mul } j \text{ } y) \\
& \} \\
\text{length} & :: (\text{List } a) \rightarrow \mathbb{N} \\
\text{length} & = \lambda xs. \text{case } xs \text{ of} \{ \\
& \quad \text{Nil} \rightarrow 0; \\
& \quad (\text{Cons } z \text{ } zs) \rightarrow \text{S}(\text{length } xs) \\
& \} \\
\\
\text{append} & :: (\text{List } a) \rightarrow (\text{List } a) \rightarrow (\text{List } a) \\
\text{append} & = \lambda xs. \lambda ys. \text{case } xs \text{ of} \{ \\
& \quad \text{Nil} \rightarrow ys; \\
& \quad (\text{Cons } z \text{ } zs) \rightarrow \text{Cons } z \text{ (append } zs \text{ } ys) \\
& \}
\end{aligned}$$

It is easy to calculate that in all but the last lemma, the subexpression on the left-hand side of the = is less fundamental than the subexpression on the right-hand side. So we will rewrite any expression of the form of left-hand side to the corresponding one on the right-hand side. In the last lemma both sides have the same fundamentality. We will discuss such matters in the next tactic.

This calculation of fundamentality should only be done once for each new lemma we encounter. The decision of which of two lemmas to apply can be done by choosing the one that results in a more fundamental expression. Such comparison between lemmas can be done only once if we maintain a sorted list. In this way we have increased the chores of the student, but it is only a mild increment. At the same time we are erasing a great source of anxiety, that is: *what* to do with the lemmas. Sorting the list of lemmas is reduced to counting, and applying a lemma is reduced to trying the lemmas in the sorted list in turn. This is, in all its meanings, a great deal.

Is it the best way to decide what to do with a lemma? We have blind faith that it is not *the* way, but it is *a* way. A way of deciding what to do with the lemmas that is both internal and external, that works in many situations in which we are interested, that does not require any disproportionated efforts on part of the student and that foments the habit of seeing lemmas as tools that must be studied in relation with the other available tools before use.

7.4 Reorder

If we consider the lemma $\text{sum } x \text{ } y = \text{sum } y \text{ } x$ in isolation, any direction would be arbitrary. If it cannot depend on the lemma, it must depend on the expression we apply the lemma to. If we are in a situation where we can apply such a lemma we will do so only if it results in a smaller expression under some (arbitrary) order. For instance, if we chose to order our expressions alphabetically, when faced with $(gx) + (fy)$ we will apply the commutative of addition, resulting in $(fy) + (gx)$. We will not apply the lemma again since that would result in a larger expression under our order.

In a future work, we would like to redesign this tactic in order to choose the more promising permutation available, but, for the moment, this somewhat naive application seems to work for our purposes.

7.5 Induction

If everything else fails, but there is at least one quantified variable, we can apply primitive induction. This tactic is the reason that tactics return a lists of new conjectures to prove, one for each case, each with its own hypotheses. In order to make use of *Fundamentalize* we have to calculate the direction as well as the order of application of each hypothesis. Notice that the direction of all the induction hypotheses will be the same since, barring the name of the variables, they are syntactically equal. So we can save us some work and calculate it only once.

In general, we try to avoid induction, since it increases the amount of sub-proofs and it does so exponentially. If there is more than one variable in which applying induction is possible, we might choose the wrong one and quickly find ourselves having no option than to apply induction as well on other variable. A good tactic is to consider the definitions of the functions involved, and to only apply induction on a variable which will allow us to perform *computation* on all the cases. If the prover chooses the wrong variable in which to apply induction, the proof works out just the same, although it might become a test to the prover's patience.

7.6 Carrying out a proof

We will attempt to prove by hand the following theorem:

$$\text{length} (\text{append } xs \ ys) = \text{sum} (\text{length } xs) (\text{length } ys)$$

with the already given definitions, and the lemmas used as example in the section of *Fundamentalize*:

1. $\text{sum } x \ \mathbf{Z} = x$
2. $\text{mul } x \ \mathbf{Z} = \mathbf{Z}$
3. $\text{sum } x \ (\mathbf{S} \ y) = \mathbf{S} (\text{sum } x \ y)$
4. $\text{mul } x \ (\text{sum } y \ z) = \text{sum} (\text{mul } x \ y)(\text{mul } x \ z)$
5. $\text{sum } x \ y = \text{sum } y \ x$

We have already sorted the lemmas as discussed in *Fundamentalize* and *Reorder* arriving at the order presented. We also know that the right-hand side of our conjecture is more fundamental than the left-hand side, and the same property will hold for any hypothesis that arises. Beginning the proof, we notice that all first four tactics fail so we apply induction. Applying induction on xs allows *Compute* to succeed in all cases on both sides of the equality. Applying induction on ys instead leaves us the subexpression $\text{append } xs \ (\mathbf{Cons} \ z \ zs)$ which forces us to apply induction again. So, not to test the patience of the reader, we prefer applying induction on xs . In the following proof, comments are surrounded with `{}`.

Base case: $xs = \mathbf{Nil}$

$$\text{length} (\text{append } \mathbf{Nil} \ ys) = \text{sum} (\text{length } \mathbf{Nil}) (\text{length } ys)$$

`{Compute succeeds with both append and length.}`

$$\text{length } ys = \text{sum } \mathbf{Z} (\text{length } ys)$$

{*Compute* succeeds with *sum*.}
 $length\ ys = length\ ys$
 {*Trivialize* succeeds, there is nothing else to do in this case.}

Inductive case: $xs = \text{Cons } z\ zs$

{In the inductive case we gain the hypothesis:}
 $length\ (\text{append } zs\ ys) = sum\ (length\ zs)\ (length\ ys)$
 {which we know that we will only apply *left-to-right* and we have to prove:}
 $length\ (\text{append } (\text{Cons } z\ zs)\ ys) = sum\ (length\ (\text{Cons } z\ zs))\ (length\ ys)$
 {*Compute* succeeds with both *append* and *length*.}
 $length\ (\text{Cons } z\ (\text{append } zs\ ys)) = sum\ (\text{S } (length\ zs))\ (length\ ys)$
 {*Compute* succeeds with both *length* and *sum*.}
 $\text{S } (length\ (\text{append } zs\ ys)) = \text{S } (sum\ (length\ zs)\ (length\ ys))$
 {*Fundamentalize* succeeds with the hypothesis.}
 $\text{S } (sum\ (length\ zs)\ (length\ ys)) = \text{S } (sum\ (length\ zs)\ (length\ ys))$
 {*Trivialize* succeeds, there is nothing else to do in this case and there are no more cases so we are done!}

Although our chosen presentation is quite verbose, the detail can be adjusted as necessary. We could combine the various applications of *Compute* into one step. We could also omit some of the comments describing which tactic we are applying. Or, going in the other direction, omit the formulas and just mention the tactics, à la Coq: “Apply *Induction* on *xs*. Solve the base case by applying *Compute* and *Trivialize*. Solve the inductive case by applying *Compute*, *Fundamentalize* with the hypothesis and *Trivialize*.”

8 – Strategies Propose

Strategies propose what is worthy of being proven, what is desired. Of course both tactics and strategy can modify our current goal but there is a difference of intention. Tactics search in order to find. Strategies search in order to keep searching. On Writing, Stephen King says “*You must do two things above all others: read a lot and write a lot*” [19]. This is a very common strategy that popular culture sums up as “*Practice makes perfect*”. Going onto other matters, Self-Defense writer Mac MacYoung advises “*Start picking up things and looking at them as possible weapons. How would you use a rolled-up newspaper?*” [20]. That is a strategy, he is telling us to understand our environment now so that we can understand how to get out of a dangerous situation later. Contrast it with the following tactic “*A guy with a baseball bat at four feet is something to worry about – but have you ever tried to wrestle while holding onto a baseball bat?*” [20]. Looking beyond the rhetoric devices, it gives us a clear course of action on a particular situation to achieve a particular purpose so it is a tactic.

On Mathematics, strategy is discussed in aesthetics terms. We say that things are interesting or beautiful. Like in Self-Defense, we want to study our environment. If we decide to study types, what are the most common types? Once we find lists, what are the most common operations on lists? What are their properties? When von Neumann says “*With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.*”, he is telling us to give

priority in our design and study to functions with a small number of parameters [21].

We notice that in our language most of the objects we are interested in have a short definition. So we can propose new objects of study by enumerating definitions and seeing if they result in something interesting. This goes somewhat against Mathematics’s tradition. We can compare with von Neumann: *“I think that it is a relatively good approximation to truth — which is much too complicated to allow anything but approximations — that mathematical ideas originate in empirics. But, once they are conceived, the subject begins to live a peculiar life of its own and is . . . governed by almost entirely aesthetic motivations. In other words, at a great distance from its empirical source, or after much “abstract” inbreeding, a mathematical subject is in danger of degeneration. Whenever this stage is reached the only remedy seems to me to be the rejuvenating return to the source: the reinjection of more or less directly empirical ideas”* [22]. This does not bother us. We are not proposing the alpha and omega of strategies, but one easy to perform that brings up results quickly.

In this way we will propose data types, functions over those data types and equations about those functions. After proposing, we attempt to prove. If we succeed in proving many properties about a function, we consider it interesting. So we use it as a base to propose more functions and more equations. For example, since the addition is interesting, we propose multiplication (among other functions that turn out not to be interesting); since multiplication is interesting, we propose the exponentiation (among other functions) and so on.

In this way, interest builds up. We must also consider that certain discoveries make other previous discoveries *less* interesting. The function that doubles a number becomes less interesting once we know multiplication. The function that counts elements of a list satisfying a predicate becomes less interesting once we know that it can be reduced to the composition of length and filter. The type (List Unit) seems to be interesting until we realize that it is isomorphic to \mathbb{N} . We will now detail how the method proposes.

8.1 Proposing Data Types

As we have said we care about enumerated types like `Bool`, inductive types like \mathbb{N} and inductive parametric types like `List a`. Notice that we care about the structure of the type, not of its name. The type `Bool = True | False` sprouts the same theory than the type `Direction = Left | Right` even if we do not want to confuse them in practice. To avoid wasted efforts reinventing the wheel we consider canonical names for types. In our notation both `Bool` and `Direction` are the type `_,_ = _,_.0 | _,_.1`

We present some types and their canonical definition:

Canonical	Traditional
<code>_,_ = _,_.0 _,_.1</code>	<code>Bool = True False</code>
<code>_,R = _,R.0 _,R.1 ..R</code>	<code>N = 0 S N</code>
<code>_,a a = _,a.0 _,a.1 a</code>	<code>Maybe a = Nothing Just a</code>
<code>_,Ra a = _,Ra.0 _,Ra.1 a (..Ra a)</code>	<code>List a = Nil Cons a (List a)</code>

All type names are a list of the parameters of their constructors separated by “,”. If a constructor has no parameters we use a “_”. If the parameter is recursive we use an “R”. If the parameter is any other fixed type we use a

lowercase letter. To further reduce the number of possible isomorphic types, we demand that the list of constructors be sorted and that we start naming the variables at “a”.

The name of each constructor is the name of the type plus a dot and its position. Many order of enumerations are suitable, but in general we would want to consider `Bool` before `N` and `Maybe`. And `List` only after considering all the others. Following von Neumann’s lead, we will not be interested in types with many constructors or many parameters per constructor. So intuitively, we can pick a very small number of parameters and add to that a small number of parameters for each constructor. We must decide for each parameter whether it is recursive or not. We also must decide if any two parameters must be of the same type. For instance we can choose three constructors, add no parameters to the first, add one variable parameter to the second one and two recursive parameters to the first. That would give us:

`_,a,RR a = _,a,RR.0 | _,a,RR.1 a | _,a,RR.2 (_,a,RR a) (_,a,RR a)`
 which after a cosmetic renaming becomes:

`Tree a = Empty | Leaf a | Node (Tree a) (Tree a)`
 i.e. the type of binary trees with information in some of its leaves.

8.2 Proposing Functions

To propose functions we must first choose those types we are interested in. This is reduced to enumerating lists of types and then replacing the “,” with “→”. For instance, some types containing both `N` and `Bool`:

`N → Bool`, `Bool → N`, `N → N → Bool`, `N → Bool → N`, `N → Bool → Bool`,
`Bool → N → Bool`, `Bool → Bool → N`, `Bool → N → N`, ...

After proposing a type, we choose on which of its parameters we want to apply the schema of recursion. This gives us a schema of recursion to be filled in. Once the filling is done, we get a function. Since we are limiting ourselves to primitive recursion, and we only care of recursion in a very small number of parameters, the number of schemas for a given type is relatively small. For instance:

With `N → N → Bool` we can choose to recur first on the first parameter, then on the second in the last branch. This would give us the schema:

$$f = \lambda n. \lambda m. \text{case } n \text{ of} \{$$

$$\quad Z \rightarrow \dots m \dots ;$$

$$\quad S j \rightarrow \text{case } m \text{ of} \{$$

$$\quad \quad Z \rightarrow \dots j \dots ;$$

$$\quad \quad S k \rightarrow \dots j \dots k \dots (f j k) \dots ;$$

$$\quad \quad \}$$

$$\quad \}$$

which has three holes each with its new subexpressions.

This schema of recursion is derived mechanically from the definition of `N`. The writing of primitive schemas of recursion is well understood, so will not discuss it further.

After arriving at the schema of recursion we get functions by filling the holes with appropriate expressions. Here some heuristics are useful: (1) we must use at least one recursive call if available (2) we will avoid using the same subexpression more than twice (3) we will try to keep the fundamentality score

of the function as low as possible (which usually means reducing the use of auxiliary functions to the minimum). In the example above, by making use of only `True` and `False` and forcing the use of the recursive function we would have (\leq) , $(>)$, $(\lambda x.\lambda y.\text{True})$ and $(\lambda x.\lambda y.\text{False})$.

8.3 Proposing Conjectures

We do not care about all theorems, we only care about those theorems that increase our tactical or strategical ability. A theorem is *ample* if we can apply it in a great number of situations. For instance: $x * (y + z) = x * y + x * z$ is ample, we can use wherever there is a `*` outside of a `+`. Since `+` and `*` are very interesting and are related, this happens very often and the one above is a theorem worthy of study. Consider now $x + 2^2 = 3 + x + 1$. It holds but we are unlikely to ever use it. Ample theorems increase our tactical ability. A theorem is *deep* if it connects two previously unrelated concepts. The classical example is Euler's identity: $e^{i\pi} + 1 = 0$. A more modest but still somewhat deep theorem is the one discussed in section (7.6) which signals the connection between addition and concatenation. Deep theorems increase our strategic ability.

When faced with a new function we first propose ample conjectures. The question we are trying to answer is: how can I maneuver when I encounter this function in the future? How does it play with the other functions I know I am likely to encounter? When faced with multiplication it is natural to ask if it is commutative, if it is associative, if it distributes over other functions, if other functions distribute over it. Conversely, when we introduce another function it is natural to ask ourselves how it interacts with multiplication.

Once we are convinced that the function has many ample theorems -and thus that it is interesting- we might look for deep theorems. Since we are trying to connect seemingly unrelated things, the search space is much larger. We have done some experiments on the search of deep theorems, but none successful enough to include here. We will only mention that by comparing sets of ample theorems, a clue for a deep theorem might be found. For instance, the theorems of concatenation are (in some sense) a subset of the theorems of addition. This suggests that there might be a somewhat deep theorem involving both.

9 – Implementation as a Computer Program

We have implemented the informal method described above as a computer program. We have done so to convince ourselves that the method is purely external. The implementation also serves as a formal specification, the previous exposition of the method, by virtue of being informal, might have omissions or ambiguities. No such thing can happen in the code. This is the reason for us writing an implementation. We care not about automated proof assistants, but we do care about expressing the method.

To us, in this work, a programming language is just that, a language. We chose Haskell as our language, which has the advantage of being a superset of our version of λ -calculus.

We will not discuss the implementation in depth, but we will establish some

correspondence between the informal method previously discussed and the code presented in the appendix A. We have specified the syntax and semantics of our language in the modules `Compute`, `Syntax` and `Type`. We have specified the tactical level in the `Prove` module. We have specified the strategical level in the `Propose` module. The operational level is scattered all over. Reading the code will make evident some of the hurdles students must face, like all the syntactical juggling needed to decide if a lemma can be applied in some expression.

To understand the tactical level, one could start by reading the definition of `mainTactic` and following its dependencies. *Trivialize* is defined as `triv`. *Compute* is defined as `bothSides (computeDontEndInCas env)`, *Fundamentalize* and *Reorder* are defined in `applyRec` and `addAffirms`, *Induction* is defined as `indc`. Following `addAffirms` one will notice that *fundamentality* is defined as `fund`.

To understand the strategical level, one could read the `Propose` module sequentially. Data types are proposed by `enumContainerRecTypes`. Functions are proposed by `enumTip`. Theorems are proposed by `findlemas`. This last functions depends on the `Prove` module in order to cut the search space as soon as a good theorem is proposed. That is, we do not suggest $x + 0 + 0 = x$ if we have already proved that $x + 0 = x$.

The `Main` module takes care of a bit of plumbing in order to put it all together.

10 – Results

We ran the method in search of functions involving \mathbb{N} and the type of lists. The results, as well as the code run, are provided in the appendices B and A respectively. On its first pass, it discovered the addition (`f0` in appendix B) and the concatenation (`f2`). It also discovered a variant of addition (`f1`), that is the function $(\lambda x y . x + y + 1)$. It did not discover length. That is because, taken on its own, length has no interesting theorems (since it cannot be combined with itself).

We then ran the method again, this time searching for functions that used addition and concatenation as auxiliary functions. It found reverse `f3` and the fact that it is its own inverse. We present the proof as developed by the program¹ :

```

rev (rev a) = a
  { Induction on a }
T) rev (rev Nil) = Nil
  { Compute ; }
rev Nil = Nil
  { Compute ; }
Nil = Nil
  { Triv }
Done.

```

¹To improve the presentation we have renamed `f3` as `rev` and `f2` as `concat`.

```

H) rev (rev k) = k
T) rev (rev (Cons j k)) = Cons j k
{ Compute ; }
rev (concat (rev k) (Cons j Nil)) =
{ Fund: rev (concat a b) = concat (rev b) (rev a) ; }
concat (rev (Cons j Nil)) (rev (rev k)) =
{ Compute ; }
concat (concat (rev Nil) (Cons j Nil)) (rev (rev k)) =
{ Compute ; }
concat (concat Nil (Cons j Nil)) (rev (rev k)) =
{ Compute ; }
concat (Cons j Nil) (rev (rev k)) =
{ Compute ; }
Cons j (concat Nil (rev (rev k))) =
{ Compute ; }
Cons j (rev (rev k)) =
{ Fund: rev (rev k) = k ; }
Cons j k = { Triv } Done.

```

The method found multiplication (f12) and many variants of it. The method could be extended to prune the variants by deleting the functions that can be defined completely in terms of others, this could be done by searching for the appropriate theorems first.

Each time we run the method it takes an order of magnitude of more time, because there are an order of magnitude more functions to combine. This is not a problem, because we can concentrate on one function as we have done in the module `Test`, which can be found in appendix A. There we searched for theorems involving exponentiation. In particular we searched for theorems involving `flipexp` $x y = y^x$. It provided the following lemmas:

1. `flipexp (suma a b) c = mult (flipexp a c) (flipexp b c)`
2. `flipexp (mult a b) c = flipexp a (flipexp b c)`

Corresponding to $c^{a+b} = (c^a)(c^b)$ and $c^{a*b} = c^{b^a}$ respectively. We present the proof of the latter:

```

flipexp (mult a b) n = flipexp a (flipexp b n)
  { Induction on a }
T) flipexp (mult 0 b) n = flipexp 0 (flipexp b n)
{ Compute ; Compute }
flipexp 0 n = S 0
{ Compute ; }
S 0 = S 0
{ Triv } Done.

```

```

H) flipexp (mult c b) n = flipexp c (flipexp b n)
T) flipexp (mult (S c) b) n = flipexp (S c) (flipexp b n)
{ Compute ; Compute }

```



```

flipexp (suma (mult c b) b) n
  = mult (flipexp c (flipexp b n)) (flipexp b n)
{ Fund: flipexp (suma a b) n = mult (flipexp a n) (flipexp b n); }
mult (flipexp (mult c b) n) (flipexp b n)
  = mult (flipexp c (flipexp b n)) (flipexp b n)
{ Fund: flipexp (mult c b) n = flipexp c (flipexp b n); }
mult (flipexp c (flipexp b n)) (flipexp b n)
  = mult (flipexp c (flipexp b n)) (flipexp b n)
{ Triv } Done.

```

11 – Concerns Against Method

There are many reasons to be suspicious of methodology. As Turing pointed out, Mathematics is not decidable, so no method will ever be enough. In a certain sense, a method is defined by what it leaves out. No two methods are equal, some stomp on any creative sprout, while others create spots where creative thought is protected and nurtured. As Betty Edwards shows on Drawing, method and creativity can form a symbiosis. Method can free us. The proper method can save the students lots of strife and hardship, and go on to quickly surprise them with what they are capable of.

There are many reasons to resist method altogether. Paul Feyerabend argues “*Successful research does not obey general standards; it relies now on one trick, now on another; the moves that advance it and the standards that define what counts as an advance are not always known to the movers. Far-reaching changes of outlook, such as the so-called ‘Copernican Revolution’ or the ‘Darwinian Revolution’, affect different areas of research in different ways and receive different impulses from them. A theory of science that devises standards and structural elements for a scientific activities and authorizes them by reference to ‘Reason’ or ‘Rationality’ may impress outsiders - but it is much too crude an instrument for the people on the spot, that is, for scientists facing some concrete research problem*” [23]. But still, the reality is that students demand method and teachers comply however they can. Forbidding and persecuting method is like outlawing *cannabis*. Demand will still be fulfilled and the perils faced by those that demand it will only have increased.

As long as the student is aware that the method is no silver bullet, there should be no problem, he will naturally drop it on his own, once mastered. The proverbial teacher says “*Do as I say; not as I do.*” and the proverbial student silently answers “*Ha! You will see once I am a master too.*”. In those cases where the student does not develop a healthy sense of skepticism about the method, in those cases where the student falls into the dogma that he learned all that is, an hour long discussion about the difference between *the map* and

the territory should be enough to put the student in the right track again.¹

12 – Current Challenges and Future Work

We have presented a method for exploring a variant of λ -calculus which is also -by way of example- a method for exploring Mathematical Methodology. Mathematics studies mathematical objects, Mathematical Methodology studies methods of doing Mathematics. By formalizing a method of doing Mathematics, we have found ourselves in the exclusive field of Mathematics of Mathematical Methodology, a field with a lot of potential of applying its methods onto itself. We suspect that doing so will provide us with an infinite supply of general abstract nonsense.¹

This puts Mathematical Methodology on much firmer ground than before, but it still is quite shaky. It is easy to see that the computer program represents an external method, and it is easy to see that the informal method is internal, but it is hard to convince ourselves that they represent the same method. We would be in much better position if we had only one version of the method, that with the same ease could be run on a computer or be performed by hand. This could be done with a programming language that put an upper bound on the amount of times each operation can be done according to how costly it would be for a human to perform it. We could call such languages, *impatient* programming languages. With such a language, a calculus could be envisioned in which to prove that certain method is a better fit for our minds than another. Once we had that, we could go onto designing methods for designing methods which would give us the infinite supply of general abstract nonsense previously mentioned. This is, of course, pure speculation. We have not stolen, like Prometheus, the fire from the gods; but we did find where they have left it unguarded. Come on, let's raid!

¹A good but -by necessity- mystic book in these matters is Prometheus Rising [24].

¹This is a technical term [25].

13 – Bibliography

- [1] P. Lockhart, “A Mathematician’s Lament,” 2002. [Online]. Available: <http://www.maa.org/devlin/LockhartsLament.pdf>
- [2] G. Polya, *How to solve it*, 2nd ed. PUP, 1957.
- [3] H. Egghart, W. Feijen, R. Leino, and all, “Computational mathematics and mathematical methodology,” <http://www.mathmeth.com/read.shtml>, [Online; accessed 5-June-2016].
- [4] R. A. Goldthwaite, *The Economy of Renaissance Florence*. JHU Press, 2009.
- [5] N. Machiavelli, *The Prince*. Public Domain Books, 1513.
- [6] A. M. Turing, “Systems of logic based on ordinals,” *Proceedings of the London Mathematical Society*, vol. 2, no. 1, pp. 161–228, 1939.
- [7] S. Roberts, “In mathematics, mistakes aren’t what they used to be,” <http://nautil.us/issue/24/error/in-mathematics-mistakes-arent-what-they-used-to-be>, 2015, [Online; accessed 5-June-2016].
- [8] J. Bach, *Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition*, ser. Oxford Series on Cognitive Models and Architectures. Oxford University Press, 2009.
- [9] D. B. Lenat and J. S. Brown, “Why AM and Eurisko appear to work,” 1983.
- [10] S. Colton, *Automated Theory Formation in Pure Mathematics*, ser. Distinguished Dissertations. Springer London, 2012.
- [11] G. Alexanderson, *The Random Walks of George Polya*, ser. MAA spectrum. Mathematical Association of America, 2000.
- [12] G. Polya, *Mathematics and Plausible Reasoning*. Princeton, 1954, vol. Volume 1.
- [13] E. W. Dijkstra, “Introducing a course on calculi,” <https://www.cs.utexas.edu/users/EWD/transcriptions/EWD12xx/EWD1213.html>, 1995, [Online; accessed 5-June-2016].
- [14] D. Gries and F. B. Schneider, *A Logical Approach to Discrete Math*, ser. Texts and Monographs in Computer Science. Springer, 1993.
- [15] Back, Ralph-Johan, Grundy, Jim, Wright, and J. von, “Structured calculational proof,” *Formal Aspects of Computing*, vol. 9, p. 469–483, 1998.
- [16] A. Tasistro, J. Michelini, and N. Szasz, *Course on a Mathematical Presentation of Functional Programming*, 2015.

- [17] R. Chandler, *The Simple Art of Murder*, ser. Vintage Crime/Black Lizard. Knopf Doubleday Publishing Group, 2002.
- [18] B. Edwards, *The New Drawing on the Right Side of the Brain*. Jeremy P. Tarcher/Putnam, 1999.
- [19] S. King, *On Writing: A Memoir Of The Craft*, ser. Scribner classics. Scribner, 2000.
- [20] M. MacYoung, *Cheap Shots, Ambushes, And Other Lessons: A Down And Dirty Book On Streetfighting & Survival*. Paladin Press, 1989.
- [21] F. Dyson, “A meeting with Enrico Fermi,” *Nature* 427, 2004.
- [22] R. B. Heywood, Ed., *The Works of the Mind*. Univeristy of Chicago Press, 1947.
- [23] P. Feyerabend, *Against Method*. Verso, 1993.
- [24] R. Wilson and I. Regardie, *Prometheus Rising*. New Falcon Publications, 1992.
- [25] S. Mac Lane, “The PNAS way back then,” *Proceedings of the National Academy of Sciences*, vol. 94, no. 12, pp. 5983–5985, 1997. [Online]. Available: <http://www.pnas.org/content/94/12/5983.short>

Appendices

A – Formalization of the Method in Haskell (Source Code)

This appendix presents the source code used to formalize the method and to obtain the results in the appendix B. It might be possible that a more up to date and cleaner version of the code can be found at <https://github.com/juanmichelini/foundations>.

Contents

A.1	Syntax.hs	30
A.2	Compute.hs	35
A.3	Type.hs	37
A.4	Prove.hs	42
A.5	Propose.hs	53
A.6	Main.hs	60
A.7	Test.hs	63
A.8	Misc.hs	74
A.9	Util.hs	75

A.1 Syntax.hs

```
1 module Syntax where
2 import Data.List
3 import qualified Data.Set as Set
4 import Util
5
6 data Term = Atm Atm | App Term Term | Abs Var Term | Cas Term [((Con, [
  Var]), Term)]
7     deriving (Ord, Show)
8
9 data Atm = Var Var | Con Con
10     deriving (Show, Eq, Ord)
11
12 type Var = String -- Variables
13 type Con = String -- Constructors
14
15
16 isACon (Atm (Con _)) = True
17 isACon _ = False
18
19 isCon (Con _) = True
20 isCon _ = False
21
```

```

22 {- ## SHOW ## -}
23
24
25
26
27 {- ## EQUALITY ## -}
28 {- ## ALPHA RULE ## -}
29
30 instance Eq Term where -- intensional equality (with alpha rule)
31   (Atm a) == (Atm b) = a == b
32   (App x y) == (App a b) = x == a && y == b
33   (Abs w y) == (Abs v z) = (swapVars w (fvar) y) == (swapVars w (fvar)
34     z)
35     where { fvar = fresh [(Abs w y), (Abs v z)]}
36   (Cas x xs) == (Cas y ys) = x == y && xs == ys
37   _ == _ = False
38
39 swapVars :: Var -> Var -> Term -> Term
40 swapVars a b (Atm (Var w)) = Atm (Var (swap a b w))
41 swapVars a b (Atm (Con c)) = Atm (Con c)
42 swapVars a b (App x y) = App (swapVars a b x) (swapVars a b y)
43 swapVars a b (Abs w y) = Abs (swap a b w) (swapVars a b y)
44 swapVars a b (Cas x xs) = Cas (swapVars a b x) (mapSnd (swapVars a b) xs
45   )
46
47 swap :: Eq a => a -> a -> a -> a
48 swap x y z = if x == z then y else (if y == z then x else z)
49
50 usedVars :: Term -> [Var]
51 usedVars (Atm (Var w)) = [w]
52 usedVars (Atm (Con _)) = []
53 usedVars (App x y) = usedVars x ++ usedVars y
54 usedVars (Abs w x) = w:(usedVars x)
55 usedVars (Cas x xs) = usedVars x ++ (concat (map (usedVars . snd) xs))
56
57 freeVars :: Term -> [Var]
58 freeVars (Atm (Var w)) = [w]
59 freeVars (Atm (Con _)) = []
60 freeVars (App x y) = freeVars x ++ freeVars y
61 freeVars (Abs w x) = filter (/=w) (freeVars x)
62 freeVars (Cas x xs) = freeVars x ++ (concat (map (\((c,vs),t) -> filter
63   (not.(‘elem‘vs)) (freeVars t)) xs))
64
65 infiniteVars :: [Var]
66 infiniteVars = concat$ map (\x -> map (++x) goodVars) $ "":(map show
67   [0..])
68   where goodVars = map (:[]) ['a'..'z']
69
70 fresh :: [Term] -> Var
71 fresh xs = head $ freshs xs
72
73 freshs :: [Term] -> [Var]
74 freshs xs = (filter (\ w -> not ( w ‘elem‘ (concatMap (usedVars) xs)))
75   infiniteVars)

```

```

71
72 {- ## Parameters Params ## -}
73 listParams :: Term -> (Term, [Term])
74 listParams (App x y) = case listParams x of { (h, hs) -> (h, hs++[y])}
75 listParams h = (h, [])
76
77 appParams :: (Term, [Term]) -> Term
78 appParams (h, x:xs) = appParams ((App h x), xs)
79 appParams (h, []) = h
80
81 {- ## BETA RULE ## -}
82
83 applyBetaRule :: Term -> Maybe Term
84 applyBetaRule (App (Abs w y) a) = Just $ substitute w a y
85 applyBetaRule _ = Nothing
86
87 substitute :: Var -> Term -> Term -> Term
88 substitute v e t = multisubstitute [(v,e)] t
89
90 multisubstitute :: [(Var, Term)] -> Term -> Term
91 multisubstitute as (Atm (Var w)) = case lookup w as of {
92     Nothing -> Atm (Var w);
93     Just y -> y;
94 }
95 multisubstitute as (Atm (Con c)) = Atm (Con c)
96 multisubstitute as (App x y) = App (multisubstitute as x) (
    multisubstitute as y)
97 multisubstitute as (Abs w y) = Abs h (multisubstitute ((w,Atm (Var h)):
    as) y)
98     where (_, h) = head (refresh [w] [y] as)
99 multisubstitute as (Cas x xs) = Cas (multisubstitute as x)
100     (map \( (c,vs), exp) -> let hs = refresh
        vs [exp] as
101                                     ths = mapSnd (
        Atm . Var)
        hs
102     in ((c, snds hs),
        multisubstitute
        (ths++as) exp
        )) xs)
103
104
105
106 refresh :: [Var] -> [Term] -> [(Var, Term)] -> [(Var, Var)]
107 refresh vs ts sus = zip vs $ freshs (concat [map (Atm . Var) vs, ts, map
    (Atm . Var . fst) sus, snds sus])
108
109 {- ## DATA TYPE DEFINITION ## -}
110
111 applyCase :: Term -> Maybe Term
112 applyCase (Cas w xs) = do (c, ys) <- getHeadCon w;
113     ((_,vars), exp) <- head' $ filter ((==c).fst.fst)
        xs;
114     return $ multisubstitute (zip vars ys) exp

```



```

115 applyCase _ = Nothing
116
117 head' [] = Nothing
118 head' (x:xs) = Just x
119
120 getHead :: Term -> Maybe (Atm, [Term])
121 getHead (Atm c) = Just (c, [])
122 getHead (App x y) = do (c, xs) <- getHead x
123                       return (c, xs++[y])
124 getHead _ = Nothing
125
126 getHeadCon :: Term -> Maybe (Con, [Term])
127 getHeadCon (Atm (Con c)) = Just (c, [])
128 getHeadCon (App x y) = do (c, xs) <- getHeadCon x
129                          return (c, y:xs)
130 getHeadCon _ = Nothing
131
132 subsHead :: Term -> Term -> Term
133 subsHead t (Atm c) = t
134 subsHead t (App x y) = App (subsHead t x) y
135 subsHead t g = g
136
137 args :: Term -> [Term]
138 args (App x y) = args x ++ [y]
139 args _ = []
140
141 paramsWithCase :: Term -> [Bool]
142 paramsWithCase (Abs w b) = (caseOn w b):(paramsWithCase b)
143 paramsWithCase _ = []
144
145 caseOn :: Var -> Term -> Bool
146 caseOn x (App y b) = caseOn x y || caseOn x b
147 caseOn x (Abs y b) = x/=y && caseOn x b
148 caseOn x (Cas y b) = (Atm (Var x))==y || any ((caseOn x) . snd ) b
149 caseOn _ _ = False
150
151
152 {- ## CONSTANT DEFINITIONS ## -}
153
154 applyDef :: Term -> [(Var, Term)] -> Maybe Term
155 applyDef (Atm (Var w)) xs = lookup w xs
156 applyDef _ _ = Nothing
157
158 {- ## LAZY APPLY ## -}
159
160 lazyApply :: (Term -> Maybe Term) -> Term -> Maybe Term
161 lazyApply f (Atm x) = f (Atm x)
162 lazyApply f (App x y) = case f (App x y) of {
163     Just a -> Just a;
164     Nothing -> fmap (flip App y) (lazyApply f x)
165 }
166 lazyApply f (Abs w y) = f (Abs w y)
167 lazyApply f (Cas x xs) = case f (Cas x xs) of {
168     Just a -> Just a;

```

```

169             Nothing -> fmap (flip Cas xs) (lazyApply f x)
170         }
171
172 altura :: Term -> Int
173 altura (Atm _) = 0
174 altura (App x y) = 1 + (max (altura x) (altura y))
175 altura (Abs w x) = 1 + (altura x)
176 altura (Cas x xs) = 1 + ((foldl1 max) $ map altura (x:(snds xs)))
177
178 alturas :: Term -> [Int]
179 alturas (Atm _) = [0]
180 alturas (App x y) = map (+1) $ concat [alturas x, alturas y]
181 alturas (Abs w x) = map (+1) $ alturas x
182 alturas (Cas x xs) = map (+1) $ concatMap alturas (x:(snds xs))
183
184 tamaño :: Term -> Int
185 tamaño (Atm _) = 0
186 tamaño (App x y) = 1 + (tamaño x) + (tamaño y)
187 tamaño (Abs w x) = 1 + (tamaño x)
188 tamaño (Cas x xs) = 1 + (sum $ map tamaño (x:(snds xs)))
189
190 contiene :: Atm -> Term -> Bool
191 contiene a (Atm x) = a == x
192 contiene a (App x y) = (contiene a x) || (contiene a y)
193 contiene a (Abs w x) = (contiene a x)
194 contiene a (Cas x xs) = (contiene a x) || any ((contiene a).snd) xs
195
196 atmsNotInTerm :: [Atm] -> Term -> [Atm]
197 atmsNotInTerm as (Atm e) = delete e as
198 atmsNotInTerm as (App x y) = intersect (atmsNotInTerm as x) (
    atmsNotInTerm as y)
199 atmsNotInTerm as (Abs w x) = (atmsNotInTerm as x)
200 atmsNotInTerm as (Cas x xs) = foldl1 intersect $ (atmsNotInTerm as x):(
    map ((atmsNotInTerm as) . snd) xs)
201
202
203 termToAtm :: Term -> [Atm]
204 termToAtm (App x y) = (termToAtm x) ++ (termToAtm y)
205 termToAtm (Abs w x) = (termToAtm x)
206 termToAtm (Cas x xs) = (concatMap termToAtm (x:(snds xs)))
207 termToAtm (Atm e) = [e]
208
209 termToAtmCas :: Term -> [Atm]
210 termToAtmCas (App x y) = (termToAtmCas x) ++ (termToAtmCas y)
211 termToAtmCas (Abs w x) = (termToAtmCas x)
212 termToAtmCas (Cas x xs) = (map (Con . fst . fst) xs) ++ (concatMap
    termToAtmCas (x:(snds xs)))
213 termToAtmCas (Atm e) = [e]
214
215 termToFreeAtm :: Term -> [Atm]
216 termToFreeAtm (App x y) = (termToFreeAtm x) ++ (termToFreeAtm y)
217 termToFreeAtm (Abs w x) = filter (/=(Var w)) (termToFreeAtm x)
218 termToFreeAtm (Cas x xs) = (termToFreeAtm x) ++ concatMap (\((c,vs),t)
    -> filter (not .('elem'(map (Var) vs))) (termToFreeAtm t)) xs

```

```

219 termToFreeAtm (Atm e) = [e]
220
221 termToFreeAtmCas :: Term -> [Atm]
222 termToFreeAtmCas (App x y) = (termToFreeAtmCas x) ++ (termToFreeAtmCas y
    )
223 termToFreeAtmCas (Abs w x) = filter (/=(Var w)) (termToFreeAtmCas x)
224 termToFreeAtmCas (Cas x xs) = (termToFreeAtmCas x) ++ concatMap \( (c,vs
    ),t) -> [Con c] ++ filter (not .('elem'(map (Var) vs))) (
    termToFreeAtmCas t)) xs
225 termToFreeAtmCas (Atm e) = [e]
226
227 termToAtmArgs :: Term -> [Atm]
228 termToAtmArgs (App (Atm _) y) = (termToAtmArgs y)
229 termToAtmArgs (App x y) = (termToAtmArgs x) ++ (termToAtmArgs y)
230 termToAtmArgs (Abs w x) = (termToAtmArgs x)
231 termToAtmArgs (Cas x xs) = (concatMap termToAtmArgs (x:(snds xs)))
232 termToAtmArgs (Atm e) = [e]

```

A.2 Compute.hs

```

1 module Compute where
2 import Syntax
3 import Util
4
5 import Data.Maybe
6 import Data.List
7 import Control.Monad
8
9
10 type Env = [(Atm, Term)]
11
12 type ComputeRule = Term -> Maybe Term
13
14 cDef :: Env -> ComputeRule
15 cDef env (Atm v) = lookup v env
16 cDef _ _ = Nothing
17
18 cApp :: ComputeRule
19 cApp (App (Abs x b) f) = Just $ multisubstitute [(x, f)] b;
20 cApp _ = Nothing
21
22 cCas :: ComputeRule
23 cCas (Cas e cs) = case listParams e of {
24     (Atm (Con c), hs) -> case listToMaybe (filter ((==
25         c) . fst . fst) cs) of{
26         Nothing -> error ("Case mal
27             formado: "++c++" | " ++ (show (
28                 Cas e cs)));
29         Just ((_,vs), g) -> Just $
30             multisubstitute (zip vs hs)
31             g;
32     };
33     _ -> Nothing;}
34 cCas _ = Nothing

```

```

30
31 cLeftAppCas :: ComputeRule -> ComputeRule
32 cLeftAppCas cr x = case cr x of {
33     Just a -> Just a;
34     Nothing -> case x of {
35         App e f -> case cLeftAppCas cr e of {
36             Nothing -> Nothing;
37             Just o -> Just $ App o f };
38         Cas e cs -> case cLeftAppCas cr e of {
39             Nothing -> Nothing;
40             Just h -> Just $ Cas h cs };
41         _ -> Nothing; }; }
42
43 cRightAppAbs :: ComputeRule -> ComputeRule
44 cRightAppAbs cr x = case cr x of {
45     Just a -> Just a;
46     Nothing -> case x of {
47         App e f -> case cRightAppAbs cr f of
48             {
49             Nothing -> Nothing;
50             Just o -> Just $ App e
51                 o };
52         Abs y b -> case cRightAppAbs cr b of
53             {
54             Just o -> Just $ Abs y
55                 o;
56             Nothing -> Nothing;
57             };
58         _ -> Nothing; }; }
59
60 cWide :: ComputeRule -> ComputeRule
61 cWide cr x = listToMaybe $ cWideL cr x
62
63 cWideL :: ComputeRule -> Term -> [Term]
64 cWideL cr x = (maybeToList $ cr x) ++ case x of {
65     App e f -> map (\(j:k:[]) -> App j k) $
66         tail $ hiliar [e:(cWideL cr e), f:(
67             cWideL cr f)];
68     Cas e cs -> map (\h -> Cas h cs) $ cWideL
69         cr e;
70     Abs y b -> map (Abs y) $ cWideL cr b ;
71     _ -> []; }
72
73 cBottomUp :: ComputeRule -> ComputeRule
74 cBottomUp cr (Atm x) = cr (Atm x)
75 cBottomUp cr (App x y) = listToMaybe $ anyJust [cBottomUp cr x >>= (
76     return . (flip App y)), cBottomUp cr y >>= (return . (App x)), cr (
77     App x y)]
78 cBottomUp cr x = error "se esperaban sólo aplicaciones"
79
80 cIf :: (Term -> Term -> Bool) -> ComputeRule -> ComputeRule
81 cIf p cr x = do y <- cr x
82     if p x y then return y else Nothing
83
84

```

```

75 cLazy :: Env -> ComputeRule
76 cLazy env e = firstJust [cDef env e, cApp e, cCas e]
77
78 cLeftLazy :: Env -> ComputeRule
79 cLeftLazy env = cLeftAppCas $ cLazy env
80
81 cWideLazy :: Env -> ComputeRule
82 cWideLazy env = cWide $ cLazy env
83
84 cRepeatButDontEndInCas :: ComputeRule -> ComputeRule
85 cRepeatButDontEndInCas f e = find (not . hasCas) $ reverse $ tail $
    repetirL f e
86
87 hasCas :: Term -> Bool
88 hasCas (Atm c) = False
89 hasCas (App e f) = (hasCas e) || (hasCas f)
90 hasCas (Abs y b) = hasCas b
91 hasCas (Cas e cs) = True
92
93 isJammedOnCase = isJust . jammedCase
94
95 jammedCase :: Term -> Maybe (Var, [(Con, [Var])])
96 jammedCase (Atm v) = Nothing
97 jammedCase (App e f) = jammedCase e
98 jammedCase (Abs y b) = Nothing
99 jammedCase (Cas (Atm (Var x)) cs) = Just (x, fsts cs)
100 jammedCase (Cas e cs) = jammedCase e
101
102 cNotCase :: Env -> ComputeRule
103 cNotCase env e = do (h, p) <- getHead e
104     x <- lookup h env
105     ls <- allJust $ map getHead $ fsts $ filter (snd) $
        zip p (paramsWithCase x)
106     a <- (repetir (\ u -> firstJust [cWide cApp u, cWide
        cCas u] )) (subsHead x e)
107     if hasCas a then Nothing else return a
108     where
109         isCon a = case a of { Con _ -> True; _ -> False}

```

A.3 Type.hs

```

1 module Type where
2 import Syntax
3 import Util
4 import Data.Maybe
5 import Data.List
6
7 {- ## TYPES ## -}
8 data Type = TAtm TAtm | TBin TBin Type Type
9     deriving (Eq, Ord, Show)
10
11 data TAtm = TVar TVar | TCon TCon
12     deriving (Eq, Ord, Show)
13

