Mechanizing Bisimulation for a Linear Functional Language: Enforcing Linearity Without Linearity

L Denney
School of Computer Science
McGill University
Montreal, Canada
lola.denney@mail.mcgill.ca

There are far more mechanizations of properties for functional languages than for linear functional languages. Mechanizations about properties of a linear functional language also struggle with inelegance when required to explicitly manage context splitting. We will word towards the property of program equivalence by adapting a mechanization of bisimulation for a functional language in Beluga from "A Case-Study in Programming Coinductive Proofs: Howe's Method"[8] to prove the same for a linear functional language. We are able to reuse many parts this mechanization by shifting our focus on the way we track the restrictions on variables from the context to the syntax of a term. The idea keep the relation defined on the intuitionist terms and then enforce linearity with a judgment that enforces a term is linear. We will apply this technique to the mechanization of bisimulation from the aforementioned paper; this is a mechanization that requires both inductive and coinductive reasoning.

1 Introduction

Many mechanizations exists for proving useful properties for functional languages, but there are significantly fewer for substructural languages. Of those few for linear languages, many suffer from inelegance when managing the context explicitly. We would benefit from a method that allows us to take advantage of an existing mechanization of a functional language and adapt it to prove the property for a substructural one, while also leaving us with a mechanization more elegant than one directly on the target substructural language.

As compared to previous mechanizations using this particular strategy, this paper focuses on the property of contextual equivalence, who's proof is a bit more complicated that straight forward induction as it relies on both inductive and coinductive reasoning. We will focus on mechanizing this proof for a simply typed linear λ-calculus, with the expectation that a similar approach could be applied for other combinations of substructural languages and properties.

Linear logic was introduced by Girard in the mid nineteen eighties and is connected to our work by the Curry Howard correspondence which associates type theory with logic. A linear logic is one in which assumptions may not be duplicated or dropped. Such a restriction is where the notion of resource control is found in a linear functional language, granting it applications in situations which require resource boundedness.

Linear functional languages provide a useful framework for managing production and consumption problems. For example, states in quantum computers have a "no-cloning" property that states that it is impossible to create perfect copies of an unknown quantum state. This can be modeled with linear logic since each state once created must be consumed. It also has applications in compilers where it can be used to model unique resources such as memory usage or channel usage in concurrent programs. These applications motivate why linear languages are worth studying, but to leverage them effectively we need properties such as program equivalence to use them for anything meaningful.

Many advancements have been made in the methods of proving program equivalence, but even now those about linear functional languages continue to suffer from inelegance during mechanization. This is unfortunate because the property of program equivalence is very useful for providing a concrete characterization of equivalence between programs, which is used to verify that optimizations made by a compiler leave the behaviour of the program unaltered.

The commonly accepted notion of program equivalence for functional languages is Morris style contextual equivalence. Previous work on contextual equivalence for linear functional languages includes [1] and [3], but unfortunately neither provide a mechanization. Since both proofs require managing the context explicitly, one would expect a mechanization to be inelegant and unnecessarily difficult. Context management is a challenge because when a multiplicative term is encountered during type-checking in a linear language, the context needs to be split between it's subterms such that they each use their partition of the context linearly. It is tricky to figure out which along which lines the context should be split such that each subterm is linear, if linearity is even possible at all.

This project is concerned with mechanizing a proof of program equivalence for a linear functional language. Since a central goal of ours is to show the value of leveraging previous mechanizations the methods in this paper depart from those of Bierman and Crole. We will try an approach that can liberate us from explicitly managing the context in a mechanization, as well as allow us to leverage a previous mechanization of the same property for a functional language by adapting the existing code to work for our desired language. It should give a result similar to those of Bierman and Crole, but with a (hopefully) elegant concrete mechanization.

Getting all the way to contextual equivalence is overly ambitious for this single-semester research project, so the scope of this report is limited to the essential first step of proving that bisimilarity is a congruence for our linear language, from which a coincidence between bisimilarity and contextual equivalence can be shown. Starting with a proof of bisimilarity for a functional language, we must adapt it for linear languages by enforcing this bisimilarity relation is only between linear terms.

The language we wish to study is presented in 2.1. We will provide background on contextual equivalence and Howe's method in 2.3 and 2.2 respectively, including the main theorems we must prove. Then we will introduce the linearity predicate in 2.4. In Section 3 we will discuss details of the mechanization: discussing how definitions were adapted, expanding on the linearity predicate, and looking at two examples to see how the mechanization changed.

2 Background

2.1 Linear Language

We begin by specifying a linear functional language by giving its types, terms, and static semantics. We will consider a linear version of a subset of a simply typed λ-calculus called PCFL following [9] (for later compatibility with the mechanization in Beluga). We have limited ourself to function space and unit and variables since the objective of this paper is not concerned with the language itself beyond it being linear. The types and terms are

Types	$\tau ::= \top \mid \tau \multimap \tau$
Terms	$M, N ::= x \mid \mathtt{lam}\, x.M \mid M\ N \mid \langle\rangle$
Values	$V ::= \mathtt{lam}\, x.M \mid \langle\rangle$

where x is from some countable set of variables, along with the typing rules below.

$$\frac{\overline{\Gamma(x) = \tau}}{\Gamma \vdash x : \tau}\ \text{var} \qquad \frac{}{\Gamma \vdash \langle\rangle : \top}\ \text{unit}$$

$$\frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \mathtt{lam}\, x.M : \tau \multimap \tau'}\ \text{lam} \qquad \frac{\Gamma_1 \vdash M : \tau \multimap \tau' \qquad \Gamma_2 \vdash N : \tau}{\Gamma_1, \Gamma_2 \vdash M\ N : \tau'}\ \text{app}$$

The big step lazy operational semantics denoted by $M \Downarrow V$ are standard and omitted here.