```

```

14 data MType = MType [TVar] Type
15
16 type TVar = String -- Variables
17 type TCon = String -- Variables
18
19 data TBin = TCom | TApp
20     deriving (Eq, Ord, Show)
21
22 {- ## SHOW ## -}
23
24 {- ## EQUALITY ## -}
25 {- ## ALPHA RULE ## -}
26
27 instance Eq MType where
28     x == y = (normalForm x) == (normalForm y)
29
30 extractTVars :: Type -> [TVar]
31 extractTVars (TAtm (TVar x)) = [x]
32 extractTVars (TAtm (TCon x)) = []
33 extractTVars (TBin b x y) = nub (concatMap extractTVars [x,y])
34
35 normalForm :: MType -> MType
36 normalForm (MType vs t) = MType newvars (multiSubs subst t)
37     where extractLigatedVars = filter ('elem' vs) $
38           extractTVars t
39           subst = zip extractLigatedVars (map (TAtm . TVar)
40                 infiniteTVars)
41           newvars = sort $ fsts subst
42
43 multiSubs :: [(TVar, Type)] -> Type -> Type
44 multiSubs xs (TAtm (TVar x)) = case lookup x xs of { Just y -> y;
45     Nothing -> TAtm $ TVar x}
46 multiSubs xs (TAtm (TCon x)) = (TAtm (TCon x))
47 multiSubs xs (TBin b x y) = TBin b (multiSubs xs x) (multiSubs xs y)
48
49 infiniteTVars :: [TVar]
50 infiniteTVars = ["a", "b", "c", "d", "e"] ++ map (show) [0..]
51
52 inferirConDadasC :: DadasC -> Term -> Maybe (Type, [(Atm, Type)])
53 inferirConDadasC datas x =
54     case inferir x of {
55         Nothing -> Nothing;
56         Just (tipo,contexto) ->
57             let sustHig = renombrar2 (snds datas) (tipo:(snds contexto
58                 ))
59                 tipoh = multiSubs sustHig tipo
60                 contextoh = mapSnd (multiSubs sustHig) contexto
61                 cs = (mapFst Con datas) ++ (contextoh)
62                 eqs = contextToEquations cs
63             in case unificar1 eqs of{
64                 Nothing -> Nothing;
65                 Just sust -> Just (multiSubs sust tipoh,
66                     filter (\(a,b) -> not (a 'elem'
67                         (map (Con . fst) datas))) $

```

```

63                                     --cosmetic filter
64                                     mapSnd (multiSubs sust)
65                                     contextoh);
66     };
67 }
68 inferir :: Term -> Maybe (Type, [(Atm, Type)])
69 inferir (Atm x) = Just ((TAtm . TVar) "t0", [(x, (TAtm . TVar) "t0")])
70 inferir (Abs w b) = case inferir b of {
71     Nothing -> Nothing;
72     Just (t, c) -> case lookup (Var w) c of{
73         Just s -> Just (TBin TCom s t,
74             filter ((/= (Var w)) . fst) c);
75         Nothing -> Just (TBin TCom ((TAtm .
76             TVar) (mFresh (t: (map (snd) c)
77             ))) t, c) ;
78     };
79 }
80 inferir (App x y) = case (inferir x, inferir y) of {
81     (Just (t0,c0), Just (t1,c1)) ->
82     let sustHig = renombrar2 (t0:(snds c0)) (t1
83         :(snds c1))
84         t1h = multiSubs sustHig t1
85         c1h = mapSnd (multiSubs sustHig) c1
86         t0h = t0
87         c0h = c0
88         cs = c0h ++ c1h
89         eqs = contextToEquations cs
90         fvars = mFreshs (t0h:t1h:(snds cs))
91         ti = (TAtm . TVar) (fvars!!0)
92         td = (TAtm . TVar) (fvars!!1)
93         neqs = ((t0h, TBin TCom ti td):(ti, t1h)
94             :eqs)
95     in case unificar1 neqs of{
96         Nothing -> Nothing;
97         Just sustUni -> Just (
98             multiSubs sustUni td,
99             mapSnd (multiSubs sustUni
100             ) cs);
101     };
102     _ -> Nothing;
103 }
104 renombrar2 :: [Type] -> [Type] -> [(TVar,Type)]
105 renombrar2 xs ys = zip varsYs (map (TAtm . TVar) (mFreshs (xs++ys)))
106     where varsYs = nub (concatMap extractTVars ys)
107
108
109
110
111
112 contextToEquations :: Eq a => [(a, b)] -> [(b,b)]
113 contextToEquations [] = []
114 contextToEquations ((v,t):xs) = case lookup v xs of {
115     Nothing -> contextToEquations xs;
116     Just y -> (t,y):(contextToEquations xs);

```

```

108                                     }
109
110
111
112 mFresh :: [Type] -> TVar
113 mFresh xs = head $ mFreshs xs
114
115 mFreshs :: [Type] -> [TVar]
116 mFreshs xs = filter (not . (flip elem (concatMap extractTVars xs)))
    infiniteTVars
117
118
119 singleVar :: Type -> Maybe TVar
120 singleVar (TAtm (TVar x)) = Just x
121 singleVar _ = Nothing
122
123
124 reps :: Eq a => [a] -> [a]
125 reps [] = []
126 reps (x:xs) = case x `elem` xs of {
127     False -> repsxs;
128     True -> x:repsxs;
129 }
130     where repsxs = reps xs
131
132
133 unificar1 :: [(Type,Type)] -> Maybe [(TVar, Type)]
134 unificar1 xs = (whileMaybes [ descomp1s, despejarD] xs) >>=
    equationsASubs
135
136 equationsASubs :: [(Type,Type)] -> Maybe [(TVar, Type)]
137 equationsASubs [] = Just []
138 equationsASubs ((TAtm (TVar x), t):xs) = (equationsASubs xs) >>= (Just .
    ((x,t):))
139 equationsASubs ((t, TAtm (TVar x)):xs) = (equationsASubs xs) >>= (Just .
    ((x,t):))
140 equationsASubs _ = error "unificar mal"
141
142
143 descomp1 :: (Type, Type) -> Maybe (Bool, [(Type, Type)])
144 descomp1 (TBin j x y, TBin k a b) = if j==k then Just (True, [(x,a),(y,b)
    ]) else Nothing
145 descomp1 (TAtm (TVar x), TAtm (TVar y)) = if x == y then Just (True, [])
    else Just (False, [(TAtm (TVar x), TAtm (TVar y))])
146 descomp1 (TAtm (TVar x), y) = if x `elem` (extractTVars y) then Nothing
    else Just (False, [(TAtm (TVar x), y)])
147 descomp1 (y, TAtm (TVar x)) = if x `elem` (extractTVars y) then Nothing
    else Just (False, [(TAtm (TVar x), y)])
148 descomp1 (x, y) = if x == y then Just (True, []) else Nothing
149
150 descomp1s :: [(Type, Type)] -> Maybe (Bool, [(Type, Type)])
151 descomp1s xs = case allJust (map descomp1 xs) of{
152     Nothing -> Nothing;
153     Just as -> case any fst as of {

```



```

154                                     False -> Just (False, xs);
155                                     True  -> Just (True, (concat . snds) as)
                                           ;
156                                     };
157                                     }
158
159 filterSingleVars :: [(Type,Type)] -> [(TVar, Type)]
160 filterSingleVars ((TAtm (TVar x), a):xs) = (x,a):(filterSingleVars xs)
161 filterSingleVars ((a, TAtm (TVar x)):xs) = (x,a):(filterSingleVars xs)
162 filterSingleVars ((a, b):xs) = (filterSingleVars xs)
163 filterSingleVars [] = []
164
165 despejarD :: [(Type, Type)] -> Maybe (Bool, [(Type, Type)])
166 despejarD xs = case singlevarsrep of {
167     ((x,t):_) -> case x `elem` (extractTVars t) of {
168         True -> Nothing;
169         False -> Just (True, (TAtm (TVar x), t):(
170             map (\(a,b) -> (multiSubs [(x, t)] a,
171                 multiSubs [(x, t)] b)) xs));
172     };
173     [] -> Just (False, xs)
174 };
175 where singlevars = filterSingleVars xs
176       allvars = concatMap (\(a,b)-> extractTVars a ++
177           extractTVars b) xs
178       allvarsrep = reps allvars
179       singlevarsrep = filter ((flip elem allvarsrep) .
180           fst) singlevars
181
182
183 -- ENUMERAR
184 type DatasH = (TVar, [(Con, [Type])]) -- estilo haskell abreviado
185 type DatasC = [(Con, Type)] -- estilo expandido
186
187 datasH2C :: DatasH -> DatasC
188 datasH2C (t, xs) = mapSnd ((foldr (TBin TCom) (TAtm (TCon t)))) xs
189
190 datasH2rec :: DatasH -> DatasC
191 datasH2rec (t, xs) = [(t ++ "prim",
192     foldr ((TBin TCom) . (foldr (TBin TCom) (TAtm (TVar
193         varnueva)))) (TAtm (TVar varnueva)) (snds xs
194         )]
195     where varnueva = (mFresh . snds . datasH2C) (t, xs)
196
197 primitiveRec :: DatasH -> Term
198 primitiveRec (n, xs) = undefined
199
200

```

```

201
202 allPossibleEquations :: Eq a => [(a, b)] -> [(b,b)]
203 allPossibleEquations [] = []
204 allPossibleEquations ((v,t):xs) = map (\(w,s) -> (t,s)) (filter ((==v) .
      fst ) xs)
205                                     ++ allPossibleEquations xs
206
207
208 returnsType :: Type -> Type -> Bool
209 returnsType t (TAtm x) = False
210 returnsType t (TBin TCom x y) = y == t || returnsType t y
211
212 returnsTypeM :: Type -> Type -> Maybe [Type]
213 returnsTypeM t o | t == o = Just []
214                   | otherwise = case o of {
215                       (TBin TCom x y) -> case returnsTypeM t y of
216                           {
217                               Nothing -> Nothing;
218                               Just as -> Just (x:as);
219                           };
219                       _ -> Nothing;
220                   }
221
222 parsAndRet :: Type -> ([Type], Type)
223 parsAndRet (TAtm x) = ([], TAtm x)
224 parsAndRet (TBin TCom x (TAtm y)) = ([x], TAtm y)
225 parsAndRet (TBin TCom x ys) = case parsAndRet ys of {
226     (pars, ret) -> (x:pars, ret);
227 }
228
229
230
231
232 cantApps :: Term -> Int
233 cantApps (Atm _) = 0
234 cantApps (App x y) = 1 + (cantApps x) + (cantApps y)
235
236
237 lastRet :: Type -> Type
238 lastRet (TAtm x) = TAtm x
239 lastRet (TBin TCom x y) = lastRet y
240
241 cantComs :: Type -> Int
242 cantComs (TAtm x) = 0
243 cantComs (TBin TCom x y) = 1 + cantComs y
244
245 type TEnv = [(Term, Type)]

```

A.4 Prove.hs

```

1 module Prove where
2 import Syntax
3 import Compute
4 import Type

```

```

5 import Util
6 import Data.Maybe
7 import Data.List
8 import Debug.Trace
9 import Control.Monad
10 import Data.Ord
11
12 data Affirm = Affirm [(Var, Type)] Term Term
13     deriving (Ord, Show)
14
15
16
17 instance Eq Affirm where
18     (Affirm vs1 y1 z1) == (Affirm vs2 y2 z2) = y1==y2 && z1==z2 && (sort
19         vs1 == sort vs2)
20
21 type Imp a = ([a], a)
22
23 type Tactic a = Imp a -> Maybe [(Comment, Imp a)]
24
25
26 fund :: Env -> TEnv -> Term -> Int
27 fund env tenv (Atm (Var x)) = case lookup (Var x) env of { Nothing -> 0;
28     Just y -> fund (filter ((/=Var x).fst) env) tenv y}
29 fund env tenv (Atm (Con x)) = 1
30 fund env tenv (App x y) = if fx > fy then fx + fy else fy
31     where (fx, fy) = (fund env tenv x, fund env tenv y)
32
33 fund env tenv (Abs x b) = 1 + (fund env tenv b)
34 fund env tenv (Cas t cts) = (length . concat . snds .fst) cts + ( (
35     foldl1 max) $ (map (fund env tenv)) $ (t:(snds cts)))
36
37
38 substA :: Affirm -> Var -> Term -> Affirm
39 substA (Affirm vs e f) v g = Affirm vs (substitute v g e) (substitute v
40     g f)
41
42 compareDep :: Env -> Affirm -> Maybe Ordering
43 compareDep env (Affirm _ b c) = case (all ('elem' bd) cd, all ('elem' cd
44     ) bd) of{
45     (True, True) -> Just EQ;
46     (True, False) -> Just GT;
47     (False, True) -> Just LT;
48     (False, False) -> Nothing;
49     } where bd = dependencies env b;
50     cd = dependencies env c;
51
52 compareDepInt :: Env -> Affirm -> Maybe Int
53 compareDepInt env (Affirm _ b c) = case (count ('elem' bd) cd, count ('
54     elem' cd) bd) of{
55     (n, m) -> if (n /= length bd) || (m
56     /= length cd)
57     then Nothing

```

```

51                                     else Just (n-m);
52                                     } where bd = dependencies env b;
53                                     cd = dependencies env c;
54                                     count p = length . (filter p)
55
56 simplifies :: Env -> Affirm -> Bool
57 simplifies env a = (Just GT == compareDep env a)
58
59 simplifiesOrd :: Env -> Affirm -> Affirm -> Ordering
60 simplifiesOrd env (Affirm a b c) (Affirm x y z) = compare (yd - zd) (bd
    - cd)
61
62                                     where bd = length $ dependencies env b; yd
    = length $ dependencies env y;
63                                     cd = length $ dependencies env c; zd
    = length $ dependencies env z;
64
64 floatsDep :: Env -> Affirm -> Bool
65 floatsDep env (Affirm a b c) = (all ('elem' bd) cd) && ((==[GT]) $ take
    1 $ filter (/=EQ) $ zipWith compare bo co)
66
67                                     where bs = termToAtm b; bd = dependencies
    env b; bo = map (length .(
68                                     dependenciesAtm env)) bs
69                                     cs = termToAtm c; cd = dependencies
    env c; co = map (length .(
70                                     dependenciesAtm env)) cs
71
68 soloFloatsDep :: Env -> Affirm -> Bool
69 soloFloatsDep env (Affirm a b c) = (sort bd == sort cd) && ((==[GT]) $
    take 1 $ filter (/=EQ) $ zipWith compare bo co)
70
71                                     where bs = termToAtm b; bd = dependencies
    env b; bo = map (length .(
72                                     dependenciesAtm env)) bs
73                                     cs = termToAtm c; cd = dependencies
    env c; co = map (length .(
74                                     dependenciesAtm env)) cs
75
74 floatsDepOrd :: Env -> Affirm -> Affirm -> Ordering
75 floatsDepOrd env (Affirm a b c) (Affirm x y z) = case filter (/=EQ) $
    zipWith compare (zipWith compare bo co) (zipWith compare yo zo) of {
76                                     [] -> EQ;
77                                     (x:_) -> x}
78
79                                     where bs = termToAtm b; bo = map (length
    .(dependenciesAtm env)) bs
80                                     cs = termToAtm c; co = map (length .(
    dependenciesAtm env)) cs
81                                     ys = termToAtm y; yo = map (length .(
    dependenciesAtm env)) ys
82                                     zs = termToAtm z; zo = map (length .(
    dependenciesAtm env)) zs
83
82 floatsEnv :: Env -> Affirm -> Bool
83 floatsEnv env (Affirm a b c) = (==[GT]) $ take 1 $ filter (/=EQ) $
    zipWith (ordEnv env) bs cs
84
85                                     where bs = termToAtm b;

```

```

86                                     cs = termToAtm c;
87
88 floatsEnvOrd :: Env -> Affirm -> Affirm -> Ordering
89 floatsEnvOrd env (Affirm a b c) (Affirm x y z) = undefined
90
91 ordEnv :: Env -> Atm -> Atm -> Ordering
92 ordEnv env x y = case (elemIndex x env' , elemIndex y env' ) of {
93     (Nothing, Nothing) -> EQ;
94     (Just _, Nothing) -> GT;
95     (Nothing, Just _) -> LT;
96     (Just x, Just y) -> compare x y;
97     } where env' = fst$ env;
98
99
100 countDiff :: (Eq a) => [a] -> Int
101 countDiff xs = length (nub xs)
102
103 dependencies :: Env -> Term -> [Atm]
104 dependencies env t = nub $ concatMap (dependenciesAtm env) (nub $
    termToFreeAtmCas t)
105
106 dependenciesAtm :: Env -> Atm -> [Atm]
107 dependenciesAtm env t = (t:)$ case lookup t env of {
108     Nothing -> [];
109     Just k-> nub $ ((concatMap (dependenciesAtm
    env)) . filter (/= t) . termToFreeAtmCas) k
    ;
110     }
111
112 idependencies :: Env -> Term -> [Atm]
113 idependencies env t = nub $ concatMap (idependenciesAtm env) (nub $
    termToFreeAtm t)
114
115 idependenciesAtm :: Env -> Atm -> [Atm]
116 idependenciesAtm env t = case lookup t env of {
117     Nothing -> [];
118     Just k-> nub $ (filter (/= t) . termToFreeAtm
    ) k;
119     }
120
121 data Reason = Simp | Floe | Flod | Reor | Basu
122     deriving (Eq,Show)
123
124 data SideComment = Compute | Lemma Reason Affirm
125     deriving (Eq,Show)
126
127 data Comment = NoIdea
128     | Triv
129     | Loop
130     | LeftComment SideComment | RightComment SideComment |
    BothComment SideComment SideComment
131     | Induction Var
132     | Caso Var
133     deriving (Eq,Show)

```

```

134
135
136 data Proof = PFail | PDone | PStep Affirm Comment Proof | PInd Affirm
      Comment [(Comment, ([Affirm], Proof))]
137     deriving (Eq, Show)
138
139 containsIndc :: Proof -> Bool
140 containsIndc (PInd _ _ _) = True
141 containsIndc _ = False
142
143 mapProof :: (Affirm -> Affirm) -> Proof -> Proof
144 mapProof f (PStep a c ps) = PStep (f a) c (mapProof f ps)
145 mapProof f (PInd a c ps) = PInd (f a) c [ (c, (map f as, mapProof f qs))
      | (c, (as, qs)) <- ps]
146 mapProof f x = x
147
148
149 data Rec = Rec
150     { simp :: [Affirm]
151     , flod :: [Affirm]
152     , floe :: [Affirm]
153     , reor :: [Affirm]
154     , basu :: [Affirm]
155     , todo :: [Affirm]
156     } deriving (Eq, Show)
157
158 emptyRec = Rec [] [] [] [] [] []
159
160 mainTactic :: [DatasH] -> Env -> Rec -> Affirm -> Proof
161 mainTactic ds env rec conj =
162     case triv conj of {
163         True -> PStep conj Triv PDone;
164         False -> case bothSides (computeDontEndInCas env) conj of {
165             Just (l,a) -> PStep conj l (mainTactic ds env rec a);
166             Nothing -> case applyRec env rec conj of {
167                 (Just (l, nconj)) -> PStep conj l (mainTactic ds
      env rec nconj);
168                 Nothing -> case indcFst ds env conj of {
169                     Just (com, xs) -> PInd conj com (map
      (\ (c, (hs, t)) -> (c, (hs,
      mainTactic ds env (addAffirms
      env rec hs) t) )) xs);
170                     Nothing -> PStep conj NoIdea PFail;
171                 }
172             };
173     }
174 }
175
176 applyRec :: Env -> Rec -> Affirm -> Maybe (Comment, Affirm)
177 applyRec env rec conj = firstJust
178     [ (bothSides (\vs t -> addReason Simp (
      applyLemmasC cWide (simp rec) vs t))
      conj)

```

```

179         , (bothSides (\vs t -> addReason Flod (
180             applyLemmasC cWide (flod rec) vs t))
            conj)
181         , (bothSides (\vs t -> addReason Floe (
182             applyLemmasC cWide (floe rec) vs t))
            conj)
183         , (bothSides (\vs t -> addReason Reor (
184             applyLemmasC (cBottomUp . cIf (\ x y ->
185                 GT == masOrd x y)) (reor rec) vs t))
            conj)
186     ]
187
188 applyRecT :: Env -> Rec -> Term -> Maybe Term
189 applyRecT env rec t = fmap (snd) $ firstJust [ applyLemmasC cWide (simp
190     rec) [] t
191     , applyLemmasC cWide (flod rec)
192     [] t
193     , applyLemmasC cWide (floe rec)
194     [] t
195     , applyLemmasC (cBottomUp . cIf
196     (\ x y -> GT == masOrd x y)
197     ) (reor rec) [] t
198     ]
199
200 triv (Affirm _ y z) = y == z
201
202 applyRecNC :: Env -> Rec -> Affirm -> Maybe (Affirm)
203 applyRecNC env rec conj = (applyRec env rec conj) >>= (Just . snd)
204
205 bothSides :: ([Var] -> Term -> Maybe (SideComment, Term)) -> Affirm ->
206     Maybe (Comment, Affirm)
207 bothSides f (Affirm vs y z) = case (f (fst vs) y, f (fst vs) z) of {
208     (Just (c,d), Just (k,j)) -> Just (
209         BothComment c k, Affirm vs d j);
210     (Nothing, Just (k,j)) -> Just (
211         RightComment k, Affirm vs y j);
212     (Just (c,d), Nothing) -> Just (LeftComment
213         c, Affirm vs d z);
214     (Nothing, Nothing) -> Nothing;
215     }
216
217 computeDontEndInCas :: Env -> ([Var] -> Term -> Maybe (SideComment, Term
218 ))
219 computeDontEndInCas env vs y = case cWide (cNotCase env) y of {
220     Just h -> Just (Compute, h);
221     Nothing -> Nothing;
222     }
223
224 applyLemmas :: [Affirm] -> [Var] -> Term -> Maybe (Affirm, Term)
225 applyLemmas ls vs e = enPos 0 $ anyJust $ map (\ u -> case aWideLemma u
226     vs e of {Just w -> Just (u, w); Nothing -> Nothing;}) ls

```

```

215
216 applyLemmasC :: (ComputeRule -> ComputeRule) -> [Affirm] -> [Var] ->
    Term -> Maybe (Affirm, Term)
217 applyLemmasC cr ls vs e = enPos 0 $ anyJust $ map (\ u -> case cr (
    aLemma u vs) e of {Just w -> Just (u, w); Nothing -> Nothing;}) ls
218
219 addReason :: Reason -> Maybe (Affirm, Term) -> Maybe (SideComment, Term)
220 addReason r Nothing = Nothing
221 addReason r (Just (l, t)) = Just (Lemma r l, t)
222
223 aLemma :: Affirm -> [Var] -> Term -> Maybe Term
224 aLemma l@(Affirm vs f1 f2) ws e = case instanciar (fst vs) f1 ws e of{
225     Just a -> Just $
        multisubstitute a f2 ;
226     Nothing -> Nothing; }
227
228 aWideLemma l ws = cWide (aLemma l ws)
229 aWideLemmaL l ws = cWideL (aLemma l ws)
230
231 instanciar :: [Var] -> Term -> [Var] -> Term -> Maybe [(Var, Term)]
232 instanciar vs e@(Atm (Var v)) ws f = case (v 'elem' vs, v 'elem' ws) of {
233     (True, _) -> Just [(v, f)];
234     (False, False) -> if e == f then
        Just [] else Nothing;
235     (False, True) -> Nothing;
236     }
237
238 instanciar vs e@(Atm (Con c)) ws f@(Atm (Con k)) = if c == k then Just
    [] else Nothing
239 instanciar vs e@(App a b) ws f@(App c d) = case allJust [instanciar vs a
    ws c, instanciar vs b ws d] of {
240     Just xs -> if isFunction (
        concat xs) then Just (
        concat xs) else Nothing;
241     Nothing -> Nothing;
242     }
243 instanciar _ _ _ _ = Nothing
244
245
246 indc :: [DatsH] -> Env -> (Var, Type) -> Affirm -> Maybe (Comment, [(
    Comment, ([Affirm], Affirm))])
247 indc ds env (v,t) e@(Affirm vs y z) = case elem (v,t) vs of {
248     False -> Nothing;
249     True -> case lookup t (mapFst (TAtm .
        TCon) ds) of {
250     Nothing -> Nothing;
251     Just a -> Just $ (Induction v
        , map (\x->(Caso v ,x))
        {- $ map (appFst(++xs))
        -})
252     $ instanciarPpio v e $
        armarPpio $ nombrarPar (
        env') $ marcarHip (

```



```

253                                     extractCon t, a));};
}where env' = (hexpsA e) ++ {- (concat (
    map hexpsA xs)) ++ -} (map (Atm .
254     fst) env) ++ (map (snd) env)
    hexpsA (Affirm vs y z) = (map ((
        Atm . Var) . fst) vs) ++ [y,z
    ]
255     extractCon (TAtm (TCon x)) = x;
256
257
258 indcFst :: [Datash] -> Env -> Affirm -> Maybe (Comment, [(Comment, ([
    Affirm], Affirm))])
259 indcFst ds env e@(Affirm vs y z) = case vs of {
260     [] -> Nothing;
261     (v:_) -> indc ds env v e;
262     }
263
264 nombrarPar :: [Term] -> [(Con, [Bool])] -> [(Con, [(Var, Bool)])]
265 nombrarPar env cs = map (\(c, bs) -> (c, zip (freshs (env)) bs)) cs
266
267 armarPpio :: [(Con, [(Var, Bool)])] -> [(Term, Term)]
268 armarPpio ppio = map (\(c,vs) -> ((map ((Atm . Var) .fst)) $ filter snd
    vs, appParams (Atm (Con c), (map ((Atm . Var) . fst) vs)))) ppio
269
270 instanciarPpio :: Var -> Affirm -> [(Term, Term)] -> [(Affirm,
    Affirm)]
271 instanciarPpio v (Affirm vs y z) ppio = map (\(hs,t) -> (map (substA e'
    v) hs, substA e' v t)) ppio
272     where vs' = filter ((/=v).fst) vs
273     e' = Affirm vs' y z
274
275 marcarHip :: Datash -> [(Con, [Bool])]
276 marcarHip (t, cvs) = mapSnd (map (== TAtm (TCon t))) cvs
277
278 -- Analyze lemmas --
279
280 permuta :: Affirm -> Bool
281 permuta (Affirm ws x y) = (sort (termToAtm x)) == (sort (termToAtm y))
282
283 soloPermuta :: Affirm -> Bool
284 soloPermuta (Affirm ws x y) = (head x') == (head y') && (sort x') == (
    sort y')
285     where x' = termToAtm x; y' = termToAtm y
286
287 masOrd :: Term -> Term -> Ordering
288 masOrd t1 t2 = case filter (/=EQ) $ zipWith (compare) (termToAtmArgs t1)
    (termToAtmArgs t2) of {
289     [] -> case take 1 $ filter (/=EQ) $ zipWith (compare)
    (alturas t1) (alturas t2) of {
290         [] -> EQ;
291         (x:xs) -> x;
292     };
293     (LT:_) -> LT;
294     (GT:_) -> GT;;

```

```

295         }
296
297 flipAffirm (Affirm vs y z) = (Affirm vs z y)
298
299 data FlowAffirm = Swap | Float | Reord
300     deriving (Eq, Ord, Show)
301
302 toJera :: Affirm -> Maybe FlowAffirm
303 toJera (Affirm vs x y) | (null sxs) && (null sys) && (head xs == head ys
    ) = Just Reord
304     | (null sxs) && (null sys) && (head xs /= head ys) =
        Just Float
305     | not (null sxs) || not (null sys) = Just Swap
306     | otherwise = Nothing
307     where xs = nub $ termToAtm x;
308           ys = nub $ termToAtm y;
309           int = intersect xs ys;
310           sxs = xs \\ int;
311           sys = ys \\ int;
312
313 toJeraAtm :: Affirm -> [(Atm, (Int, Int, Int))]
314 toJeraAtm a@(Affirm vs x y) = case toJera a of{
315     (Just Swap) -> map (\x -> (x,(1,0,0))) (
        nub ((termToAtm x) ++ (termToAtm y)));
316     (Just Float) -> map (\x -> (x,(0,1,0))) (
        nub ((termToAtm x) ++ (termToAtm y)));
317     (Just Reord) -> map (\x -> (x,(0,0,1))) (
        nub ((termToAtm x) ++ (termToAtm y)));
318     Nothing -> [];
319     }
320
321
322
323 addAffirm :: Env -> Rec -> Affirm -> Rec
324 addAffirm env rec l@(Affirm vs a b)
325     | fa > fb = rec {simp = insertBy (compF) l (simp rec)}
326     | fa < fb = rec {simp = insertBy (compF) l' (simp rec)}
327     | permuta l      = rec {reor = insertBy compSnd l (
        insertBy compSnd l' (reor rec))}
328     | otherwise = rec {basu = l:(basu rec)}
329     where l' = flipAffirm l
330           compSnd (Affirm _ x y) (Affirm _ j k) = masOrd y k
331           compF (Affirm _ x y) (Affirm _ j k) = compare ((
        fund env [] x) - (fund env [] y)) ((fund env
        [] j) - (fund env [] k))
332           fa = fund env [] a
333           fb = fund env [] b
334
335 addAffirms :: Env -> Rec -> [Affirm] -> Rec
336 addAffirms env rec ls = foldl (addAffirm env) rec ls
337
338 countSub :: (Term -> Bool) -> Term -> Int
339 countSub p x = case x of {

```

```

340         App e f -> (if p x then 1 else 0) +(countSub p e) + (
              countSub p f);
341         Cas e cs -> (if p x then 1 else 0) +countSub p e;
342         Abs y b -> (if p x then 1 else 0) +countSub p b ;
343         _ -> 0;}
344
345 headProof :: Proof -> Affirm
346 headProof (PStep x _ _) = x
347 headProof (PInd x _ _) = x
348
349 cutSingleStepCycles :: Proof -> Proof
350 cutSingleStepCycles (PStep x c e@(PStep y k (PStep u j zs))) = if x==u
              then PStep x c (PStep y k (PStep u Loop PFail))
351                                                    else
                                                    PStep
                                                    x c
                                                    (
                                                    cutSingleStepCycles
                                                    e)
352 cutSingleStepCycles (PStep x c (PStep y k zs)) = if x==y then PStep x c
              (PStep y k PFail) else PStep x c (cutSingleStepCycles (PStep y k zs)
              )
353 cutSingleStepCycles (PStep x c a) = PStep x c (cutSingleStepCycles a)
354 cutSingleStepCycles (PInd x s xs) = PInd x s (map (\(c,(hs,t)) -> (c,(hs
              , cutSingleStepCycles t))) xs)
355 cutSingleStepCycles a = a
356
357 giveUpAfter :: Int -> Proof -> Proof
358 giveUpAfter _ PDone = PDone
359 giveUpAfter _ PFail = PFail
360 giveUpAfter _ (PInd _ _ []) = error "casos mal formados"
361 giveUpAfter n (PStep x s xs)
362     | n == 0 = PFail
363     | n >= 0 = PStep x s (giveUpAfter (n-1) xs)
364 giveUpAfter n (PInd x s xs)
365     | n == 0 = PFail
366     | n >= 0 = PInd x s (map (\(c,(hs,t)) -> (c,(hs, giveUpAfter (n'div
              '2) t))) xs)
367
368                                     where cant = length xs
369
369 giveUp :: Int -> Proof -> Maybe (Proof)
370 giveUp _ PDone = Nothing
371 giveUp _ PFail = Just PFail
372 giveUp _ (PInd _ _ []) = error "casos mal formados"
373 giveUp n (PStep x s xs) | n <= 0 = Just PFail
374                       | n > 0 = do k <- (giveUp (n-1) xs);
375                                   return (PStep x s k)
376 giveUp n (PInd x s xs) | n <= 0 = Just PFail
377                       | n > 0 = do ks <- f (n-1) xs;
378                                   return (PInd x s ks)
379
380                                     where f n [] = Just []
381                                     f n ((c, (hs,t)):xs) =
382                                     case giveUp n t of{
383                                     (Just a) -> Just [(c,(hs, a))];

```

```

383         Nothing -> case f (n - 1 - (largo t)) xs
                    of {
384             Nothing -> Nothing;
385             Just [] -> Nothing;
386             Just (a:as) -> Just ((c,
                                (hs,t)):a:as);
387         }}
388 largo :: Proof -> Int
389 largo PDone = 0
390 largo PFail = 0
391 largo (PInd x s xs) = 1 + ( sum (map (\(c,(hs,t)) -> largo t) xs) )
392 largo (PStep x _ xs) = 1 + (largo xs)
393
394 proofToList :: Proof -> [Affirm]
395 proofToList PDone = []
396 proofToList PFail = []
397 proofToList (PStep x _ xs) = x:(proofToList xs)
398 proofToList (PInd x _ xs) = (x:) $ concatMap (proofToList . snd . snd)
    xs
399
400
401 proofToHojas :: Proof -> [[Affirm], Affirm]
402 proofToHojas PDone = []
403 proofToHojas PFail = []
404 proofToHojas (PStep x _ xs) = case proofToHojas xs of {[] ->[[[]],x]; a
    -> a}
405 proofToHojas (PInd x _ xs) = ([[[]], x):) $ concat $ map (\(c, (hs, x)) ->
    map (\(hs',t) -> (hs++hs',t)) (proofToHojas x)) xs
406
407 proofCompleto :: Proof -> Bool
408 proofCompleto PDone = True
409 proofCompleto PFail = False
410 proofCompleto (PStep x _ xs) = proofCompleto xs
411 proofCompleto (PInd x _ xs) = all (proofCompleto . snd . snd) xs
412
413
414
415
416 prove :: Int -> [Datash] -> Env -> Rec -> Affirm -> Proof
417 prove n d env rec t = (try (giveUp n)) $ cutSingleStepCycles $ (
    mainTactic d env rec t)
418
419
420
421 provell :: Env -> TEnv -> [Datash] -> Int -> Rec -> [[Affirm]] -> [Proof
    ]
422 provell env tenv dats n rec (ls:lls) = case find proofCompleto $
423     map (prove n dats env rec) $
424     filter (isNothing . (applyRec env (
        rec))) ls of {
425     Nothing -> provell env tenv dats n
        rec (lls);
426     Just k -> (k:) $ provell env tenv
        dats n (addAffirms env rec [

```

```

427                                     headProof k]) (lls);
428 provell env tenv dats n rec [] = []

```

A.5 Propose.hs

```

1 module Propose where
2 import Syntax
3 import Util
4 import Type
5 import Data.List
6 import Data.Maybe
7 import Compute
8 import Prove (Affirm(Affirm), Rec, Proof, simplifies, soloPermuta,
9               masOrd, provell, idependencies, dependencies, computeDontEndInCas,
10              applyRec, applyRecT, emptyRec, fund)
9 import Debug.Trace
10 import PrettyPrint
11
12 -- Proposing data types:
13 enumSimpleFiniteTypes :: Int -> DatasH
14 enumSimpleFiniteTypes n = ("F"++(show n), [( "F"++(show n)++"."++(show m
15   ) , [] ) | m<-[1..n]] )
16
17 enumContainerFiniteTypes :: Int -> Int -> [DatasH]
18 enumContainerFiniteTypes n c = [ ("C"++(most2 d) , [( "C"++(most2 d)++"."
19   ++(most1 e), (map (TAtm . TVar) e)) | e <- d ]) | d <- dist n (take
20   c (infiniteTVars))]
21
22 enumRecursiveTypes :: Int -> Int -> [DatasH]
23 enumRecursiveTypes n r = [ ("C"++(most2 d), [( "C"++(most2 d)++"."++(show
24   $ fromJust (elemIndex e d)), map (TAtm . TVar) e | e <- d ]) | d <-
25   dist n (take r (repeat "rec"))]
26
27 enumContainerRecTypes :: Int -> Int -> Int -> [DatasH]
28 enumContainerRecTypes n c r = [ ("C"++(most2 d), [( "C"++(most2 d)++"."++
29   k, e) | (e,k) <- d''])
30   | rs <- nub $ map sort $ dist n (take r (
31     repeat "R")),
32     cs <- nub $ map sort $ dist n (take c (
33     infiniteTVars)),
34     d <- [zipWith (++) cs rs],
35     d' <- [map (map (swap (TAtm (TVar "R")) (
36       TAtm (TCon ("C"++(most2 d)))))) (map (
37       map (TAtm . TVar)) d)],
38     d'' <- [zip d' (map (show) [0..])] ]
39
40 most1 :: [String] -> String
41 most1 [] = "_"
42 most1 xs = filter (/= 't') $ concat $ (intersperse "_") xs
43
44 most2 :: [[String]] -> String
45 most2 xs = concat $ (intersperse ",") $ map (most1) xs
46

```

```

37 -- Proposing function types:
38 armarTip :: Int -> [Type] -> [Type]
39 armarTip n xs | n==0 = xs
40             | n/=0 = [ TBin TCom x t | x <- xs, t <- armarTip (n-1) xs]
41
42 armarPar :: Type -> (Term -> Term, [(Var, Type)])
43 armarPar (TAtm x) = ( id, [] )
44 armarPar (TBin TCom x y) = ( (Abs var) . a, (var, x):b )
45     where var = head $ filter (not . ('elem' fb)) infiniteVars
46           (a,b) = armarPar y
47           fb = fst b
48
49 -- Proposing recursion schemes:
50
51 dataAEsquema :: DatasH -> [Var] -> (Term, Type) -> (Term -> [Term] ->
    Term, [(Term, Type)])
52 dataAEsquema (o, ds) vs (f, TBin TCom a b) = (\t ts -> Cas t (zip
    consvarpara ts), disp)
53     where cons = fst ds
54           tipopara = snds ds
55           para = map (zip vs) tipopara
56           varpara = map fst para
57           tvarpara = map (mapFst (Atm . Var)) para
58           recs = if (TAtm (TCon o)) == a
59                 then map (map ((\r -> (r,b)) . App f . fst)
    . filter ((==a). snd)) tvarpara
60                 else error "rec call must match data"
61           disp = zipWith (++) recs tvarpara
62           consvarpara = zip cons varpara
63
64
65 -- Proposing expresions of a type:
66
67 enumApp :: Int -> Type -> [(Term, Type)] -> [Term]
68 enumApp n s tts = (fst base) ++ if n == 0 then [] else {- nub $ filter (
    not . ('elem' (fst tts))) -} d
69     where rel = mapSnd fromJust $ filter (isJust . snd) $
    mapSnd (returnsTypeM s) tts
70           (base, funs) = partition (null . snd) rel
71           fargs = filter (not . (any null) . snd) $ mapSnd (map
    (\u -> enumApp (n-1) u tts)) funs
72           d = [appParams (f, cargs) | (f, args) <- fargs,
    cargs <- hilar args]
73
74 enumAppRec :: Rec -> Int -> Type -> [(Term, Type)] -> [Term]
75 enumAppRec rec n s tts = filter (isNothing . (applyRecT [] (rec))) (
    enumApp n s tts)
76
77 enumValues :: Type -> TEnv -> [Term]
78 enumValues t tenv = enumApp 22 t (filter (isACon . fst) tenv)
79
80 enumAppStartingRec :: Rec -> (Term, Type) -> TEnv -> [Term]
81 enumAppStartingRec rec (e,t) tenv = filter (isNothing . (applyRecT [] (
    rec))) (enumAppStarting (e,t) tenv)

```

```

82
83 enumAppStarting :: (Term, Type) -> TEnv -> [Term]
84 enumAppStarting (e,t) tenv = if null pars then [] else (if any null args
      then [] else d)
85           where (pars, ret) = parsAndRet t
86                   args = map (\u -> enumApp (2) u tenv) pars
87                   d = [appParams (e, cargos) | cargos <- hilar
                        args]
88
89 armarUsarTodo :: Type -> [(Term, Type)] -> [Term]
90 armarUsarTodo s [(x,t)] = if s == t then [x] else []
91 armarUsarTodo s tts = llenar
92           where posraices = nub $ mapSnd fromJust $ filter (isJust
      . snd) $ mapSnd (returnsTypeM s) tts
93           distresto (a, b) = filter (not.([]'elem')) $ filter
      (/=[] ) $ dist0 (length b) (deleteFstThat ((==a
      ) . fst) tts)
94           llenar = [ addApps a args | (a,b)<- posraices, cs
      <- distresto (a,b), args <- hilar $ (zipWith (
      armarUsarTodo) b) cs]
95           addApps = foldl App
96
97
98
99 armarUsarTodoHead :: Type -> (Term, Type) -> [(Term, Type)] -> [Term]
100 armarUsarTodoHead s h tts = takeWhile ((==(fst h)) . Atm . fst .
      fromJust . getHead) (armarUsarTodo s (h:tts))
101
102 -- Proposing instance of schemes of recursion
103
104
105 repiteAtmHasta :: Int -> [Term] -> [Term]
106 repiteAtmHasta n ts = filter (\t -> null $ filter (>=n) $ map length $
      group $ sort $ termToAtm t) ts
107
108
109 enumFun :: Rec -> Var -> Type -> [DahasH] -> [(Term, Type)] -> [Term]
110 enumFun rec f tipo@(TBin TCom (TAtm (TCon x)) y) datas ls =
      repiteAtmHasta 3 $ map par apps
111           where (par, vs) = armarPar (tipo)
112                   vs2 = mapFst (Atm . Var) vs
113                   dat = (x, fromJust $ lookup x datas)
114                   (dae, cas) = dataAESquema dat (filter (not . ('elem' (fst
      vs))) infiniteVars) (Atm (Var f), tipo)
115                   (_, res) = parsAndRet tipo
116                   cas' = [[ if (Atm (Var f)) == g
117                             then (appParams (Atm (Var f), gs ++ (fst
      .tail
      vs2) , res)
118                             else (e,t) | (e, t) <- cs, (g, gs) <- [listParams
      e] ] | cs <- cas ]
119           apps = map (dae (fst (vs2!!0))) $ hilar $ map (\c ->
      enumAppRec rec 2 (lastRet (tipo)) ((tail vs2) ++ c ++
      ls) cas'

```

```

120         cant xs = sum $ map (uncurry (+)) $ map (\(a,b) -> (
              cantApps a, cantComs b)) xs
121
122
123
124
125 -- Proposing equations
126
127 conjeturasSimp :: Rec -> Env -> TEnv -> PartTypedTerm -> [Affirm]
128 conjeturasSimp rec env tenv a = filter (\ (Affirm _ a b) -> fund env
              tenv a > fund env tenv b) (conjeturasSoloConDep rec env tenv a)
129
130
131 conjeturasPerm :: Rec -> Env -> TEnv -> PartTypedTerm -> [Affirm]
132 conjeturasPerm rec env tenv a@((e,t), vts) = filter (\(Affirm _ e f) ->
              e /= f )
133
134
135
136
137
138
139
140
141
142 -- example Sum x y :: N [ x :: N, y :: N]
143 type PartTypedTerm = ((Term, Type), [(Var, Type)])
144
145 -- example f = \ x -> case x of { O -> h :: Bool []; S j -> k :: Bool [ f j ::
              Bool] } :: Bool
146 type Scheme = (Var, ((Term, Type), [(Var, [(Term, Type)])))
147
148 saturateTerm :: (Term, Type) -> PartTypedTerm
149 saturateTerm (e, t) = ((foldl1 App (e:(map (Atm. Var) vars)), ret),
              typedvars)
150
151
152
153
154 allSaturations :: (Term, Type) -> [PartTypedTerm]
155 allSaturations (e, t) = [(foldl1 App (e:(map (Atm. Var) (vars n))), ret
              n), typedvars n | n <- [0..length pars]]
156
157
158
159
160
161 toPTT :: (Term, Type) -> PartTypedTerm
162 toPTT = head . allSaturations
163
164 joinVar :: Var -> PartTypedTerm -> [PartTypedTerm]

```



```

165 joinVar v ((e, t), vts) = case lookup v vts of {
166     Nothing -> [];
167     Just s -> let vts' = delete (v,s) vts
168         in [ ((multisubstitute [(v, Atm (Var w
169             ))) e, t), vts') | (w, r) <- vts',
170             r==s];
171
172 permutVars :: Type -> PartTypedTerm -> [PartTypedTerm]
173 permutVars s ((e, t), vts) = let vs = (fst (filter ((==s) . snd) vts));
174     pts = permutations vs
175     in case pts of {
176     [] -> [];
177     (x:xs) -> [ ((multisubstitute (zip
178         x p) e, t), vts) | p <- map (
179         map (Atm . Var)) xs]
180     }
181
182 substitutePTT :: Var -> PartTypedTerm -> PartTypedTerm -> PartTypedTerm
183 substitutePTT v ((e, t), vts) ((f, s), wts)
184     = case lookup v wts of {
185     Nothing -> error $ "uninstantiable variable" ;
186     Just r -> if r/=t
187         then error $ "types do not match"
188         else ((multisubstitute subs f , s) , vts'++wts
189             ')
190     }
191
192 where newvars = freshs $ concat $ [[e, f], map (Atm .
193     Var . fst) vts, map (Atm . Var . fst) wts]
194     wtssinv = filter ((/=v).fst) wts
195     svts = zip (fst vts) newvars
196     swts = zip (fst wtssinv) (drop (length svts)
197         newvars)
198     vts' = zip (snd svts) (snd vts)
199     wts' = zip (snd swts) (snd wtssinv)
200     e' = multisubstitute (mapSnd (Atm . Var) svts) e
201     subs = (v,e'):(mapSnd (Atm . Var) swts)
202
203 ampleDep :: Env -> TEnv -> Atm -> [PartTypedTerm]
204 ampleDep env tenv v = map renameReorderVars $
205     filter ((<=3).length.snd) $
206     a:[ ( substitutePTT w d a) | d@((_,r),_) <- sdep, (w
207         , s) <- ets, s == r]
208
209 where a@((e,t), ets) = saturateTerm (Atm v, fromJust (
210     lookup (Atm v) tenv))
211     deps = sortBy (\x y -> compare (length (
212         dependencies env (Atm x))) (length (
213         dependencies env (Atm y)))) $ (++ [v]) $ (
214         delete v) $ dependencies env (Atm v)
215     depst = map (\x -> (Atm x, fromJust (lookup (Atm
216         x) tenv))) deps
217     sdep = map saturateTerm depst

```

```

206
207 amplePerm :: Env -> TEnv -> [Atm] -> [PartTypedTerm]
208 amplePerm env tenv vs = map renameReorderVars $
209     filter ((<=3).length.snd) res
210     where vst = map (\x -> (Atm x, fromJust (lookup (
211         Atm x) tenv))) vs
211         svs = map saturateTerm vst
212         res = concatMap (instanceAny env tenv svs)
                svs
213
214 ampleAll :: Env -> TEnv -> [Atm] -> [PartTypedTerm]
215 ampleAll env tenv as = map renameReorderVars res
216     where ast = map (\x -> (Atm x, fromJust (lookup (Atm x
217         ) tenv))) as
217         svs = map (\x -> (x, saturateTerm x)) ast
218         res = concatMap (\(x,s) -> tail $ instanceAny
                env tenv (snds (filter ((/=x).fst) svs)) s)
                svs
219
220
221 conjeturasSoloConDep :: Rec -> Env -> TEnv -> PartTypedTerm -> [Affirm]
222 conjeturasSoloConDep rec env tenv ((e,t), vts) = map (Affirm vts e) (
223     enumAppRec rec 2 t (vts' ++ depst))
223     where deps = sortBy (compWith (length . (dependencies
224         env))) $ map Atm $ dependencies env e
224         depst = filter (\(g,_) -> g 'elem' deps) tenv
225         vts' = mapFst (Atm . Var) vts
226
227 conjeturasFloatsDep :: Rec -> Env -> TEnv -> PartTypedTerm -> [Affirm]
228 conjeturasFloatsDep rec env tenv ((e,t), vts) = map (Affirm vts e)
229     $ concat
230     $ filter (not . null)
231     $ map (filter ((tamaño e + 3 >=)
232         . tamaño))
232     $ map (take 1000) -- change
233     $ [ enumAppStartingRec rec (a, b
234         ) (vts' ++ ((e',t'):depst))
235         | (a,b) <- depst ]
234     where
235         e'' = fst $ fromJust (
236             getHead e)
236         e' = Atm $ e''
237         t' = fromJust $ lookup e'
238             tenv
238         id = idependencies env e'
239         deps = map Atm $ id
240         depst = filter (\(g,_) ->
241             g 'elem' deps) tenv
241         vts' = mapFst (Atm . Var)
242             vts
242         vtsid = e'' : ((map Var (
243             fsts vts)) ++ id)
243
244

```

```

245 conjeturasAll :: Rec -> Env -> TEnv -> PartTypedTerm -> [Affirm]
246 conjeturasAll rec env tenv a@((e,t), vts) = filter (\(Affirm _ e f) -> e
    /= f )
247                                     $ map (Affirm vts e)
248                                     $ take 1000 -- cambiar
249                                     $ enumAppRec rec 2 t
250                                     $ ((mapFst (Atm . Var) vts) ++)
251                                     $ (\ q -> filter (\(g,_) -> g 'elem'
    q) tenv)
252                                     $ (map Atm (nub (termToAtm e)))
253
254 renameReorderVars :: PartTypedTerm -> PartTypedTerm
255 renameReorderVars ((e,t),vts) = ((multisubstitute z e, t), w)
256     where f = nub $ freeVars e
257           o = filter ('elem'(fst vts)) f -- ligadas
258           s = sort $ take (length o) (freshs [e]) --
    nuevas
259           z = zip o (map (Atm . Var) s) -- subs
260           w = [(k,t) | (v,t) <- vts, (w, Atm (Var k))
    <- z, w==v]
261
262
263
264 instanceAll :: Env -> TEnv -> [PartTypedTerm] -> PartTypedTerm -> [
    PartTypedTerm]
265 instanceAll env tenv ins q@((e,t), []) = [(e,t), []]
266 instanceAll env tenv ins q@((e,t), (v,r):vts) = concat [ map ((
    substitutePTT v y) . (\ (a,b) -> (a,(v,r):b)) (instanceAll env
    tenv ins ((e,t),vts)) | y <- (filter ((==r) . snd . fst) ins)]
267
268 instanceAny :: Env -> TEnv -> [PartTypedTerm] -> PartTypedTerm -> [
    PartTypedTerm]
269 instanceAny env tenv ins q@((e,t), []) = [(e,t), []]
270 instanceAny env tenv ins q@((e,t), (v,r):vts) = concat [ [(a,(v,r):b),
    substitutePTT v y (a, (v,r):b) ] | y <- (filter ((==r) . snd . fst)
    ins), (a,b) <- (instanceAny env tenv ins ((e,t),vts))]
271
272
273
274
275 disprove :: Env -> TEnv -> Affirm -> Maybe Affirm
276 disprove env tenv (Affirm vs xs ys) = undefined -- instanciar vs con
    enumvalues, computar hasta que sea solo cons todo, comparar.
277
278
279
280
281 findlemas :: Env -> TEnv -> [DahasH] -> Int -> Rec -> Atm -> [Proof]
282 findlemas env tenv datas n rec a = provell env tenv datas n rec $ concat
    $
283     (map (conjeturasSimp rec env tenv) (
    ampleDep env tenv a)):
284     (map (conjeturasSimp rec env tenv) (
    ampleDep env tenv a)):

```

```

285             (map (conjeturasPerm rec env tenv) (
                amplePerm env tenv [a]))):
286             []
287
288 plemas :: Env -> TEnv -> [Datash] -> Int -> Rec -> Atm -> [[Affirm]]
289 plemas env tenv datas n rec a = concat $
290             (map (conjeturasSimp rec env tenv) (
                ampleDep env tenv a)):
291             (map (conjeturasSimp rec env tenv) (
                ampleDep env tenv a)):
292             (map (conjeturasPerm rec env tenv) (
                amplePerm env tenv [a]))):
293             []
294
295 findlemasAll :: Env -> TEnv -> [Datash] -> Int -> Rec -> [Atm] -> [Proof
    ]
296 findlemasAll env tenv datas n rec as = provell env tenv datas n rec $ (
    map (conjeturasAll rec env tenv) (ampleAll env tenv as))
297
298 pfindlemasAll :: Env -> TEnv -> [Datash] -> Int -> Rec -> [Atm] -> [
    Affirm]
299 pfindlemasAll env tenv datas n rec as = concat (map (conjeturasAll rec
    env tenv) (ampleAll env tenv as))

```

A.6 Main.hs

```

1 module Main where
2 import Syntax
3 import Util
4 import Type
5 import Data.List
6 import Data.Maybe
7 import Compute
8 import Prove
9 import Debug.Trace
10 import Propose
11 import PrettyPrint
12
13 main = putStrLn $ "\\section{First Search}" ++
14         (concatMap (\((nom, def), t), ls) ->
15             ("\\n\\n % Ini Fun \\n\\n" ++
16              "\\subsection{$" ++ toLatex (Atm nom) ++ " =
17              " ++ (toLatex def) ++ "$}\\n\\n" ++
18              "$" ++ (toLatex t) ++ "$\\n\\n" ++
19              (concatMap (\l -> "\\subsubsection{""++toLatex
20              (headProof l) ++ "}"\\n\\n" ++ toLatex l
21              ++ "\\n\\n") ls) ++
22              "\\n\\n % Fin Fun \\n\\n")) namedFstSel)
23         ++
24         "\\section{Second Search}" ++
25         (concatMap (\((nom, def), t), ls) ->
26             ("\\n\\n % Ini Fun \\n\\n" ++
27              "\\subsection{$" ++ toLatex (Atm nom) ++ " =
28              " ++ (toLatex def) ++ "$}\\n\\n" ++

```

```

25         "$" ++ (toLatex t) ++ "$\n\n" ++
26         (concatMap (\l -> "\subsubsection{""++toLatex
          (headProof l) ++ ""}\n\n" ++ toLatex l ++
          "\n\n") ls) ++
27         "\n\n % Fin Fun \n\n")) namedSndSel)
28
29 datas :: [DatasH]
30 datas = concat [enumContainerRecTypes cantcons canrec canemb | cantcons
  <- [2], canrec <-[0..1], canemb <- [0..1]]
31
32 tiposfununi = armarTip 1 (map (TAtm . TCon . fst) datas)
33 tiposfunbin = armarTip 2 (map (TAtm . TCon . fst) datas)
34
35 tipbb = tiposfununi!!0
36 tipnb = tiposfununi!!4
37 tipnn = tiposfununi!!5
38 tiplb = tiposfununi!!12
39 tipln = tiposfununi!!13
40 tipll = tiposfununi!!14
41
42 paciencia :: Int
43 paciencia = 30
44
45 cerothTEnv :: TEnv
46 cerothTEnv = mapFst (Atm . Con) $ concatMap datasH2C datas
47
48 cerothRec :: Rec
49 cerothRec = emptyRec
50
51 candidatosFst :: [(Term, Type)]
52 candidatosFst = filter (\ (e,t) -> {- not (shouldContainRecursiveCall t) ||
  -} (contiene (Var "f") e))
53         [ (f, tipo) | aridad <- [1..2],
54           tipo <- (armarTip aridad (map (TAtm .
55             TCon . fst) [datas!!1, datas!!3] )),
56           f <- enumFun cerothRec "f" tipo datas
57             cerothTEnv ]
58
59
60
61 candidatosFstTeo :: [((Term, Type), [Proof])]
62 candidatosFstTeo = [(f, t), findlemas [(Var "f", f)] ((Atm (Var "f"), t)
  :cerothTEnv) datas paciencia cerothRec (Var "f")) | (f, t) <-
  candidatosFst]
63
64 candidatosFstSel :: [((Term, Type), [Proof])]
65 candidatosFstSel = filter ((>3) . length . map headProof {- . filter
  containsIndc -} . snd ) candidatosFstTeo
66         where transformLemas (Affirm _ _ c) = (Var "f") '
          elem' (termToAtm c)
67
68 namedFstSel :: [(((Atm, Term), Type), [Proof])]

```

```

69 namedFstSel = map (\ (((def, typ), ls), n) -> let newf = (Var "f" ++ (
      show n)))
70
71
72
73
74
75 fstEnv :: Env
76 fstEnv = [(nom, def) | (((nom, def), typ), ls) <- namedFstSel]
77
78 fstTEnv :: TEnv
79 fstTEnv = cerothTEnv ++ [ (Atm nom, typ) | (((nom, def), typ), ls) <-
      namedFstSel]
80
81 fstRec :: Rec
82 fstRec = addAffirms fstEnv cerothRec ( map headProof $ concat $ snds $
      namedFstSel )
83
84 candidatosSnd :: [(Term, Type)]
85 candidatosSnd = candidatosSndUni ++ candidatosSndBin
86
87 candidatosSndUni :: [(Term, Type)]
88 candidatosSndUni = filter (\ (e,t) -> (contiene (Var "f") e) && any (flip
      contiene e) (fstS fstEnv))
89
90
91
92 candidatosSndBin :: [(Term, Type)]
93 candidatosSndBin = filter (\ (e,t) -> (contiene (Var "f") e) &&
94
95
96
97
98
99
100 candidatosSndTeo :: [((Term, Type), [Proof])]
101 candidatosSndTeo = [((f, t), findlemas ((Var "f", f):fstEnv) ((Atm (Var "
      f"), t):fstTEnv) datas paciencia fstRec (Var "f")) | (f, t) <-
      candidatosSnd]
102
103 candidatosSndSel :: [((Term, Type), [Proof])]
104 candidatosSndSel = filter ((>=2) . length . map headProof . filter
      containsIndc . snd) candidatosSndTeo

```

```

105             where transformLemas (Affirm _ _ c) = (Var "f") `
                elem' (termToAtm c)
106
107 namedSndSel :: [((Atm, Term), Type), [Proof]]]
108 namedSndSel = map (\ ((def, typ), ls), n) -> let newf = (Var ("f" ++ (
                show n)))
109
110             anewf = Atm newf
111             in ((newf, substitute "f"
                anewf def), typ), map (
                substituteProof "f" anewf)
                ls))
111 $ zip candidatosSndSel [(length fstEnv)..]
112     where substituteProof x y = mapProof (\ (Affirm
                a b c) -> Affirm a (substitute x y b) (
                substitute x y c))

```

A.7 Test.hs

```

1 module Test where
2 import Syntax
3 import Util
4 import Type
5 import Data.List
6 import Data.Maybe
7 import Compute
8 import Prove
9 import Debug.Trace
10 import Propose
11 import PrettyPrint
12
13 main = putStrLn (show $ (provelemas 30))
14
15
16 ----- Strategy
17 -- list types with 2 constructors, 1 recursive call, 0 embeded types
18 dats0 = enumContainerRecTypes 2 0 0
19 -- only one..
20 boo = dats0!!0
21
22 (boonom, _) = boo
23 -- pass from abbreviation to expansion
24 booExt = datasH2C boo
25
26 -- list types with 2 constructors, 1 recursive call, 0 embeded types
27 dats1 = enumContainerRecTypes 2 0 1
28 nat = dats1!!0
29
30 (natnom, _) = nat
31 -- pass from abbreviation to expansion
32 natExt = datasH2C nat
33
34 -- list types with 2 constructors, 1 recursive call, 1 embeded types
35 dats2 = enumContainerRecTypes 2 1 1
36 lis = dats2!!0

```

```

37
38 (lisnom, _) = lis
39 -- pass from abbreviation to expansion
40 lisExt = datasH2C lis
41
42 tipos = armarTip 2 [TAtm (TCon boonom), TAtm (TCon natnom), TAtm (TCon
    lisnom)]
43 tipo = tipos!!26
44
45 pruebaEnum = enumFun emptyRec "f" tipo [nat, boo] []
46
47 candidatos :: [Term]
48 candidatos = enumFun emptyRec "f" tipo [nat, boo, lis] (mapFst (Atm . Con
    ) natExt)
49
50 candidatosFst :: [Term]
51 candidatosFst = filter (\t -> all ('elem' termToAtm t) [Var "x", Var "f"]
    ) $ enumFun emptyRec "f" tipo [boo,nat,lis] (mapFst (Atm . Con) (
    natExt++lisExt++booExt))
52
53 candidatosRev :: [Term]
54 candidatosRev = filter (\t -> all ('elem' termToAtm t) [Var "concat", Var
    "f"] ) $ enumFun emptyRec "f" (TBin TCom (TAtm (TCon "C_,a_R")) (TAtm
    (TCon "C_,a_R"))) [boo,nat,lis] ((Atm (Var "concat"), TBin TCom (
    TAtm (TCon "C_,a_R")) (TBin TCom (TAtm (TCon "C_,a_R")) (TAtm (TCon
    "C_,a_R")))):(mapFst (Atm . Con) (natExt++lisExt++booExt)))
55
56 -- prueba conjetura
57 expsconcandidatos :: Int -> [Term]
58 expsconcandidatos a = enumApp a (TAtm (TCon natnom)) ([ (Atm (Var "x"),
    TAtm (TCon natnom)), (Atm (Var "y"), TAtm (TCon natnom)), (Atm (Var
    "f"), tipo)] ++ ( mapFst (Atm . Con) $ concatMap datasH2C [nat, boo
    ]))
59
60
61 conjeturasFst :: [Affirm]
62 conjeturasFst = conjeturas (TAtm (TCon natnom)) ((Atm (Var "f"), tipo) :
    ( mapFst (Atm . Con) $ concatMap datasH2C [nat, boo])) [(Var "f")]
63
64 conjeturasSnd :: [Affirm]
65 conjeturasSnd = conjeturas (TAtm (TCon natnom)) ((Atm (Var "g"), tipo) :
    (Atm (Var "f"), tipo) : ( mapFst (Atm . Con) $ concatMap datasH2C [
    nat, boo])) [(Var "f1")]
66
67 conjeturas0 :: [Affirm]
68 conjeturas0 = filter (\(Affirm _ x y) -> case (getHeadCon x, getHeadCon
    y) of { (Just (a,_), Just (b,_)) -> a /= b; _ -> True;}) $
69     -- for any constructor S, (S x = S y <=> x = y) so we can safely
    ignore it
70     filter (\(Affirm _ x y) -> all ('elem' (termToAtm x ++
    termToAtm y)) [Var "x", Var "f"] ) $
71     -- ignore what does not apply f to a quantified variable
72     filter (\a -> a /= flipAffirm a) $
73     -- avoid repetitions

```



```

74     map (\(y,z) -> Affirm [("x", (TAtm . TCon) natnom), ("y", (
      TAtm . TCon) natnom)] y z) $
75     armarParejas (expsconcandidates 3)
76
77
78 conjeturas2 = let (sim0, nosim0) = partition (simplifies env)
      conjeturas0;
79     (sim1, nosim) = partition ((simplifies env).flipAffirm)
      nosim0;
80     sim = sim0 ++ (map flipAffirm sim1);
81     (floDep0, nofloDep0) = partition (floatsDep env) nosim;
82     (floDep1, nofloDep) = partition ((floatsDep env).
      flipAffirm) nofloDep0;
83     floDep = floDep0 ++ (map flipAffirm floDep1);
84     (perm, _) = partition (soloPermuta) nofloDep;
85     in ((sortBy (simplifiesOrd env) sim ) ++ (sortBy (
      floatsDepOrd env) floDep ) ++ (filter soloPermuta perm
      ))
86     where env = [(Var "f", candidatos!!48)]
87
88
89 suma = (candidatos)!!48
90
91
92 conjeturas :: Type -> [(Term, Type)] -> [Atm] -> [Affirm]
93 conjeturas t exps must = filter (\(Affirm vs x y) -> (any ('elem'must) (
      termToAtm x ++ termToAtm y))) $
94     filter (\a -> a /= flipAffirm a) ${-sortBy comp $
      -}
95     map (\(y,z) -> Affirm vars y z) $ armarParejas (
      repiteAtmHasta 3 $ enumApp 3 (TAtm (TCon
      natnom)) (exps++(map (\(a,b) -> ((Atm .Var) a,
      b)) vars)))
96     where vars = [("x", (TAtm . TCon) natnom), ("y", (
      TAtm . TCon) natnom)]
97
98
99 --For testing tactics without testing strategy:
100
101 --Nat
102
103 hTrue = Atm (Con sTrue)
104 hFalse = Atm (Con sFalse)
105 sTrue = "True"
106 sFalse = "False"
107 h0 = Atm (Con s0)
108 hS = Atm (Con sS)
109 s0 = "0"
110 sS = "S"
111 fS = App hS
112 hNil = Atm (Con sNil)
113 hCons = Atm (Con sCons)
114 sNil = "Nil"
115 sCons = "Cons"

```

```

116 fCons x xs = App ((App hCons) x) xs
117
118 dataBool = ("Bool", [(sTrue,[]),(sFalse,[])])
119 dataNat = ("Nat", [(s0,[]),(sS,[TAtm (TCon "Nat")])])
120 dataList = ("List", [(sNil, []), (sCons, [TAtm (TVar "a"), TAtm (TCon "
List")]) ])
121 datas = [dataBool, dataNat, dataList]
122
123 envToVars :: Env -> [Var]
124 envToVars = map (\(Var x) -> x) . filter (not . isCon) . fstS
125
126 env :: Env
127 env =
128   (Var "suma" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x")) [(s0, []),
Atm (Var "y")], ((sS, ["k"]), fS (App (App (Atm (Var "suma")) (
Atm (Var "k")))) (Atm (Var "y"))))]) :
129   (Var "mult" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x"))
130     [(s0, []), h0],
131     ((sS, ["k"]), App (App (Atm (Var "suma"
)) (App (App (Atm (Var "mult")) (Atm
(Var "k")))) (Atm (Var "y")))) (Atm
(Var "y"))]) :
132   (Var "flipexp" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x"))
133     [(s0, []), mkAp [Con "S", Con "O"]],
134     ((sS, ["k"]), App (App (Atm (Var "mult"
) (App (App (Atm (Var "flipexp")) (
Atm (Var "k")))) (Atm (Var "y")))) (
Atm (Var "y"))]) :
135   (Var "not" , Abs "x" $ Cas (Atm (Var "x")) [(sTrue, []), hFalse], ((
sFalse, []), hTrue])) :
136   (Var "and" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x")) [(sTrue , []),
Atm (Var "y")], ((sFalse, []), hFalse])) :
137   (Var "or" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x")) [(sFalse, []),
Atm (Var "y")], ((sTrue , []), hTrue])) :
138   (Var "ssi" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x"))
139     [(sTrue , []), Atm (Var "y")],
140     ((sFalse, []), Cas (Atm (Var "y"))
141       [(sTrue, []), hFalse],
142       ((sFalse, []), hTrue))]) :
143   (Var "xor" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x")) [(sTrue, []),
mkAp [ Var "not", Var "y"]], ((sFalse, []), Atm (Var "y"))]) :
144   (Var "mayoria", Abs "x" $ Abs "y" $ Abs "z" $ Cas (Atm (Var "x"))
145     [(sFalse, []), Cas (Atm (
Var "y"))
146     [(sFalse,
[]),
hFalse
),
147     ((sTrue ,
[]),
Atm
(Var
"z"))
],)

```

```

148         ((sTrue , []), Cas (Atm (
149             Var "y"))
150             [(sFalse,
151                 []),
152                 Atm (
153                     Var "z
154                     ")),
155                 ((sTrue ,
156                     []),
157                     hTrue
158                     )]]))
159             :
160 (Var "par" , Abs "x" $ Cas (Atm (Var "x")) [(s0, []), hTrue), ((sS,
161     ["k"]), (App (Atm (Var "not")) (App (Atm (Var "par")) (Atm (Var "
162     k"))))))]) :
163 (Var "doble" , Abs "x" $ Cas (Atm (Var "x")) [(s0, []), h0), ((sS, [
164     "k"]), App (Atm (Con "S")) (App (Atm (Con "S")) (App (Atm (Var "
165     doble")) (Atm (Var "k"))))))]) :
166 (Var "pred" , Abs "x" $ Cas (Atm (Var "x")) [(s0, []), h0), ((sS, ["
167     k"]), (App (Atm (Var "pred")) (Atm (Var "k"))))] ) :
168 (Var "min" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x"))
169     [(s0, []), (Atm (Con "O"))],
170     ((sS, ["k"]), Cas (Atm (Var "y"))
171     [(s0, []), Atm (Con "O")],
172     ((sS, ["j"]), mkAp1 [ Atm (
173         Con "S"), mkAp [ Var "
174         min", Var "k", Var "j"]]]
175     ])) :
176 (Var "max" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x"))
177     [(s0, []), (Atm (Var "y"))],
178     ((sS, ["k"]), Cas (Atm (Var "y"))
179     [(s0, []), Atm (Var "x")],
180     ((sS, ["j"]), mkAp1 [ Atm (
181         Con "S"), mkAp [ Var "
182         max", Var "k", Var "j"
183         ]]]))] ) :
184 (Var "absr" , Abs "x" $ Abs "y" $ Cas (Atm (Var "x"))
185     [(s0, []), (Atm (Var "y"))],
186     ((sS, ["k"]), Cas (Atm (Var "y"))
187     [(s0, []), Atm (Var "x")],
188     ((sS, ["j"]), mkAp1 [ mkAp [
189         Var "absr", Var "k", Var
190         "j"]]]))] ) :
191 (Var "concat" , Abs "xs" $ Abs "ys" $ Cas (Atm (Var "xs")) [ ((sNil,
192     []), Atm (Var "ys")), ((sCons, ["k", "ks"]), fCons (Atm (Var "k")
193     ) (App (App (Atm (Var "concat")) (Atm (Var "ks"))) (Atm (Var "ys"
194     ))))] ) :
195 (Var "largo" , Abs "xs" $ Cas (Atm (Var "xs")) [ ((sNil, []), h0), ((
196     sCons, ["k", "ks"]), fS (App (Atm (Var "largo")) (Atm (Var "ks")))
197     )] ) :
198 (Var "rev" , Abs "xs" $ Cas (Atm (Var "xs")) [ ((sNil, []), hNil), ((
199     sCons, ["k", "ks"]), (hConcat (App (Atm (Var "rev")) (Atm (Var "
200     ks"))) (fCons (Atm (Var "k")) hNil))] ) :

```

```

172     []
173
174
175
176
177 intToS n | n == 0 = h0
178         | otherwise = fS (intToS (n-1))
179
180 sToInt (Atm (Con "O")) = 0
181 sToInt (App (Atm (Con "S")) x) = 1 + (sToInt x)
182 sToInt _ = error "no es un número"
183
184 -- Probar, ejemplo:
185 omasy = App (App (Atm (Var "suma")) (Atm (Con "O"))) (Atm (Var "y"))
186 xmaso = App (App (Atm (Var "suma")) (Atm (Var "x"))) (Atm (Con "O"))
187 sxmasy = App (App (Atm (Var "suma")) (App (Atm (Con "S")) (Atm (Var "x"
188         )))) (Atm (Var "y"))
189 xmassy = App (App (Atm (Var "suma")) (Atm (Var "x"))) (App (Atm (Con "S"
190         )) (Atm (Var "y")))
191 xmasy = App (App (Atm (Var "suma")) (Atm (Var "x"))) (Atm (Var "y"))
192 ymasx = App (App (Atm (Var "suma")) (Atm (Var "y"))) (Atm (Var "x"))
193 ymasz = App (App (Atm (Var "suma")) (Atm (Var "y"))) (Atm (Var "z"))
194 xmas_ymasz = App (App (Atm (Var "suma")) (Atm (Var "x"))) ymasz
195 xmas_y_masx = App (App (Atm (Var "suma")) (Atm (Var "x"))) xmasz
196 xmas_y_masx = App (App (Atm (Var "suma")) (Atm (Var "x"))) xmasz
197 s_xmasy = App (Atm (Con "S")) (xmasz)
198 opory = App (App (Atm (Var "mult")) (Atm (Con "O"))) (Atm (Var "y"))
199 xporo = App (App (Atm (Var "mult")) (Atm (Var "x"))) (Atm (Con "O"))
200 sxpory = App (App (Atm (Var "mult")) (App (Atm (Con "S")) (Atm (Var "x"
201         )))) (Atm (Var "y"))
202 xporsy = App (App (Atm (Var "mult")) (Atm (Var "x"))) (App (Atm (Con "S"
203         )) (Atm (Var "y")))
204 xpory = App (App (Atm (Var "mult")) (Atm (Var "x"))) (Atm (Var "y"))
205 xporz = App (App (Atm (Var "mult")) (Atm (Var "x"))) (Atm (Var "z"))
206 yporx = App (App (Atm (Var "mult")) (Atm (Var "y"))) (Atm (Var "x"))
207 yporz = App (App (Atm (Var "mult")) (Atm (Var "y"))) (Atm (Var "z"))
208 s_xpory = App (Atm (Con "S")) (xpory)
209 xmas_xpory = App (App (Atm (Var "suma")) (Atm (Var "x"))) (xpory)
210 ymas_xpory = App (App (Atm (Var "suma")) (Atm (Var "y"))) (xpory)
211 xpor_ymasz = App (App (Atm (Var "mult")) (Atm (Var "x"))) (ymasz)
212 xpory_mas_xporz = App (App (Atm (Var "suma")) (xpory)) (xporz)
213 ymasz_porx = App (App (Atm (Var "mult")) ymasz) (Atm (Var "x"))
214 xporymas_x = App (App (Atm (Var "suma")) xpory) (Atm (Var "x"))
215 xporymas_y = App (App (Atm (Var "suma")) xpory) (Atm (Var "y"))
216 xpor_yporz = App (App (Atm (Var "mult")) (Atm (Var "x"))) yporz
217 xpory_porz = App (App (Atm (Var "mult")) xpory) (Atm (Var "z"))
218 nilconcatxs = App (App (Atm (Var "concat")) hNil) (Atm (Var "xs"))
219 xsconcatnil = App (App (Atm (Var "concat")) (Atm (Var "xs"))) hNil
220 xsconcatys = App (App (Atm (Var "concat")) (Atm (Var "xs"))) (Atm (Var "
221         ys"))
222 ysconcatxs = App (App (Atm (Var "concat")) (Atm (Var "ys"))) (Atm (Var "
223         xs"))
224 ysconcatzs = App (App (Atm (Var "concat")) (Atm (Var "ys"))) (Atm (Var "
225         zs"))

```

```

218 xsconcat_ysconcatzs = App (App (Atm (Var "concat")) (Atm (Var "xs")))
    ysconcatzs
219 xsconcatys_concatzs = App (App (Atm (Var "concat")) xsconcatys) (Atm (Var
    "zs"))
220 x_xsconcatys = fCons (Atm (Var "x")) xsconcatys
221
222 asocmas :: Int -> Term
223 asocmas 0 = Atm (Var "0")
224 asocmas n = App (App (Atm (Var "suma")) (Atm (Var (show n)))) (asocmas (
    n-1))
225 -- probar mas
226
227 mkAp = (foldl1 App) . (map Atm)
228 mkAp1 = (foldl1 App)
229
230 lemascomp :: [Affirm]
231 lemascomp =
232     Affirm [("y", (TAtm . TCon) "Nat")]
233         (mkAp [Var "suma", Con "O", Var "y"])
234         (Atm (Var "y")) :
235     Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
236         (mkAp1 [Atm (Con "S"), (mkAp [Var "suma
    ", Var "x", Var "y"])]])
237         (mkAp1 [Atm (Var "suma"), mkAp [Con "S"
    , Var "x", Atm (Var "y")]] :
238     Affirm [("y", (TAtm . TCon) "Nat")] opory (Atm (Con "O")) :
239     Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
240         (mkAp1 [Atm (Var "mult"), mkAp [Con "S",
    Var "x", Atm (Var "y")]]
241         (mkAp1 [mkAp [Var "suma", Var "y"],
    mkAp [Var "mult", Var "x", Var "y"
    ]]) :
242     Affirm [("xs", (TAtm . TCon) "List")] nilconcatxs (Atm (Var "xs"))
    :
243     []
244
245 lemas :: [Affirm]
246 lemas =
247     Affirm [("b1", (TAtm . TCon) "Bool"), ("b2", (TAtm . TCon) "Bool")]
248         (mkAp [Var "mayoria", Var "b1", Var "b1"
    , Var "b2"])
249         (Atm (Var "b1")) :
250 --     Affirm [("x", (TAtm . TCon) "Bool")]
251 --         (mkAp1 [Atm (Var "not"), mkAp [Var "not
    ", Var "x"]])
252 --         (Atm (Var "x")) :
253     Affirm [("x", (TAtm . TCon) "Bool")]
254         (mkAp [Var "and", Var "x", Var "x"])
255         (Atm (Var "x")) :
256     Affirm [("x", (TAtm . TCon) "Bool")]
257         (mkAp [Var "ssi", Var "x", Var "x"])
258         (Atm (Con "True")) :
259     Affirm [("x", (TAtm . TCon) "Bool"), ("y", (TAtm . TCon) "Bool")]

```

```

260         (mkAp1 [Atm (Var "not"), mkAp [Var "ssi",
261           Var "x", Var "y"]])
262         (mkAp1 [Atm (Var "ssi"), mkAp [Var "not",
263           Var "x"], Atm (Var "y")]) :
264 Affirm [("x", (TAtm . TCon) "Bool"), ("y", (TAtm . TCon) "Bool")]
265         (mkAp [Var "ssi", Var "x", Var "y"])
266         (mkAp [Var "ssi", Var "y", Var "x"]) :
267 Affirm [("x", (TAtm . TCon) "Bool"), ("y", (TAtm . TCon) "Bool")]
268         (mkAp1 [Atm (Var "not"), mkAp [Var "xor"
269           , Var "x", Var "y"]])
270         (mkAp1 [Atm (Var "xor"), mkAp [Var "not"
271           , Var "x"], Atm (Var "y")]) :
272 Affirm [("x", (TAtm . TCon) "Bool"), ("y", (TAtm . TCon) "Bool")]
273         (mkAp [Var "xor", Var "x", Var "y"])
274         (mkAp [Var "xor", Var "y", Var "x"]) :
275 Affirm [("x", (TAtm . TCon) "Bool"), ("y", (TAtm . TCon) "Bool"), (
276   "z", (TAtm . TCon) "Bool")]
277         (mkAp1 [mkAp [Var "xor", Var "x"], mkAp
278           [Var "xor", Var "y", Var "z"]])
279         (mkAp1 [Atm (Var "xor"), mkAp [Var "xor"
280           , Var "x", Var "y"], Atm (Var "z")])
281         :
282 Affirm [("x", (TAtm . TCon) "Nat")]
283         (mkAp [Var "suma", Var "x", Con "O"])
284         (Atm (Var "x")) :
285 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
286         (mkAp1 [Atm (Con "S"), (mkAp [Var "suma"
287           , Var "x", Var "y"])]])
288         (mkAp1 [Atm (Var "suma"), Atm (Var "x")
289           , mkAp [Con "S", Var "y"]]) :
290 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
291         (mkAp [Var "suma", Var "x", Var "y"])
292         (mkAp [Var "suma", Var "y", Var "x"]) :
293 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat"), ("z"
294   ", (TAtm . TCon) "Nat")]
295         (mkAp1 [mkAp [Var "suma", Var "x"],
296           mkAp [Var "suma", Var "y", Var "z"
297           ]])
298         (mkAp1 [Atm (Var "suma"), mkAp [Var "
299   suma", Var "x", Var "y"], Atm (Var
300   "z")]) :
301 Affirm [("y", (TAtm . TCon) "Nat")] opory (Atm (Con "O")) :
302 Affirm [("x", (TAtm . TCon) "Nat")] xporo (Atm (Con "O")) :
303 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
304         (mkAp1 [Atm (Var "mult"), mkAp [Con "S",
305           Var "x"], Atm (Var "y")])
306         (mkAp1 [mkAp [Var "suma", Var "y"],
307           mkAp [Var "mult", Var "x", Var "y"
308           ]]) :
309 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
310         xporo ymas_xporo :
311 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
312         xporsy xmas_xporo :

```

```

293 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
      sxyory xpyorymas_y :
294 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
      xporsy xpyorymas_x :
295 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat"), ("z
      ", (TAtm . TCon) "Nat")] xpor_ymasz xpyory_mas_xporz :
296 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat"), ("z
      ", (TAtm . TCon) "Nat")] ymasz_porx xpyory_mas_xporz :
297 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat")]
      xpyory yporx :
298 Affirm [("x", (TAtm . TCon) "Nat"), ("y", (TAtm . TCon) "Nat"), ("z
      ", (TAtm . TCon) "Nat")] xpor_yporz xpyory_porz :
299 Affirm [("x", (TAtm . TCon) "Nat")] (App (Atm(Var "doble")) (Atm (
      Var "x")) (App (App (Atm(Var "suma")) (Atm (Var "x")) (Atm
      (Var "x"))):
300 Affirm [("xs", (TAtm . TCon) "List")] nilconcatxs (Atm (Var "xs"))
      :
301 Affirm [("xs", (TAtm . TCon) "List")] xsconcatnil (Atm (Var "xs"))
      :
302 Affirm [("x", (TAtm . TCon) "a"), ("xs", (TAtm . TCon) "List"), ("
      ys", (TAtm . TCon) "List")]
303 (mkAp1 [Atm (Var "concat"), mkAp
      [Con "Cons", Var "x", Var "
      xs"], mkAp [Var "ys"]])
304 (mkAp1 [Atm (Con "Cons"), mkAp [
      Var "x"], mkAp [Var "concat",
      Var "xs", Var "ys"]]):
305 Affirm [("xs", (TAtm . TCon) "List"), ("ys", (TAtm . TCon) "List"), (
      "zs", (TAtm . TCon) "List")] xsconcat_ysconcatzs
      xsconcatys_concatzs :
306 Affirm [("x", (TAtm . TCon) "a"), ("xs", (TAtm . TCon) "List")]
307 (mkAp1 [Atm (Var "rev"), mkAp
      [Con "Cons", Var "x",
      Var "xs"]])
308 (mkAp1 [Atm (Var "concat"),
      mkAp [Var "rev", Var "xs"
      ], mkAp [Con "Cons", Var
      "x", Con "Nil"]]):
309 Affirm [("x", (TAtm . TCon) "a") ]
310 (mkAp1 [Atm (Var "rev"), mkAp
      [Con "Cons", Var "x",
      Con "Nil"]])
311 (mkAp [Con "Cons", Var "x",
      Con "Nil"]):
312 Affirm [] ((App (Atm (Var "rev")) (Atm (Con "Nil")))) (Atm (Con "
      Nil")) :
313 Affirm [("xs", (TAtm . TCon) "List"), ("ys", (TAtm . TCon) "List")] (
      App (Atm (Var "rev")) xsconcatys) (hConcat (App (Atm (Var "
      rev")) (Atm (Var "ys")) (App (Atm (Var "rev")) (Atm (Var "xs"
      )))) :
314 Affirm [("xs", (TAtm . TCon) "List")] (App (Atm (Var "rev")) (App (
      Atm (Var "rev")) (Atm (Var "xs")))) (Atm (Var "xs")) :
315 Affirm [("xs", (TAtm . TCon) "List"), ("ys", (TAtm . TCon) "List")]

```

```

316         (App (Atm (Var "largo")) xsconcatys) (hSuma (App (Atm (Var
"largo")) (Atm (Var "xs"))) (App (Atm (Var "largo")) (
Atm (Var "ys")))) :
317     Affirm [("xs", (TAtm . TCon) "List"), ("ys", (TAtm . TCon) "List")] (
App (Atm (Var "largo")) xsconcatys) (App (Atm (Var "largo"))
ysconcatxs) :
318     []
319
320 hSuma x y = App (App (Atm (Var "suma")) x) y
321 hConcat x y = App (App (Atm (Var "concat")) x) y
322
323 --ie
324 -- prove ([], lema1)
325 -- prove ([], lema2)
326 -- prove ([lema1], lema2)
327
328
329 tenv :: [(Term, Type)]
330 tenv =
331     (Atm (Con "O"), TAtm (TCon "Nat")) :
332     (Atm (Con "S"), TBin TCom (TAtm (TCon "Nat")) (TAtm (TCon "Nat")
)) :
333     (Atm (Var "suma"), TBin TCom (TAtm (TCon "Nat")) (TBin TCom (
TAtm (TCon "Nat")) (TAtm (TCon "Nat")))) :
334     (Atm (Var "mult"), TBin TCom (TAtm (TCon "Nat")) (TBin TCom (
TAtm (TCon "Nat")) (TAtm (TCon "Nat")))) :
335     (Atm (Var "flipexp"), TBin TCom (TAtm (TCon "Nat")) (TBin TCom (
TAtm (TCon "Nat")) (TAtm (TCon "Nat")))) :
336     (Atm (Con "True"), TAtm (TCon "Bool")) :
337     (Atm (Con "False"), TAtm (TCon "Bool")) :
338     (Atm (Var "par"), TBin TCom (TAtm (TCon "Nat")) (TAtm (TCon "
Bool")))) :
339     (Atm (Var "doble"), TBin TCom (TAtm (TCon "Nat")) (TAtm (TCon "
Nat")))) :
340     (Atm (Var "not"), TBin TCom (TAtm (TCon "Bool")) (TAtm (TCon "
Bool")))) :
341     (Atm (Var "mayoria"), foldr1 (TBin TCom) $ map TAtm $ [TCon "
Bool", TCon "Bool", TCon "Bool", TCon "Bool"]) :
342     (Atm (Var "and"), foldr1 (TBin TCom) $ map TAtm $ [TCon "Bool",
TCon "Bool", TCon "Bool"]) :
343     (Atm (Var "or"), foldr1 (TBin TCom) $ map TAtm $ [TCon "Bool",
TCon "Bool", TCon "Bool"]) :
344     (Atm (Var "ssi"), foldr1 (TBin TCom) $ map TAtm $ [TCon "Bool",
TCon "Bool", TCon "Bool"]) :
345     (Atm (Var "xor"), foldr1 (TBin TCom) $ map TAtm $ [TCon "Bool",
TCon "Bool", TCon "Bool"]) :
346     (Atm (Var "min"), TBin TCom (TAtm (TCon "Nat")) (TBin TCom (TAtm
(TCon "Nat")) (TAtm (TCon "Nat")))) :
347     (Atm (Var "max"), TBin TCom (TAtm (TCon "Nat")) (TBin TCom (
TAtm (TCon "Nat")) (TAtm (TCon "Nat")))) :
348     (Atm (Var "absr"), TBin TCom (TAtm (TCon "Nat")) (TBin TCom (
TAtm (TCon "Nat")) (TAtm (TCon "Nat")))) :
349     (Atm (Con "Nil"), TAtm (TCon "List")) :

```



```

350     (Atm (Con "Cons"), TBin TCom (TAtm (TCon "a")) (TBin TCom (TAtm
351         (TCon "List")) (TAtm (TCon "List")))) :
352     (Atm (Var "concat"), TBin TCom (TAtm (TCon "List")) (TBin TCom (
353         TAtm (TCon "List")) (TAtm (TCon "List")))) :
354     (Atm (Var "rev"), TBin TCom (TAtm (TCon "List")) (TAtm (TCon "List
355         ")))) :
356     (Atm (Var "largo"), TBin TCom (TAtm (TCon "List")) (TAtm (TCon "
357         Nat")))) :
358     (Atm (Con "Hoja"), TBin TCom (TAtm (TCon "Nat")) (TAtm (TCon "
359         TreeNat")))) :
360     (Atm (Con "Nodo"), TBin TCom (TAtm (TCon "TreeNat")) (TBin TCom
361         (TAtm (TCon "TreeNat")) (TAtm (TCon "TreeNat")))) :
362     []
363
364 lsumad = findlemas (take 3 env) (take 5 tenv) datas 30 emptyRec (Var "
365     suma")
366 lsumal = plemas (take 3 env) (take 5 tenv) datas 30 emptyRec (Var "suma"
367     )
368 lsuma = map headProof lsumad
369 lsumar = addAffirms env emptyRec lsuma
370
371 lmultd = findlemas (take 4 env) (take 6 tenv) datas 30 lsumar (Var "mult
372     ")
373 lmultl = plemas (take 4 env) (take 6 tenv) datas 30 lsumar (Var "mult")
374 lmult = map headProof lmultd
375 lmultr = addAffirms env lsumar lmult
376
377 lexd = findlemas (take 5 env) (take 6 tenv) datas 30 lmultr (Var "
378     flipexp")
379 lexp1 = plemas (take 5 env) (take 6 tenv) datas 30 lmultr (Var "flipexp")
380 lexp = map headProof lexd
381 lexr = addAffirms env lmultr lexp
382
383 lmind = findlemas env tenv datas 30 emptyRec (Var "min")
384 lmin = map headProof lmind
385 lminr = addAffirms env emptyRec lmin
386
387 lmaxd = findlemas env tenv datas 30 emptyRec (Var "max")
388 lmax = map headProof lmaxd
389 lmaxr = addAffirms env emptyRec lmax
390
391 labsrd = findlemas env tenv datas 30 emptyRec (Var "absr")
392 labsr = map headProof labsrd
393 labsrr = addAffirms env emptyRec labsr
394
395 landd = findlemas env tenv datas 30 emptyRec (Var "and")
396 land = map headProof landd
397 landr = addAffirms env emptyRec land
398
399 lord = findlemas env tenv datas 30 emptyRec (Var "or")

```

```

394 lor = map headProof lord
395 lorr = addAffirms env emptyRec lor
396
397 lnotd = findlemas env tenv datas 30 emptyRec (Var "not")
398 lnot = map headProof lnotd
399 lnotr = addAffirms env emptyRec lnot
400
401
402 lconcatd = findlemas env tenv datas 30 emptyRec (Var "concat")
403 lconcatl = plemas env tenv datas 30 emptyRec (Var "concat")
404 lconcat = map headProof lconcatd
405 lconcatr = addAffirms env emptyRec lconcat
406
407 lre vd = findlemas env tenv datas 30 lconcatr (Var "rev")
408 lre vl = plemas env tenv datas 30 lconcatr (Var "rev")
409 lre v = map headProof lre vd
410 lre vr = addAffirms env lconcatr lre v
411
412 provelemas m = [prove m datas env (addAffirms env emptyRec (lemascomp ++
      take n lemas)) (lemas!!n) | n<-[0..((length lemas )-1)]]
413
414 (provedlemas, unprovedlemas) = partition proofCompleto $ provelemas 30
415
416 cliprovedlemas = mapM_ putStrLn $ map (toCLI . headProof) provedlemas
417 cliunprovedlemas = mapM_ putStrLn $ map (toCLI . headProof)
      unprovedlemas

```

A.8 Misc.hs

```

1 module Misc where
2 import Data.Maybe
3 import Data.List
4
5 lookup2 :: Eq a => a -> [(a,b,c)] -> Maybe (b,c)
6 lookup2 x [] = Nothing
7 lookup2 x ((a,b,c):xs) = if x==a then Just (b,c) else lookup2 x xs
8
9 repetir :: a -> (a -> Maybe a) -> Maybe a
10 repetir x f = case f x of {
11     Nothing -> Nothing;
12     Just y -> case repetir y f of {
13         Nothing -> Just y;
14         Just z -> Just z; }}
15
16 repetirc :: a -> (a -> Maybe a) -> a
17 repetirc x f = case repetir x f of {
18     Nothing -> x;
19     Just z -> z;
20     }
21
22 repetirL :: a -> (a -> Maybe a) -> [a]
23 repetirL x f = [x] ++ case f x of {
24     Nothing -> [];
25     Just u -> repetirL u f;

```

```

26         }
27
28 firstJust :: [Maybe a] -> Maybe a
29 firstJust = listToMaybe . catMaybes
30
31 enPos :: Int -> [a] -> Maybe a
32 enPos 0 [] = Nothing
33 enPos 0 (x:xs) = Just x
34 enPos n (x:xs) = enPos (n-1) xs
35
36 firstJustF :: [a -> Maybe b] -> a -> Maybe b
37 firstJustF fs x = firstJust (map (\f -> f x) fs)
38
39 try :: (a -> Maybe a) -> a -> a
40 try f x = case f x of { Just y -> y; Nothing -> x}
41
42 anyJust :: [Maybe a] -> [a]
43 anyJust xs = map (fromJust) $ filter (isJust) xs
44
45 allJust :: [Maybe a] -> Maybe [a]
46 allJust [] = Just []
47 allJust (Nothing:_) = Nothing
48 allJust ((Just u):xs) = case allJust xs of {
49     Just us -> Just (u:us);
50     Nothing -> Nothing;
51     }
52
53 isFunction :: Eq a => Eq b => [(a,b)] -> Bool
54 isFunction [] = True
55 isFunction ((x,y):xs) = case lookup x xs of{
56     Just z -> (z==y)&&(isFunction xs);
57     Nothing -> isFunction xs;
58     }
59
60 zipFunction :: Eq a => [(a,b)] -> [(a,c)] -> [(a,b,c)]
61 zipFunction [] _ = []
62 zipFunction ((a,b):xs) ys = case lookup a ys of {
63     Just c -> (a,b,c):(zipFunction xs ys);
64     Nothing -> error "zipFunction mal formada
65     ";
66     }
67
68 matchLookup :: Eq a => Eq b => [(a,b)] -> [(a,b)] -> Bool
69 matchLookup [] ys = True
70 matchLookup ((a,b):xs) ys = case lookup a ys of {
71     Just w -> w==b && (matchLookup xs ys);
72     Nothing -> False;
73     }
74
75 removeFromFunction :: Eq a => a -> [(a,b)] -> [(a,b)]
76 removeFromFunction a xs = filter ((/=a) . fst) xs

```

A.9 Util.hs

```

1 module Util where
2 import Data.Maybe
3 import Data.List
4 import Debug.Trace
5
6 -- Listas de Pares
7
8 fstS :: [(a,b)] -> [a]
9 fstS = fst . unzip
10
11 snds :: [(a,b)] -> [b]
12 snds = snd . unzip
13
14 flippair :: (a,b) -> (b,a)
15 flippair (a,b) = (b,a)
16
17 mapFst :: (a -> c) -> [(a,b)] -> [(c,b)]
18 mapFst f xs = map (\(fst,snd)->(f fst, snd)) xs
19
20 mapSnd :: (b -> c) -> [(a,b)] -> [(a,c)]
21 mapSnd f xs = map (\(fst,snd)->(fst,f snd)) xs
22
23 mapFstSnd :: (a -> c) -> (b -> d) -> [(a,b)] -> [(c,d)]
24 mapFstSnd f1 f2 xs = map (\(fst,snd)->(f1 fst,f2 snd)) xs
25
26 mapBoth :: (a -> b) -> [(a,a)] -> [(b,b)]
27 mapBoth f = mapFstSnd f f
28
29 cartesianprod :: [a] -> [b] -> [(a,b)]
30 cartesianprod [] _ = []
31 cartesianprod (x:xs) ys = (map (\y -> (x,y)) ys) ++ cartesianprod xs ys
32
33
34
35
36 allJust :: [Maybe a] -> Maybe [a]
37 allJust [] = Just []
38 allJust (Nothing:_) = Nothing
39 allJust ((Just x):xs) = case allJust xs of
40     Nothing -> Nothing
41     Just ys -> Just (x:ys)
42
43 anyJust :: [Maybe a] -> [a]
44 anyJust [] = []
45 anyJust (Nothing:xs) = anyJust xs
46 anyJust ((Just x):xs) = x:(anyJust xs)
47
48
49 while :: (a -> (Bool, a)) -> a -> a
50 while f a = case f a of {
51     (False, b) -> b;
52     (True, b) -> while f b;
53 }
54

```

```

55 whileMaybe :: (a -> Maybe (Bool, a)) -> a -> Maybe a
56 whileMaybe f a = case f a of {
57     Just (False, b) -> Just b;
58     Just (True, b) -> whileMaybe f b;
59     Nothing -> Nothing;
60 }
61
62 revapply :: a -> (a -> b) -> b
63 revapply a f = f a
64
65
66 whileMaybes :: [a -> Maybe (Bool, a)] -> a -> Maybe a
67 whileMaybes fs a = case filter (\y -> (isJust y) && fst(fromJust y)) fsa
    of{
68     [] -> case all isJust fsa of {
69         True -> Just a;
70         False -> Nothing;
71     };
72     ((Just(True,b)):_ ) -> whileMaybes fs b;
73 }
74     where fsa = map (revapply a) fs
75
76 parens :: String -> String
77 parens s = "("++s++")"
78
79 separar :: a -> [a] -> [a]
80 separar a [] = []
81 separar a (x:xs) = x:(concat $ zipWith (\u b -> [u,b]) (repeat a) xs)
82
83 hilar :: [[a]] -> [[a]]
84 hilar [] = [[]]
85 hilar (xs:xss) = [ x:hs | hs <- hilar xss, x <- xs]
86
87 --xs finite, xs infinite
88 hilarI :: [[a]] -> [[a]]
89 hilarI [] = []
90 hilarI [xs] = map (:[]) xs
91 hilarI (xs:ys:yss) = map (\(a,b) -> a:b) (comb xs (hilarI (ys:yss)))
92
93 comb :: [a] -> [b] -> [(a,b)]
94 comb (xs) (ys) = diag $ map (\x -> map (\y -> (x,y)) ys) xs
95
96 dist :: Int -> [a] -> [[[a]]]
97 dist 0 xs = []
98 dist 1 xs = [[xs]]
99 dist n [] = [take n (repeat [])]
100 dist n xs = [(take m xs):ys | m <- [0..(length xs)], ys <- dist (n-1) (
    drop m xs) ]
101
102 dist0 :: Int -> [a] -> [[[a]]]
103 dist0 0 xs = []
104 dist0 1 xs = [[xs]]
105 dist0 n [] = [take n (repeat [])]

```

```

106 dist0 n xs = [ (j):rec | m <- [0..(length xs)], (j,k) <- postake m xs,
    rec <- dist0 (n-1) k]
107
108 postake :: Int -> [a] -> [[a],[a]]
109 postake 0 xs = [([],xs)]
110 postake n [] = []
111 postake n (x:xs) = mapFst (x:) (postake (n-1) xs) ++ mapSnd (x:) (
    postake n xs)
112
113 armarParejas :: [a] -> [(a,a)]
114 armarParejas [] = []
115 armarParejas (x:xs) = (map (\y->(x,y)) xs) ++ (armarParejas xs)
116
117
118
119 lookup2 :: Eq a => a -> [(a,b,c)] -> Maybe (b,c)
120 lookup2 x [] = Nothing
121 lookup2 x ((a,b,c):xs) = if x==a then Just (b,c) else lookup2 x xs
122
123 repetir :: (a -> Maybe a) -> a -> Maybe a
124 repetir f x = case f x of {
125     Nothing -> Nothing;
126     Just y -> case repetir f y of {
127         Nothing -> Just y;
128         Just z -> Just z; }}
129
130 repetirc :: (a -> Maybe a) -> a -> a
131 repetirc f x = case repetir f x of {
132     Nothing -> x;
133     Just z -> z;
134 }
135
136 repetirc' :: Show a => (a -> Maybe a) -> a -> a
137 repetirc' f x = foldr1 (trace . show) ls
138     where ls = repetirL f x
139
140 repetirL :: (a -> Maybe a) -> a -> [a]
141 repetirL f x = [x] ++ case f x of {
142     Nothing -> [];
143     Just u -> repetirL f u;
144 }
145
146 firstJust :: [Maybe a] -> Maybe a
147 firstJust = listToMaybe . catMaybes
148
149 enPos :: Int -> [a] -> Maybe a
150 enPos _ [] = Nothing
151 enPos 0 (x:xs) = Just x
152 enPos n (x:xs) = enPos (n-1) xs
153
154 firstJustF :: [a -> Maybe b] -> a -> Maybe b
155 firstJustF fs x = firstJust (map (\f -> f x) fs)
156
157 try :: (a -> Maybe a) -> a -> a

```

```

158 try f x = case f x of { Just y -> y; Nothing -> x}
159
160
161 isFunction :: Eq a => Eq b => [(a,b)] -> Bool
162 isFunction [] = True
163 isFunction ((x,y):xs) = case lookup x xs of{
164     Just z -> (z==y)&&(isFunction xs);
165     Nothing -> isFunction xs;
166     }
167
168 zipFunction :: Eq a => [(a,b)] -> [(a,c)] -> [(a,b,c)]
169 zipFunction [] _ = []
170 zipFunction ((a,b):xs) ys = case lookup a ys of {
171     Just c -> (a,b,c):(zipFunction xs ys);
172     Nothing -> error "zipFunction mal formada
173     ";
174     }
175
176 matchLookup :: Eq a => Eq b => [(a,b)] -> [(a,b)] -> Bool
177 matchLookup [] ys = True
178 matchLookup ((a,b):xs) ys = case lookup a ys of {
179     Just w -> w==b && (matchLookup xs ys);
180     Nothing -> False;
181     }
182
183 removeFromFunction :: Eq a => a -> [(a,b)] -> [(a,b)]
184 removeFromFunction a xs = filter ((/=a) . fst) xs
185
186 appFst :: (a -> c) -> (a,b) -> (c,b)
187 appFst f (a,b) = (f a, b)
188
189 appSnd :: (b -> c) -> (a,b) -> (a,c)
190 appSnd f (a,b) = (a, f b)
191
192 appBoth :: (a -> c) -> (b -> d) -> (a,b) -> (c,d)
193 appBoth f g (a,b) = (f a, g b)
194
195 appBoth2 :: (a -> b) -> (a,a) -> (b,b)
196 appBoth2 f (a1,a2) = (f a1, f a2)
197
198 oppOrd :: Ordering -> Ordering
199 oppOrd LT = GT
200 oppOrd GT = LT
201 oppOrd EQ = EQ
202
203 compWith :: (a -> Int) -> (a -> a -> Ordering)
204 compWith f x y = compare (f x) (f y)
205
206 deleteFstThat :: (a -> Bool) -> [a] -> [a]
207 deleteFstThat p [] = []
208 deleteFstThat p (x:xs) = if p x then xs else x:(deleteFstThat p xs)
209
210 diag :: [[a]] -> [a]
211 diag xs = diag' 1 xs

```

```
211
212 diag' :: Int -> [[a]] -> [a]
213 diag' n xs = (map head ys) ++ (diag' (n+1) (filter (not . null) ((map
      tail ys) ++ zs)))
214           where (ys, zs) = splitAt n xs
215
216 infiniteMatrix = ([0..]):(map (map (+1)) infiniteMatrix)
217
218 -- diag infiniteMatrix = [0, 1, 1, 2, 2, 2 ...]
```


B – Results

This appendix presents the results obtained by running the code in `Main.hs` from appendix A. Each section presents a function found deemed of interest by the program with a list of theorems proved. To aid the reader we have titled each section with an informal definition of the function.

Contents

B.1	First Search	82
B.1.1	<code>f0</code> $x\ y = x + y$	82
B.1.2	<code>f1</code> $x\ y = x + y + 1$	86
B.1.3	<code>f2</code> $[x_0, x_1, \dots, x_n]\ [y_0, y_1, \dots, y_n] = [x_0, x_1, \dots, x_n, y_0, y_1, \dots, y_n]$	91
B.2	Second Search	92
B.2.1	<code>f3</code> $[x_0, x_1, \dots, x_n] = [x_n, \dots, x_1, x_0]$	92
B.2.2	<code>f4</code> $x\ y = x * y + x$	93
B.2.3	<code>f5</code> $x_ = 2^x - 1$	97
B.2.4	<code>f6</code> $x_ = 2^{x+1} - 1$	99
B.2.5	<code>f7</code> $x\ y = y + (0 + 1 + 2 + \dots + x)$	101
B.2.6	<code>f8</code> $x\ y = 1 + y + (0 + 1 + 2 + \dots + x)$	103
B.2.7	<code>f9</code> $x_ = 3 * 2^x - 2$	105
B.2.8	<code>f10</code> $x\ y = x + y + (0 + 1 + 2 + \dots + x)$	106
B.2.9	<code>f11</code> = <code>f7</code>	108
B.2.10	<code>f12</code> $x\ y = x * y$	110
B.2.11	<code>f13</code> $x\ y = x * y + 1$	114
B.2.12	<code>f14</code> $x\ y = x * y + 2$	117
B.2.13	<code>f15</code> $x_ = 2^{x+1}$	119
B.2.14	<code>f16</code> $x_ = 2^{x+2}$	121
B.2.15	<code>f17</code> = <code>f7</code>	124
B.2.16	<code>f18</code> = <code>f8</code>	126
B.2.17	<code>f19</code> $x\ y = 2 + y + (0 + 1 + 2 + \dots + x)$	128
B.2.18	<code>f20</code> = <code>f6</code>	130
B.2.19	<code>f21</code> = <code>f7</code>	132
B.2.20	<code>f22</code> = <code>f8</code>	134
B.2.21	<code>f23</code> = <code>f19</code>	136
B.2.22	<code>f24</code> $xs\ n = n * ((length\ xs) + 1)$	138
B.2.23	<code>f25</code> $[x_0, x_1, \dots, x_n]\ ys = [x_0 : ys, x_1 : ys, \dots, x_n : ys]$	141
B.2.24	<code>f26</code> $[0, 1, 2, 3]\ _ = [0, 1, 2, 3, 3, 2, 3, 3, 1, 2, 3, 3, 2, 3, 3]$	142
B.2.25	<code>f27</code> $xs\ ys = ys ++ ys ++ ys \dots ++ ys\ ((length\ xs)\ times)$	145
B.2.26	<code>f28</code> = <code>f27</code>	146
B.2.27	<code>f29</code> $[x_0, x_1, \dots, x_n]\ ys = [ys ++ [x_0], ys ++ [x_1], \dots, ys ++ [x_n]]$	149
B.2.28	<code>f30</code> = <code>f25</code>	150

B.1 First Search

B.1.1 $f0\ x\ y = x + y$

$f0 :: C_{-}, R \rightarrow C_{-}, R \rightarrow C_{-}, R$

$f0 = \lambda d \rightarrow \lambda e \rightarrow case\ d\ of\ \{C_{-}, R.0 \rightarrow e; C_{-}, R.1 g \rightarrow C_{-}, R.1 (f0\ g\ e)\}$

$f0\ C_{-}, R.0\ a = a$

$f0\ C_{-}, R.0\ a = a$

{ Compute ; }

$a = a$

{ Triv }

Done.

$f0\ a\ C_{-}, R.0 = a$

$f0\ a\ C_{-}, R.0 = a$

{ Induction on a }

$T) f0\ C_{-}, R.0\ C_{-}, R.0 = C_{-}, R.0$

{ Compute ; }

$C_{-}, R.0 = C_{-}, R.0$

{ Triv }

Done.

$H) f0\ d\ C_{-}, R.0 = d$

$T) f0\ (C_{-}, R.1\ d)\ C_{-}, R.0 = C_{-}, R.1\ d$

{ Compute ; }

$C_{-}, R.1\ (f0\ d\ C_{-}, R.0) = C_{-}, R.1\ d$

{ Fund: $f0\ d\ C_{-}, R.0 = d$; }

$C_{-}, R.1\ d = C_{-}, R.1\ d$

{ Triv }

Done.

$f0\ (C_{-}, R.1\ a)\ b = C_{-}, R.1\ (f0\ b\ a)$

$f0\ (C_{-}, R.1\ a)\ b = C_{-}, R.1\ (f0\ b\ a)$

{ Compute ; }

$C_{-}, R.1\ (f0\ a\ b) = C_{-}, R.1\ (f0\ b\ a)$

{ Induction on a }

$T) C_{-}, R.1\ (f0\ C_{-}, R.0\ b) = C_{-}, R.1\ (f0\ b\ C_{-}, R.0)$

{ Compute ; }

$C_{-}, R.1\ b = C_{-}, R.1\ (f0\ b\ C_{-}, R.0)$

{ ; Fund: $f0\ a\ C_{-}, R.0 = a$ }

$C_{-}, R.1\ b = C_{-}, R.1\ b$

{ Triv }

Done.

$H) C_{-}, R.1\ (f0\ d\ b) = C_{-}, R.1\ (f0\ b\ d)$

$T) C_{-}, R.1\ (f0\ (C_{-}, R.1\ d)\ b) = C_{-}, R.1\ (f0\ b\ (C_{-}, R.1\ d))$

{ Compute ; }
 $C_{-,R.1}(C_{-,R.1}(f0\ d\ b)) = C_{-,R.1}(f0\ b\ (C_{-,R.1}\ d))$
 { Reor: $C_{-,R.1}(f0\ d\ b) = C_{-,R.1}(f0\ b\ d)$; }
 $C_{-,R.1}(C_{-,R.1}(f0\ b\ d)) = C_{-,R.1}(f0\ b\ (C_{-,R.1}\ d))$
 { Induction on b }

$T) C_{-,R.1}(C_{-,R.1}(f0\ C_{-,R.0}\ d)) = C_{-,R.1}(f0\ C_{-,R.0}\ (C_{-,R.1}\ d))$
 { Compute ; Compute }
 $C_{-,R.1}(C_{-,R.1}\ d) = C_{-,R.1}(C_{-,R.1}\ d)$
 { Triv }
 Done.

$H) C_{-,R.1}(C_{-,R.1}(f0\ e\ d)) = C_{-,R.1}(f0\ e\ (C_{-,R.1}\ d))$

$T) C_{-,R.1}(C_{-,R.1}(f0\ (C_{-,R.1}\ e)\ d)) = C_{-,R.1}(f0\ (C_{-,R.1}\ e)\ (C_{-,R.1}\ d))$
 { Compute ; Compute }
 $C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f0\ e\ d))) = C_{-,R.1}(C_{-,R.1}(f0\ e\ (C_{-,R.1}\ d)))$
 { ; Fund: $C_{-,R.1}(f0\ e\ (C_{-,R.1}\ d)) = C_{-,R.1}(C_{-,R.1}(f0\ e\ d))$ }
 $C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f0\ e\ d))) = C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f0\ e\ d)))$
 { Triv }
 Done.

$f0\ a\ (C_{-,R.1}\ b) = C_{-,R.1}(f0\ a\ b)$
 $f0\ a\ (C_{-,R.1}\ b) = C_{-,R.1}(f0\ a\ b)$
 { Induction on b }
 $T) f0\ a\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}(f0\ a\ C_{-,R.0})$
 { ; Fund: $f0\ a\ C_{-,R.0} = a$ }
 $f0\ a\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ a$
 { Induction on a }

$T) f0\ C_{-,R.0}\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ C_{-,R.0}$
 { Compute ; }
 $C_{-,R.1}\ C_{-,R.0} = C_{-,R.1}\ C_{-,R.0}$
 { Triv }
 Done.

$H) f0\ d\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ d$

$T) f0\ (C_{-,R.1}\ d)\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}(C_{-,R.1}\ d)$
 { Compute ; }
 $C_{-,R.1}(f0\ d\ (C_{-,R.1}\ C_{-,R.0})) = C_{-,R.1}(C_{-,R.1}\ d)$
 { Fund: $f0\ d\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ d$; }
 $C_{-,R.1}(C_{-,R.1}\ d) = C_{-,R.1}(C_{-,R.1}\ d)$
 { Triv }

Done.

H) $f0\ a\ (C_,\ R.1\ d) = C_,\ R.1\ (f0\ a\ d)$
 T) $f0\ a\ (C_,\ R.1\ (C_,\ R.1\ d)) = C_,\ R.1\ (f0\ a\ (C_,\ R.1\ d))$
{ ; Fund: $f0\ a\ (C_,\ R.1\ d) = C_,\ R.1\ (f0\ a\ d)$ }
 $f0\ a\ (C_,\ R.1\ (C_,\ R.1\ d)) = C_,\ R.1\ (C_,\ R.1\ (f0\ a\ d))$
{ Induction on a }

T) $f0\ C_,\ R.0\ (C_,\ R.1\ (C_,\ R.1\ d)) = C_,\ R.1\ (C_,\ R.1\ (f0\ C_,\ R.0\ d))$
{ Compute ; Compute }
 $C_,\ R.1\ (C_,\ R.1\ d) = C_,\ R.1\ (C_,\ R.1\ d)$
{ Triv }
Done.

H) $f0\ e\ (C_,\ R.1\ (C_,\ R.1\ d)) = C_,\ R.1\ (C_,\ R.1\ (f0\ e\ d))$

T) $f0\ (C_,\ R.1\ e)\ (C_,\ R.1\ (C_,\ R.1\ d)) = C_,\ R.1\ (C_,\ R.1\ (f0\ (C_,\ R.1\ e)\ d))$
{ Compute ; Compute }
 $C_,\ R.1\ (f0\ e\ (C_,\ R.1\ (C_,\ R.1\ d))) = C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ e\ d)))$
{ Fund: $f0\ e\ (C_,\ R.1\ (C_,\ R.1\ d)) = C_,\ R.1\ (C_,\ R.1\ (f0\ e\ d))$; }
 $C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ e\ d))) = C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ e\ d)))$
{ Triv }
Done.

$f0\ c\ d = f0\ d\ c$

$f0\ c\ d = f0\ d\ c$

{ Induction on c }

T) $f0\ C_,\ R.0\ d = f0\ d\ C_,\ R.0$

{ Compute ; }

$d = f0\ d\ C_,\ R.0$

{ ; Fund: $f0\ a\ C_,\ R.0 = a$ }

$d = d$

{ Triv }

Done.

H) $f0\ e\ d = f0\ d\ e$

T) $f0\ (C_,\ R.1\ e)\ d = f0\ d\ (C_,\ R.1\ e)$

{ Compute ; }

$C_,\ R.1\ (f0\ e\ d) = f0\ d\ (C_,\ R.1\ e)$

{ ; Fund: $f0\ a\ (C_,\ R.1\ b) = C_,\ R.1\ (f0\ a\ b)$ }

$C_,\ R.1\ (f0\ e\ d) = C_,\ R.1\ (f0\ d\ e)$

{ Reor: $f0\ e\ d = f0\ d\ e$; }

$C_,\ R.1\ (f0\ d\ e) = C_,\ R.1\ (f0\ d\ e)$

{ Triv }

Done.

$$f0 (f0 a b) g = f0 (f0 a g) b$$

$$f0 (f0 a b) g = f0 (f0 a g) b$$

{ Induction on a }

$$T) f0 (f0 C_{-}, R.0 b) g = f0 (f0 C_{-}, R.0 g) b$$

{ Compute ; Compute }

$$f0 b g = f0 g b$$

{ ; Reor: $f0 d c = f0 c d$ }

$$f0 b g = f0 b g$$

{ Triv }

Done.

$$H) f0 (f0 d b) g = f0 (f0 d g) b$$

$$T) f0 (f0 (C_{-}, R.1 d) b) g = f0 (f0 (C_{-}, R.1 d) g) b$$

{ Compute ; Compute }

$$f0 (C_{-}, R.1 (f0 d b)) g = f0 (C_{-}, R.1 (f0 d g)) b$$

{ Compute ; Compute }

$$C_{-}, R.1 (f0 (f0 d b) g) = C_{-}, R.1 (f0 (f0 d g) b)$$

{ Reor: $f0 d c = f0 c d$; Reor: $f0 d c = f0 c d$ }

$$C_{-}, R.1 (f0 (f0 d b) g) = C_{-}, R.1 (f0 b (f0 d g))$$

{ Induction on b }

$$T) C_{-}, R.1 (f0 (f0 C_{-}, R.0 d) g) = C_{-}, R.1 (f0 C_{-}, R.0 (f0 d g))$$

{ Compute ; Compute }

$$C_{-}, R.1 (f0 d g) = C_{-}, R.1 (f0 d g)$$

{ Triv }

Done.

$$H) C_{-}, R.1 (f0 (f0 e d) g) = C_{-}, R.1 (f0 e (f0 d g))$$

$$T) C_{-}, R.1 (f0 (f0 (C_{-}, R.1 e) d) g) = C_{-}, R.1 (f0 (C_{-}, R.1 e) (f0 d g))$$

{ Compute ; Compute }

$$C_{-}, R.1 (f0 (C_{-}, R.1 (f0 e d)) g) = C_{-}, R.1 (C_{-}, R.1 (f0 e (f0 d g)))$$

{ Compute ; }

$$C_{-}, R.1 (C_{-}, R.1 (f0 (f0 e d) g)) = C_{-}, R.1 (C_{-}, R.1 (f0 e (f0 d g)))$$

{ Reor: $f0 d c = f0 c d$; Reor: $f0 d c = f0 c d$ }

$$C_{-}, R.1 (C_{-}, R.1 (f0 (f0 d e) g)) = C_{-}, R.1 (C_{-}, R.1 (f0 (f0 d g) e))$$

{ ; Reor: $f0 (f0 d g) b = f0 (f0 d b) g$ }

$$C_{-}, R.1 (C_{-}, R.1 (f0 (f0 d e) g)) = C_{-}, R.1 (C_{-}, R.1 (f0 (f0 d e) g))$$

{ Triv }

Done.

$f0\ b\ (f0\ e\ g) = f0\ (f0\ b\ e)\ g$
 $f0\ b\ (f0\ e\ g) = f0\ (f0\ b\ e)\ g$
 { Induction on b }
T) $f0\ C_{-,R.0}\ (f0\ e\ g) = f0\ (f0\ C_{-,R.0}\ e)\ g$
 { Compute ; Compute }
 $f0\ e\ g = f0\ e\ g$
 { Triv }
 Done.

H) $f0\ d\ (f0\ e\ g) = f0\ (f0\ d\ e)\ g$
T) $f0\ (C_{-,R.1}\ d)\ (f0\ e\ g) = f0\ (f0\ (C_{-,R.1}\ d)\ e)\ g$
 { Compute ; Compute }
 $C_{-,R.1}\ (f0\ d\ (f0\ e\ g)) = f0\ (C_{-,R.1}\ (f0\ d\ e))\ g$
 { ; Compute }
 $C_{-,R.1}\ (f0\ d\ (f0\ e\ g)) = C_{-,R.1}\ (f0\ (f0\ d\ e)\ g)$
 { ; Reor: $f0\ (f0\ d\ e)\ g = f0\ d\ (f0\ e\ g)$ }
 $C_{-,R.1}\ (f0\ d\ (f0\ e\ g)) = C_{-,R.1}\ (f0\ d\ (f0\ e\ g))$
 { Triv }
 Done.

B.1.2 $f1\ x\ y = x + y + 1$

$f1::C_{-,R} \rightarrow C_{-,R} \rightarrow C_{-,R}$
 $f1 = \lambda d \rightarrow \lambda e \rightarrow case\ d\ of\ \{C_{-,R.0} \rightarrow C_{-,R.1}\ e;\ C_{-,R.1}\ g \rightarrow C_{-,R.1}\ (f1\ g\ e)\}$

$f1\ C_{-,R.0}\ a = C_{-,R.1}\ a$

$f1\ C_{-,R.0}\ a = C_{-,R.1}\ a$

{ Compute ; }

$C_{-,R.1}\ a = C_{-,R.1}\ a$

{ Triv }

Done.

$f1\ a\ C_{-,R.0} = C_{-,R.1}\ a$

$f1\ a\ C_{-,R.0} = C_{-,R.1}\ a$

{ Induction on a }

T) $f1\ C_{-,R.0}\ C_{-,R.0} = C_{-,R.1}\ C_{-,R.0}$

{ Compute ; }

$C_{-,R.1}\ C_{-,R.0} = C_{-,R.1}\ C_{-,R.0}$

{ Triv }

Done.

H) $f1\ d\ C_{-,R.0} = C_{-,R.1}\ d$

T) $f1\ (C_{-,R.1}\ d)\ C_{-,R.0} = C_{-,R.1}\ (C_{-,R.1}\ d)$

{ Compute ; }

$C_{-,R.1}\ (f1\ d\ C_{-,R.0}) = C_{-,R.1}\ (C_{-,R.1}\ d)$

{ Fund: $f1\ d\ C_{-,R.0} = C_{-,R.1}\ d$; }

$C_{-,R.1}\ (C_{-,R.1}\ d) = C_{-,R.1}\ (C_{-,R.1}\ d)$

{ Triv }
Done.

$f1 (C_{-}, R.1 a) b = C_{-}, R.1 (f1 b a)$
 $f1 (C_{-}, R.1 a) b = C_{-}, R.1 (f1 b a)$
 { Compute ; }
 $C_{-}, R.1 (f1 a b) = C_{-}, R.1 (f1 b a)$
 { Induction on a }
 $T) C_{-}, R.1 (f1 C_{-}, R.0 b) = C_{-}, R.1 (f1 b C_{-}, R.0)$
 { Compute ; }
 $C_{-}, R.1 (C_{-}, R.1 b) = C_{-}, R.1 (f1 b C_{-}, R.0)$
 { ; Fund: $f1 a C_{-}, R.0 = C_{-}, R.1 a$ }
 $C_{-}, R.1 (C_{-}, R.1 b) = C_{-}, R.1 (C_{-}, R.1 b)$
 { Triv }
Done.

$H) C_{-}, R.1 (f1 d b) = C_{-}, R.1 (f1 b d)$
 $T) C_{-}, R.1 (f1 (C_{-}, R.1 d) b) = C_{-}, R.1 (f1 b (C_{-}, R.1 d))$
 { Compute ; }
 $C_{-}, R.1 (C_{-}, R.1 (f1 d b)) = C_{-}, R.1 (f1 b (C_{-}, R.1 d))$
 { Reor: $C_{-}, R.1 (f1 d b) = C_{-}, R.1 (f1 b d)$; }
 $C_{-}, R.1 (C_{-}, R.1 (f1 b d)) = C_{-}, R.1 (f1 b (C_{-}, R.1 d))$
 { Induction on b }

 $T) C_{-}, R.1 (C_{-}, R.1 (f1 C_{-}, R.0 d)) = C_{-}, R.1 (f1 C_{-}, R.0 (C_{-}, R.1 d))$
 { Compute ; Compute }
 $C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 d)) = C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 d))$
 { Triv }
Done.

$H) C_{-}, R.1 (C_{-}, R.1 (f1 e d)) = C_{-}, R.1 (f1 e (C_{-}, R.1 d))$

 $T) C_{-}, R.1 (C_{-}, R.1 (f1 (C_{-}, R.1 e) d)) = C_{-}, R.1 (f1 (C_{-}, R.1 e) (C_{-}, R.1 d))$
 { Compute ; Compute }
 $C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 (f1 e d))) = C_{-}, R.1 (C_{-}, R.1 (f1 e (C_{-}, R.1 d)))$
 { ; Fund: $C_{-}, R.1 (f1 e (C_{-}, R.1 d)) = C_{-}, R.1 (C_{-}, R.1 (f1 e d))$ }
 $C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 (f1 e d))) = C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 (f1 e d)))$
 { Triv }
Done.

$f1 a (C_{-}, R.1 b) = C_{-}, R.1 (f1 a b)$
 $f1 a (C_{-}, R.1 b) = C_{-}, R.1 (f1 a b)$
 { Induction on b }
 $T) f1 a (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f1 a C_{-}, R.0)$

```

{ ; Fund: f1 a C_,R.0 = C_,R.1 a }
f1 a (C_,R.1 C_,R.0) = C_,R.1 (C_,R.1 a)
{ Induction on a }

```

```

T) f1 C_,R.0 (C_,R.1 C_,R.0) = C_,R.1 (C_,R.1 C_,R.0)
{ Compute ; }
C_,R.1 (C_,R.1 C_,R.0) = C_,R.1 (C_,R.1 C_,R.0)
{ Triv }
Done.

```

```

H) f1 d (C_,R.1 C_,R.0) = C_,R.1 (C_,R.1 d)

```

```

T) f1 (C_,R.1 d) (C_,R.1 C_,R.0) = C_,R.1 (C_,R.1 (C_,R.1 d))
{ Compute ; }
C_,R.1 (f1 d (C_,R.1 C_,R.0)) = C_,R.1 (C_,R.1 (C_,R.1 d))
{ Fund: f1 d (C_,R.1 C_,R.0) = C_,R.1 (C_,R.1 d) ; }
C_,R.1 (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (C_,R.1 d))
{ Triv }
Done.

```

```

H) f1 a (C_,R.1 d) = C_,R.1 (f1 a d)
T) f1 a (C_,R.1 (C_,R.1 d)) = C_,R.1 (f1 a (C_,R.1 d))
{ ; Fund: f1 a (C_,R.1 d) = C_,R.1 (f1 a d) }
f1 a (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (f1 a d))
{ Induction on a }

```

```

T) f1 C_,R.0 (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (f1 C_,R.0 d))
{ Compute ; Compute }
C_,R.1 (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (C_,R.1 d))
{ Triv }
Done.

```

```

H) f1 e (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (f1 e d))

```

```

T) f1 (C_,R.1 e) (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (f1 (C_,R.1 e) d))
{ Compute ; Compute }
C_,R.1 (f1 e (C_,R.1 (C_,R.1 d))) = C_,R.1 (C_,R.1 (C_,R.1 (f1 e d)))
{ Fund: f1 e (C_,R.1 (C_,R.1 d)) = C_,R.1 (C_,R.1 (f1 e d)) ; }
C_,R.1 (C_,R.1 (C_,R.1 (f1 e d))) = C_,R.1 (C_,R.1 (C_,R.1 (f1 e d)))
{ Triv }
Done.

```


$f1\ c\ d = f1\ d\ c$
 $f1\ c\ d = f1\ d\ c$
 { Induction on c }
T) $f1\ C_{-,R.0}\ d = f1\ d\ C_{-,R.0}$
 { Compute ; }
 $C_{-,R.1}\ d = f1\ d\ C_{-,R.0}$
 { ; Fund: $f1\ a\ C_{-,R.0} = C_{-,R.1}\ a$ }
 $C_{-,R.1}\ d = C_{-,R.1}\ d$
 { Triv }
 Done.

H) $f1\ e\ d = f1\ d\ e$
T) $f1\ (C_{-,R.1}\ e)\ d = f1\ d\ (C_{-,R.1}\ e)$
 { Compute ; }
 $C_{-,R.1}\ (f1\ e\ d) = f1\ d\ (C_{-,R.1}\ e)$
 { ; Fund: $f1\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f1\ a\ b)$ }
 $C_{-,R.1}\ (f1\ e\ d) = C_{-,R.1}\ (f1\ d\ e)$
 { Reor: $f1\ e\ d = f1\ d\ e$; }
 $C_{-,R.1}\ (f1\ d\ e) = C_{-,R.1}\ (f1\ d\ e)$
 { Triv }
 Done.

$f1\ (f1\ a\ b)\ g = f1\ (f1\ a\ g)\ b$
 $f1\ (f1\ a\ b)\ g = f1\ (f1\ a\ g)\ b$
 { Induction on a }
T) $f1\ (f1\ C_{-,R.0}\ b)\ g = f1\ (f1\ C_{-,R.0}\ g)\ b$
 { Compute ; Compute }
 $f1\ (C_{-,R.1}\ b)\ g = f1\ (C_{-,R.1}\ g)\ b$
 { Compute ; Compute }
 $C_{-,R.1}\ (f1\ b\ g) = C_{-,R.1}\ (f1\ g\ b)$
 { ; Reor: $f1\ d\ c = f1\ c\ d$ }
 $C_{-,R.1}\ (f1\ b\ g) = C_{-,R.1}\ (f1\ b\ g)$
 { Triv }
 Done.

H) $f1\ (f1\ d\ b)\ g = f1\ (f1\ d\ g)\ b$
T) $f1\ (f1\ (C_{-,R.1}\ d)\ b)\ g = f1\ (f1\ (C_{-,R.1}\ d)\ g)\ b$
 { Compute ; Compute }
 $f1\ (C_{-,R.1}\ (f1\ d\ b))\ g = f1\ (C_{-,R.1}\ (f1\ d\ g))\ b$
 { Compute ; Compute }
 $C_{-,R.1}\ (f1\ (f1\ d\ b)\ g) = C_{-,R.1}\ (f1\ (f1\ d\ g)\ b)$
 { Reor: $f1\ d\ c = f1\ c\ d$; Reor: $f1\ d\ c = f1\ c\ d$ }
 $C_{-,R.1}\ (f1\ (f1\ b\ d)\ g) = C_{-,R.1}\ (f1\ b\ (f1\ d\ g))$
 { Induction on b }

T) $C_{-,R.1}\ (f1\ (f1\ C_{-,R.0}\ d)\ g) = C_{-,R.1}\ (f1\ C_{-,R.0}\ (f1\ d\ g))$
 { Compute ; Compute }

$C_{-,R.1} (f1 (C_{-,R.1} d) g) = C_{-,R.1} (C_{-,R.1} (f1 d g))$
 { Compute ; }
 $C_{-,R.1} (C_{-,R.1} (f1 d g)) = C_{-,R.1} (C_{-,R.1} (f1 d g))$
 { Triv }
 Done.

$H)C_{-,R.1} (f1 (f1 e d) g) = C_{-,R.1} (f1 e (f1 d g))$

$T)C_{-,R.1} (f1 (f1 (C_{-,R.1} e) d) g) = C_{-,R.1} (f1 (C_{-,R.1} e) (f1 d g))$
 { Compute ; Compute }
 $C_{-,R.1} (f1 (C_{-,R.1} (f1 e d)) g) = C_{-,R.1} (C_{-,R.1} (f1 e (f1 d g)))$
 { Compute ; }
 $C_{-,R.1} (C_{-,R.1} (f1 (f1 e d) g)) = C_{-,R.1} (C_{-,R.1} (f1 e (f1 d g)))$
 { Reor: $f1 d c = f1 c d$; Reor: $f1 d c = f1 c d$ }
 $C_{-,R.1} (C_{-,R.1} (f1 (f1 d e) g)) = C_{-,R.1} (C_{-,R.1} (f1 (f1 d g) e))$
 { ; Reor: $f1 (f1 d g) b = f1 (f1 d b) g$ }
 $C_{-,R.1} (C_{-,R.1} (f1 (f1 d e) g)) = C_{-,R.1} (C_{-,R.1} (f1 (f1 d e) g))$
 { Triv }
 Done.

$f1 b (f1 e g) = f1 (f1 b e) g$

$f1 b (f1 e g) = f1 (f1 b e) g$

{ Induction on b }

$T)f1 C_{-,R.0} (f1 e g) = f1 (f1 C_{-,R.0} e) g$

{ Compute ; Compute }

$C_{-,R.1} (f1 e g) = f1 (C_{-,R.1} e) g$

{ ; Compute }

$C_{-,R.1} (f1 e g) = C_{-,R.1} (f1 e g)$

{ Triv }

Done.

$H)f1 d (f1 e g) = f1 (f1 d e) g$

$T)f1 (C_{-,R.1} d) (f1 e g) = f1 (f1 (C_{-,R.1} d) e) g$

{ Compute ; Compute }

$C_{-,R.1} (f1 d (f1 e g)) = f1 (C_{-,R.1} (f1 d e)) g$

{ ; Compute }

$C_{-,R.1} (f1 d (f1 e g)) = C_{-,R.1} (f1 (f1 d e) g)$

{ ; Reor: $f1 (f1 d e) g = f1 d (f1 e g)$ }

$C_{-,R.1} (f1 d (f1 e g)) = C_{-,R.1} (f1 d (f1 e g))$

{ Triv }

Done.

B.1.3 $f2 [x_0, x_1, \dots x_n] [y_0, y_1, \dots y_n] = [x_0, x_1, \dots x_n, y_0, y_1, \dots y_n]$

$f2 :: C_{-, a.R} \rightarrow C_{-, a.R} \rightarrow C_{-, a.R}$

$f2 = \lambda e \rightarrow \lambda g \rightarrow \text{case } e \text{ of } \{ C_{-, a.R.0} \rightarrow g; C_{-, a.R.1hi} \rightarrow C_{-, a.R.1} h (f2 \text{ i } g) \}$

$f2 \ C_{-, a.R.0} \ a = a$

$f2 \ C_{-, a.R.0} \ a = a$

{ Compute ; }

$a = a$

{ Triv }

Done.

$f2 \ a \ C_{-, a.R.0} = a$

$f2 \ a \ C_{-, a.R.0} = a$

{ Induction on a }

$T) f2 \ C_{-, a.R.0} \ C_{-, a.R.0} = C_{-, a.R.0}$

{ Compute ; }

$C_{-, a.R.0} = C_{-, a.R.0}$

{ Triv }

Done.

$H) f2 \ g \ C_{-, a.R.0} = g$

$T) f2 \ (C_{-, a.R.1} \ e \ g) \ C_{-, a.R.0} = C_{-, a.R.1} \ e \ g$

{ Compute ; }

$C_{-, a.R.1} \ e \ (f2 \ g \ C_{-, a.R.0}) = C_{-, a.R.1} \ e \ g$

{ Fund: $f2 \ g \ C_{-, a.R.0} = g$; }

$C_{-, a.R.1} \ e \ g = C_{-, a.R.1} \ e \ g$

{ Triv }

Done.

$f2 \ (C_{-, a.R.1} \ a \ b) \ g = C_{-, a.R.1} \ a \ (f2 \ b \ g)$

$f2 \ (C_{-, a.R.1} \ a \ b) \ g = C_{-, a.R.1} \ a \ (f2 \ b \ g)$

{ Compute ; }

$C_{-, a.R.1} \ a \ (f2 \ b \ g) = C_{-, a.R.1} \ a \ (f2 \ b \ g)$

{ Triv }

Done.

$f2 \ b \ (f2 \ e \ g) = f2 \ (f2 \ b \ e) \ g$

$f2 \ b \ (f2 \ e \ g) = f2 \ (f2 \ b \ e) \ g$

{ Induction on b }

$T) f2 \ C_{-, a.R.0} \ (f2 \ e \ g) = f2 \ (f2 \ C_{-, a.R.0} \ e) \ g$

{ Compute ; Compute }

$f2 \ e \ g = f2 \ e \ g$

{ Triv }

Done.

H) $f2\ i\ (f2\ e\ g) = f2\ (f2\ i\ e)\ g$
T) $f2\ (C_{-}, a.R.1\ h\ i)\ (f2\ e\ g) = f2\ (f2\ (C_{-}, a.R.1\ h\ i)\ e)\ g$
 { Compute ; Compute }
 $C_{-}, a.R.1\ h\ (f2\ i\ (f2\ e\ g)) = f2\ (C_{-}, a.R.1\ h\ (f2\ i\ e))\ g$
 { ; Compute }
 $C_{-}, a.R.1\ h\ (f2\ i\ (f2\ e\ g)) = C_{-}, a.R.1\ h\ (f2\ (f2\ i\ e)\ g)$
 { ; Reor: $f2\ (f2\ i\ e)\ g = f2\ i\ (f2\ e\ g)$ }
 $C_{-}, a.R.1\ h\ (f2\ i\ (f2\ e\ g)) = C_{-}, a.R.1\ h\ (f2\ i\ (f2\ e\ g))$
 { Triv }
 Done.

B.2 Second Search

B.2.1 $f3 = [x_0, x_1, \dots, x_n] = [x_n, \dots, x_1, x_0]$

$f3: C_{-}, a.R \rightarrow C_{-}, a.R$
 $f3 = \lambda d \rightarrow case\ d\ of\ \{C_{-}, a.R.0 \rightarrow C_{-}, a.R.0; C_{-}, a.R.1\ e\ g \rightarrow f2\ (f3\ g)\ (C_{-}, a.R.1\ e\ C_{-}, a.R.0)\}$

$f3\ C_{-}, a.R.0 = C_{-}, a.R.0$

$f3\ C_{-}, a.R.0 = C_{-}, a.R.0$

{ Compute ; }

$C_{-}, a.R.0 = C_{-}, a.R.0$

{ Triv }

Done.

$f3\ (f2\ a\ b) = f2\ (f3\ b)\ (f3\ a)$

$f3\ (f2\ a\ b) = f2\ (f3\ b)\ (f3\ a)$

{ Induction on a }

T) $f3\ (f2\ C_{-}, a.R.0\ b) = f2\ (f3\ b)\ (f3\ C_{-}, a.R.0)$

{ Compute ; Compute }

$f3\ b = f2\ (f3\ b)\ C_{-}, a.R.0$

{ ; Fund: $f2\ a\ C_{-}, a.R.0 = a$ }

$f3\ b = f3\ b$

{ Triv }

Done.

H) $f3\ (f2\ k\ b) = f2\ (f3\ b)\ (f3\ k)$

T) $f3\ (f2\ (C_{-}, a.R.1\ j\ k)\ b) = f2\ (f3\ b)\ (f3\ (C_{-}, a.R.1\ j\ k))$

{ Compute ; Compute }

$f3\ (C_{-}, a.R.1\ j\ (f2\ k\ b)) = f2\ (f3\ b)\ (f2\ (f3\ k)\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0))$

{ Compute ; }

$f2\ (f3\ (f2\ k\ b))\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0) = f2\ (f3\ b)\ (f2\ (f3\ k)\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0))$

{ Fund: $f3\ (f2\ k\ b) = f2\ (f3\ b)\ (f3\ k)$; }

$f2\ (f2\ (f3\ b)\ (f3\ k))\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0) = f2\ (f3\ b)\ (f2\ (f3\ k)\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0))$

{ Reor: $f2\ (f2\ b\ e)\ g = f2\ b\ (f2\ e\ g)$; }

$f2\ (f3\ b)\ (f2\ (f3\ k)\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0)) = f2\ (f3\ b)\ (f2\ (f3\ k)\ (C_{-}, a.R.1\ j\ C_{-}, a.R.0))$

{ Triv }

Done.

```
f3 (f3 a) = a
f3 (f3 a) = a
  { Induction on a }
T) f3 (f3 C-, aR.0) = C-, aR.0
{ Compute ; }
f3 C-, aR.0 = C-, aR.0
{ Compute ; }
C-, aR.0 = C-, aR.0
{ Triv }
Done.
```

```
H) f3 (f3 k) = k
T) f3 (f3 (C-, aR.1 j k)) = C-, aR.1 j k
{ Compute ; }
f3 (f2 (f3 k) (C-, aR.1 j C-, aR.0)) = C-, aR.1 j k
{ Fund: f3 (f2 a b) = f2 (f3 b) (f3 a) ; }
f2 (f3 (C-, aR.1 j C-, aR.0)) (f3 (f3 k)) = C-, aR.1 j k
{ Compute ; }
f2 (f2 (f3 C-, aR.0) (C-, aR.1 j C-, aR.0)) (f3 (f3 k)) = C-, aR.1 j k
{ Compute ; }
f2 (f2 C-, aR.0 (C-, aR.1 j C-, aR.0)) (f3 (f3 k)) = C-, aR.1 j k
{ Compute ; }
f2 (C-, aR.1 j C-, aR.0) (f3 (f3 k)) = C-, aR.1 j k
{ Compute ; }
C-, aR.1 j (f2 C-, aR.0 (f3 (f3 k))) = C-, aR.1 j k
{ Compute ; }
C-, aR.1 j (f3 (f3 k)) = C-, aR.1 j k
{ Fund: f3 (f3 k) = k ; }
C-, aR.1 j k = C-, aR.1 j k
{ Triv }
Done.
```

B.2.2 $f4 x y = x * y + x$

```
f4::C-, R → C-, R → C-, R
f4 = λd → λe → case d of {C-, R.0 → C-, R.0; C-, R.1g → C-, R.1 (f0 e (f4 g e))}

f4 C-, R.0 a = C-, R.0
f4 C-, R.0 a = C-, R.0
  { Compute ; }
C-, R.0 = C-, R.0
  { Triv }
Done.
```

f4 a C_{-,R.0} = a

f4 a C_{-,R.0} = a

{ Induction on a }

T) f4 C_{-,R.0} C_{-,R.0} = C_{-,R.0}

{ Compute ; }

C_{-,R.0} = C_{-,R.0}

{ Triv }

Done.

H) f4 j C_{-,R.0} = j

T) f4 (C_{-,R.1} j) C_{-,R.0} = C_{-,R.1} j

{ Compute ; }

C_{-,R.1} (f0 C_{-,R.0} (f4 j C_{-,R.0})) = C_{-,R.1} j

{ Compute ; }

C_{-,R.1} (f4 j C_{-,R.0}) = C_{-,R.1} j

{ Fund: f4 j C_{-,R.0} = j ; }

C_{-,R.1} j = C_{-,R.1} j

{ Triv }

Done.

f4 a (C_{-,R.1} b) = f0 a (f4 a b)

f4 a (C_{-,R.1} b) = f0 a (f4 a b)

{ Induction on b }

T) f4 a (C_{-,R.1} C_{-,R.0}) = f0 a (f4 a C_{-,R.0})

{ ; Fund: f4 a C_{-,R.0} = a }

f4 a (C_{-,R.1} C_{-,R.0}) = f0 a a

{ Induction on a }

T) f4 C_{-,R.0} (C_{-,R.1} C_{-,R.0}) = f0 C_{-,R.0} C_{-,R.0}

{ Compute ; Compute }

C_{-,R.0} = C_{-,R.0}

{ Triv }

Done.

H) f4 j (C_{-,R.1} C_{-,R.0}) = f0 j j

T) f4 (C_{-,R.1} j) (C_{-,R.1} C_{-,R.0}) = f0 (C_{-,R.1} j) (C_{-,R.1} j)

{ Compute ; Compute }

C_{-,R.1} (f0 (C_{-,R.1} C_{-,R.0}) (f4 j (C_{-,R.1} C_{-,R.0}))) = C_{-,R.1} (f0 j (C_{-,R.1} j))

{ Compute ; }

C_{-,R.1} (C_{-,R.1} (f0 C_{-,R.0} (f4 j (C_{-,R.1} C_{-,R.0})))) = C_{-,R.1} (f0 j (C_{-,R.1} j))

{ Compute ; }

C_{-,R.1} (C_{-,R.1} (f4 j (C_{-,R.1} C_{-,R.0}))) = C_{-,R.1} (f0 j (C_{-,R.1} j))

{ Fund: f4 j (C_{-,R.1} C_{-,R.0}) = f0 j j ; Fund: f0 a (C_{-,R.1} b) = C_{-,R.1} (f0 a b) }

C_{-,R.1} (C_{-,R.1} (f0 j j)) = C_{-,R.1} (C_{-,R.1} (f0 j j))

{ Triv }

Done.

H) $f4\ a\ (C_,\ R.1\ j) = f0\ a\ (f4\ a\ j)$
 T) $f4\ a\ (C_,\ R.1\ (C_,\ R.1\ j)) = f0\ a\ (f4\ a\ (C_,\ R.1\ j))$
{ ; Fund: $f4\ a\ (C_,\ R.1\ j) = f0\ a\ (f4\ a\ j)$ }
 $f4\ a\ (C_,\ R.1\ (C_,\ R.1\ j)) = f0\ a\ (f0\ a\ (f4\ a\ j))$
{ Induction on a }

T) $f4\ C_,\ R.0\ (C_,\ R.1\ (C_,\ R.1\ j)) = f0\ C_,\ R.0\ (f0\ C_,\ R.0\ (f4\ C_,\ R.0\ j))$
{ Compute ; Compute }
 $C_,\ R.0 = f0\ C_,\ R.0\ (f4\ C_,\ R.0\ j)$
{ ; Compute }
 $C_,\ R.0 = f4\ C_,\ R.0\ j$
{ ; Compute }
 $C_,\ R.0 = C_,\ R.0$
{ Triv }
Done.

H) $f4\ k\ (C_,\ R.1\ (C_,\ R.1\ j)) = f0\ k\ (f0\ k\ (f4\ k\ j))$

T) $f4\ (C_,\ R.1\ k)\ (C_,\ R.1\ (C_,\ R.1\ j)) = f0\ (C_,\ R.1\ k)\ (f0\ (C_,\ R.1\ k)\ (f4\ (C_,\ R.1\ k)\ j))$
{ Compute ; Compute }
 $(\forall)(C_,\ R.1\ (f0\ (C_,\ R.1\ (C_,\ R.1\ j))\ (f4\ k\ (C_,\ R.1\ (C_,\ R.1\ j)))))$
 $= C_,\ R.1\ (f0\ k\ (f0\ (C_,\ R.1\ k)\ (f4\ (C_,\ R.1\ k)\ j)))$
{ Compute ; Compute }
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (f0\ (C_,\ R.1\ j)\ (f4\ k\ (C_,\ R.1\ (C_,\ R.1\ j)))))$
 $= C_,\ R.1\ (f0\ k\ (C_,\ R.1\ (f0\ k\ (f4\ (C_,\ R.1\ k)\ j))))$
{ Compute ; Compute }
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j\ (f4\ k\ (C_,\ R.1\ (C_,\ R.1\ j)))))$
 $= C_,\ R.1\ (f0\ k\ (C_,\ R.1\ (f0\ k\ (C_,\ R.1\ (f0\ j\ (f4\ k\ j)))))$
{ Fund: $f4\ k\ (C_,\ R.1\ (C_,\ R.1\ j)) = f0\ k\ (f0\ k\ (f4\ k\ j))$; Fund: $f0\ a\ (C_,\ R.1\ b) = C_,\ R.1\ (f0\ a\ b)$ }
}
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j\ (f0\ k\ (f0\ k\ (f4\ k\ j)))))$
 $= C_,\ R.1\ (C_,\ R.1\ (f0\ k\ (f0\ k\ (C_,\ R.1\ (f0\ j\ (f4\ k\ j)))))$
{ ; Fund: $f0\ a\ (C_,\ R.1\ b) = C_,\ R.1\ (f0\ a\ b)$ }
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j\ (f0\ k\ (f0\ k\ (f4\ k\ j)))))$
 $= C_,\ R.1\ (C_,\ R.1\ (f0\ k\ (C_,\ R.1\ (f0\ k\ (f0\ j\ (f4\ k\ j)))))$
{ ; Fund: $f0\ a\ (C_,\ R.1\ b) = C_,\ R.1\ (f0\ a\ b)$ }
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j\ (f0\ k\ (f0\ k\ (f4\ k\ j)))))$
 $= C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ k\ (f0\ k\ (f0\ j\ (f4\ k\ j)))))$
{ Reor: $f0\ d\ c = f0\ c\ d$; Reor: $f0\ d\ c = f0\ c\ d$ }
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j\ (f0\ k\ (f0\ (f4\ k\ j)\ k)))))$
 $= C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ k\ (f0\ (f0\ j\ (f4\ k\ j)\ k)))))$
{ Reor: $f0\ d\ c = f0\ c\ d$; Reor: $f0\ (f0\ b\ e)\ g = f0\ b\ (f0\ e\ g)$ }
 $(\forall)(C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j\ (f0\ (f0\ (f4\ k\ j)\ k)\ k)))))$
 $= C_,\ R.1\ (C_,\ R.1\ (C_,\ R.1\ (f0\ k\ (f0\ j\ (f0\ (f4\ k\ j)\ k)))))$
{ Reor: $f0\ (f0\ b\ e)\ g = f0\ b\ (f0\ e\ g)$; Reor: $f0\ d\ c = f0\ c\ d$ }

$(\forall)(C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f_0 j (f_0 (f_4 k j) (f_0 k k))))))$
 $= C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f_0 (f_0 j (f_0 (f_4 k j) k) k))))$
 { ; Reor: $f_0 (f_0 b e) g = f_0 b (f_0 e g)$ }
 $(\forall)(C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f_0 j (f_0 (f_4 k j) (f_0 k k))))))$
 $= C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f_0 j (f_0 (f_0 (f_4 k j) k) k))))$
 { ; Reor: $f_0 (f_0 b e) g = f_0 b (f_0 e g)$ }
 $(\forall)(C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f_0 j (f_0 (f_4 k j) (f_0 k k))))))$
 $= C_{-,R.1}(C_{-,R.1}(C_{-,R.1}(f_0 j (f_0 (f_4 k j) (f_0 k k))))))$
 { Triv }
 Done.

$f_4 (f_0 a b) g = f_0 (f_4 a g) (f_4 b g)$
 $f_4 (f_0 a b) g = f_0 (f_4 a g) (f_4 b g)$
 { Induction on a }
 $T) f_4 (f_0 C_{-,R.0} b) g = f_0 (f_4 C_{-,R.0} g) (f_4 b g)$
 { Compute ; Compute }
 $f_4 b g = f_0 C_{-,R.0} (f_4 b g)$
 { ; Compute }
 $f_4 b g = f_4 b g$
 { Triv }
 Done.

$H) f_4 (f_0 j b) g = f_0 (f_4 j g) (f_4 b g)$
 $T) f_4 (f_0 (C_{-,R.1} j) b) g = f_0 (f_4 (C_{-,R.1} j) g) (f_4 b g)$
 { Compute ; Compute }
 $f_4 (C_{-,R.1} (f_0 j b)) g = f_0 (C_{-,R.1} (f_0 g (f_4 j g))) (f_4 b g)$
 { Compute ; Compute }
 $C_{-,R.1} (f_0 g (f_4 (f_0 j b) g)) = C_{-,R.1} (f_0 (f_0 g (f_4 j g)) (f_4 b g))$
 { Fund: $f_4 (f_0 j b) g = f_0 (f_4 j g) (f_4 b g)$; }
 $C_{-,R.1} (f_0 g (f_0 (f_4 j g) (f_4 b g))) = C_{-,R.1} (f_0 (f_0 g (f_4 j g)) (f_4 b g))$
 { Reor: $f_0 d c = f_0 c d$; Reor: $f_0 (f_0 a g) b = f_0 (f_0 a b) g$ }
 $C_{-,R.1} (f_0 g (f_0 (f_4 b g) (f_4 j g))) = C_{-,R.1} (f_0 (f_0 g (f_4 b g)) (f_4 j g))$
 { Reor: $f_0 d c = f_0 c d$; Reor: $f_0 (f_0 b e) g = f_0 b (f_0 e g)$ }
 $C_{-,R.1} (f_0 (f_0 (f_4 b g) (f_4 j g)) g) = C_{-,R.1} (f_0 g (f_0 (f_4 b g) (f_4 j g)))$
 { Reor: $f_0 (f_0 a g) b = f_0 (f_0 a b) g$; Reor: $f_0 d c = f_0 c d$ }
 $C_{-,R.1} (f_0 (f_0 (f_4 b g) g) (f_4 j g)) = C_{-,R.1} (f_0 (f_0 (f_4 b g) (f_4 j g)) g)$
 { Reor: $f_0 (f_0 b e) g = f_0 b (f_0 e g)$; Reor: $f_0 (f_0 a g) b = f_0 (f_0 a b) g$ }
 $C_{-,R.1} (f_0 (f_4 b g) (f_0 g (f_4 j g))) = C_{-,R.1} (f_0 (f_0 (f_4 b g) g) (f_4 j g))$
 { ; Reor: $f_0 (f_0 b e) g = f_0 b (f_0 e g)$ }
 $C_{-,R.1} (f_0 (f_4 b g) (f_0 g (f_4 j g))) = C_{-,R.1} (f_0 (f_4 b g) (f_0 g (f_4 j g)))$
 { Triv }
 Done.

B.2.3 $f5\ x\ _ = 2^x - 1$

$f5::C_{-},R \rightarrow C_{-},R \rightarrow C_{-},R$

$f5 = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{C_{-},R.0 \rightarrow C_{-},R.0; C_{-},R.1g \rightarrow C_{-},R.1 (f0 (f5\ g\ e) (f5\ g\ e))\}$

$f5\ C_{-},R.0\ a = C_{-},R.0$

$f5\ C_{-},R.0\ a = C_{-},R.0$

{ Compute ; }

$C_{-},R.0 = C_{-},R.0$

{ Triv }

Done.

$f5\ a\ (f0\ b\ g) = f5\ a\ b$

$f5\ a\ (f0\ b\ g) = f5\ a\ b$

{ Induction on b }

$T) f5\ a\ (f0\ C_{-},R.0\ g) = f5\ a\ C_{-},R.0$

{ Compute ; }

$f5\ a\ g = f5\ a\ C_{-},R.0$

{ Induction on g }

$T) f5\ a\ C_{-},R.0 = f5\ a\ C_{-},R.0$

{ Triv }

Done.

$H) f5\ a\ j = f5\ a\ C_{-},R.0$

$T) f5\ a\ (C_{-},R.1\ j) = f5\ a\ C_{-},R.0$

{ ; Fund: $f5\ a\ C_{-},R.0 = f5\ a\ j$ }

$f5\ a\ (C_{-},R.1\ j) = f5\ a\ j$

{ Induction on a }

$T) f5\ C_{-},R.0\ (C_{-},R.1\ j) = f5\ C_{-},R.0\ j$

{ Compute ; Compute }

$C_{-},R.0 = C_{-},R.0$

{ Triv }

Done.

$H) f5\ k\ (C_{-},R.1\ j) = f5\ k\ j$

$T) f5\ (C_{-},R.1\ k)\ (C_{-},R.1\ j) = f5\ (C_{-},R.1\ k)\ j$

{ Compute ; Compute }

$C_{-},R.1\ (f0\ (f5\ k\ (C_{-},R.1\ j))\ (f5\ k\ (C_{-},R.1\ j))) = C_{-},R.1\ (f0\ (f5\ k\ j)\ (f5\ k\ j))$

{ Fund: $f5\ k\ (C_{-},R.1\ j) = f5\ k\ j$; }

$C_{-},R.1\ (f0\ (f5\ k\ j)\ (f5\ k\ (C_{-},R.1\ j))) = C_{-},R.1\ (f0\ (f5\ k\ j)\ (f5\ k\ j))$

{ Fund: $f5\ k\ (C_{-},R.1\ j) = f5\ k\ j$; }

$C_{-},R.1\ (f0\ (f5\ k\ j)\ (f5\ k\ j)) = C_{-},R.1\ (f0\ (f5\ k\ j)\ (f5\ k\ j))$

{ Triv }

Done.

H) $f5\ a\ (f0\ j\ g) = f5\ a\ j$
T) $f5\ a\ (f0\ (C_,R.1\ j)\ g) = f5\ a\ (C_,R.1\ j)$
{ Compute ; }
 $f5\ a\ (C_,R.1\ (f0\ j\ g)) = f5\ a\ (C_,R.1\ j)$
{ Reor: $f0\ d\ c = f0\ c\ d$; }
 $f5\ a\ (C_,R.1\ (f0\ g\ j)) = f5\ a\ (C_,R.1\ j)$
{ Induction on g }

T) $f5\ a\ (C_,R.1\ (f0\ C_,R.0\ j)) = f5\ a\ (C_,R.1\ j)$
{ Compute ; }
 $f5\ a\ (C_,R.1\ j) = f5\ a\ (C_,R.1\ j)$
{ Triv }
Done.

H) $f5\ a\ (C_,R.1\ (f0\ k\ j)) = f5\ a\ (C_,R.1\ j)$

T) $f5\ a\ (C_,R.1\ (f0\ (C_,R.1\ k)\ j)) = f5\ a\ (C_,R.1\ j)$
{ Compute ; }
 $f5\ a\ (C_,R.1\ (C_,R.1\ (f0\ k\ j))) = f5\ a\ (C_,R.1\ j)$
{ Reor: $f0\ d\ c = f0\ c\ d$; }
 $f5\ a\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))) = f5\ a\ (C_,R.1\ j)$
{ Induction on a }

T) $f5\ C_,R.0\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))) = f5\ C_,R.0\ (C_,R.1\ j)$
{ Compute ; Compute }
 $C_,R.0 = C_,R.0$
{ Triv }
Done.

H) $(\forall)(f5\ l\ (C_,R.1\ (C_,R.1\ (f0\ j\ k)))$
 $= f5\ l\ (C_,R.1\ j))$

T) $(\forall)(f5\ (C_,R.1\ l)\ (C_,R.1\ (C_,R.1\ (f0\ j\ k)))$
 $= f5\ (C_,R.1\ l)\ (C_,R.1\ j))$
{ Compute ; Compute }
 $(\forall)(C_,R.1\ (f0\ (f5\ l\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))))\ (f5\ l\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))))))$
 $= C_,R.1\ (f0\ (f5\ l\ (C_,R.1\ j))\ (f5\ l\ (C_,R.1\ j)))$
{ Fund: $f5\ l\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))) = f5\ l\ (C_,R.1\ j)$; }
 $(\forall)(C_,R.1\ (f0\ (f5\ l\ (C_,R.1\ j))\ (f5\ l\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))))))$
 $= C_,R.1\ (f0\ (f5\ l\ (C_,R.1\ j))\ (f5\ l\ (C_,R.1\ j)))$
{ Fund: $f5\ l\ (C_,R.1\ (C_,R.1\ (f0\ j\ k))) = f5\ l\ (C_,R.1\ j)$; }
 $(\forall)(C_,R.1\ (f0\ (f5\ l\ (C_,R.1\ j))\ (f5\ l\ (C_,R.1\ j))))$
 $= C_,R.1\ (f0\ (f5\ l\ (C_,R.1\ j))\ (f5\ l\ (C_,R.1\ j)))$
{ Triv }

Done.

f5 b (f5 e g) = f5 b (f5 g e)
f5 b (f5 e g) = f5 b (f5 g e)
 { Induction on b }
T) f5 C_{-,R.0} (f5 e g) = f5 C_{-,R.0} (f5 g e)
 { Compute ; Compute }
C_{-,R.0} = C_{-,R.0}
 { Triv }
Done.

H) f5 j (f5 e g) = f5 j (f5 g e)
T) f5 (C_{-,R.1} j) (f5 e g) = f5 (C_{-,R.1} j) (f5 g e)
 { Compute ; Compute }
C_{-,R.1} (f0 (f5 j (f5 e g)) (f5 j (f5 e g))) = C_{-,R.1} (f0 (f5 j (f5 g e)) (f5 j (f5 g e)))
 { ; Reor: f5 j (f5 g e) = f5 j (f5 e g) }
C_{-,R.1} (f0 (f5 j (f5 e g)) (f5 j (f5 e g))) = C_{-,R.1} (f0 (f5 j (f5 e g)) (f5 j (f5 g e)))
 { ; Reor: f5 j (f5 g e) = f5 j (f5 e g) }
C_{-,R.1} (f0 (f5 j (f5 e g)) (f5 j (f5 e g))) = C_{-,R.1} (f0 (f5 j (f5 e g)) (f5 j (f5 e g)))
 { Triv }
Done.

B.2.4 f6 x _ = 2^{x+1} - 1

f6::C_{-,R} → C_{-,R} → C_{-,R}
f6 = λd → λe → case dof {C_{-,R.0} → C_{-,R.1} C_{-,R.0}; C_{-,R.1}g → C_{-,R.1} (f0 (f6 g e) (f6 g e))}

f6 C_{-,R.0} a = C_{-,R.1} C_{-,R.0}
f6 C_{-,R.0} a = C_{-,R.1} C_{-,R.0}
 { Compute ; }
C_{-,R.1} C_{-,R.0} = C_{-,R.1} C_{-,R.0}
 { Triv }
Done.

f6 a (C_{-,R.1} b) = f6 a (f6 b b)
f6 a (C_{-,R.1} b) = f6 a (f6 b b)
 { Induction on b }
T) f6 a (C_{-,R.1} C_{-,R.0}) = f6 a (f6 C_{-,R.0} C_{-,R.0})
 { ; Compute }
f6 a (C_{-,R.1} C_{-,R.0}) = f6 a (C_{-,R.1} C_{-,R.0})
 { Triv }
Done.

H) f6 a (C_{-,R.1} j) = f6 a (f6 j j)

$T) f6\ a\ (C_,R.1\ (C_,R.1\ j)) = f6\ a\ (f6\ (C_,R.1\ j)\ (C_,R.1\ j))$
 { ; Compute }
 $f6\ a\ (C_,R.1\ (C_,R.1\ j)) = f6\ a\ (C_,R.1\ (f0\ (f6\ j\ (C_,R.1\ j))\ (f6\ j\ (C_,R.1\ j))))$
 { ; Fund: $f6\ a\ (C_,R.1\ j) = f6\ a\ (f6\ j\ j)$ }
 $f6\ a\ (C_,R.1\ (C_,R.1\ j)) = f6\ a\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (C_,R.1\ j))))$
 { ; Fund: $f6\ a\ (C_,R.1\ j) = f6\ a\ (f6\ j\ j)$ }
 $f6\ a\ (C_,R.1\ (C_,R.1\ j)) = f6\ a\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$
 { Induction on a }

 $T) f6\ C_,R.0\ (C_,R.1\ (C_,R.1\ j)) = f6\ C_,R.0\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$
 { Compute ; Compute }
 $C_,R.1\ C_,R.0 = C_,R.1\ C_,R.0$
 { Triv }
 Done.

$H) f6\ k\ (C_,R.1\ (C_,R.1\ j)) = f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$

$T) f6\ (C_,R.1\ k)\ (C_,R.1\ (C_,R.1\ j)) = f6\ (C_,R.1\ k)\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$
 { Compute ; Compute }
 $(\forall\ (C_,R.1\ (f0\ (f6\ k\ (C_,R.1\ (C_,R.1\ j))))\ (f6\ k\ (C_,R.1\ (C_,R.1\ j))))$
 $= C_,R.1\ (f0\ (f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))\ (f6\ j\ (f6\ j\ j))))$
 $(f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$
 { Fund: $(\forall\ (f6\ k\ (C_,R.1\ (C_,R.1\ j))$
 $= f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j)))) ; }$
 $(\forall\ (C_,R.1\ (f0\ (f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))\ (f6\ j\ (f6\ j\ j))))$
 $= C_,R.1\ (f0\ (f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))\ (f6\ j\ (f6\ j\ j))))$
 $(f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$
 { Fund: $(\forall\ (f6\ k\ (C_,R.1\ (C_,R.1\ j))$
 $= f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j)))) ; }$
 $(\forall\ (C_,R.1\ (f0\ (f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))\ (f6\ j\ (f6\ j\ j))))$
 $(f6\ j\ (f6\ j\ j))))$
 $= C_,R.1\ (f0\ (f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))\ (f6\ j\ (f6\ j\ j))))$
 $(f6\ k\ (C_,R.1\ (f0\ (f6\ j\ (f6\ j\ j))\ (f6\ j\ (f6\ j\ j))))$
 { Triv }
 Done.

$f6\ b\ (f6\ e\ g) = f6\ b\ (f6\ g\ e)$

$f6\ b\ (f6\ e\ g) = f6\ b\ (f6\ g\ e)$

{ Induction on b }

$T) f6\ C_,R.0\ (f6\ e\ g) = f6\ C_,R.0\ (f6\ g\ e)$

{ Compute ; Compute }

$C_,R.1\ C_,R.0 = C_,R.1\ C_,R.0$

{ Triv }

Done.

$H) f6\ j\ (f6\ e\ g) = f6\ j\ (f6\ g\ e)$

$T) f6 (C_{-}, R.1 j) (f6 e g) = f6 (C_{-}, R.1 j) (f6 g e)$
 { Compute ; Compute }
 $C_{-}, R.1 (f0 (f6 j (f6 e g)) (f6 j (f6 e g))) = C_{-}, R.1 (f0 (f6 j (f6 g e)) (f6 j (f6 g e)))$
 { ; Reor: $f6 j (f6 g e) = f6 j (f6 e g)$ }
 $C_{-}, R.1 (f0 (f6 j (f6 e g)) (f6 j (f6 e g))) = C_{-}, R.1 (f0 (f6 j (f6 e g)) (f6 j (f6 g e)))$
 { ; Reor: $f6 j (f6 g e) = f6 j (f6 e g)$ }
 $C_{-}, R.1 (f0 (f6 j (f6 e g)) (f6 j (f6 e g))) = C_{-}, R.1 (f0 (f6 j (f6 e g)) (f6 j (f6 e g)))$
 { Triv }
 Done.

B.2.5 $f7 x y = y + (0 + 1 + 2 + \dots + x)$

$f7 :: C_{-}, R \rightarrow C_{-}, R \rightarrow C_{-}, R$
 $f7 = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{ C_{-}, R.0 \rightarrow e ; C_{-}, R.1 g \rightarrow C_{-}, R.1 (f0 (f7 g e) g) \}$

$f7 C_{-}, R.0 a = a$

$f7 C_{-}, R.0 a = a$
 { Compute ; }
 $a = a$
 { Triv }
 Done.

$f7 a (C_{-}, R.1 b) = C_{-}, R.1 (f7 a b)$

$f7 a (C_{-}, R.1 b) = C_{-}, R.1 (f7 a b)$
 { Induction on b }

$T) f7 a (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f7 a C_{-}, R.0)$
 { Induction on a }

$T) f7 C_{-}, R.0 (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f7 C_{-}, R.0 C_{-}, R.0)$
 { Compute ; Compute }
 $C_{-}, R.1 C_{-}, R.0 = C_{-}, R.1 C_{-}, R.0$
 { Triv }
 Done.

$H) f7 j (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f7 j C_{-}, R.0)$

$T) f7 (C_{-}, R.1 j) (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f7 (C_{-}, R.1 j) C_{-}, R.0)$
 { Compute ; Compute }
 $C_{-}, R.1 (f0 (f7 j (C_{-}, R.1 C_{-}, R.0)) j) = C_{-}, R.1 (C_{-}, R.1 (f0 (f7 j C_{-}, R.0) j))$
 { Reor: $f0 d c = f0 c d$; Reor: $f0 d c = f0 c d$ }
 $C_{-}, R.1 (f0 j (f7 j (C_{-}, R.1 C_{-}, R.0))) = C_{-}, R.1 (C_{-}, R.1 (f0 j (f7 j C_{-}, R.0)))$
 { Reor: $f7 j (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f7 j C_{-}, R.0)$; }
 $C_{-}, R.1 (f0 j (C_{-}, R.1 (f7 j C_{-}, R.0))) = C_{-}, R.1 (C_{-}, R.1 (f0 j (f7 j C_{-}, R.0)))$
 { Fund: $f0 a (C_{-}, R.1 b) = C_{-}, R.1 (f0 a b)$; }
 $C_{-}, R.1 (C_{-}, R.1 (f0 j (f7 j C_{-}, R.0))) = C_{-}, R.1 (C_{-}, R.1 (f0 j (f7 j C_{-}, R.0)))$
 { Triv }

Done.

H) $f7\ a\ (C_{-}, R.1\ j) = C_{-}, R.1\ (f7\ a\ j)$
T) $f7\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (f7\ a\ (C_{-}, R.1\ j))$
{ ; Fund: $f7\ a\ (C_{-}, R.1\ j) = C_{-}, R.1\ (f7\ a\ j)$ }
 $f7\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f7\ a\ j))$
{ Induction on a }

T) $f7\ C_{-}, R.0\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f7\ C_{-}, R.0\ j))$
{ Compute ; Compute }
 $C_{-}, R.1\ (C_{-}, R.1\ j) = C_{-}, R.1\ (C_{-}, R.1\ j)$
{ Triv }
Done.

H) $f7\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f7\ k\ j))$
T) $f7\ (C_{-}, R.1\ k)\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f7\ (C_{-}, R.1\ k)\ j))$
{ Compute ; Compute }
 $C_{-}, R.1\ (f0\ (f7\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j))))\ k = C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f0\ (f7\ k\ j)\ k)))$
{ Fund: $f7\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f7\ k\ j))$; }
 $C_{-}, R.1\ (f0\ (C_{-}, R.1\ (C_{-}, R.1\ (f7\ k\ j))))\ k = C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f0\ (f7\ k\ j)\ k)))$
{ Compute ; }
 $C_{-}, R.1\ (C_{-}, R.1\ (f0\ (C_{-}, R.1\ (f7\ k\ j)\ k))) = C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f0\ (f7\ k\ j)\ k)))$
{ Compute ; }
 $C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f0\ (f7\ k\ j)\ k))) = C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f0\ (f7\ k\ j)\ k)))$
{ Triv }
Done.

$f7\ a\ (f0\ b\ g) = f0\ (f7\ a\ g)\ b$
 $f7\ a\ (f0\ b\ g) = f0\ (f7\ a\ g)\ b$
{ Induction on b }
T) $f7\ a\ (f0\ C_{-}, R.0\ g) = f0\ (f7\ a\ g)\ C_{-}, R.0$
{ Compute ; }
 $f7\ a\ g = f0\ (f7\ a\ g)\ C_{-}, R.0$
{ ; Fund: $f0\ a\ C_{-}, R.0 = a$ }
 $f7\ a\ g = f7\ a\ g$
{ Triv }
Done.

H) $f7\ a\ (f0\ j\ g) = f0\ (f7\ a\ g)\ j$
T) $f7\ a\ (f0\ (C_{-}, R.1\ j)\ g) = f0\ (f7\ a\ g)\ (C_{-}, R.1\ j)$
{ Compute ; }
 $f7\ a\ (C_{-}, R.1\ (f0\ j\ g)) = f0\ (f7\ a\ g)\ (C_{-}, R.1\ j)$
{ Fund: $f7\ a\ (C_{-}, R.1\ b) = C_{-}, R.1\ (f7\ a\ b)$; Fund: $f0\ a\ (C_{-}, R.1\ b) = C_{-}, R.1\ (f0\ a\ b)$ }
 $C_{-}, R.1\ (f7\ a\ (f0\ j\ g)) = C_{-}, R.1\ (f0\ (f7\ a\ g)\ j)$

{ Fund: $f7\ a\ (f0\ j\ g) = f0\ (f7\ a\ g)\ j$; }
 $C_{-,R.1}\ (f0\ (f7\ a\ g)\ j) = C_{-,R.1}\ (f0\ (f7\ a\ g)\ j)$
 { Triv }
 Done.

B.2.6 $f8\ x\ y = 1 + y + (0 + 1 + 2 + \dots + x)$

$f8::C_{-,R} \rightarrow C_{-,R} \rightarrow C_{-,R}$
 $f8 = \lambda d \rightarrow \lambda e \rightarrow case\ d\ of\ \{C_{-,R.0} \rightarrow C_{-,R.1}\ e; C_{-,R.1}g \rightarrow C_{-,R.1}\ (f0\ (f8\ g\ e)\ g)\}$

$f8\ C_{-,R.0}\ a = C_{-,R.1}\ a$

$f8\ C_{-,R.0}\ a = C_{-,R.1}\ a$

{ Compute ; }

$C_{-,R.1}\ a = C_{-,R.1}\ a$

{ Triv }

Done.

$f8\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f8\ a\ b)$

$f8\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f8\ a\ b)$

{ Induction on b }

$T) f8\ a\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (f8\ a\ C_{-,R.0})$

{ Induction on a }

$T) f8\ C_{-,R.0}\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (f8\ C_{-,R.0}\ C_{-,R.0})$

{ Compute ; Compute }

$C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0})$

{ Triv }

Done.

$H) f8\ j\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (f8\ j\ C_{-,R.0})$

$T) f8\ (C_{-,R.1}\ j)\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (f8\ (C_{-,R.1}\ j)\ C_{-,R.0})$

{ Compute ; Compute }

$C_{-,R.1}\ (f0\ (f8\ j\ (C_{-,R.1}\ C_{-,R.0}))\ j) = C_{-,R.1}\ (C_{-,R.1}\ (f0\ (f8\ j\ C_{-,R.0})\ j))$

{ Reor: $f0\ d\ c = f0\ c\ d$; Reor: $f0\ d\ c = f0\ c\ d$ }

$C_{-,R.1}\ (f0\ j\ (f8\ j\ (C_{-,R.1}\ C_{-,R.0}))) = C_{-,R.1}\ (C_{-,R.1}\ (f0\ j\ (f8\ j\ C_{-,R.0})))$

{ Reor: $f8\ j\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (f8\ j\ C_{-,R.0})$; }

$C_{-,R.1}\ (f0\ j\ (C_{-,R.1}\ (f8\ j\ C_{-,R.0}))) = C_{-,R.1}\ (C_{-,R.1}\ (f0\ j\ (f8\ j\ C_{-,R.0})))$

{ Fund: $f0\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f0\ a\ b)$; }

$C_{-,R.1}\ (C_{-,R.1}\ (f0\ j\ (f8\ j\ C_{-,R.0}))) = C_{-,R.1}\ (C_{-,R.1}\ (f0\ j\ (f8\ j\ C_{-,R.0})))$

{ Triv }

Done.

$H) f8\ a\ (C_{-,R.1}\ j) = C_{-,R.1}\ (f8\ a\ j)$

$T) f8\ a\ (C_{-,R.1}\ (C_{-,R.1}\ j)) = C_{-,R.1}\ (f8\ a\ (C_{-,R.1}\ j))$

{ ; Fund: $f8\ a\ (C_{-},R.1\ j) = C_{-},R.1\ (f8\ a\ j)$ }
 $f8\ a\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f8\ a\ j))$
 { Induction on a }

T) $f8\ C_{-},R.0\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f8\ C_{-},R.0\ j))$
 { Compute ; Compute }
 $C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ j))$
 { Triv }
 Done.

H) $f8\ k\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f8\ k\ j))$

T) $f8\ (C_{-},R.1\ k)\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f8\ (C_{-},R.1\ k)\ j))$
 { Compute ; Compute }
 $C_{-},R.1\ (f0\ (f8\ k\ (C_{-},R.1\ (C_{-},R.1\ j))))\ k = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f8\ k\ j)\ k)))$
 { Fund: $f8\ k\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f8\ k\ j))$; }
 $C_{-},R.1\ (f0\ (C_{-},R.1\ (C_{-},R.1\ (f8\ k\ j))))\ k = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f8\ k\ j)\ k)))$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ (f0\ (C_{-},R.1\ (f8\ k\ j)\ k))) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f8\ k\ j)\ k)))$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f8\ k\ j)\ k))) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f8\ k\ j)\ k)))$
 { Triv }
 Done.

$f8\ a\ (f0\ b\ g) = f0\ (f8\ a\ g)\ b$
 $f8\ a\ (f0\ b\ g) = f0\ (f8\ a\ g)\ b$
 { Induction on b }
T) $f8\ a\ (f0\ C_{-},R.0\ g) = f0\ (f8\ a\ g)\ C_{-},R.0$
 { Compute ; }
 $f8\ a\ g = f0\ (f8\ a\ g)\ C_{-},R.0$
 { ; Fund: $f0\ a\ C_{-},R.0 = a$ }
 $f8\ a\ g = f8\ a\ g$
 { Triv }
 Done.

H) $f8\ a\ (f0\ j\ g) = f0\ (f8\ a\ g)\ j$
T) $f8\ a\ (f0\ (C_{-},R.1\ j)\ g) = f0\ (f8\ a\ g)\ (C_{-},R.1\ j)$
 { Compute ; }
 $f8\ a\ (C_{-},R.1\ (f0\ j\ g)) = f0\ (f8\ a\ g)\ (C_{-},R.1\ j)$
 { Fund: $f8\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f8\ a\ b)$; Fund: $f0\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f0\ a\ b)$ }
 $C_{-},R.1\ (f8\ a\ (f0\ j\ g)) = C_{-},R.1\ (f0\ (f8\ a\ g)\ j)$
 { Fund: $f8\ a\ (f0\ j\ g) = f0\ (f8\ a\ g)\ j$; }
 $C_{-},R.1\ (f0\ (f8\ a\ g)\ j) = C_{-},R.1\ (f0\ (f8\ a\ g)\ j)$
 { Triv }
 Done.

B.2.7 $f9\ x\ _ = 3 * 2^x - 2$

$f9 :: C_{-}, R \rightarrow C_{-}, R \rightarrow C_{-}, R$

$f9 = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{C_{-}, R.0 \rightarrow C_{-}, R.1\ C_{-}, R.0; C_{-}, R.1g \rightarrow C_{-}, R.1\ (f1\ (f9\ g\ e)\ (f9\ g\ e))\}$

$f9\ C_{-}, R.0\ a = C_{-}, R.1\ C_{-}, R.0$

$f9\ C_{-}, R.0\ a = C_{-}, R.1\ C_{-}, R.0$

{ Compute ; }

$C_{-}, R.1\ C_{-}, R.0 = C_{-}, R.1\ C_{-}, R.0$

{ Triv }

Done.

$f9\ a\ (C_{-}, R.1\ b) = f9\ a\ (f9\ b\ b)$

$f9\ a\ (C_{-}, R.1\ b) = f9\ a\ (f9\ b\ b)$

{ Induction on b }

$T) f9\ a\ (C_{-}, R.1\ C_{-}, R.0) = f9\ a\ (f9\ C_{-}, R.0\ C_{-}, R.0)$

{ ; Compute }

$f9\ a\ (C_{-}, R.1\ C_{-}, R.0) = f9\ a\ (C_{-}, R.1\ C_{-}, R.0)$

{ Triv }

Done.

$H) f9\ a\ (C_{-}, R.1\ j) = f9\ a\ (f9\ j\ j)$

$T) f9\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ a\ (f9\ (C_{-}, R.1\ j)\ (C_{-}, R.1\ j))$

{ ; Compute }

$f9\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ a\ (C_{-}, R.1\ (f1\ (f9\ j\ (C_{-}, R.1\ j))\ (f9\ j\ (C_{-}, R.1\ j))))$

{ ; Fund: $f9\ a\ (C_{-}, R.1\ j) = f9\ a\ (f9\ j\ j)$ }

$f9\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ a\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (C_{-}, R.1\ j))))$

{ ; Fund: $f9\ a\ (C_{-}, R.1\ j) = f9\ a\ (f9\ j\ j)$ }

$f9\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ a\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j))))$

{ Induction on a }

$T) f9\ C_{-}, R.0\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ C_{-}, R.0\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j))))$

{ Compute ; Compute }

$C_{-}, R.1\ C_{-}, R.0 = C_{-}, R.1\ C_{-}, R.0$

{ Triv }

Done.

$H) f9\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ k\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j))))$

$T) f9\ (C_{-}, R.1\ k)\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = f9\ (C_{-}, R.1\ k)\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j))))$

{ Compute ; Compute }

$(\forall) (C_{-}, R.1\ (f1\ (f9\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j))))\ (f9\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j))))$

$= C_{-}, R.1\ (f1\ (f9\ k\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j))))$

$(f9\ k\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j))))$

{ Fund: $(\forall) (f9\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)))$

$= f9\ k\ (C_{-}, R.1\ (f1\ (f9\ j\ (f9\ j\ j))\ (f9\ j\ (f9\ j\ j)))) ; }$

```

(∀)(C-,R.1 (f1 (f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j)))) (f9 k (C-,R.1 (C-,R.1 j))))))
  = C-,R.1 (f1 (f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j)))) (f9 j (f9 j j))))
(f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j))))))
{ Fund: (∀)(f9 k (C-,R.1 (C-,R.1 j))
  = f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j)))) ; }
(∀)(C-,R.1 (f1 (f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j)))) (f9 k (C-,R.1 (f1 (f9 j (f9 j j))
(f9 j (f9 j j))))))
  = C-,R.1 (f1 (f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j)))) (f9 j (f9 j j))))
(f9 k (C-,R.1 (f1 (f9 j (f9 j j)) (f9 j (f9 j j))))))
{ Triv }
Done.

```

```

f9 b (f9 e g) = f9 b (f9 g e)
f9 b (f9 e g) = f9 b (f9 g e)
  { Induction on b }
T) f9 C-,R.0 (f9 e g) = f9 C-,R.0 (f9 g e)
{ Compute ; Compute }
C-,R.1 C-,R.0 = C-,R.1 C-,R.0
{ Triv }
Done.

```

```

H) f9 j (f9 e g) = f9 j (f9 g e)
T) f9 (C-,R.1 j) (f9 e g) = f9 (C-,R.1 j) (f9 g e)
{ Compute ; Compute }
C-,R.1 (f1 (f9 j (f9 e g)) (f9 j (f9 e g))) = C-,R.1 (f1 (f9 j (f9 g e)) (f9 j (f9 g e)))
{ ; Reor: f9 j (f9 g e) = f9 j (f9 e g) }
C-,R.1 (f1 (f9 j (f9 e g)) (f9 j (f9 e g))) = C-,R.1 (f1 (f9 j (f9 e g)) (f9 j (f9 g e)))
{ ; Reor: f9 j (f9 g e) = f9 j (f9 e g) }
C-,R.1 (f1 (f9 j (f9 e g)) (f9 j (f9 e g))) = C-,R.1 (f1 (f9 j (f9 e g)) (f9 j (f9 e g)))
{ Triv }
Done.

```

B.2.8 $f_{10} x y = x + y + (0 + 1 + 2 + \dots + x)$

```

f10::C-,R → C-,R → C-,R
f10 = λd → λe → case d of {C-,R.0 → e; C-,R.1g → C-,R.1 (f1 (f10 g e) g)}

```

```

f10 C-,R.0 a = a
f10 C-,R.0 a = a
  { Compute ; }
a = a
  { Triv }
Done.

```

$f10\ a\ (C_{-}, R.1\ b) = C_{-}, R.1\ (f10\ a\ b)$
 $f10\ a\ (C_{-}, R.1\ b) = C_{-}, R.1\ (f10\ a\ b)$
 { Induction on b }
T) $f10\ a\ (C_{-}, R.1\ C_{-}, R.0) = C_{-}, R.1\ (f10\ a\ C_{-}, R.0)$
 { Induction on a }

T) $f10\ C_{-}, R.0\ (C_{-}, R.1\ C_{-}, R.0) = C_{-}, R.1\ (f10\ C_{-}, R.0\ C_{-}, R.0)$
 { Compute ; Compute }
 $C_{-}, R.1\ C_{-}, R.0 = C_{-}, R.1\ C_{-}, R.0$
 { Triv }
 Done.

H) $f10\ j\ (C_{-}, R.1\ C_{-}, R.0) = C_{-}, R.1\ (f10\ j\ C_{-}, R.0)$

T) $f10\ (C_{-}, R.1\ j)\ (C_{-}, R.1\ C_{-}, R.0) = C_{-}, R.1\ (f10\ (C_{-}, R.1\ j)\ C_{-}, R.0)$
 { Compute ; Compute }
 $C_{-}, R.1\ (f1\ (f10\ j\ (C_{-}, R.1\ C_{-}, R.0))\ j) = C_{-}, R.1\ (C_{-}, R.1\ (f1\ (f10\ j\ C_{-}, R.0)\ j))$
 { Reor: $f1\ d\ c = f1\ c\ d$; Reor: $f1\ d\ c = f1\ c\ d$ }
 $C_{-}, R.1\ (f1\ j\ (f10\ j\ (C_{-}, R.1\ C_{-}, R.0))) = C_{-}, R.1\ (C_{-}, R.1\ (f1\ j\ (f10\ j\ C_{-}, R.0)))$
 { Reor: $f10\ j\ (C_{-}, R.1\ C_{-}, R.0) = C_{-}, R.1\ (f10\ j\ C_{-}, R.0)$; }
 $C_{-}, R.1\ (f1\ j\ (C_{-}, R.1\ (f10\ j\ C_{-}, R.0))) = C_{-}, R.1\ (C_{-}, R.1\ (f1\ j\ (f10\ j\ C_{-}, R.0)))$
 { Fund: $f1\ a\ (C_{-}, R.1\ b) = C_{-}, R.1\ (f1\ a\ b)$; }
 $C_{-}, R.1\ (C_{-}, R.1\ (f1\ j\ (f10\ j\ C_{-}, R.0))) = C_{-}, R.1\ (C_{-}, R.1\ (f1\ j\ (f10\ j\ C_{-}, R.0)))$
 { Triv }
 Done.

H) $f10\ a\ (C_{-}, R.1\ j) = C_{-}, R.1\ (f10\ a\ j)$
T) $f10\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (f10\ a\ (C_{-}, R.1\ j))$
 { ; Fund: $f10\ a\ (C_{-}, R.1\ j) = C_{-}, R.1\ (f10\ a\ j)$ }
 $f10\ a\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f10\ a\ j))$
 { Induction on a }

T) $f10\ C_{-}, R.0\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f10\ C_{-}, R.0\ j))$
 { Compute ; Compute }
 $C_{-}, R.1\ (C_{-}, R.1\ j) = C_{-}, R.1\ (C_{-}, R.1\ j)$
 { Triv }
 Done.

H) $f10\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f10\ k\ j))$

T) $f10\ (C_{-}, R.1\ k)\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f10\ (C_{-}, R.1\ k)\ j))$
 { Compute ; Compute }
 $C_{-}, R.1\ (f1\ (f10\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)))\ k) = C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f1\ (f10\ k\ j)\ k)))$
 { Fund: $f10\ k\ (C_{-}, R.1\ (C_{-}, R.1\ j)) = C_{-}, R.1\ (C_{-}, R.1\ (f10\ k\ j))$; }
 $C_{-}, R.1\ (f1\ (C_{-}, R.1\ (C_{-}, R.1\ (f10\ k\ j)))\ k) = C_{-}, R.1\ (C_{-}, R.1\ (C_{-}, R.1\ (f1\ (f10\ k\ j)\ k)))$
 { Compute ; }

$C_{-,R.1} (C_{-,R.1} (f1 (C_{-,R.1} (f10 k j)) k)) = C_{-,R.1} (C_{-,R.1} (C_{-,R.1} (f1 (f10 k j) k)))$
 { Compute ; }
 $C_{-,R.1} (C_{-,R.1} (C_{-,R.1} (f1 (f10 k j) k))) = C_{-,R.1} (C_{-,R.1} (C_{-,R.1} (f1 (f10 k j) k)))$
 { Triv }
 Done.

$f10 a (f1 b g) = f1 (f10 a g) b$
 $f10 a (f1 b g) = f1 (f10 a g) b$
 { Induction on b }
T) $f10 a (f1 C_{-,R.0} g) = f1 (f10 a g) C_{-,R.0}$
 { Compute ; }
 $f10 a (C_{-,R.1} g) = f1 (f10 a g) C_{-,R.0}$
 { Fund: $f10 a (C_{-,R.1} b) = C_{-,R.1} (f10 a b)$; Fund: $f1 a C_{-,R.0} = C_{-,R.1} a$ }
 $C_{-,R.1} (f10 a g) = C_{-,R.1} (f10 a g)$
 { Triv }
 Done.

H) $f10 a (f1 j g) = f1 (f10 a g) j$
T) $f10 a (f1 (C_{-,R.1} j) g) = f1 (f10 a g) (C_{-,R.1} j)$
 { Compute ; }
 $f10 a (C_{-,R.1} (f1 j g)) = f1 (f10 a g) (C_{-,R.1} j)$
 { Fund: $f10 a (C_{-,R.1} b) = C_{-,R.1} (f10 a b)$; Fund: $f1 a (C_{-,R.1} b) = C_{-,R.1} (f1 a b)$ }
 $C_{-,R.1} (f10 a (f1 j g)) = C_{-,R.1} (f1 (f10 a g) j)$
 { Fund: $f10 a (f1 j g) = f1 (f10 a g) j$; }
 $C_{-,R.1} (f1 (f10 a g) j) = C_{-,R.1} (f1 (f10 a g) j)$
 { Triv }
 Done.

B.2.9 $f11 = f7$

$f11::C_{-,R} \rightarrow C_{-,R} \rightarrow C_{-,R}$
 $f11 = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{C_{-,R.0} \rightarrow C_{-,R.1} e; C_{-,R.1} g \rightarrow C_{-,R.1} (f1 (f11 g e) g)\}$

$f11 C_{-,R.0} a = C_{-,R.1} a$
 $f11 C_{-,R.0} a = C_{-,R.1} a$
 { Compute ; }
 $C_{-,R.1} a = C_{-,R.1} a$
 { Triv }
 Done.

$f11 a (C_{-,R.1} b) = C_{-,R.1} (f11 a b)$
 $f11 a (C_{-,R.1} b) = C_{-,R.1} (f11 a b)$
 { Induction on b }
T) $f11 a (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} (f11 a C_{-,R.0})$
 { Induction on a }

T) $f11\ C_{-},R.0\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (f11\ C_{-},R.0\ C_{-},R.0)$
 { Compute ; Compute }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Triv }
 Done.

H) $f11\ j\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (f11\ j\ C_{-},R.0)$

T) $f11\ (C_{-},R.1\ j)\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (f11\ (C_{-},R.1\ j)\ C_{-},R.0)$
 { Compute ; Compute }
 $C_{-},R.1\ (f1\ (f11\ j\ (C_{-},R.1\ C_{-},R.0))\ j) = C_{-},R.1\ (C_{-},R.1\ (f1\ (f11\ j\ C_{-},R.0)\ j))$
 { Reor: $f1\ d\ c = f1\ c\ d$; Reor: $f1\ d\ c = f1\ c\ d$ }
 $C_{-},R.1\ (f1\ j\ (f11\ j\ (C_{-},R.1\ C_{-},R.0))) = C_{-},R.1\ (C_{-},R.1\ (f1\ j\ (f11\ j\ C_{-},R.0)))$
 { Reor: $f11\ j\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (f11\ j\ C_{-},R.0)$; }
 $C_{-},R.1\ (f1\ j\ (C_{-},R.1\ (f11\ j\ C_{-},R.0))) = C_{-},R.1\ (C_{-},R.1\ (f1\ j\ (f11\ j\ C_{-},R.0)))$
 { Fund: $f1\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f1\ a\ b)$; }
 $C_{-},R.1\ (C_{-},R.1\ (f1\ j\ (f11\ j\ C_{-},R.0))) = C_{-},R.1\ (C_{-},R.1\ (f1\ j\ (f11\ j\ C_{-},R.0)))$
 { Triv }
 Done.

H) $f11\ a\ (C_{-},R.1\ j) = C_{-},R.1\ (f11\ a\ j)$
T) $f11\ a\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (f11\ a\ (C_{-},R.1\ j))$
 { ; Fund: $f11\ a\ (C_{-},R.1\ j) = C_{-},R.1\ (f11\ a\ j)$ }
 $f11\ a\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f11\ a\ j))$
 { Induction on a }

T) $f11\ C_{-},R.0\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f11\ C_{-},R.0\ j))$
 { Compute ; Compute }
 $C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ j))$
 { Triv }
 Done.

H) $f11\ k\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f11\ k\ j))$

T) $f11\ (C_{-},R.1\ k)\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f11\ (C_{-},R.1\ k)\ j))$
 { Compute ; Compute }
 $C_{-},R.1\ (f1\ (f11\ k\ (C_{-},R.1\ (C_{-},R.1\ j)))\ k) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f1\ (f11\ k\ j)\ k)))$
 { Fund: $f11\ k\ (C_{-},R.1\ (C_{-},R.1\ j)) = C_{-},R.1\ (C_{-},R.1\ (f11\ k\ j))$; }
 $C_{-},R.1\ (f1\ (C_{-},R.1\ (C_{-},R.1\ (f11\ k\ j)))\ k) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f1\ (f11\ k\ j)\ k)))$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ (f1\ (C_{-},R.1\ (f11\ k\ j))\ k)) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f1\ (f11\ k\ j)\ k)))$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f1\ (f11\ k\ j)\ k))) = C_{-},R.1\ (C_{-},R.1\ (C_{-},R.1\ (f1\ (f11\ k\ j)\ k)))$
 { Triv }

Done.

$f11\ a\ (f1\ b\ g) = f1\ (f11\ a\ g)\ b$
 $f11\ a\ (f1\ b\ g) = f1\ (f11\ a\ g)\ b$
 { Induction on b }
 $T) f11\ a\ (f1\ C_{-,R.0}\ g) = f1\ (f11\ a\ g)\ C_{-,R.0}$
 { Compute ; }
 $f11\ a\ (C_{-,R.1}\ g) = f1\ (f11\ a\ g)\ C_{-,R.0}$
 { Fund: $f11\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f11\ a\ b)$; Fund: $f1\ a\ C_{-,R.0} = C_{-,R.1}\ a$ }
 $C_{-,R.1}\ (f11\ a\ g) = C_{-,R.1}\ (f11\ a\ g)$
 { Triv }
Done.

$H) f11\ a\ (f1\ j\ g) = f1\ (f11\ a\ g)\ j$
 $T) f11\ a\ (f1\ (C_{-,R.1}\ j)\ g) = f1\ (f11\ a\ g)\ (C_{-,R.1}\ j)$
 { Compute ; }
 $f11\ a\ (C_{-,R.1}\ (f1\ j\ g)) = f1\ (f11\ a\ g)\ (C_{-,R.1}\ j)$
 { Fund: $f11\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f11\ a\ b)$; Fund: $f1\ a\ (C_{-,R.1}\ b) = C_{-,R.1}\ (f1\ a\ b)$ }
 $C_{-,R.1}\ (f11\ a\ (f1\ j\ g)) = C_{-,R.1}\ (f1\ (f11\ a\ g)\ j)$
 { Fund: $f11\ a\ (f1\ j\ g) = f1\ (f11\ a\ g)\ j$; }
 $C_{-,R.1}\ (f1\ (f11\ a\ g)\ j) = C_{-,R.1}\ (f1\ (f11\ a\ g)\ j)$
 { Triv }
Done.

B.2.10 $f12\ x\ y = x * y$

$f12::C_{-,R} \rightarrow C_{-,R} \rightarrow C_{-,R}$
 $f12 = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{C_{-,R.0} \rightarrow C_{-,R.0}; C_{-,R.1}g \rightarrow f0\ e\ (f12\ g\ e)\}$
 $f12\ C_{-,R.0}\ a = C_{-,R.0}$
 $f12\ C_{-,R.0}\ a = C_{-,R.0}$
 { Compute ; }
 $C_{-,R.0} = C_{-,R.0}$
 { Triv }
Done.

$f12\ a\ C_{-,R.0} = C_{-,R.0}$
 $f12\ a\ C_{-,R.0} = C_{-,R.0}$
 { Induction on a }
 $T) f12\ C_{-,R.0}\ C_{-,R.0} = C_{-,R.0}$
 { Compute ; }
 $C_{-,R.0} = C_{-,R.0}$
 { Triv }
Done.

H) $f_{12} j C_{-,R.0} = C_{-,R.0}$
T) $f_{12} (C_{-,R.1} j) C_{-,R.0} = C_{-,R.0}$
 { Compute ; }
 $f_0 C_{-,R.0} (f_{12} j C_{-,R.0}) = C_{-,R.0}$
 { Compute ; }
 $f_{12} j C_{-,R.0} = C_{-,R.0}$
 { Fund: $f_{12} j C_{-,R.0} = C_{-,R.0}$; }
 $C_{-,R.0} = C_{-,R.0}$
 { Triv }
 Done.

$f_{12} (C_{-,R.1} a) b = f_0 (f_{12} a b) b$
 $f_{12} (C_{-,R.1} a) b = f_0 (f_{12} a b) b$
 { Compute ; }
 $f_0 b (f_{12} a b) = f_0 (f_{12} a b) b$
 { Reor: $f_0 d c = f_0 c d$; }
 $f_0 (f_{12} a b) b = f_0 (f_{12} a b) b$
 { Triv }
 Done.

$f_{12} a (C_{-,R.1} b) = f_0 a (f_{12} a b)$
 $f_{12} a (C_{-,R.1} b) = f_0 a (f_{12} a b)$
 { Induction on b }
T) $f_{12} a (C_{-,R.1} C_{-,R.0}) = f_0 a (f_{12} a C_{-,R.0})$
 { ; Fund: $f_{12} a C_{-,R.0} = C_{-,R.0}$ }
 $f_{12} a (C_{-,R.1} C_{-,R.0}) = f_0 a C_{-,R.0}$
 { ; Fund: $f_0 a C_{-,R.0} = a$ }
 $f_{12} a (C_{-,R.1} C_{-,R.0}) = a$
 { Induction on a }

T) $f_{12} C_{-,R.0} (C_{-,R.1} C_{-,R.0}) = C_{-,R.0}$
 { Compute ; }
 $C_{-,R.0} = C_{-,R.0}$
 { Triv }
 Done.

H) $f_{12} j (C_{-,R.1} C_{-,R.0}) = j$

T) $f_{12} (C_{-,R.1} j) (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} j$
 { Compute ; }
 $f_0 (C_{-,R.1} C_{-,R.0}) (f_{12} j (C_{-,R.1} C_{-,R.0})) = C_{-,R.1} j$
 { Compute ; }
 $C_{-,R.1} (f_0 C_{-,R.0} (f_{12} j (C_{-,R.1} C_{-,R.0}))) = C_{-,R.1} j$
 { Compute ; }
 $C_{-,R.1} (f_{12} j (C_{-,R.1} C_{-,R.0})) = C_{-,R.1} j$
 { Fund: $f_{12} j (C_{-,R.1} C_{-,R.0}) = j$; }

$C_{-,R.1} j = C_{-,R.1} j$
 { Triv }
 Done.

$H) f12 a (C_{-,R.1} j) = f0 a (f12 a j)$
 $T) f12 a (C_{-,R.1} (C_{-,R.1} j)) = f0 a (f12 a (C_{-,R.1} j))$
 { ; Fund: $f12 a (C_{-,R.1} j) = f0 a (f12 a j)$ }
 $f12 a (C_{-,R.1} (C_{-,R.1} j)) = f0 a (f0 a (f12 a j))$
 { Induction on a }

$T) f12 C_{-,R.0} (C_{-,R.1} (C_{-,R.1} j)) = f0 C_{-,R.0} (f0 C_{-,R.0} (f12 C_{-,R.0} j))$
 { Compute ; Compute }
 $C_{-,R.0} = f0 C_{-,R.0} (f12 C_{-,R.0} j)$
 { ; Compute }
 $C_{-,R.0} = f12 C_{-,R.0} j$
 { ; Compute }
 $C_{-,R.0} = C_{-,R.0}$
 { Triv }
 Done.

$H) f12 k (C_{-,R.1} (C_{-,R.1} j)) = f0 k (f0 k (f12 k j))$

$T) f12 (C_{-,R.1} k) (C_{-,R.1} (C_{-,R.1} j)) = f0 (C_{-,R.1} k) (f0 (C_{-,R.1} k) (f12 (C_{-,R.1} k) j))$
 { Compute ; Compute }
 $f0 (C_{-,R.1} (C_{-,R.1} j)) (f12 k (C_{-,R.1} (C_{-,R.1} j))) = C_{-,R.1} (f0 k (f0 (C_{-,R.1} k) (f12 (C_{-,R.1} k) j)))$
 { Compute ; Compute }
 $C_{-,R.1} (f0 (C_{-,R.1} j)) (f12 k (C_{-,R.1} (C_{-,R.1} j))) = C_{-,R.1} (f0 k (C_{-,R.1} (f0 k (f12 (C_{-,R.1} k) j))))$
 { Compute ; Compute }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f12 k (C_{-,R.1} (C_{-,R.1} j)))))) = C_{-,R.1} (f0 k (C_{-,R.1} (f0 k (f0 j (f12 k j))))))$
 { Fund: $f12 k (C_{-,R.1} (C_{-,R.1} j)) = f0 k (f0 k (f12 k j))$; Fund: $f0 a (C_{-,R.1} b) = C_{-,R.1} (f0 a b)$ }
 }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f0 k (f0 k (f12 k j)))))) = C_{-,R.1} (C_{-,R.1} (f0 k (f0 k (f0 j (f12 k j))))))$
 { Reor: $f0 d c = f0 c d$; Reor: $f0 d c = f0 c d$ }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f0 k (f0 (f12 k j) k)))) = C_{-,R.1} (C_{-,R.1} (f0 k (f0 (f0 j (f12 k j) k))))$
 { Reor: $f0 d c = f0 c d$; Reor: $f0 (f0 b e) g = f0 b (f0 e g)$ }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f0 (f0 (f12 k j) k) k))) = C_{-,R.1} (C_{-,R.1} (f0 k (f0 j (f0 (f12 k j) k))))$
 { Reor: $f0 (f0 b e) g = f0 b (f0 e g)$; Reor: $f0 d c = f0 c d$ }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f0 (f12 k j) (f0 k k)))) = C_{-,R.1} (C_{-,R.1} (f0 (f0 j (f0 (f12 k j) k) k)))$
 { ; Reor: $f0 (f0 b e) g = f0 b (f0 e g)$ }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f0 (f12 k j) (f0 k k)))) = C_{-,R.1} (C_{-,R.1} (f0 j (f0 (f0 (f12 k j) k) k)))$
 { ; Reor: $f0 (f0 b e) g = f0 b (f0 e g)$ }
 $C_{-,R.1} (C_{-,R.1} (f0 j (f0 (f12 k j) (f0 k k)))) = C_{-,R.1} (C_{-,R.1} (f0 j (f0 (f12 k j) (f0 k k))))$
 { Triv }
 Done.

$f12 (f0 a b) g = f0 (f12 b g) (f12 g a)$
 $f12 (f0 a b) g = f0 (f12 b g) (f12 g a)$
 { Induction on a }
T) $f12 (f0 C_{-}, R.0 b) g = f0 (f12 b g) (f12 g C_{-}, R.0)$
 { Compute ; }
 $f12 b g = f0 (f12 b g) (f12 g C_{-}, R.0)$
 { ; Fund: $f12 a C_{-}, R.0 = C_{-}, R.0$ }
 $f12 b g = f0 (f12 b g) C_{-}, R.0$
 { ; Fund: $f0 a C_{-}, R.0 = a$ }
 $f12 b g = f12 b g$
 { Triv }
 Done.

H) $f12 (f0 j b) g = f0 (f12 b g) (f12 g j)$
T) $f12 (f0 (C_{-}, R.1 j) b) g = f0 (f12 b g) (f12 g (C_{-}, R.1 j))$
 { Compute ; }
 $f12 (C_{-}, R.1 (f0 j b)) g = f0 (f12 b g) (f12 g (C_{-}, R.1 j))$
 { Compute ; }
 $f0 g (f12 (f0 j b) g) = f0 (f12 b g) (f12 g (C_{-}, R.1 j))$
 { Fund: $f12 (f0 j b) g = f0 (f12 b g) (f12 g j)$; Fund: $f12 a (C_{-}, R.1 b) = f0 a (f12 a b)$ }
 $f0 g (f0 (f12 b g) (f12 g j)) = f0 (f12 b g) (f0 g (f12 g j))$
 { Reor: $f0 d c = f0 c d$; }
 $f0 (f0 (f12 b g) (f12 g j)) g = f0 (f12 b g) (f0 g (f12 g j))$
 { Reor: $f0 (f0 a g) b = f0 (f0 a b) g$; }
 $f0 (f0 (f12 b g) g) (f12 g j) = f0 (f12 b g) (f0 g (f12 g j))$
 { Reor: $f0 (f0 b e) g = f0 b (f0 e g)$; }
 $f0 (f12 b g) (f0 g (f12 g j)) = f0 (f12 b g) (f0 g (f12 g j))$
 { Triv }
 Done.

$f12 a (f0 b g) = f0 (f12 a b) (f12 a g)$
 $f12 a (f0 b g) = f0 (f12 a b) (f12 a g)$
 { Induction on b }
T) $f12 a (f0 C_{-}, R.0 g) = f0 (f12 a C_{-}, R.0) (f12 a g)$
 { Compute ; }
 $f12 a g = f0 (f12 a C_{-}, R.0) (f12 a g)$
 { ; Fund: $f12 a C_{-}, R.0 = C_{-}, R.0$ }
 $f12 a g = f0 C_{-}, R.0 (f12 a g)$
 { ; Compute }
 $f12 a g = f12 a g$
 { Triv }
 Done.

H) $f12 a (f0 j g) = f0 (f12 a j) (f12 a g)$
T) $f12 a (f0 (C_{-}, R.1 j) g) = f0 (f12 a (C_{-}, R.1 j)) (f12 a g)$
 { Compute ; }
 $f12 a (C_{-}, R.1 (f0 j g)) = f0 (f12 a (C_{-}, R.1 j)) (f12 a g)$

```

{ Fund: f12 a (C-,R.1 b) = f0 a (f12 a b) ; Fund: f12 a (C-,R.1 b) = f0 a (f12 a b) }
f0 a (f12 a (f0 j g)) = f0 (f0 a (f12 a j)) (f12 a g)
{ Fund: f12 a (f0 j g) = f0 (f12 a j) (f12 a g) ; }
f0 a (f0 (f12 a j) (f12 a g)) = f0 (f0 a (f12 a j)) (f12 a g)
{ Reor: f0 d c = f0 c d ; Reor: f0 (f0 a g) b = f0 (f0 a b) g }
f0 a (f0 (f12 a g) (f12 a j)) = f0 (f0 a (f12 a g)) (f12 a j)
{ ; Reor: f0 (f0 b e) g = f0 b (f0 e g) }
f0 a (f0 (f12 a g) (f12 a j)) = f0 a (f0 (f12 a g) (f12 a j))
{ Triv }
Done.

```

```

f12 c d = f12 d c
f12 c d = f12 d c
{ Induction on c }
T) f12 C-,R.0 d = f12 d C-,R.0
{ Compute ; }
C-,R.0 = f12 d C-,R.0
{ ; Fund: f12 a C-,R.0 = C-,R.0 }
C-,R.0 = C-,R.0
{ Triv }
Done.

```

```

H) f12 j d = f12 d j
T) f12 (C-,R.1 j) d = f12 d (C-,R.1 j)
{ Compute ; }
f0 d (f12 j d) = f12 d (C-,R.1 j)
{ ; Fund: f12 a (C-,R.1 b) = f0 a (f12 a b) }
f0 d (f12 j d) = f0 d (f12 d j)
{ Reor: f12 j d = f12 d j ; }
f0 d (f12 d j) = f0 d (f12 d j)
{ Triv }
Done.

```

B.2.11 $f_{13} x y = x * y + 1$

```

f13::C-,R → C-,R → C-,R
f13 = λd → λe → case d of {C-,R.0 → C-,R.1 C-,R.0; C-,R.1g → f0 e (f13 g e)}

```

```

f13 C-,R.0 a = C-,R.1 C-,R.0
f13 C-,R.0 a = C-,R.1 C-,R.0
{ Compute ; }
C-,R.1 C-,R.0 = C-,R.1 C-,R.0
{ Triv }
Done.

```

f13 a C_{-,R.0} = C_{-,R.1} C_{-,R.0}

f13 a C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Induction on a }

T) f13 C_{-,R.0} C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Compute ; }

C_{-,R.1} C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Triv }

Done.

H) f13 j C_{-,R.0} = C_{-,R.1} C_{-,R.0}

T) f13 (C_{-,R.1} j) C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Compute ; }

f0 C_{-,R.0} (f13 j C_{-,R.0}) = C_{-,R.1} C_{-,R.0}

{ Compute ; }

f13 j C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Fund: f13 j C_{-,R.0} = C_{-,R.1} C_{-,R.0} ; }

C_{-,R.1} C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Triv }

Done.

f13 (C_{-,R.1} a) b = f0 (f13 a b) b

f13 (C_{-,R.1} a) b = f0 (f13 a b) b

{ Compute ; }

f0 b (f13 a b) = f0 (f13 a b) b

{ Reor: f0 d c = f0 c d ; }

f0 (f13 a b) b = f0 (f13 a b) b

{ Triv }

Done.

f13 a (C_{-,R.1} b) = f0 a (f13 a b)

f13 a (C_{-,R.1} b) = f0 a (f13 a b)

{ Induction on b }

T) f13 a (C_{-,R.1} C_{-,R.0}) = f0 a (f13 a C_{-,R.0})

{ ; Fund: f13 a C_{-,R.0} = C_{-,R.1} C_{-,R.0} }

f13 a (C_{-,R.1} C_{-,R.0}) = f0 a (C_{-,R.1} C_{-,R.0})

{ ; Fund: f0 a (C_{-,R.1} b) = C_{-,R.1} (f0 a b) }

f13 a (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} (f0 a C_{-,R.0})

{ ; Fund: f0 a C_{-,R.0} = a }

f13 a (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} a

{ Induction on a }

T) f13 C_{-,R.0} (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} C_{-,R.0}

{ Compute ; }

C_{-,R.1} C_{-,R.0} = C_{-,R.1} C_{-,R.0}

{ Triv }

Done.

H)f13 j (C₋,R.1 C₋,R.0) = C₋,R.1 j

T)f13 (C₋,R.1 j) (C₋,R.1 C₋,R.0) = C₋,R.1 (C₋,R.1 j)
{ Compute ; }
f0 (C₋,R.1 C₋,R.0) (f13 j (C₋,R.1 C₋,R.0)) = C₋,R.1 (C₋,R.1 j)
{ Compute ; }
C₋,R.1 (f0 C₋,R.0 (f13 j (C₋,R.1 C₋,R.0))) = C₋,R.1 (C₋,R.1 j)
{ Compute ; }
C₋,R.1 (f13 j (C₋,R.1 C₋,R.0)) = C₋,R.1 (C₋,R.1 j)
{ Fund: f13 j (C₋,R.1 C₋,R.0) = C₋,R.1 j ; }
C₋,R.1 (C₋,R.1 j) = C₋,R.1 (C₋,R.1 j)
{ Triv }
Done.

H)f13 a (C₋,R.1 j) = f0 a (f13 a j)
T)f13 a (C₋,R.1 (C₋,R.1 j)) = f0 a (f13 a (C₋,R.1 j))
{ ; Fund: f13 a (C₋,R.1 j) = f0 a (f13 a j) }
f13 a (C₋,R.1 (C₋,R.1 j)) = f0 a (f0 a (f13 a j))
{ Induction on a }

T)f13 C₋,R.0 (C₋,R.1 (C₋,R.1 j)) = f0 C₋,R.0 (f0 C₋,R.0 (f13 C₋,R.0 j))
{ Compute ; Compute }
C₋,R.1 C₋,R.0 = f0 C₋,R.0 (f13 C₋,R.0 j)
{ ; Compute }
C₋,R.1 C₋,R.0 = f13 C₋,R.0 j
{ ; Compute }
C₋,R.1 C₋,R.0 = C₋,R.1 C₋,R.0
{ Triv }
Done.

H)f13 k (C₋,R.1 (C₋,R.1 j)) = f0 k (f0 k (f13 k j))

T)f13 (C₋,R.1 k) (C₋,R.1 (C₋,R.1 j)) = f0 (C₋,R.1 k) (f0 (C₋,R.1 k) (f13 (C₋,R.1 k) j))
{ Compute ; Compute }
f0 (C₋,R.1 (C₋,R.1 j)) (f13 k (C₋,R.1 (C₋,R.1 j))) = C₋,R.1 (f0 k (f0 (C₋,R.1 k) (f13 (C₋,R.1 k) j)))
{ Compute ; Compute }
C₋,R.1 (f0 (C₋,R.1 j) (f13 k (C₋,R.1 (C₋,R.1 j)))) = C₋,R.1 (f0 k (C₋,R.1 (f0 k (f13 (C₋,R.1 k) j))))
{ Compute ; Compute }
C₋,R.1 (C₋,R.1 (f0 j (f13 k (C₋,R.1 (C₋,R.1 j))))) = C₋,R.1 (f0 k (C₋,R.1 (f0 k (f0 j (f13 k j)))))
{ Fund: f13 k (C₋,R.1 (C₋,R.1 j)) = f0 k (f0 k (f13 k j)) ; Fund: f0 a (C₋,R.1 b) = C₋,R.1 (f0 a b) }
}
C₋,R.1 (C₋,R.1 (f0 j (f0 k (f0 k (f13 k j))))) = C₋,R.1 (C₋,R.1 (f0 k (f0 k (f0 j (f13 k j)))))
{ Reor: f0 d c = f0 c d ; Reor: f0 d c = f0 c d }
C₋,R.1 (C₋,R.1 (f0 j (f0 k (f0 (f13 k j) k)))) = C₋,R.1 (C₋,R.1 (f0 k (f0 (f0 j (f13 k j)) k)))
{ Reor: f0 d c = f0 c d ; Reor: f0 (f0 b e) g = f0 b (f0 e g) }
C₋,R.1 (C₋,R.1 (f0 j (f0 (f0 (f13 k j) k) k))) = C₋,R.1 (C₋,R.1 (f0 k (f0 j (f0 (f13 k j) k))))
{ Reor: f0 (f0 b e) g = f0 b (f0 e g) ; Reor: f0 d c = f0 c d }

```

C-,R.1 (C-,R.1 (f0 j (f0 (f13 k j) (f0 k k)))) = C-,R.1 (C-,R.1 (f0 (f0 j (f0 (f13 k j) k) k))
{ ; Reor: f0 (f0 b e) g = f0 b (f0 e g) }
C-,R.1 (C-,R.1 (f0 j (f0 (f13 k j) (f0 k k)))) = C-,R.1 (C-,R.1 (f0 j (f0 (f0 (f13 k j) k) k)))
{ ; Reor: f0 (f0 b e) g = f0 b (f0 e g) }
C-,R.1 (C-,R.1 (f0 j (f0 (f13 k j) (f0 k k)))) = C-,R.1 (C-,R.1 (f0 j (f0 (f13 k j) (f0 k k))))
{ Triv }
Done.

```

```

f13 c d = f13 d c
f13 c d = f13 d c
{ Induction on c }
T) f13 C-,R.0 d = f13 d C-,R.0
{ Compute ; }
C-,R.1 C-,R.0 = f13 d C-,R.0
{ ; Fund: f13 a C-,R.0 = C-,R.1 C-,R.0 }
C-,R.1 C-,R.0 = C-,R.1 C-,R.0
{ Triv }
Done.

```

```

H) f13 j d = f13 d j
T) f13 (C-,R.1 j) d = f13 d (C-,R.1 j)
{ Compute ; }
f0 d (f13 j d) = f13 d (C-,R.1 j)
{ ; Fund: f13 a (C-,R.1 b) = f0 a (f13 a b) }
f0 d (f13 j d) = f0 d (f13 d j)
{ Reor: f13 j d = f13 d j ; }
f0 d (f13 d j) = f0 d (f13 d j)
{ Triv }
Done.

```

B.2.12 $f_{14} x y = x * y + 2$

```

f14::C-,R → C-,R → C-,R
f14 = λd → λe → case d of {C-,R.0 → C-,R.1 (C-,R.1 C-,R.0); C-,R.1g → f0 e (f14 g e)}

f14 C-,R.0 a = C-,R.1 (C-,R.1 C-,R.0)
f14 C-,R.0 a = C-,R.1 (C-,R.1 C-,R.0)
{ Compute ; }
C-,R.1 (C-,R.1 C-,R.0) = C-,R.1 (C-,R.1 C-,R.0)
{ Triv }
Done.

```

$f14\ a\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 $f14\ a\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Induction on a }
 $T) f14\ C_{-},R.0\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Triv }
 Done.

$H) f14\ j\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 $T) f14\ (C_{-},R.1\ j)\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Compute ; }
 $f0\ C_{-},R.0\ (f14\ j\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Compute ; }
 $f14\ j\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Fund: $f14\ j\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$; }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Triv }
 Done.

$f14\ (C_{-},R.1\ a)\ b = f0\ (f14\ a\ b)\ b$
 $f14\ (C_{-},R.1\ a)\ b = f0\ (f14\ a\ b)\ b$
 { Compute ; }
 $f0\ b\ (f14\ a\ b) = f0\ (f14\ a\ b)\ b$
 { Reor: $f0\ d\ c = f0\ c\ d$; }
 $f0\ (f14\ a\ b)\ b = f0\ (f14\ a\ b)\ b$
 { Triv }
 Done.

$f14\ c\ d = f14\ d\ c$
 $f14\ c\ d = f14\ d\ c$
 { Induction on c }
 $T) f14\ C_{-},R.0\ d = f14\ d\ C_{-},R.0$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = f14\ d\ C_{-},R.0$
 { ; Fund: $f14\ a\ C_{-},R.0 = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$ }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Triv }
 Done.

$H) f14\ j\ d = f14\ d\ j$
 $T) f14\ (C_{-},R.1\ j)\ d = f14\ d\ (C_{-},R.1\ j)$
 { Compute ; }
 $f0\ d\ (f14\ j\ d) = f14\ d\ (C_{-},R.1\ j)$
 { Reor: $f14\ j\ d = f14\ d\ j$; }
 $f0\ d\ (f14\ d\ j) = f14\ d\ (C_{-},R.1\ j)$
 { Induction on d }

T) $f0\ C_{-},R.0\ (f14\ C_{-},R.0\ j) = f14\ C_{-},R.0\ (C_{-},R.1\ j)$
 { Compute ; Compute }
 $f14\ C_{-},R.0\ j = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Compute ; }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Triv }
 Done.

H) $f0\ k\ (f14\ k\ j) = f14\ k\ (C_{-},R.1\ j)$

T) $f0\ (C_{-},R.1\ k)\ (f14\ (C_{-},R.1\ k)\ j) = f14\ (C_{-},R.1\ k)\ (C_{-},R.1\ j)$
 { Compute ; Compute }
 $C_{-},R.1\ (f0\ k\ (f14\ (C_{-},R.1\ k)\ j)) = f0\ (C_{-},R.1\ j)\ (f14\ k\ (C_{-},R.1\ j))$
 { Compute ; Compute }
 $C_{-},R.1\ (f0\ k\ (f0\ j\ (f14\ k\ j))) = C_{-},R.1\ (f0\ j\ (f14\ k\ (C_{-},R.1\ j)))$
 { ; Fund: $f14\ k\ (C_{-},R.1\ j) = f0\ k\ (f14\ k\ j)$ }
 $C_{-},R.1\ (f0\ k\ (f0\ j\ (f14\ k\ j))) = C_{-},R.1\ (f0\ j\ (f0\ k\ (f14\ k\ j)))$
 { Reor: $f0\ d\ c = f0\ c\ d$; Reor: $f0\ d\ c = f0\ c\ d$ }
 $C_{-},R.1\ (f0\ (f0\ j\ (f14\ k\ j))\ k) = C_{-},R.1\ (f0\ j\ (f0\ (f14\ k\ j)\ k))$
 { Reor: $f0\ (f0\ b\ e)\ g = f0\ b\ (f0\ e\ g)$; Reor: $f14\ d\ j = f14\ j\ d$ }
 $C_{-},R.1\ (f0\ j\ (f0\ (f14\ k\ j)\ k)) = C_{-},R.1\ (f0\ j\ (f0\ (f14\ j\ k)\ k))$
 { Reor: $f14\ d\ j = f14\ j\ d$; }
 $C_{-},R.1\ (f0\ j\ (f0\ (f14\ j\ k)\ k)) = C_{-},R.1\ (f0\ j\ (f0\ (f14\ j\ k)\ k))$
 { Triv }
 Done.

B.2.13 $f15\ x\ _ = 2^{x+1}$

$f15::C_{-},R \rightarrow C_{-},R \rightarrow C_{-},R$
 $f15 = \lambda d \rightarrow \lambda e \rightarrow case\ d\ of\ \{C_{-},R.0 \rightarrow C_{-},R.1\ C_{-},R.0; C_{-},R.1g \rightarrow f0\ (f15\ g\ e)\ (f15\ g\ e)\}$

$f15\ C_{-},R.0\ a = C_{-},R.1\ C_{-},R.0$

$f15\ C_{-},R.0\ a = C_{-},R.1\ C_{-},R.0$

{ Compute ; }

$C_{-},R.1\ C_{-},R.0 = C_{-},R.1\ C_{-},R.0$

{ Triv }

Done.

$f15\ (C_{-},R.1\ a)\ b = f0\ (f15\ a\ b)\ (f15\ a\ b)$

$f15\ (C_{-},R.1\ a)\ b = f0\ (f15\ a\ b)\ (f15\ a\ b)$

{ Compute ; }

$f0\ (f15\ a\ b)\ (f15\ a\ b) = f0\ (f15\ a\ b)\ (f15\ a\ b)$

{ Triv }

Done.

f15 a (C_,R.1 b) = f15 a (f15 b b)

f15 a (C_,R.1 b) = f15 a (f15 b b)

{ Induction on b }

(T) f15 a (C_,R.1 C_,R.0) = f15 a (f15 C_,R.0 C_,R.0)

{ ; Compute }

f15 a (C_,R.1 C_,R.0) = f15 a (C_,R.1 C_,R.0)

{ Triv }

Done.

(H) f15 a (C_,R.1 j) = f15 a (f15 j j)

(T) f15 a (C_,R.1 (C_,R.1 j)) = f15 a (f15 (C_,R.1 j) (C_,R.1 j))

{ ; Compute }

f15 a (C_,R.1 (C_,R.1 j)) = f15 a (f0 (f15 j (C_,R.1 j)) (f15 j (C_,R.1 j)))

{ ; Fund: f15 a (C_,R.1 j) = f15 a (f15 j j) }

f15 a (C_,R.1 (C_,R.1 j)) = f15 a (f0 (f15 j (f15 j j)) (f15 j (C_,R.1 j)))

{ ; Fund: f15 a (C_,R.1 j) = f15 a (f15 j j) }

f15 a (C_,R.1 (C_,R.1 j)) = f15 a (f0 (f15 j (f15 j j)) (f15 j (f15 j j)))

{ Induction on a }

(T) f15 C_,R.0 (C_,R.1 (C_,R.1 j)) = f15 C_,R.0 (f0 (f15 j (f15 j j)) (f15 j (f15 j j)))

{ Compute ; Compute }

C_,R.1 C_,R.0 = C_,R.1 C_,R.0

{ Triv }

Done.

(H) f15 k (C_,R.1 (C_,R.1 j)) = f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j)))

(T) f15 (C_,R.1 k) (C_,R.1 (C_,R.1 j)) = f15 (C_,R.1 k) (f0 (f15 j (f15 j j)) (f15 j (f15 j j)))

HOLA

{ Compute ; Compute }

(∀)(f0 (f15 k (C_,R.1 (C_,R.1 j))) (f15 k (C_,R.1 (C_,R.1 j))))

= f0 (f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

(f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

{ Fund: (∀)(f15 k (C_,R.1 (C_,R.1 j)))

= f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))) ; }

(∀)(f0 (f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j)))) (f15 k (C_,R.1 (C_,R.1 j)))

= f0 (f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

(f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

{ Fund: (∀)(f15 k (C_,R.1 (C_,R.1 j)))

= f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))) ; }

(∀)(f0 (f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j)))) (f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

= f0 (f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

(f15 k (f0 (f15 j (f15 j j)) (f15 j (f15 j j))))

{ Triv }

Done.

$f15\ b\ (f15\ e\ g) = f15\ b\ (f15\ g\ e)$
 $f15\ b\ (f15\ e\ g) = f15\ b\ (f15\ g\ e)$
 { Induction on b }
T) $f15\ C_{-,R.0}\ (f15\ e\ g) = f15\ C_{-,R.0}\ (f15\ g\ e)$
 { Compute ; Compute }
 $C_{-,R.1}\ C_{-,R.0} = C_{-,R.1}\ C_{-,R.0}$
 { Triv }
 Done.

H) $f15\ j\ (f15\ e\ g) = f15\ j\ (f15\ g\ e)$
T) $f15\ (C_{-,R.1}\ j)\ (f15\ e\ g) = f15\ (C_{-,R.1}\ j)\ (f15\ g\ e)$
 { Compute ; Compute }
 $f0\ (f15\ j\ (f15\ e\ g))\ (f15\ j\ (f15\ e\ g)) = f0\ (f15\ j\ (f15\ g\ e))\ (f15\ j\ (f15\ g\ e))$
 { ; Reor: $f15\ j\ (f15\ g\ e) = f15\ j\ (f15\ e\ g)$ }
 $f0\ (f15\ j\ (f15\ e\ g))\ (f15\ j\ (f15\ e\ g)) = f0\ (f15\ j\ (f15\ e\ g))\ (f15\ j\ (f15\ g\ e))$
 { ; Reor: $f15\ j\ (f15\ g\ e) = f15\ j\ (f15\ e\ g)$ }
 $f0\ (f15\ j\ (f15\ e\ g))\ (f15\ j\ (f15\ e\ g)) = f0\ (f15\ j\ (f15\ e\ g))\ (f15\ j\ (f15\ e\ g))$
 { Triv }
 Done.

B.2.14 $f16\ x\ _ = 2^{x+2}$

$f16::C_{-,R} \rightarrow C_{-,R} \rightarrow C_{-,R}$
 $f16 = \lambda d \rightarrow \lambda e \rightarrow case\ d\ of\ \{C_{-,R.0} \rightarrow C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0}); C_{-,R.1}g \rightarrow f0\ (f16\ g\ e)\ (f16\ g\ e)\}$

$f16\ C_{-,R.0}\ a = C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0})$
 $f16\ C_{-,R.0}\ a = C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0})$
 { Compute ; }
 $C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0}) = C_{-,R.1}\ (C_{-,R.1}\ C_{-,R.0})$
 { Triv }
 Done.

$f16\ (C_{-,R.1}\ a)\ b = f0\ (f16\ a\ b)\ (f16\ a\ b)$
 $f16\ (C_{-,R.1}\ a)\ b = f0\ (f16\ a\ b)\ (f16\ a\ b)$
 { Compute ; }
 $f0\ (f16\ a\ b)\ (f16\ a\ b) = f0\ (f16\ a\ b)\ (f16\ a\ b)$
 { Triv }
 Done.

$f16\ a\ (f0\ b\ g) = f16\ a\ b$
 $f16\ a\ (f0\ b\ g) = f16\ a\ b$
 { Induction on b }
T) $f16\ a\ (f0\ C_{-,R.0}\ g) = f16\ a\ C_{-,R.0}$
 { Compute ; }
 $f16\ a\ g = f16\ a\ C_{-,R.0}$
 { Induction on g }

T) $f16\ a\ C_{-},R.0 = f16\ a\ C_{-},R.0$
 { Triv }
 Done.

H) $f16\ a\ j = f16\ a\ C_{-},R.0$

T) $f16\ a\ (C_{-},R.1\ j) = f16\ a\ C_{-},R.0$
 { ; Fund: $f16\ a\ C_{-},R.0 = f16\ a\ j$ }
 $f16\ a\ (C_{-},R.1\ j) = f16\ a\ j$
 { Induction on a }

T) $f16\ C_{-},R.0\ (C_{-},R.1\ j) = f16\ C_{-},R.0\ j$
 { Compute ; Compute }
 $C_{-},R.1\ (C_{-},R.1\ C_{-},R.0) = C_{-},R.1\ (C_{-},R.1\ C_{-},R.0)$
 { Triv }
 Done.

H) $f16\ k\ (C_{-},R.1\ j) = f16\ k\ j$

T) $f16\ (C_{-},R.1\ k)\ (C_{-},R.1\ j) = f16\ (C_{-},R.1\ k)\ j$
 { Compute ; Compute }
 $f0\ (f16\ k\ (C_{-},R.1\ j))\ (f16\ k\ (C_{-},R.1\ j)) = f0\ (f16\ k\ j)\ (f16\ k\ j)$
 { Fund: $f16\ k\ (C_{-},R.1\ j) = f16\ k\ j$; }
 $f0\ (f16\ k\ j)\ (f16\ k\ (C_{-},R.1\ j)) = f0\ (f16\ k\ j)\ (f16\ k\ j)$
 { Fund: $f16\ k\ (C_{-},R.1\ j) = f16\ k\ j$; }
 $f0\ (f16\ k\ j)\ (f16\ k\ j) = f0\ (f16\ k\ j)\ (f16\ k\ j)$
 { Triv }
 Done.

H) $f16\ a\ (f0\ j\ g) = f16\ a\ j$
T) $f16\ a\ (f0\ (C_{-},R.1\ j)\ g) = f16\ a\ (C_{-},R.1\ j)$
 { Compute ; }
 $f16\ a\ (C_{-},R.1\ (f0\ j\ g)) = f16\ a\ (C_{-},R.1\ j)$
 { Reor: $f0\ d\ c = f0\ c\ d$; }
 $f16\ a\ (C_{-},R.1\ (f0\ g\ j)) = f16\ a\ (C_{-},R.1\ j)$
 { Induction on g }

T) $f16\ a\ (C_{-},R.1\ (f0\ C_{-},R.0\ j)) = f16\ a\ (C_{-},R.1\ j)$
 { Compute ; }
 $f16\ a\ (C_{-},R.1\ j) = f16\ a\ (C_{-},R.1\ j)$
 { Triv }
 Done.

H) $f16\ a\ (C_{-},R.1\ (f0\ k\ j)) = f16\ a\ (C_{-},R.1\ j)$

$T) f16\ a\ (C_,\ R.1\ (f0\ (C_,\ R.1\ k)\ j)) = f16\ a\ (C_,\ R.1\ j)$
 { Compute ; }
 $f16\ a\ (C_,\ R.1\ (C_,\ R.1\ (f0\ k)\ j)) = f16\ a\ (C_,\ R.1\ j)$
 { Reor: $f0\ d\ c = f0\ c\ d$; }
 $f16\ a\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)) = f16\ a\ (C_,\ R.1\ j)$
 { Induction on a }

$T) f16\ C_,\ R.0\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)) = f16\ C_,\ R.0\ (C_,\ R.1\ j)$
 { Compute ; Compute }
 $C_,\ R.1\ (C_,\ R.1\ C_,\ R.0) = C_,\ R.1\ (C_,\ R.1\ C_,\ R.0)$
 { Triv }
 Done.

$H) f16\ l\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)) = f16\ l\ (C_,\ R.1\ j)$

$T) f16\ (C_,\ R.1\ l)\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)) = f16\ (C_,\ R.1\ l)\ (C_,\ R.1\ j)$
 { Compute ; Compute }
 $(\forall)(f0\ (f16\ l\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k))))\ (f16\ l\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)))$
 $= f0\ (f16\ l\ (C_,\ R.1\ j))\ (f16\ l\ (C_,\ R.1\ j))$
 { Fund: $(\forall)(f16\ l\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)))$
 $= f16\ l\ (C_,\ R.1\ j)$; }
 $(\forall)(f0\ (f16\ l\ (C_,\ R.1\ j))\ (f16\ l\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k))))$
 $= f0\ (f16\ l\ (C_,\ R.1\ j))\ (f16\ l\ (C_,\ R.1\ j))$
 { Fund: $(\forall)(f16\ l\ (C_,\ R.1\ (C_,\ R.1\ (f0\ j)\ k)))$
 $= f16\ l\ (C_,\ R.1\ j)$; }
 $(\forall)(f0\ (f16\ l\ (C_,\ R.1\ j))\ (f16\ l\ (C_,\ R.1\ j)))$
 $= f0\ (f16\ l\ (C_,\ R.1\ j))\ (f16\ l\ (C_,\ R.1\ j))$
 { Triv }
 Done.

$f16\ b\ (f16\ e\ g) = f16\ b\ (f16\ g\ e)$
 $f16\ b\ (f16\ e\ g) = f16\ b\ (f16\ g\ e)$
 { Induction on b }
 $T) f16\ C_,\ R.0\ (f16\ e\ g) = f16\ C_,\ R.0\ (f16\ g\ e)$
 { Compute ; Compute }
 $C_,\ R.1\ (C_,\ R.1\ C_,\ R.0) = C_,\ R.1\ (C_,\ R.1\ C_,\ R.0)$
 { Triv }
 Done.

$H) f16\ j\ (f16\ e\ g) = f16\ j\ (f16\ g\ e)$
 $T) f16\ (C_,\ R.1\ j)\ (f16\ e\ g) = f16\ (C_,\ R.1\ j)\ (f16\ g\ e)$
 { Compute ; Compute }
 $f0\ (f16\ j\ (f16\ e\ g))\ (f16\ j\ (f16\ g\ e)) = f0\ (f16\ j\ (f16\ g\ e))\ (f16\ j\ (f16\ e\ g))$
 { ; Reor: $f16\ j\ (f16\ g\ e) = f16\ j\ (f16\ e\ g)$ }

```

f0 (f16 j (f16 e g)) (f16 j (f16 e g)) = f0 (f16 j (f16 e g)) (f16 j (f16 g e))
{ ; Reor: f16 j (f16 g e) = f16 j (f16 e g) }
f0 (f16 j (f16 e g)) (f16 j (f16 e g)) = f0 (f16 j (f16 e g)) (f16 j (f16 e g))
{ Triv }
Done.

```

B.2.15 f17 = f7

f17::C₋,R → C₋,R → C₋,R

f17 = λd → λe → case d of {C₋,R.0 → e; C₋,R.1g → f0 (f17 g e) g}

f17 C₋,R.0 a = a

```

f17 C-,R.0 a = a
  { Compute ; }
a = a
  { Triv }
Done.

```

f17 (C₋,R.1 a) b = f0 a (f17 a b)

```

f17 (C-,R.1 a) b = f0 a (f17 a b)
  { Compute ; }
f0 (f17 a b) a = f0 a (f17 a b)
  { Reor: f0 d c = f0 c d ; }
f0 a (f17 a b) = f0 a (f17 a b)
  { Triv }
Done.

```

f17 a (C₋,R.1 b) = C₋,R.1 (f17 a b)

f17 a (C₋,R.1 b) = C₋,R.1 (f17 a b)

{ Induction on b }

T) f17 a (C₋,R.1 C₋,R.0) = C₋,R.1 (f17 a C₋,R.0)

{ Induction on a }

T) f17 C₋,R.0 (C₋,R.1 C₋,R.0) = C₋,R.1 (f17 C₋,R.0 C₋,R.0)

{ Compute ; Compute }

C₋,R.1 C₋,R.0 = C₋,R.1 C₋,R.0

{ Triv }

Done.

H) f17 j (C₋,R.1 C₋,R.0) = C₋,R.1 (f17 j C₋,R.0)

T) f17 (C₋,R.1 j) (C₋,R.1 C₋,R.0) = C₋,R.1 (f17 (C₋,R.1 j) C₋,R.0)

{ Compute ; Compute }

f0 (f17 j (C₋,R.1 C₋,R.0)) j = C₋,R.1 (f0 (f17 j C₋,R.0) j)

{ Reor: f0 d c = f0 c d ; Reor: f0 d c = f0 c d }

f0 j (f17 j (C₋,R.1 C₋,R.0)) = C₋,R.1 (f0 j (f17 j C₋,R.0))

```

{ Reor: f17 j (C_, R.1 C_, R.0) = C_, R.1 (f17 j C_, R.0) ; }
f0 j (C_, R.1 (f17 j C_, R.0)) = C_, R.1 (f0 j (f17 j C_, R.0))
{ Fund: f0 a (C_, R.1 b) = C_, R.1 (f0 a b) ; }
C_, R.1 (f0 j (f17 j C_, R.0)) = C_, R.1 (f0 j (f17 j C_, R.0))
{ Triv }
Done.

```

```

H)f17 a (C_, R.1 j) = C_, R.1 (f17 a j)
T)f17 a (C_, R.1 (C_, R.1 j)) = C_, R.1 (f17 a (C_, R.1 j))
{ ; Fund: f17 a (C_, R.1 j) = C_, R.1 (f17 a j) }
f17 a (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f17 a j))
{ Induction on a }

```

```

T)f17 C_, R.0 (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f17 C_, R.0 j))
{ Compute ; Compute }
C_, R.1 (C_, R.1 j) = C_, R.1 (C_, R.1 j)
{ Triv }
Done.

```

```

H)f17 k (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f17 k j))

```

```

T)f17 (C_, R.1 k) (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f17 (C_, R.1 k) j))
{ Compute ; Compute }
f0 (f17 k (C_, R.1 (C_, R.1 j))) k = C_, R.1 (C_, R.1 (f0 (f17 k j) k))
{ Fund: f17 k (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f17 k j)) ; }
f0 (C_, R.1 (C_, R.1 (f17 k j))) k = C_, R.1 (C_, R.1 (f0 (f17 k j) k))
{ Compute ; }
C_, R.1 (f0 (C_, R.1 (f17 k j)) k) = C_, R.1 (C_, R.1 (f0 (f17 k j) k))
{ Compute ; }
C_, R.1 (C_, R.1 (f0 (f17 k j) k)) = C_, R.1 (C_, R.1 (f0 (f17 k j) k))
{ Triv }
Done.

```

```

f17 a (f0 b g) = f0 (f17 a g) b
f17 a (f0 b g) = f0 (f17 a g) b
{ Induction on b }
T)f17 a (f0 C_, R.0 g) = f0 (f17 a g) C_, R.0
{ Compute ; }
f17 a g = f0 (f17 a g) C_, R.0
{ ; Fund: f0 a C_, R.0 = a }
f17 a g = f17 a g
{ Triv }
Done.

```

```

H)f17 a (f0 j g) = f0 (f17 a g) j

```

$T) f_{17} a (f_0 (C_{-}, R.1 j) g) = f_0 (f_{17} a g) (C_{-}, R.1 j)$
 { Compute ; }
 $f_{17} a (C_{-}, R.1 (f_0 j g)) = f_0 (f_{17} a g) (C_{-}, R.1 j)$
 { Fund: $f_{17} a (C_{-}, R.1 b) = C_{-}, R.1 (f_{17} a b)$; Fund: $f_0 a (C_{-}, R.1 b) = C_{-}, R.1 (f_0 a b)$ }
 $C_{-}, R.1 (f_{17} a (f_0 j g)) = C_{-}, R.1 (f_0 (f_{17} a g) j)$
 { Fund: $f_{17} a (f_0 j g) = f_0 (f_{17} a g) j$; }
 $C_{-}, R.1 (f_0 (f_{17} a g) j) = C_{-}, R.1 (f_0 (f_{17} a g) j)$
 { Triv }
 Done.

B.2.16 $f_{18} = f_8$

$f_{18} :: C_{-}, R \rightarrow C_{-}, R \rightarrow C_{-}, R$
 $f_{18} = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{ C_{-}, R.0 \rightarrow C_{-}, R.1 e ; C_{-}, R.1 g \rightarrow f_0 (f_{18} g e) g \}$

$f_{18} C_{-}, R.0 a = C_{-}, R.1 a$

$f_{18} C_{-}, R.0 a = C_{-}, R.1 a$

{ Compute ; }

$C_{-}, R.1 a = C_{-}, R.1 a$

{ Triv }

Done.

$f_{18} (C_{-}, R.1 a) b = f_0 a (f_{18} a b)$

$f_{18} (C_{-}, R.1 a) b = f_0 a (f_{18} a b)$

{ Compute ; }

$f_0 (f_{18} a b) a = f_0 a (f_{18} a b)$

{ Reor: $f_0 d c = f_0 c d$; }

$f_0 a (f_{18} a b) = f_0 a (f_{18} a b)$

{ Triv }

Done.

$f_{18} a (C_{-}, R.1 b) = C_{-}, R.1 (f_{18} a b)$

$f_{18} a (C_{-}, R.1 b) = C_{-}, R.1 (f_{18} a b)$

{ Induction on b }

$T) f_{18} a (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{18} a C_{-}, R.0)$

{ Induction on a }

$T) f_{18} C_{-}, R.0 (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{18} C_{-}, R.0 C_{-}, R.0)$

{ Compute ; Compute }

$C_{-}, R.1 (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (C_{-}, R.1 C_{-}, R.0)$

{ Triv }

Done.

$H) f_{18} j (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{18} j C_{-}, R.0)$

$T) f_{18} (C_{-}, R.1 j) (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{18} (C_{-}, R.1 j) C_{-}, R.0)$

```

{ Compute ; Compute }
f0 (f18 j (C_, R.1 C_, R.0)) j = C_, R.1 (f0 (f18 j C_, R.0) j)
{ Reor: f0 d c = f0 c d ; Reor: f0 d c = f0 c d }
f0 j (f18 j (C_, R.1 C_, R.0)) = C_, R.1 (f0 j (f18 j C_, R.0))
{ Reor: f18 j (C_, R.1 C_, R.0) = C_, R.1 (f18 j C_, R.0) ; }
f0 j (C_, R.1 (f18 j C_, R.0)) = C_, R.1 (f0 j (f18 j C_, R.0))
{ Fund: f0 a (C_, R.1 b) = C_, R.1 (f0 a b) ; }
C_, R.1 (f0 j (f18 j C_, R.0)) = C_, R.1 (f0 j (f18 j C_, R.0))
{ Triv }
Done.

```

```

H) f18 a (C_, R.1 j) = C_, R.1 (f18 a j)
T) f18 a (C_, R.1 (C_, R.1 j)) = C_, R.1 (f18 a (C_, R.1 j))
{ ; Fund: f18 a (C_, R.1 j) = C_, R.1 (f18 a j) }
f18 a (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f18 a j))
{ Induction on a }

```

```

T) f18 C_, R.0 (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f18 C_, R.0 j))
{ Compute ; Compute }
C_, R.1 (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (C_, R.1 j))
{ Triv }
Done.

```

```

H) f18 k (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f18 k j))

```

```

T) f18 (C_, R.1 k) (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f18 (C_, R.1 k) j))
{ Compute ; Compute }
f0 (f18 k (C_, R.1 (C_, R.1 j))) k = C_, R.1 (C_, R.1 (f0 (f18 k j) k))
{ Fund: f18 k (C_, R.1 (C_, R.1 j)) = C_, R.1 (C_, R.1 (f18 k j)) ; }
f0 (C_, R.1 (C_, R.1 (f18 k j))) k = C_, R.1 (C_, R.1 (f0 (f18 k j) k))
{ Compute ; }
C_, R.1 (f0 (C_, R.1 (f18 k j)) k) = C_, R.1 (C_, R.1 (f0 (f18 k j) k))
{ Compute ; }
C_, R.1 (C_, R.1 (f0 (f18 k j) k)) = C_, R.1 (C_, R.1 (f0 (f18 k j) k))
{ Triv }
Done.

```

```

f18 a (f0 b g) = f0 (f18 a g) b
f18 a (f0 b g) = f0 (f18 a g) b
{ Induction on b }
T) f18 a (f0 C_, R.0 g) = f0 (f18 a g) C_, R.0
{ Compute ; }
f18 a g = f0 (f18 a g) C_, R.0
{ ; Fund: f0 a C_, R.0 = a }
f18 a g = f18 a g

```

{ Triv }

Done.

H) $f_{18} a (f_0 j g) = f_0 (f_{18} a g) j$

T) $f_{18} a (f_0 (C_{-,R.1} j) g) = f_0 (f_{18} a g) (C_{-,R.1} j)$

{ Compute ; }

$f_{18} a (C_{-,R.1} (f_0 j g)) = f_0 (f_{18} a g) (C_{-,R.1} j)$

{ Fund: $f_{18} a (C_{-,R.1} b) = C_{-,R.1} (f_{18} a b)$; Fund: $f_0 a (C_{-,R.1} b) = C_{-,R.1} (f_0 a b)$ }

$C_{-,R.1} (f_{18} a (f_0 j g)) = C_{-,R.1} (f_0 (f_{18} a g) j)$

{ Fund: $f_{18} a (f_0 j g) = f_0 (f_{18} a g) j$; }

$C_{-,R.1} (f_0 (f_{18} a g) j) = C_{-,R.1} (f_0 (f_{18} a g) j)$

{ Triv }

Done.

B.2.17 $f_{19} x y = 2 + y + (0 + 1 + 2 + \dots + x)$

$f_{19} :: C_{-,R} \rightarrow C_{-,R} \rightarrow C_{-,R}$

$f_{19} = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{ C_{-,R.0} \rightarrow C_{-,R.1} (C_{-,R.1} e); C_{-,R.1} g \rightarrow f_0 (f_{19} g e) g \}$

$f_{19} C_{-,R.0} a = C_{-,R.1} (C_{-,R.1} a)$

$f_{19} C_{-,R.0} a = C_{-,R.1} (C_{-,R.1} a)$

{ Compute ; }

$C_{-,R.1} (C_{-,R.1} a) = C_{-,R.1} (C_{-,R.1} a)$

{ Triv }

Done.

$f_{19} (C_{-,R.1} a) b = f_0 a (f_{19} a b)$

$f_{19} (C_{-,R.1} a) b = f_0 a (f_{19} a b)$

{ Compute ; }

$f_0 (f_{19} a b) a = f_0 a (f_{19} a b)$

{ Reor: $f_0 d c = f_0 c d$; }

$f_0 a (f_{19} a b) = f_0 a (f_{19} a b)$

{ Triv }

Done.

$f_{19} a (C_{-,R.1} b) = C_{-,R.1} (f_{19} a b)$

$f_{19} a (C_{-,R.1} b) = C_{-,R.1} (f_{19} a b)$

{ Induction on b }

T) $f_{19} a (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} (f_{19} a C_{-,R.0})$

{ Induction on a }

B) $f_{19} C_{-,R.0} (C_{-,R.1} C_{-,R.0}) = C_{-,R.1} (f_{19} C_{-,R.0} C_{-,R.0})$

{ Compute ; Compute }

$C_{-,R.1} (C_{-,R.1} (C_{-,R.1} C_{-,R.0})) = C_{-,R.1} (C_{-,R.1} (C_{-,R.1} C_{-,R.0}))$

{ Triv }

Done.

H) $f_{19} j (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{19} j C_{-}, R.0)$

T) $f_{19} (C_{-}, R.1 j) (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{19} (C_{-}, R.1 j) C_{-}, R.0)$
{ Compute ; Compute }
 $f_0 (f_{19} j (C_{-}, R.1 C_{-}, R.0)) j = C_{-}, R.1 (f_0 (f_{19} j C_{-}, R.0) j)$
{ Reor: $f_0 d c = f_0 c d$; Reor: $f_0 d c = f_0 c d$ }
 $f_0 j (f_{19} j (C_{-}, R.1 C_{-}, R.0)) = C_{-}, R.1 (f_0 j (f_{19} j C_{-}, R.0))$
{ Reor: $f_{19} j (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f_{19} j C_{-}, R.0)$; }
 $f_0 j (C_{-}, R.1 (f_{19} j C_{-}, R.0)) = C_{-}, R.1 (f_0 j (f_{19} j C_{-}, R.0))$
{ Fund: $f_0 a (C_{-}, R.1 b) = C_{-}, R.1 (f_0 a b)$; }
 $C_{-}, R.1 (f_0 j (f_{19} j C_{-}, R.0)) = C_{-}, R.1 (f_0 j (f_{19} j C_{-}, R.0))$
{ Triv }
Done.

H) $f_{19} a (C_{-}, R.1 j) = C_{-}, R.1 (f_{19} a j)$
 T) $f_{19} a (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (f_{19} a (C_{-}, R.1 j))$
{ ; Fund: $f_{19} a (C_{-}, R.1 j) = C_{-}, R.1 (f_{19} a j)$ }
 $f_{19} a (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f_{19} a j))$
{ Induction on a }

T) $f_{19} C_{-}, R.0 (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f_{19} C_{-}, R.0 j))$
{ Compute ; Compute }
 $C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 j))) = C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 (C_{-}, R.1 j)))$
{ Triv }
Done.

H) $f_{19} k (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f_{19} k j))$

T) $f_{19} (C_{-}, R.1 k) (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f_{19} (C_{-}, R.1 k) j))$
{ Compute ; Compute }
 $f_0 (f_{19} k (C_{-}, R.1 (C_{-}, R.1 j))) k = C_{-}, R.1 (C_{-}, R.1 (f_0 (f_{19} k j) k))$
{ Fund: $f_{19} k (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f_{19} k j))$; }
 $f_0 (C_{-}, R.1 (C_{-}, R.1 (f_{19} k j))) k = C_{-}, R.1 (C_{-}, R.1 (f_0 (f_{19} k j) k))$
{ Compute ; }
 $C_{-}, R.1 (f_0 (C_{-}, R.1 (f_{19} k j)) k) = C_{-}, R.1 (C_{-}, R.1 (f_0 (f_{19} k j) k))$
{ Compute ; }
 $C_{-}, R.1 (C_{-}, R.1 (f_0 (f_{19} k j) k)) = C_{-}, R.1 (C_{-}, R.1 (f_0 (f_{19} k j) k))$
{ Triv }
Done.

$f_{19} a (f_0 b g) = f_0 (f_{19} a g) b$

$f_{19} a (f_0 b g) = f_0 (f_{19} a g) b$

{ Induction on b }

T) $f_{19} a (f_0 C_{-}, R.0 g) = f_0 (f_{19} a g) C_{-}, R.0$

```

{ Compute ; }
f19 a g = f0 (f19 a g) C-,R.0
{ ; Fund: f0 a C-,R.0 = a }
f19 a g = f19 a g
{ Triv }
Done.

```

```

H)f19 a (f0 j g) = f0 (f19 a g) j
T)f19 a (f0 (C-,R.1 j) g) = f0 (f19 a g) (C-,R.1 j)
{ Compute ; }
f19 a (C-,R.1 (f0 j g)) = f0 (f19 a g) (C-,R.1 j)
{ Fund: f19 a (C-,R.1 b) = C-,R.1 (f19 a b) ; Fund: f0 a (C-,R.1 b) = C-,R.1 (f0 a b) }
C-,R.1 (f19 a (f0 j g)) = C-,R.1 (f0 (f19 a g) j)
{ Fund: f19 a (f0 j g) = f0 (f19 a g) j ; }
C-,R.1 (f0 (f19 a g) j) = C-,R.1 (f0 (f19 a g) j)
{ Triv }
Done.

```

B.2.18 f20 = f6

f20::C_{-,R} → C_{-,R} → C_{-,R}
f20 = λd → λe → case d of {C_{-,R.0} → C_{-,R.1} C_{-,R.0}; C_{-,R.1}g → f1 (f20 g e) (f20 g e)}

```

f20 C-,R.0 a = C-,R.1 C-,R.0
f20 C-,R.0 a = C-,R.1 C-,R.0
{ Compute ; }
C-,R.1 C-,R.0 = C-,R.1 C-,R.0
{ Triv }
Done.

```

```

f20 (C-,R.1 a) b = f1 (f20 a b) (f20 a b)
f20 (C-,R.1 a) b = f1 (f20 a b) (f20 a b)
{ Compute ; }
f1 (f20 a b) (f20 a b) = f1 (f20 a b) (f20 a b)
{ Triv }
Done.

```

```

f20 a (C-,R.1 b) = f20 a (f20 b b)
f20 a (C-,R.1 b) = f20 a (f20 b b)
{ Induction on b }
T)f20 a (C-,R.1 C-,R.0) = f20 a (f20 C-,R.0 C-,R.0)
{ ; Compute }
f20 a (C-,R.1 C-,R.0) = f20 a (C-,R.1 C-,R.0)
{ Triv }
Done.

```

```

H)f20 a (C-,R.1 j) = f20 a (f20 j j)

```

$T) f_{20} a (C_{-,R.1} (C_{-,R.1} j)) = f_{20} a (f_{20} (C_{-,R.1} j) (C_{-,R.1} j))$
 { ; Compute }
 $f_{20} a (C_{-,R.1} (C_{-,R.1} j)) = f_{20} a (f_1 (f_{20} j (C_{-,R.1} j)) (f_{20} j (C_{-,R.1} j)))$
 { ; Fund: $f_{20} a (C_{-,R.1} j) = f_{20} a (f_{20} j j)$ }
 $f_{20} a (C_{-,R.1} (C_{-,R.1} j)) = f_{20} a (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (C_{-,R.1} j)))$
 { ; Fund: $f_{20} a (C_{-,R.1} j) = f_{20} a (f_{20} j j)$ }
 $f_{20} a (C_{-,R.1} (C_{-,R.1} j)) = f_{20} a (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j)))$
 { Induction on a }

 $T) f_{20} C_{-,R.0} (C_{-,R.1} (C_{-,R.1} j)) = f_{20} C_{-,R.0} (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j)))$
 { Compute ; Compute }
 $C_{-,R.1} C_{-,R.0} = C_{-,R.1} C_{-,R.0}$
 { Triv }
 Done.

$H) f_{20} k (C_{-,R.1} (C_{-,R.1} j)) = f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j)))$

$T) f_{20} (C_{-,R.1} k) (C_{-,R.1} (C_{-,R.1} j)) = f_{20} (C_{-,R.1} k) (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j)))$
 { Compute ; Compute }
 $(\forall (f_1 (f_{20} k (C_{-,R.1} (C_{-,R.1} j)))) (f_{20} k (C_{-,R.1} (C_{-,R.1} j)))$
 $= f_1 (f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 $(f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 { Fund: $(\forall (f_{20} k (C_{-,R.1} (C_{-,R.1} j)))$
 $= f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))) ; }$
 $(\forall (f_1 (f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j)))) (f_{20} k (C_{-,R.1} (C_{-,R.1} j)))$
 $= f_1 (f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 $(f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 { Fund: $(\forall (f_{20} k (C_{-,R.1} (C_{-,R.1} j)))$
 $= f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))) ; }$
 $(\forall (f_1 (f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j)))) (f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 $= f_1 (f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 $(f_{20} k (f_1 (f_{20} j (f_{20} j j)) (f_{20} j (f_{20} j j))))$
 { Triv }
 Done.

$f_{20} b (f_{20} e g) = f_{20} b (f_{20} g e)$

$f_{20} b (f_{20} e g) = f_{20} b (f_{20} g e)$

{ Induction on b }

$T) f_{20} C_{-,R.0} (f_{20} e g) = f_{20} C_{-,R.0} (f_{20} g e)$

{ Compute ; Compute }

$C_{-,R.1} C_{-,R.0} = C_{-,R.1} C_{-,R.0}$

{ Triv }

Done.

$H) f_{20} j (f_{20} e g) = f_{20} j (f_{20} g e)$

$T) f_{20} (C_{-,R.1} j) (f_{20} e g) = f_{20} (C_{-,R.1} j) (f_{20} g e)$

```

{ Compute ; Compute }
f1 (f20 j (f20 e g)) (f20 j (f20 e g)) = f1 (f20 j (f20 g e)) (f20 j (f20 g e))
{ ; Reor: f20 j (f20 g e) = f20 j (f20 e g) }
f1 (f20 j (f20 e g)) (f20 j (f20 e g)) = f1 (f20 j (f20 e g)) (f20 j (f20 g e))
{ ; Reor: f20 j (f20 g e) = f20 j (f20 e g) }
f1 (f20 j (f20 e g)) (f20 j (f20 e g)) = f1 (f20 j (f20 e g)) (f20 j (f20 e g))
{ Triv }
Done.

```

B.2.19 $f_{21} = f_7$

$f_{21} :: C_-, R \rightarrow C_-, R \rightarrow C_-, R$

$f_{21} = \lambda d \rightarrow \lambda e \rightarrow \text{case } d \text{ of } \{ C_-, R.0 \rightarrow e; C_-, R.1 g \rightarrow f_1 (f_{21} g e) g \}$

$f_{21} C_-, R.0 a = a$

```

f21 C_-, R.0 a = a
  { Compute ; }
a = a
  { Triv }
Done.

```

$f_{21} (C_-, R.1 a) b = f_1 a (f_{21} a b)$

```

f21 (C_-, R.1 a) b = f1 a (f21 a b)
  { Compute ; }
f1 (f21 a b) a = f1 a (f21 a b)
  { Reor: f1 d c = f1 c d ; }
f1 a (f21 a b) = f1 a (f21 a b)
  { Triv }
Done.

```

$f_{21} a (C_-, R.1 b) = C_-, R.1 (f_{21} a b)$

$f_{21} a (C_-, R.1 b) = C_-, R.1 (f_{21} a b)$

{ Induction on b }

$T) f_{21} a (C_-, R.1 C_-, R.0) = C_-, R.1 (f_{21} a C_-, R.0)$

{ Induction on a }

$T) f_{21} C_-, R.0 (C_-, R.1 C_-, R.0) = C_-, R.1 (f_{21} C_-, R.0 C_-, R.0)$

{ Compute ; Compute }

$C_-, R.1 C_-, R.0 = C_-, R.1 C_-, R.0$

{ Triv }

Done.

$H) f_{21} j (C_-, R.1 C_-, R.0) = C_-, R.1 (f_{21} j C_-, R.0)$

$T) f_{21} (C_-, R.1 j) (C_-, R.1 C_-, R.0) = C_-, R.1 (f_{21} (C_-, R.1 j) C_-, R.0)$

{ Compute ; Compute }

$f1 (f21 j (C_{-}, R.1 C_{-}, R.0)) j = C_{-}, R.1 (f1 (f21 j C_{-}, R.0) j)$
 { Reor: $f1 d c = f1 c d$; Reor: $f1 d c = f1 c d$ }
 $f1 j (f21 j (C_{-}, R.1 C_{-}, R.0)) = C_{-}, R.1 (f1 j (f21 j C_{-}, R.0))$
 { Reor: $f21 j (C_{-}, R.1 C_{-}, R.0) = C_{-}, R.1 (f21 j C_{-}, R.0)$; }
 $f1 j (C_{-}, R.1 (f21 j C_{-}, R.0)) = C_{-}, R.1 (f1 j (f21 j C_{-}, R.0))$
 { Fund: $f1 a (C_{-}, R.1 b) = C_{-}, R.1 (f1 a b)$; }
 $C_{-}, R.1 (f1 j (f21 j C_{-}, R.0)) = C_{-}, R.1 (f1 j (f21 j C_{-}, R.0))$
 { Triv }
 Done.

H) $f21 a (C_{-}, R.1 j) = C_{-}, R.1 (f21 a j)$
T) $f21 a (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (f21 a (C_{-}, R.1 j))$
 { ; Fund: $f21 a (C_{-}, R.1 j) = C_{-}, R.1 (f21 a j)$ }
 $f21 a (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f21 a j))$
 { Induction on a }

T) $f21 C_{-}, R.0 (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f21 C_{-}, R.0 j))$
 { Compute ; Compute }
 $C_{-}, R.1 (C_{-}, R.1 j) = C_{-}, R.1 (C_{-}, R.1 j)$
 { Triv }
 Done.

H) $f21 k (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f21 k j))$

T) $f21 (C_{-}, R.1 k) (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f21 (C_{-}, R.1 k) j))$
 { Compute ; Compute }
 $f1 (f21 k (C_{-}, R.1 (C_{-}, R.1 j))) k = C_{-}, R.1 (C_{-}, R.1 (f1 (f21 k j) k))$
 { Fund: $f21 k (C_{-}, R.1 (C_{-}, R.1 j)) = C_{-}, R.1 (C_{-}, R.1 (f21 k j))$; }
 $f1 (C_{-}, R.1 (C_{-}, R.1 (f21 k j))) k = C_{-}, R.1 (C_{-}, R.1 (f1 (f21 k j) k))$
 { Compute ; }
 $C_{-}, R.1 (f1 (C_{-}, R.1 (f21 k j)) k) = C_{-}, R.1 (C_{-}, R.1 (f1 (f21 k j) k))$
 { Compute ; }
 $C_{-}, R.1 (C_{-}, R.1 (f1 (f21 k j) k)) = C_{-}, R.1 (C_{-}, R.1 (f1 (f21 k j) k))$
 { Triv }
 Done.

$f21 a (f1 b g) = f1 (f21 a g) b$
 $f21 a (f1 b g) = f1 (f21 a g) b$
 { Induction on b }
T) $f21 a (f1 C_{-}, R.0 g) = f1 (f21 a g) C_{-}, R.0$
 { Compute ; }
 $f21 a (C_{-}, R.1 g) = f1 (f21 a g) C_{-}, R.0$
 { Fund: $f21 a (C_{-}, R.1 b) = C_{-}, R.1 (f21 a b)$; Fund: $f1 a C_{-}, R.0 = C_{-}, R.1 a$ }
 $C_{-}, R.1 (f21 a g) = C_{-}, R.1 (f21 a g)$
 { Triv }

Done.

```
H)f21 a (f1 j g) = f1 (f21 a g) j
T)f21 a (f1 (C-,R.1 j) g) = f1 (f21 a g) (C-,R.1 j)
{ Compute ; }
f21 a (C-,R.1 (f1 j g)) = f1 (f21 a g) (C-,R.1 j)
{ Fund: f21 a (C-,R.1 b) = C-,R.1 (f21 a b) ; Fund: f1 a (C-,R.1 b) = C-,R.1 (f1 a b) }
C-,R.1 (f21 a (f1 j g)) = C-,R.1 (f1 (f21 a g) j)
{ Fund: f21 a (f1 j g) = f1 (f21 a g) j ; }
C-,R.1 (f1 (f21 a g) j) = C-,R.1 (f1 (f21 a g) j)
{ Triv }
Done.
```

B.2.20 f22 = f8

```
f22::C-,R → C-,R → C-,R
f22 = λd → λe → case d of {C-,R.0 → C-,R.1 e; C-,R.1g → f1 (f22 g e) g}
```

```
f22 C-,R.0 a = C-,R.1 a
f22 C-,R.0 a = C-,R.1 a
{ Compute ; }
C-,R.1 a = C-,R.1 a
{ Triv }
Done.
```

```
f22 (C-,R.1 a) b = f1 a (f22 a b)
f22 (C-,R.1 a) b = f1 a (f22 a b)
{ Compute ; }
f1 (f22 a b) a = f1 a (f22 a b)
{ Reor: f1 d c = f1 c d ; }
f1 a (f22 a b) = f1 a (f22 a b)
{ Triv }
Done.
```

```
f22 a (C-,R.1 b) = C-,R.1 (f22 a b)
f22 a (C-,R.1 b) = C-,R.1 (f22 a b)
{ Induction on b }
T)f22 a (C-,R.1 C-,R.0) = C-,R.1 (f22 a C-,R.0)
{ Induction on a }
```

```
T)f22 C-,R.0 (C-,R.1 C-,R.0) = C-,R.1 (f22 C-,R.0 C-,R.0)
{ Compute ; Compute }
C-,R.1 (C-,R.1 C-,R.0) = C-,R.1 (C-,R.1 C-,R.0)
{ Triv }
Done.
```

H) $f22\ j\ (C_, R.1\ C_, R.0) = C_, R.1\ (f22\ j\ C_, R.0)$

T) $f22\ (C_, R.1\ j)\ (C_, R.1\ C_, R.0) = C_, R.1\ (f22\ (C_, R.1\ j)\ C_, R.0)$
{ Compute ; Compute }
 $f1\ (f22\ j\ (C_, R.1\ C_, R.0))\ j = C_, R.1\ (f1\ (f22\ j\ C_, R.0)\ j)$
{ Reor: $f1\ d\ c = f1\ c\ d$; Reor: $f1\ d\ c = f1\ c\ d$ }
 $f1\ j\ (f22\ j\ (C_, R.1\ C_, R.0)) = C_, R.1\ (f1\ j\ (f22\ j\ C_, R.0))$
{ Reor: $f22\ j\ (C_, R.1\ C_, R.0) = C_, R.1\ (f22\ j\ C_, R.0)$; }
 $f1\ j\ (C_, R.1\ (f22\ j\ C_, R.0)) = C_, R.1\ (f1\ j\ (f22\ j\ C_, R.0))$
{ Fund: $f1\ a\ (C_, R.1\ b) = C_, R.1\ (f1\ a\ b)$; }
 $C_, R.1\ (f1\ j\ (f22\ j\ C_, R.0)) = C_, R.1\ (f1\ j\ (f22\ j\ C_, R.0))$
{ Triv }
Done.

H) $f22\ a\ (C_, R.1\ j) = C_, R.1\ (f22\ a\ j)$
T) $f22\ a\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (f22\ a\ (C_, R.1\ j))$
{ ; Fund: $f22\ a\ (C_, R.1\ j) = C_, R.1\ (f22\ a\ j)$ }
 $f22\ a\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (C_, R.1\ (f22\ a\ j))$
{ Induction on a }

T) $f22\ C_, R.0\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (C_, R.1\ (f22\ C_, R.0\ j))$
{ Compute ; Compute }
 $C_, R.1\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (C_, R.1\ (C_, R.1\ j))$
{ Triv }
Done.

H) $f22\ k\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (C_, R.1\ (f22\ k\ j))$

T) $f22\ (C_, R.1\ k)\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (C_, R.1\ (f22\ (C_, R.1\ k)\ j))$
{ Compute ; Compute }
 $f1\ (f22\ k\ (C_, R.1\ (C_, R.1\ j)))\ k = C_, R.1\ (C_, R.1\ (f1\ (f22\ k\ j)\ k))$
{ Fund: $f22\ k\ (C_, R.1\ (C_, R.1\ j)) = C_, R.1\ (C_, R.1\ (f22\ k\ j))$; }
 $f1\ (C_, R.1\ (C_, R.1\ (f22\ k\ j)))\ k = C_, R.1\ (C_, R.1\ (f1\ (f22\ k\ j)\ k))$
{ Compute ; }
 $C_, R.1\ (f1\ (C_, R.1\ (f22\ k\ j))\ k) = C_, R.1\ (C_, R.1\ (f1\ (f22\ k\ j)\ k))$
{ Compute ; }
 $C_, R.1\ (C_, R.1\ (f1\ (f22\ k\ j)\ k)) = C_, R.1\ (C_, R.1\ (f1\ (f22\ k\ j)\ k))$
{ Triv }
Done.

$f22\ a\ (f1\ b\ g) = f1\ (f22\ a\ g)\ b$

$f22\ a\ (f1\ b\ g) = f1\ (f22\ a\ g)\ b$

{ Induction on b }

T) $f22\ a\ (f1\ C_, R.0\ g) = f1\ (f22\ a\ g)\ C_, R.0$

{ Compute ; }

$f22\ a\ (C_,R.1\ g) = f1\ (f22\ a\ g)\ C_,R.0$
 { Fund: $f22\ a\ (C_,R.1\ b) = C_,R.1\ (f22\ a\ b)$; Fund: $f1\ a\ C_,R.0 = C_,R.1\ a$ }
 $C_,R.1\ (f22\ a\ g) = C_,R.1\ (f22\ a\ g)$
 { Triv }
 Done.

$H)f22\ a\ (f1\ j\ g) = f1\ (f22\ a\ g)\ j$
 $T)f22\ a\ (f1\ (C_,R.1\ j)\ g) = f1\ (f22\ a\ g)\ (C_,R.1\ j)$
 { Compute ; }
 $f22\ a\ (C_,R.1\ (f1\ j\ g)) = f1\ (f22\ a\ g)\ (C_,R.1\ j)$
 { Fund: $f22\ a\ (C_,R.1\ b) = C_,R.1\ (f22\ a\ b)$; Fund: $f1\ a\ (C_,R.1\ b) = C_,R.1\ (f1\ a\ b)$ }
 $C_,R.1\ (f22\ a\ (f1\ j\ g)) = C_,R.1\ (f1\ (f22\ a\ g)\ j)$
 { Fund: $f22\ a\ (f1\ j\ g) = f1\ (f22\ a\ g)\ j$; }
 $C_,R.1\ (f1\ (f22\ a\ g)\ j) = C_,R.1\ (f1\ (f22\ a\ g)\ j)$
 { Triv }
 Done.

B.2.21 $f23 = f19$

$f23::C_,R \rightarrow C_,R \rightarrow C_,R$
 $f23 = \lambda d \rightarrow \lambda e \rightarrow case\ d\ of\ \{C_,R.0 \rightarrow C_,R.1\ (C_,R.1\ e); C_,R.1g \rightarrow f1\ (f23\ g\ e)\ g\}$

$f23\ C_,R.0\ a = C_,R.1\ (C_,R.1\ a)$
 $f23\ C_,R.0\ a = C_,R.1\ (C_,R.1\ a)$
 { Compute ; }
 $C_,R.1\ (C_,R.1\ a) = C_,R.1\ (C_,R.1\ a)$
 { Triv }
 Done.

$f23\ (C_,R.1\ a)\ b = f1\ a\ (f23\ a\ b)$
 $f23\ (C_,R.1\ a)\ b = f1\ a\ (f23\ a\ b)$
 { Compute ; }
 $f1\ (f23\ a\ b)\ a = f1\ a\ (f23\ a\ b)$
 { Reor: $f1\ d\ c = f1\ c\ d$; }
 $f1\ a\ (f23\ a\ b) = f1\ a\ (f23\ a\ b)$
 { Triv }
 Done.

$f23\ a\ (C_,R.1\ b) = C_,R.1\ (f23\ a\ b)$
 $f23\ a\ (C_,R.1\ b) = C_,R.1\ (f23\ a\ b)$
 { Induction on b }
 $T)f23\ a\ (C_,R.1\ C_,R.0) = C_,R.1\ (f23\ a\ C_,R.0)$
 { Induction on a }

$T)f23\ C_,R.0\ (C_,R.1\ C_,R.0) = C_,R.1\ (f23\ C_,R.0\ C_,R.0)$
 { Compute ; Compute }
 $C_,R.1\ (C_,R.1\ (C_,R.1\ C_,R.0)) = C_,R.1\ (C_,R.1\ (C_,R.1\ C_,R.0))$

{ Triv }
Done.

H)f23 j (C₋,R.1 C₋,R.0) = C₋,R.1 (f23 j C₋,R.0)

T)f23 (C₋,R.1 j) (C₋,R.1 C₋,R.0) = C₋,R.1 (f23 (C₋,R.1 j) C₋,R.0)
{ Compute ; Compute }
f1 (f23 j (C₋,R.1 C₋,R.0)) j = C₋,R.1 (f1 (f23 j C₋,R.0) j)
{ Reor: f1 d c = f1 c d ; Reor: f1 d c = f1 c d }
f1 j (f23 j (C₋,R.1 C₋,R.0)) = C₋,R.1 (f1 j (f23 j C₋,R.0))
{ Reor: f23 j (C₋,R.1 C₋,R.0) = C₋,R.1 (f23 j C₋,R.0) ; }
f1 j (C₋,R.1 (f23 j C₋,R.0)) = C₋,R.1 (f1 j (f23 j C₋,R.0))
{ Fund: f1 a (C₋,R.1 b) = C₋,R.1 (f1 a b) ; }
C₋,R.1 (f1 j (f23 j C₋,R.0)) = C₋,R.1 (f1 j (f23 j C₋,R.0))
{ Triv }
Done.

H)f23 a (C₋,R.1 j) = C₋,R.1 (f23 a j)
T)f23 a (C₋,R.1 (C₋,R.1 j)) = C₋,R.1 (f23 a (C₋,R.1 j))
{ ; Fund: f23 a (C₋,R.1 j) = C₋,R.1 (f23 a j) }
f23 a (C₋,R.1 (C₋,R.1 j)) = C₋,R.1 (C₋,R.1 (f23 a j))
{ Induction on a }

T)f23 C₋,R.0 (C₋,R.1 (C₋,R.1 j)) = C₋,R.1 (C₋,R.1 (f23 C₋,R.0 j))
{ Compute ; Compute }
C₋,R.1 (C₋,R.1 (C₋,R.1 (C₋,R.1 j))) = C₋,R.1 (C₋,R.1 (C₋,R.1 (C₋,R.1 j)))
{ Triv }
Done.

H)f23 k (C₋,R.1 (C₋,R.1 j)) = C₋,R.1 (C₋,R.1 (f23 k j))

T)f23 (C₋,R.1 k) (C₋,R.1 (C₋,R.1 j)) = C₋,R.1 (C₋,R.1 (f23 (C₋,R.1 k) j))
{ Compute ; Compute }
f1 (f23 k (C₋,R.1 (C₋,R.1 j))) k = C₋,R.1 (C₋,R.1 (f1 (f23 k j) k))
{ Fund: f23 k (C₋,R.1 (C₋,R.1 j)) = C₋,R.1 (C₋,R.1 (f23 k j)) ; }
f1 (C₋,R.1 (C₋,R.1 (f23 k j))) k = C₋,R.1 (C₋,R.1 (f1 (f23 k j) k))
{ Compute ; }
C₋,R.1 (f1 (C₋,R.1 (f23 k j)) k) = C₋,R.1 (C₋,R.1 (f1 (f23 k j) k))
{ Compute ; }
C₋,R.1 (C₋,R.1 (f1 (f23 k j) k)) = C₋,R.1 (C₋,R.1 (f1 (f23 k j) k))
{ Triv }
Done.

$f23\ a\ (f1\ b\ g) = f1\ (f23\ a\ g)\ b$
 $f23\ a\ (f1\ b\ g) = f1\ (f23\ a\ g)\ b$
 { Induction on b }
 $T) f23\ a\ (f1\ C_{-},R.0\ g) = f1\ (f23\ a\ g)\ C_{-},R.0$
 { Compute ; }
 $f23\ a\ (C_{-},R.1\ g) = f1\ (f23\ a\ g)\ C_{-},R.0$
 { Fund: $f23\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f23\ a\ b)$; Fund: $f1\ a\ C_{-},R.0 = C_{-},R.1\ a$ }
 $C_{-},R.1\ (f23\ a\ g) = C_{-},R.1\ (f23\ a\ g)$
 { Triv }
 Done.

$H) f23\ a\ (f1\ j\ g) = f1\ (f23\ a\ g)\ j$
 $T) f23\ a\ (f1\ (C_{-},R.1\ j)\ g) = f1\ (f23\ a\ g)\ (C_{-},R.1\ j)$
 { Compute ; }
 $f23\ a\ (C_{-},R.1\ (f1\ j\ g)) = f1\ (f23\ a\ g)\ (C_{-},R.1\ j)$
 { Fund: $f23\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f23\ a\ b)$; Fund: $f1\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f1\ a\ b)$ }
 $C_{-},R.1\ (f23\ a\ (f1\ j\ g)) = C_{-},R.1\ (f1\ (f23\ a\ g)\ j)$
 { Fund: $f23\ a\ (f1\ j\ g) = f1\ (f23\ a\ g)\ j$; }
 $C_{-},R.1\ (f1\ (f23\ a\ g)\ j) = C_{-},R.1\ (f1\ (f23\ a\ g)\ j)$
 { Triv }
 Done.

B.2.22 $f24\ xs\ n = n * ((length\ xs) + 1)$

$f24::C_{-}, a.R \rightarrow C_{-}, R \rightarrow C_{-}, R$
 $f24 = \lambda c \rightarrow \lambda e \rightarrow case\ c\ of\ \{C_{-}, a.R.0 \rightarrow C_{-}, R.0; C_{-}, a.R.1gh \rightarrow f0\ e\ (f24\ h\ e)\}$

$f24\ C_{-}, a.R.0\ a = C_{-}, R.0$

$f24\ C_{-}, a.R.0\ a = C_{-}, R.0$

{ Compute ; }

$C_{-}, R.0 = C_{-}, R.0$

{ Triv }

Done.

$f24\ a\ C_{-}, R.0 = C_{-}, R.0$

$f24\ a\ C_{-}, R.0 = C_{-}, R.0$

{ Induction on a }

$T) f24\ C_{-}, a.R.0\ C_{-}, R.0 = C_{-}, R.0$

{ Compute ; }

$C_{-}, R.0 = C_{-}, R.0$

{ Triv }

Done.

$H) f24\ j\ C_{-}, R.0 = C_{-}, R.0$

$T) f24\ (C_{-}, a.R.1\ c\ j)\ C_{-}, R.0 = C_{-}, R.0$

{ Compute ; }

$f0\ C_{-}, R.0\ (f24\ j\ C_{-}, R.0) = C_{-}, R.0$

```

{ Compute ; }
f24 j C-, R.0 = C-, R.0
{ Fund: f24 j C-, R.0 = C-, R.0 ; }
C-, R.0 = C-, R.0
{ Triv }
Done.

```

```

f24 (C-, a.R.1 a b) g = f0 (f24 b g) g
f24 (C-, a.R.1 a b) g = f0 (f24 b g) g
  { Compute ; }
f0 g (f24 b g) = f0 (f24 b g) g
  { Reor: f0 d c = f0 c d ; }
f0 (f24 b g) g = f0 (f24 b g) g
  { Triv }
Done.

```

```

f24 a (f0 b g) = f0 (f24 a b) (f24 a g)
f24 a (f0 b g) = f0 (f24 a b) (f24 a g)
  { Induction on b }
T) f24 a (f0 C-, R.0 g) = f0 (f24 a C-, R.0) (f24 a g)
  { Compute ; }
f24 a g = f0 (f24 a C-, R.0) (f24 a g)
  { ; Fund: f24 a C-, R.0 = C-, R.0 }
f24 a g = f0 C-, R.0 (f24 a g)
  { ; Compute }
f24 a g = f24 a g
  { Triv }
Done.

```

```

H) f24 a (f0 c g) = f0 (f24 a c) (f24 a g)
T) f24 a (f0 (C-, R.1 c) g) = f0 (f24 a (C-, R.1 c)) (f24 a g)
  { Compute ; }
f24 a (C-, R.1 (f0 c g)) = f0 (f24 a (C-, R.1 c)) (f24 a g)
  { Induction on g }

```

```

T) f24 a (C-, R.1 (f0 c C-, R.0)) = f0 (f24 a (C-, R.1 c)) (f24 a C-, R.0)
  { Fund: f0 a C-, R.0 = a ; Fund: f24 a C-, R.0 = C-, R.0 }
f24 a (C-, R.1 c) = f0 (f24 a (C-, R.1 c)) C-, R.0
  { ; Fund: f0 a C-, R.0 = a }
f24 a (C-, R.1 c) = f24 a (C-, R.1 c)
  { Triv }
Done.

```

```

H) f24 a (C-, R.1 (f0 c j)) = f0 (f24 a (C-, R.1 c)) (f24 a j)

```

```

T) f24 a (C-, R.1 (f0 c (C-, R.1 j))) = f0 (f24 a (C-, R.1 c)) (f24 a (C-, R.1 j))

```

{ Fund: $f0\ a\ (C_{-},R.1\ b) = C_{-},R.1\ (f0\ a\ b)$; }
 $f24\ a\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j))) = f0\ (f24\ a\ (C_{-},R.1\ c))\ (f24\ a\ (C_{-},R.1\ j))$
 { Induction on a }

T) $f24\ C_{-},a.R.0\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j))) = f0\ (f24\ C_{-},a.R.0\ (C_{-},R.1\ c))\ (f24\ C_{-},a.R.0\ (C_{-},R.1\ j))$
 { Compute ; Compute }
 $C_{-},R.0 = f0\ C_{-},R.0\ (f24\ C_{-},a.R.0\ (C_{-},R.1\ j))$
 { ; Compute }
 $C_{-},R.0 = f24\ C_{-},a.R.0\ (C_{-},R.1\ j)$
 { ; Compute }
 $C_{-},R.0 = C_{-},R.0$
 { Triv }
 Done.

H) $f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j))) = f0\ (f24\ l\ (C_{-},R.1\ c))\ (f24\ l\ (C_{-},R.1\ j))$

T) $f24\ (C_{-},a.R.1\ k\ l)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j))) = f0\ (f24\ (C_{-},a.R.1\ k\ l)\ (C_{-},R.1\ c))\ (f24\ (C_{-},a.R.1\ k\ l)\ (C_{-},R.1\ j))$
 { Compute ; Compute }
 $(\forall)\ (f0\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j)))\ (f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j))))$
 $= f0\ (f0\ (C_{-},R.1\ c)\ (f24\ l\ (C_{-},R.1\ c)))\ (f24\ (C_{-},a.R.1\ k\ l)\ (C_{-},R.1\ j))$
 { Compute ; Compute }
 $(\forall)\ (C_{-},R.1\ (f0\ (C_{-},R.1\ (f0\ c\ j))\ (f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j)))))$
 $= f0\ (C_{-},R.1\ (f0\ c\ (f24\ l\ (C_{-},R.1\ c))))\ (f24\ (C_{-},a.R.1\ k\ l)\ (C_{-},R.1\ j))$
 { Compute ; Compute }
 $(\forall)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f0\ c\ j)\ (f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j)))))$
 $= C_{-},R.1\ (f0\ (f0\ c\ (f24\ l\ (C_{-},R.1\ c)))\ (f24\ (C_{-},a.R.1\ k\ l)\ (C_{-},R.1\ j)))$
 { ; Compute }
 $(\forall)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f0\ c\ j)\ (f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j)))))$
 $= C_{-},R.1\ (f0\ (f0\ c\ (f24\ l\ (C_{-},R.1\ c)))\ (f0\ (C_{-},R.1\ j)\ (f24\ l\ (C_{-},R.1\ j))))$
 { ; Compute }
 $(\forall)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f0\ c\ j)\ (f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j)))))$
 $= C_{-},R.1\ (f0\ (f0\ c\ (f24\ l\ (C_{-},R.1\ c)))\ (C_{-},R.1\ (f0\ j\ (f24\ l\ (C_{-},R.1\ j))))$
 { Fund: $(\forall)\ (f24\ l\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ j)))$
 $= f0\ (f24\ l\ (C_{-},R.1\ c))\ (f24\ l\ (C_{-},R.1\ j))$; Fund: $(\forall b::C_{-},R, a::C_{-},R)\ (f0\ a\ (C_{-},R.1\ b)$
 $= C_{-},R.1\ (f0\ a\ b)$ }
 $(\forall)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ (f0\ c\ j)\ (f0\ (f24\ l\ (C_{-},R.1\ c))\ (f24\ l\ (C_{-},R.1\ j))))$
 $= C_{-},R.1\ (C_{-},R.1\ (f0\ (f0\ c\ (f24\ l\ (C_{-},R.1\ c)))\ (f0\ j\ (f24\ l\ (C_{-},R.1\ j))))$
 { Reor: $(\forall b::C_{-},R, e::C_{-},R, g::C_{-},R)\ (f0\ (f0\ b\ e)\ g$
 $= f0\ b\ (f0\ e\ g)$; Reor: $(\forall a::C_{-},R, b::C_{-},R, g::C_{-},R)\ (f0\ (f0\ a\ g)\ b$
 $= f0\ (f0\ a\ b)\ g$ }
 $(\forall)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ (f0\ j\ (f0\ (f24\ l\ (C_{-},R.1\ c))\ (f24\ l\ (C_{-},R.1\ j)))))$
 $= C_{-},R.1\ (C_{-},R.1\ (f0\ (f0\ c\ (f0\ j\ (f24\ l\ (C_{-},R.1\ j))))\ (f24\ l\ (C_{-},R.1\ c))))$
 { ; Reor: $(\forall b::C_{-},R, e::C_{-},R, g::C_{-},R)\ (f0\ (f0\ b\ e)\ g$
 $= f0\ b\ (f0\ e\ g)$ }
 $(\forall)\ (C_{-},R.1\ (C_{-},R.1\ (f0\ c\ (f0\ j\ (f0\ (f24\ l\ (C_{-},R.1\ c))\ (f24\ l\ (C_{-},R.1\ j)))))$
 $= C_{-},R.1\ (C_{-},R.1\ (f0\ c\ (f0\ (f0\ j\ (f24\ l\ (C_{-},R.1\ j))\ (f24\ l\ (C_{-},R.1\ c))))$
 { ; Reor: $(\forall a::C_{-},R, b::C_{-},R, g::C_{-},R)\ (f0\ (f0\ a\ g)\ b$
 $= f0\ (f0\ a\ b)\ g$ }

```

(∀)(C_,R.1 (C_,R.1 (f0 c (f0 j (f0 (f24 l (C_,R.1 c)) (f24 l (C_,R.1 j)))))))
      = C_,R.1 (C_,R.1 (f0 c (f0 (f0 j (f24 l (C_,R.1 c)) (f24 l (C_,R.1 j))))))
{ ; Reor: (∀b::C_,R, e::C_,R, g::C_,R)(f0 (f0 b e) g
          = f0 b (f0 e g) ) }
(∀)(C_,R.1 (C_,R.1 (f0 c (f0 j (f0 (f24 l (C_,R.1 c)) (f24 l (C_,R.1 j)))))))
      = C_,R.1 (C_,R.1 (f0 c (f0 j (f0 (f24 l (C_,R.1 c)) (f24 l (C_,R.1 j))))))
{ Triv }
Done.

```

B.2.23 $f25 [x_0, x_1, \dots, x_n] ys = [x_0 : ys, x_1 : ys, \dots, x_n : ys]$

```
f25::C_, a.R → C_, a.R → C_, a.R
```

```
f25 = λe → λg → case e of {C_, a.R.0 → C_, a.R.0; C_, a.R.1 hi → C_, a.R.1 h (f2 g (f25 i g))}
```

```
f25 C_, a.R.0 a = C_, a.R.0
```

```
f25 C_, a.R.0 a = C_, a.R.0
```

```
{ Compute ; }
```

```
C_, a.R.0 = C_, a.R.0
```

```
{ Triv }
```

```
Done.
```

```
f25 a C_, a.R.0 = a
```

```
f25 a C_, a.R.0 = a
```

```
{ Induction on a }
```

```
T) f25 C_, a.R.0 C_, a.R.0 = C_, a.R.0
```

```
{ Compute ; }
```

```
C_, a.R.0 = C_, a.R.0
```

```
{ Triv }
```

```
Done.
```

```
H) f25 k C_, a.R.0 = k
```

```
T) f25 (C_, a.R.1 j k) C_, a.R.0 = C_, a.R.1 j k
```

```
{ Compute ; }
```

```
C_, a.R.1 j (f2 C_, a.R.0 (f25 k C_, a.R.0)) = C_, a.R.1 j k
```

```
{ Compute ; }
```

```
C_, a.R.1 j (f25 k C_, a.R.0) = C_, a.R.1 j k
```

```
{ Fund: f25 k C_, a.R.0 = k ; }
```

```
C_, a.R.1 j k = C_, a.R.1 j k
```

```
{ Triv }
```

```
Done.
```

$f_{25} (f_2 a b) g = f_2 (f_{25} a g) (f_{25} b g)$
 $f_{25} (f_2 a b) g = f_2 (f_{25} a g) (f_{25} b g)$
 { Induction on a }
T) $f_{25} (f_2 C_{-, a.R.0} b) g = f_2 (f_{25} C_{-, a.R.0} g) (f_{25} b g)$
 { Compute ; Compute }
 $f_{25} b g = f_2 C_{-, a.R.0} (f_{25} b g)$
 { ; Compute }
 $f_{25} b g = f_{25} b g$
 { Triv }
 Done.

H) $f_{25} (f_2 k b) g = f_2 (f_{25} k g) (f_{25} b g)$
T) $f_{25} (f_2 (C_{-, a.R.1} j k) b) g = f_2 (f_{25} (C_{-, a.R.1} j k) g) (f_{25} b g)$
 { Compute ; Compute }
 $f_{25} (C_{-, a.R.1} j (f_2 k b)) g = f_2 (C_{-, a.R.1} j (f_2 g (f_{25} k g))) (f_{25} b g)$
 { Compute ; Compute }
 $C_{-, a.R.1} j (f_2 g (f_{25} (f_2 k b) g)) = C_{-, a.R.1} j (f_2 (f_2 g (f_{25} k g)) (f_{25} b g))$
 { Fund: $f_{25} (f_2 k b) g = f_2 (f_{25} k g) (f_{25} b g)$; }
 $C_{-, a.R.1} j (f_2 g (f_2 (f_{25} k g) (f_{25} b g))) = C_{-, a.R.1} j (f_2 (f_2 g (f_{25} k g)) (f_{25} b g))$
 { ; Reor: $f_2 (f_2 b e) g = f_2 b (f_2 e g)$ }
 $C_{-, a.R.1} j (f_2 g (f_2 (f_{25} k g) (f_{25} b g))) = C_{-, a.R.1} j (f_2 g (f_2 (f_{25} k g) (f_{25} b g)))$
 { Triv }
 Done.

B.2.24 $f_{26} [0, 1, 2, 3] _ = [0, 1, 2, 3, 3, 2, 3, 3, 1, 2, 3, 3, 2, 3, 3]$

$f_{26} :: C_{-, a.R} \rightarrow C_{-, a.R} \rightarrow C_{-, a.R}$
 $f_{26} = \lambda e \rightarrow \lambda g \rightarrow \text{case } e \text{ of } \{ C_{-, a.R.0} \rightarrow C_{-, a.R.0}; C_{-, a.R.1hi} \rightarrow C_{-, a.R.1} h (f_2 (f_{26} i g) (f_{26} i g)) \}$

$f_{26} C_{-, a.R.0} a = C_{-, a.R.0}$
 $f_{26} C_{-, a.R.0} a = C_{-, a.R.0}$
 { Compute ; }
 $C_{-, a.R.0} = C_{-, a.R.0}$
 { Triv }
 Done.

$f_{26} a (f_2 b g) = f_{26} a b$
 $f_{26} a (f_2 b g) = f_{26} a b$
 { Induction on b }
T) $f_{26} a (f_2 C_{-, a.R.0} g) = f_{26} a C_{-, a.R.0}$
 { Compute ; }
 $f_{26} a g = f_{26} a C_{-, a.R.0}$
 { Induction on g }

T) $f_{26} a C_{-, a.R.0} = f_{26} a C_{-, a.R.0}$
 { Triv }

Done.

$$H) f26 a k = f26 a C_, a.R.0$$

$$T) f26 a (C_, a.R.1 j k) = f26 a C_, a.R.0$$

{ ; Fund: $f26 a C_, a.R.0 = f26 a k$ }

$$f26 a (C_, a.R.1 j k) = f26 a k$$

{ Induction on a }

$$T) f26 C_, a.R.0 (C_, a.R.1 j k) = f26 C_, a.R.0 k$$

{ Compute ; Compute }

$$C_, a.R.0 = C_, a.R.0$$

{ Triv }

Done.

$$H) f26 m (C_, a.R.1 j k) = f26 m k$$

$$T) f26 (C_, a.R.1 l m) (C_, a.R.1 j k) = f26 (C_, a.R.1 l m) k$$

{ Compute ; Compute }

$$C_, a.R.1 l (f2 (f26 m (C_, a.R.1 j k)) (f26 m (C_, a.R.1 j k))) = C_, a.R.1 l (f2 (f26 m k) (f26 m k))$$

{ Fund: $f26 m (C_, a.R.1 j k) = f26 m k$; }

$$C_, a.R.1 l (f2 (f26 m k) (f26 m (C_, a.R.1 j k))) = C_, a.R.1 l (f2 (f26 m k) (f26 m k))$$

{ Fund: $f26 m (C_, a.R.1 j k) = f26 m k$; }

$$C_, a.R.1 l (f2 (f26 m k) (f26 m k)) = C_, a.R.1 l (f2 (f26 m k) (f26 m k))$$

{ Triv }

Done.

$$H) f26 a (f2 k g) = f26 a k$$

$$T) f26 a (f2 (C_, a.R.1 j k) g) = f26 a (C_, a.R.1 j k)$$

{ Compute ; }

$$f26 a (C_, a.R.1 j (f2 k g)) = f26 a (C_, a.R.1 j k)$$

{ Induction on g }

$$T) f26 a (C_, a.R.1 j (f2 k C_, a.R.0)) = f26 a (C_, a.R.1 j k)$$

{ Fund: $f2 a C_, a.R.0 = a$; }

$$f26 a (C_, a.R.1 j k) = f26 a (C_, a.R.1 j k)$$

{ Triv }

Done.

$$H) f26 a (C_, a.R.1 j (f2 k m)) = f26 a (C_, a.R.1 j k)$$

$$T) f26 a (C_, a.R.1 j (f2 k (C_, a.R.1 l m))) = f26 a (C_, a.R.1 j k)$$

{ Induction on a }

$$T) f26 C_, a.R.0 (C_, a.R.1 j (f2 k (C_, a.R.1 l m))) = f26 C_, a.R.0 (C_, a.R.1 j k)$$

{ Compute ; Compute }
 $C_ , a_R.0 = C_ , a_R.0$
 { Triv }
 Done.

H) $f26 \circ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m))) = f26 \circ (C_ , a_R.1 \ j \ k)$

T) $f26 \ (C_ , a_R.1 \ n \ o) \ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m))) = f26 \ (C_ , a_R.1 \ n \ o) \ (C_ , a_R.1 \ j \ k)$
 { Compute ; Compute }
 $(\forall) (C_ , a_R.1 \ n \ (f2 \ (f26 \ o \ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m)))) \ (f26 \ o \ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m))))))$
 $= C_ , a_R.1 \ n \ (f2 \ (f26 \ o \ (C_ , a_R.1 \ j \ k)) \ (f26 \ o \ (C_ , a_R.1 \ j \ k)))$
 { Fund: $(\forall) (f26 \ o \ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m)))$
 $= f26 \ o \ (C_ , a_R.1 \ j \ k) ; \}$ }
 $(\forall) (C_ , a_R.1 \ n \ (f2 \ (f26 \ o \ (C_ , a_R.1 \ j \ k)) \ (f26 \ o \ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m))))))$
 $= C_ , a_R.1 \ n \ (f2 \ (f26 \ o \ (C_ , a_R.1 \ j \ k)) \ (f26 \ o \ (C_ , a_R.1 \ j \ k)))$
 { Fund: $(\forall) (f26 \ o \ (C_ , a_R.1 \ j \ (f2 \ k \ (C_ , a_R.1 \ l \ m)))$
 $= f26 \ o \ (C_ , a_R.1 \ j \ k) ; \}$ }
 $(\forall) (C_ , a_R.1 \ n \ (f2 \ (f26 \ o \ (C_ , a_R.1 \ j \ k)) \ (f26 \ o \ (C_ , a_R.1 \ j \ k))))$
 $= C_ , a_R.1 \ n \ (f2 \ (f26 \ o \ (C_ , a_R.1 \ j \ k)) \ (f26 \ o \ (C_ , a_R.1 \ j \ k)))$
 { Triv }
 Done.

$f26 \ b \ (f26 \ e \ g) = f26 \ b \ (f26 \ g \ e)$

$f26 \ b \ (f26 \ e \ g) = f26 \ b \ (f26 \ g \ e)$

{ Induction on b }

T) $f26 \ C_ , a_R.0 \ (f26 \ e \ g) = f26 \ C_ , a_R.0 \ (f26 \ g \ e)$

{ Compute ; Compute }

$C_ , a_R.0 = C_ , a_R.0$

{ Triv }

Done.

H) $f26 \ k \ (f26 \ e \ g) = f26 \ k \ (f26 \ g \ e)$

T) $(\forall e::C_ , a_R, g::C_ , a_R) \ (f26 \ (C_ , a_R.1 \ j \ k) \ (f26 \ e \ g))$

$= f26 \ (C_ , a_R.1 \ j \ k) \ (f26 \ g \ e)$

{ Compute ; Compute }

$(\forall e::C_ , a_R, g::C_ , a_R) \ (C_ , a_R.1 \ j \ (f2 \ (f26 \ k \ (f26 \ e \ g)) \ (f26 \ k \ (f26 \ e \ g))))$
 $= C_ , a_R.1 \ j \ (f2 \ (f26 \ k \ (f26 \ g \ e)) \ (f26 \ k \ (f26 \ g \ e)))$

{ ; Reor: $(\forall e::C_ , a_R, g::C_ , a_R) \ (f26 \ k \ (f26 \ g \ e))$

$= f26 \ k \ (f26 \ e \ g) \}$ }

$(\forall e::C_ , a_R, g::C_ , a_R) \ (C_ , a_R.1 \ j \ (f2 \ (f26 \ k \ (f26 \ e \ g)) \ (f26 \ k \ (f26 \ e \ g))))$
 $= C_ , a_R.1 \ j \ (f2 \ (f26 \ k \ (f26 \ g \ e)) \ (f26 \ k \ (f26 \ g \ e)))$

{ ; Reor: $(\forall e::C_ , a_R, g::C_ , a_R) \ (f26 \ k \ (f26 \ g \ e))$

$= f26 \ k \ (f26 \ e \ g) \}$ }

$(\forall e::C_ , a_R, g::C_ , a_R) \ (C_ , a_R.1 \ j \ (f2 \ (f26 \ k \ (f26 \ e \ g)) \ (f26 \ k \ (f26 \ e \ g))))$
 $= C_ , a_R.1 \ j \ (f2 \ (f26 \ k \ (f26 \ g \ e)) \ (f26 \ k \ (f26 \ g \ e)))$


```
{ Triv }
Done.
```

B.2.25 $f27\ xs\ ys = ys ++ ys ++ ys \cdots ++ ys \quad ((length\ xs)\ times)$

```
f27::C_, a.R → C_, a.R → C_, a.R
```

```
f27 = λc → λe → case c of {C_, a.R.0 → C_, a.R.0; C_, a.R.1gh → f2 (f27 h e) e}
```

```
f27 C_, a.R.0 a = C_, a.R.0
```

```
f27 C_, a.R.0 a = C_, a.R.0
```

```
{ Compute ; }
```

```
C_, a.R.0 = C_, a.R.0
```

```
{ Triv }
```

```
Done.
```

```
f27 a C_, a.R.0 = C_, a.R.0
```

```
f27 a C_, a.R.0 = C_, a.R.0
```

```
{ Induction on a }
```

```
T) f27 C_, a.R.0 C_, a.R.0 = C_, a.R.0
```

```
{ Compute ; }
```

```
C_, a.R.0 = C_, a.R.0
```

```
{ Triv }
```

```
Done.
```

```
H) f27 j C_, a.R.0 = C_, a.R.0
```

```
T) f27 (C_, a.R.1 c j) C_, a.R.0 = C_, a.R.0
```

```
{ Compute ; }
```

```
f2 (f27 j C_, a.R.0) C_, a.R.0 = C_, a.R.0
```

```
{ Fund: f2 a C_, a.R.0 = a ; }
```

```
f27 j C_, a.R.0 = C_, a.R.0
```

```
{ Fund: f27 j C_, a.R.0 = C_, a.R.0 ; }
```

```
C_, a.R.0 = C_, a.R.0
```

```
{ Triv }
```

```
Done.
```

```
f27 (C_, a.R.1 a b) g = f2 (f27 b g) g
```

```
f27 (C_, a.R.1 a b) g = f2 (f27 b g) g
```

```
{ Compute ; }
```

```
f2 (f27 b g) g = f2 (f27 b g) g
```

```
{ Triv }
```

```
Done.
```

```

f27 (f2 a b) g = f2 (f27 b g) (f27 a g)
f27 (f2 a b) g = f2 (f27 b g) (f27 a g)
  { Induction on a }
T) f27 (f2 C_, a.R.0 b) g = f2 (f27 b g) (f27 C_, a.R.0 g)
  { Compute ; Compute }
f27 b g = f2 (f27 b g) C_, a.R.0
  { ; Fund: f2 a C_, a.R.0 = a }
f27 b g = f27 b g
  { Triv }
Done.

H) f27 (f2 j b) g = f2 (f27 b g) (f27 j g)
T) f27 (f2 (C_, a.R.1 c j) b) g = f2 (f27 b g) (f27 (C_, a.R.1 c j) g)
  { Compute ; Compute }
f27 (C_, a.R.1 c (f2 j b)) g = f2 (f27 b g) (f2 (f27 j g) g)
  { Compute ; }
f2 (f27 (f2 j b) g) g = f2 (f27 b g) (f2 (f27 j g) g)
  { Fund: f27 (f2 j b) g = f2 (f27 b g) (f27 j g) ; }
f2 (f2 (f27 b g) (f27 j g)) g = f2 (f27 b g) (f2 (f27 j g) g)
  { Reor: f2 (f2 b e) g = f2 b (f2 e g) ; }
f2 (f27 b g) (f2 (f27 j g) g) = f2 (f27 b g) (f2 (f27 j g) g)
  { Triv }
Done.

```

B.2.26 f28 = f27

```

f28::C_, a.R → C_, a.R → C_, a.R
f28 = λc → λe → case c of {C_, a.R.0 → C_, a.R.0; C_, a.R.1gh → f2 e (f28 h e)}

f28 C_, a.R.0 a = C_, a.R.0
f28 C_, a.R.0 a = C_, a.R.0
  { Compute ; }
C_, a.R.0 = C_, a.R.0
  { Triv }
Done.

f28 a C_, a.R.0 = C_, a.R.0
f28 a C_, a.R.0 = C_, a.R.0
  { Induction on a }
T) f28 C_, a.R.0 C_, a.R.0 = C_, a.R.0
  { Compute ; }
C_, a.R.0 = C_, a.R.0
  { Triv }
Done.

H) f28 j C_, a.R.0 = C_, a.R.0
T) f28 (C_, a.R.1 c j) C_, a.R.0 = C_, a.R.0

```

```

{ Compute ; }
f2 C_, a.R.0 (f28 j C_, a.R.0) = C_, a.R.0
{ Compute ; }
f28 j C_, a.R.0 = C_, a.R.0
{ Fund: f28 j C_, a.R.0 = C_, a.R.0 ; }
C_, a.R.0 = C_, a.R.0
{ Triv }
Done.

```

```

f28 (C_, a.R.1 a b) g = f2 g (f28 b g)
f28 (C_, a.R.1 a b) g = f2 g (f28 b g)
  { Compute ; }
f2 g (f28 b g) = f2 g (f28 b g)
  { Triv }
Done.

```

```

f28 (f2 a b) g = f2 (f28 b g) (f28 a g)
f28 (f2 a b) g = f2 (f28 b g) (f28 a g)
  { Induction on a }
T) f28 (f2 C_, a.R.0 b) g = f2 (f28 b g) (f28 C_, a.R.0 g)
  { Compute ; Compute }
f28 b g = f2 (f28 b g) C_, a.R.0
  { ; Fund: f2 a C_, a.R.0 = a }
f28 b g = f28 b g
  { Triv }
Done.

```

```

H) f28 (f2 j b) g = f2 (f28 b g) (f28 j g)
T) f28 (f2 (C_, a.R.1 c j) b) g = f2 (f28 b g) (f28 (C_, a.R.1 c j) g)
  { Compute ; Compute }
f28 (C_, a.R.1 c (f2 j b)) g = f2 (f28 b g) (f2 g (f28 j g))
  { Compute ; }
f2 g (f28 (f2 j b) g) = f2 (f28 b g) (f2 g (f28 j g))
  { Fund: f28 (f2 j b) g = f2 (f28 b g) (f28 j g) ; }
f2 g (f2 (f28 b g) (f28 j g)) = f2 (f28 b g) (f2 g (f28 j g))
  { Induction on b }

```

```

T) f2 g (f2 (f28 C_, a.R.0 g) (f28 j g)) = f2 (f28 C_, a.R.0 g) (f2 g (f28 j g))
  { Compute ; Compute }
f2 g (f2 C_, a.R.0 (f28 j g)) = f2 C_, a.R.0 (f2 g (f28 j g))
  { Compute ; Compute }
f2 g (f28 j g) = f2 g (f28 j g)
  { Triv }
Done.

```

```

H) f2 g (f2 (f28 k g) (f28 j g)) = f2 (f28 k g) (f2 g (f28 j g))

```

T) $f2\ g\ (f2\ (f28\ (C_,\ a.R.1\ c\ k)\ g)\ (f28\ j\ g)) = f2\ (f28\ (C_,\ a.R.1\ c\ k)\ g)\ (f2\ g\ (f28\ j\ g))$
 { Compute ; Compute }
 $f2\ g\ (f2\ (f2\ g\ (f28\ k\ g))\ (f28\ j\ g)) = f2\ (f2\ g\ (f28\ k\ g))\ (f2\ g\ (f28\ j\ g))$
 { Reor: $f2\ (f2\ b\ e)\ g = f2\ b\ (f2\ e\ g)$; Reor: $f2\ (f2\ b\ e)\ g = f2\ b\ (f2\ e\ g)$ }
 $f2\ g\ (f2\ g\ (f2\ (f28\ k\ g)\ (f28\ j\ g))) = f2\ g\ (f2\ (f28\ k\ g)\ (f2\ g\ (f28\ j\ g)))$
 { ; Reor: $f2\ (f28\ k\ g)\ (f2\ g\ (f28\ j\ g)) = f2\ g\ (f2\ (f28\ k\ g)\ (f28\ j\ g))$ }
 $f2\ g\ (f2\ g\ (f2\ (f28\ k\ g)\ (f28\ j\ g))) = f2\ g\ (f2\ g\ (f2\ (f28\ k\ g)\ (f28\ j\ g)))$
 { Triv }
 Done.

$f28\ b\ (f28\ e\ g) = f28\ (f28\ e\ b)\ g$
 $f28\ b\ (f28\ e\ g) = f28\ (f28\ e\ b)\ g$
 { Induction on b }
T) $f28\ C_,\ a.R.0\ (f28\ e\ g) = f28\ (f28\ e\ C_,\ a.R.0)\ g$
 { Compute ; }
 $C_,\ a.R.0 = f28\ (f28\ e\ C_,\ a.R.0)\ g$
 { ; Fund: $f28\ a\ C_,\ a.R.0 = C_,\ a.R.0$ }
 $C_,\ a.R.0 = f28\ C_,\ a.R.0\ g$
 { ; Compute }
 $C_,\ a.R.0 = C_,\ a.R.0$
 { Triv }
 Done.

H) $f28\ j\ (f28\ e\ g) = f28\ (f28\ e\ j)\ g$
T) $f28\ (C_,\ a.R.1\ c\ j)\ (f28\ e\ g) = f28\ (f28\ e\ (C_,\ a.R.1\ c\ j))\ g$
 { Compute ; }
 $f2\ (f28\ e\ g)\ (f28\ j\ (f28\ e\ g)) = f28\ (f28\ e\ (C_,\ a.R.1\ c\ j))\ g$
 { Reor: $f28\ j\ (f28\ e\ g) = f28\ (f28\ e\ j)\ g$; }
 $f2\ (f28\ e\ g)\ (f28\ (f28\ e\ j)\ g) = f28\ (f28\ e\ (C_,\ a.R.1\ c\ j))\ g$
 { Induction on e }

T) $f2\ (f28\ C_,\ a.R.0\ g)\ (f28\ (f28\ C_,\ a.R.0\ j)\ g) = f28\ (f28\ C_,\ a.R.0\ (C_,\ a.R.1\ c\ j))\ g$
 { Compute ; Compute }
 $f2\ C_,\ a.R.0\ (f28\ (f28\ C_,\ a.R.0\ j)\ g) = f28\ C_,\ a.R.0\ g$
 { Compute ; Compute }
 $f28\ (f28\ C_,\ a.R.0\ j)\ g = C_,\ a.R.0$
 { Compute ; }
 $f28\ C_,\ a.R.0\ g = C_,\ a.R.0$
 { Compute ; }
 $C_,\ a.R.0 = C_,\ a.R.0$
 { Triv }
 Done.

H) $f2\ (f28\ l\ g)\ (f28\ (f28\ l\ j)\ g) = f28\ (f28\ l\ (C_,\ a.R.1\ c\ j))\ g$

$T) (\forall g::C_ , a_R) (f2 (f28 (C_ , a_R.1 k l) g) (f28 (f28 (C_ , a_R.1 k l) j) g))$
 $= f28 (f28 (C_ , a_R.1 k l) (C_ , a_R.1 c j)) g)$
 { Compute ; Compute }
 $(\forall g::C_ , a_R) (f2 (f2 g (f28 l g) (f28 (f28 (C_ , a_R.1 k l) j) g))$
 $= f28 (f2 (C_ , a_R.1 c j) (f28 l (C_ , a_R.1 c j))) g)$
 { Compute ; Compute }
 $(\forall g::C_ , a_R) (f2 (f2 g (f28 l g) (f28 (f2 j (f28 l j)) g))$
 $= f28 (C_ , a_R.1 c (f2 j (f28 l (C_ , a_R.1 c j)))) g)$
 { ; Compute }
 $(\forall g::C_ , a_R) (f2 (f2 g (f28 l g) (f28 (f2 j (f28 l j)) g))$
 $= f2 g (f28 (f2 j (f28 l (C_ , a_R.1 c j))) g)$
 { Fund: $(\forall a::C_ , a_R, b::C_ , a_R, g::C_ , a_R) (f28 (f2 a b) g$
 $= f2 (f28 b g) (f28 a g)) ; Fund: (\forall a::C_ , a_R, b::C_ , a_R, g::C_ , a_R) (f28 (f2 a b) g$
 $= f2 (f28 b g) (f28 a g)) \}$
 $(\forall g::C_ , a_R) (f2 (f2 g (f28 l g) (f2 (f28 (f28 l j) g) (f28 j g))$
 $= f2 g (f2 (f28 (f28 l (C_ , a_R.1 c j)) g) (f28 j g)))$
 { ; Fund: $(\forall g::C_ , a_R) (f28 (f28 l (C_ , a_R.1 c j)) g$
 $= f2 (f28 l g) (f28 (f28 l j) g)) \}$
 $(\forall g::C_ , a_R) (f2 (f2 g (f28 l g) (f2 (f28 (f28 l j) g) (f28 j g))$
 $= f2 g (f2 (f2 (f28 l g) (f28 (f28 l j) g)) (f28 j g)))$
 { Reor: $(\forall b::C_ , a_R, e::C_ , a_R, g::C_ , a_R) (f2 (f2 b e) g$
 $= f2 b (f2 e g)) ; Reor: (\forall b::C_ , a_R, e::C_ , a_R, g::C_ , a_R) (f2 (f2 b e) g$
 $= f2 b (f2 e g)) \}$
 $(\forall g::C_ , a_R) (f2 g (f2 (f28 l g) (f2 (f28 (f28 l j) g) (f28 j g)))$
 $= f2 g (f2 (f28 l g) (f2 (f28 (f28 l j) g) (f28 j g))))$
 { Triv }
 Done.

B.2.27 $f29 [x_0, x_1, \dots, x_n] ys = [ys ++ [x_0], ys ++ [x_1], \dots, ys ++ [x_n]]$

$f29::C_ , a_R \rightarrow C_ , a_R \rightarrow C_ , a_R$
 $f29 = \lambda e \rightarrow \lambda g \rightarrow \text{case } e \text{ of } \{ C_ , a_R.0 \rightarrow C_ , a_R.0 ; C_ , a_R.1hi \rightarrow f2 g (C_ , a_R.1 h (f29 i g)) \}$

$f29 C_ , a_R.0 a = C_ , a_R.0$

$f29 C_ , a_R.0 a = C_ , a_R.0$

{ Compute ; }

$C_ , a_R.0 = C_ , a_R.0$

{ Triv }

Done.

$f29 a C_ , a_R.0 = a$

$f29 a C_ , a_R.0 = a$

{ Induction on a }

$T) f29 C_ , a_R.0 C_ , a_R.0 = C_ , a_R.0$

{ Compute ; }

$C_{-, a.R.0} = C_{-, a.R.0}$

{ Triv }

Done.

H) $f_{29} k C_{-, a.R.0} = k$

T) $f_{29} (C_{-, a.R.1} j k) C_{-, a.R.0} = C_{-, a.R.1} j k$

{ Compute ; }

$f_2 C_{-, a.R.0} (C_{-, a.R.1} j (f_{29} k C_{-, a.R.0})) = C_{-, a.R.1} j k$

{ Compute ; }

$C_{-, a.R.1} j (f_{29} k C_{-, a.R.0}) = C_{-, a.R.1} j k$

{ Fund: $f_{29} k C_{-, a.R.0} = k$; }

$C_{-, a.R.1} j k = C_{-, a.R.1} j k$

{ Triv }

Done.

$f_{29} (f_2 a b) g = f_2 (f_{29} a g) (f_{29} b g)$

$f_{29} (f_2 a b) g = f_2 (f_{29} a g) (f_{29} b g)$

{ Induction on a }

T) $f_{29} (f_2 C_{-, a.R.0} b) g = f_2 (f_{29} C_{-, a.R.0} g) (f_{29} b g)$

{ Compute ; Compute }

$f_{29} b g = f_2 C_{-, a.R.0} (f_{29} b g)$

{ ; Compute }

$f_{29} b g = f_{29} b g$

{ Triv }

Done.

H) $f_{29} (f_2 k b) g = f_2 (f_{29} k g) (f_{29} b g)$

T) $f_{29} (f_2 (C_{-, a.R.1} j k) b) g = f_2 (f_{29} (C_{-, a.R.1} j k) g) (f_{29} b g)$

{ Compute ; Compute }

$f_{29} (C_{-, a.R.1} j (f_2 k b)) g = f_2 (f_2 g (C_{-, a.R.1} j (f_{29} k g))) (f_{29} b g)$

{ Compute ; }

$f_2 g (C_{-, a.R.1} j (f_{29} (f_2 k b) g)) = f_2 (f_2 g (C_{-, a.R.1} j (f_{29} k g))) (f_{29} b g)$

{ Fund: $f_{29} (f_2 k b) g = f_2 (f_{29} k g) (f_{29} b g)$; }

$f_2 g (C_{-, a.R.1} j (f_2 (f_{29} k g) (f_{29} b g))) = f_2 (f_2 g (C_{-, a.R.1} j (f_{29} k g))) (f_{29} b g)$

{ ; Reor: $f_2 (f_2 b e) g = f_2 b (f_2 e g)$ }

$f_2 g (C_{-, a.R.1} j (f_2 (f_{29} k g) (f_{29} b g))) = f_2 g (f_2 (C_{-, a.R.1} j (f_{29} k g)) (f_{29} b g))$

{ ; Compute }

$f_2 g (C_{-, a.R.1} j (f_2 (f_{29} k g) (f_{29} b g))) = f_2 g (C_{-, a.R.1} j (f_2 (f_{29} k g) (f_{29} b g)))$

{ Triv }

Done.

B.2.28 $f_{30} = f_{25}$

$f_{30} :: C_{-, a.R} \rightarrow C_{-, a.R} \rightarrow C_{-, a.R}$

$f_{30} = \lambda e \rightarrow \lambda g \rightarrow \text{case } e \text{ of } \{ C_{-, a.R.0} \rightarrow C_{-, a.R.0}; C_{-, a.R.1} hi \rightarrow f_2 (f_{30} i g) (C_{-, a.R.1} h (f_{30} i g)) \}$

f30 C_{-,a.R.0} a = C_{-,a.R.0}

f30 C_{-,a.R.0} a = C_{-,a.R.0}

{ Compute ; }

C_{-,a.R.0} = C_{-,a.R.0}

{ Triv }

Done.

f30 a (f2 b g) = f30 a b

f30 a (f2 b g) = f30 a b

{ Induction on b }

T) f30 a (f2 C_{-,a.R.0} g) = f30 a C_{-,a.R.0}

{ Compute ; }

f30 a g = f30 a C_{-,a.R.0}

{ Induction on g }

T) f30 a C_{-,a.R.0} = f30 a C_{-,a.R.0}

{ Triv }

Done.

H) f30 a k = f30 a C_{-,a.R.0}

T) f30 a (C_{-,a.R.1} j k) = f30 a C_{-,a.R.0}

{ ; Fund: f30 a C_{-,a.R.0} = f30 a k }

f30 a (C_{-,a.R.1} j k) = f30 a k

{ Induction on a }

T) f30 C_{-,a.R.0} (C_{-,a.R.1} j k) = f30 C_{-,a.R.0} k

{ Compute ; Compute }

C_{-,a.R.0} = C_{-,a.R.0}

{ Triv }

Done.

H) f30 m (C_{-,a.R.1} j k) = f30 m k

T) f30 (C_{-,a.R.1} l m) (C_{-,a.R.1} j k) = f30 (C_{-,a.R.1} l m) k

{ Compute ; Compute }

f2 (f30 m (C_{-,a.R.1} j k)) (C_{-,a.R.1} l (f30 m (C_{-,a.R.1} j k))) = f2 (f30 m k) (C_{-,a.R.1} l (f30 m k))

{ Fund: f30 m (C_{-,a.R.1} j k) = f30 m k ; }

f2 (f30 m k) (C_{-,a.R.1} l (f30 m (C_{-,a.R.1} j k))) = f2 (f30 m k) (C_{-,a.R.1} l (f30 m k))

{ Fund: f30 m (C_{-,a.R.1} j k) = f30 m k ; }

f2 (f30 m k) (C_{-,a.R.1} l (f30 m k)) = f2 (f30 m k) (C_{-,a.R.1} l (f30 m k))

{ Triv }

Done.

$H) f30\ a\ (f2\ k\ g) = f30\ a\ k$
 $T) f30\ a\ (f2\ (C_{-}, a.R.1\ j\ k)\ g) = f30\ a\ (C_{-}, a.R.1\ j\ k)$
 { Compute ; }
 $f30\ a\ (C_{-}, a.R.1\ j\ (f2\ k\ g)) = f30\ a\ (C_{-}, a.R.1\ j\ k)$
 { Induction on g }

$T) f30\ a\ (C_{-}, a.R.1\ j\ (f2\ k\ C_{-}, a.R.0)) = f30\ a\ (C_{-}, a.R.1\ j\ k)$
 { Fund: $f2\ a\ C_{-}, a.R.0 = a$; }
 $f30\ a\ (C_{-}, a.R.1\ j\ k) = f30\ a\ (C_{-}, a.R.1\ j\ k)$
 { Triv }
 Done.

$H) f30\ a\ (C_{-}, a.R.1\ j\ (f2\ k\ m)) = f30\ a\ (C_{-}, a.R.1\ j\ k)$

$T) f30\ a\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))) = f30\ a\ (C_{-}, a.R.1\ j\ k)$
 { Induction on a }

$T) f30\ C_{-}, a.R.0\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))) = f30\ C_{-}, a.R.0\ (C_{-}, a.R.1\ j\ k)$
 { Compute ; Compute }
 $C_{-}, a.R.0 = C_{-}, a.R.0$
 { Triv }
 Done.

$H) f30\ o\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))) = f30\ o\ (C_{-}, a.R.1\ j\ k)$

$T) f30\ (C_{-}, a.R.1\ n\ o)\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))) = f30\ (C_{-}, a.R.1\ n\ o)\ (C_{-}, a.R.1\ j\ k)$
 { Compute ; Compute }
 $f2\ (f30\ o\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))))\ (C_{-}, a.R.1\ n\ (f30\ o\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))))))$
 $= f2\ (f30\ o\ (C_{-}, a.R.1\ j\ k))\ (C_{-}, a.R.1\ n\ (f30\ o\ (C_{-}, a.R.1\ j\ k)))$
 { Fund: $f30\ o\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))) = f30\ o\ (C_{-}, a.R.1\ j\ k)$; }
 $f2\ (f30\ o\ (C_{-}, a.R.1\ j\ k))\ (C_{-}, a.R.1\ n\ (f30\ o\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))))))$
 $= f2\ (f30\ o\ (C_{-}, a.R.1\ j\ k))\ (C_{-}, a.R.1\ n\ (f30\ o\ (C_{-}, a.R.1\ j\ k)))$
 { Fund: $f30\ o\ (C_{-}, a.R.1\ j\ (f2\ k\ (C_{-}, a.R.1\ l\ m))) = f30\ o\ (C_{-}, a.R.1\ j\ k)$; }
 $(\forall)\ (f2\ (f30\ o\ (C_{-}, a.R.1\ j\ k))\ (C_{-}, a.R.1\ n\ (f30\ o\ (C_{-}, a.R.1\ j\ k))))$
 $\quad = f2\ (f30\ o\ (C_{-}, a.R.1\ j\ k))\ (C_{-}, a.R.1\ n\ (f30\ o\ (C_{-}, a.R.1\ j\ k)))$
 { Triv }
 Done.

$f30\ b\ (f30\ e\ g) = f30\ b\ (f30\ g\ e)$

$f30\ b\ (f30\ e\ g) = f30\ b\ (f30\ g\ e)$
 { Induction on b }

$T) f30\ C_{-}, a.R.0\ (f30\ e\ g) = f30\ C_{-}, a.R.0\ (f30\ g\ e)$
 { Compute ; Compute }
 $C_{-}, a.R.0 = C_{-}, a.R.0$

{ Triv }

Done.

H) $f_{30} k (f_{30} e g) = f_{30} k (f_{30} g e)$

T) $f_{30} (C_, a.R.1 j k) (f_{30} e g) = f_{30} (C_, a.R.1 j k) (f_{30} g e)$

{ Compute ; Compute }

$(\forall e::C_, a.R, g::C_, a.R) (f_2 (f_{30} k (f_{30} e g)) (C_, a.R.1 j (f_{30} k (f_{30} e g))))$
 $= f_2 (f_{30} k (f_{30} g e)) (C_, a.R.1 j (f_{30} k (f_{30} g e)))$

{ ; Reor: $(\forall e::C_, a.R, g::C_, a.R) (f_{30} k (f_{30} g e) = f_{30} k (f_{30} e g))$ }

$(\forall e::C_, a.R, g::C_, a.R) (f_2 (f_{30} k (f_{30} e g)) (C_, a.R.1 j (f_{30} k (f_{30} e g))))$
 $= f_2 (f_{30} k (f_{30} e g)) (C_, a.R.1 j (f_{30} k (f_{30} g e)))$

{ ; Reor: $(\forall e::C_, a.R, g::C_, a.R) (f_{30} k (f_{30} g e) = f_{30} k (f_{30} e g))$ }

$(\forall e::C_, a.R, g::C_, a.R) (f_2 (f_{30} k (f_{30} e g)) (C_, a.R.1 j (f_{30} k (f_{30} e g))))$
 $= f_2 (f_{30} k (f_{30} e g)) (C_, a.R.1 j (f_{30} k (f_{30} e g)))$

{ Triv }

Done.