As we saw in the introduction, a linear logic is one where all assumptions are used exactly once. It can also be understood as intuitionist logic with no weakening or contraction; weakening meaning extra assumptions don't matter (i.e. we can drop assumptions), and contraction meaning we can use an assumption as often as we want (i.e. we can duplicate assumptions).

Due to the loss of weakening and contraction, when typechecking multiplicative terms one must split the context between the subterms to ensure linearity is preserved. This is because when a multiplicative term is reached during the typechecking traversal, in order to continue type and linearity -checking it's subterms, the context must be partitioned between those checks to ensure we arent using any variables in the context twice. The frustration arises when a multiplicative term is arrived at and one must decide along which lines the context should be split so that the typechecking of each subterm has exactly the variables it requires in the partition of the context allotted to give linearity. The only multiplicative term present in the language we are considering is function application, but the hope is we can avoid the same trouble with the other multiplicative constructors like lazy pairs if we were to extend the language to something closer to a full linear version of PCFL.

2.2 Contextual Equivalence

Let us begin the discussion on program equivalence without worrying about linearity.

We employ Morris style contextual equivalence as our characterization of program equivalence since it is commonly accepted as a natural notion of equivalence for functional languages. It equates programs such that one program can be exchanged for another in any larger program without affecting overall observable behaviour. Intuitively, this will give us an equivalence between programs that can be used in place of each other in all possible situations.

If we attempt to mechanize contextual equivalence directly, we will find that the statement “in any larger programs” makes the proof tricky since it requires quantifying over all possible contexts. Thankfully, the useful property of contextual equivalence can be shown by instead proving applicative bisimilarity, which is often more manageable. (Applicative) bisimilarity is a coinductively[1] defined relation over terms which can be shown to be a congruence, and then a coincidence can be established between it and contextual equivalence.

We say that two programs are (applicatively) bisimilar if whenever the first evaluates to a value, so does the second one, and all its sub-programs are also bisimilar and vise versa. Applicative simply asserts that the observation evaluates function applications.

The general method for operational program equivalence follows 4 steps:

1. Define a type theory, language, and operational semantics to make the definitions of "program" and "evaluation" tangible.
2. Use contextual equivalence to equate programs by their observable behaviour when interchanged in all larger programs.
3. Define a notion of bisimilarity over programs where making an observation on (evaluating) each program produces the same value, and the immediate subprograms of the observation are themselves bisimilar.
4. Prove that bisimilarity and contextual equivalence coincide.

An important property of bisimilarity is that it is a congruence. This means it is both compatible (respects the construction of terms) and transitive. This property is important for proving that two programs are applicatively bisimilar since it allows for equational reasoning. Therefore our first step (and the step that consumed all the time allotted to this project) is to establish that bisimilarity on linear terms is a congruence.

2.3 Howe’s Method

Before discussing a method of proving that bisimilarity is a congruence, we must first establish a few things. Rather than bisimilarity, we will focus on the notion of (applicative) similarity from which bisimilarity can be obtained by symmetry. This is possible thanks to the determination of evaluation. Applicative simulation is a family of typed relations on programs that are related by the result of CBN evaluation, and the union of two applicative simulations is a simulation. We define applicative similarity to be the largest of those. We also will refer to open similarity, which is the similarity definition extended for open terms by grounding substitutions. Now we may proceed to the discussion of Howe’s method.

The solution Howe offered to proving that bisimilarity is a congruence in 1996 was to introduce a candidate Howe relation[5] that contains open similarity. The idea is to prove it is an almost substitutive precongruence (where almost refers to semi-transitive), and then show it coincides with similarity. Pre-congruence is defined as a compatible transitive relation, informally a relation respecting the way terms are constructed, and notably it requires substitutivity. The

[1]A good resource on coinduction is [6]

reason Howe's method is used is because a direct attempt to prove that open similarity is a pre-congruence breaks down when dealing with function application and proving that open similarity is substitutive. The Howe relation lets us show that similarity is a precongruence without having to prove the substitutivity of it directly.

Howe's method requires a few main theorems to be proved:

- the Howe relation is closed under substitution
- the Howe relation is a pre-congruence
- coincidence of the Howe relation and open similarity

We will build on the mechanization of Howe's method for an functional language in Beluga from [8].

This mechanization uses a simply types λ-calculus named PCFL following Pitts (1997) [9]. We will only work on the terms in PCFL corresponding to those in our linear language defined back in 2.1 and leave the rest for another time. The language is lazily evaluated and has the usual typing rules.

$$\begin{array}{lr}
\text{Types} & \tau ::= \top \mid \tau \to \tau \\
\text{Terms} & M, N ::= x \mid \texttt{lam}\, x.M \mid M\ N \mid \langle\rangle \\
\text{Values} & V ::= \texttt{lam}\, x.M \mid \langle\rangle
\end{array}$$

2.4 Linearity Predicate

The key to reasoning about the terms of a linear functional language in a non-linear functional language is to take an different perspective when enforcing the relevant sub-structural properties in the sub-structural logic. Instead of focusing on the context, we will the individual assumptions in the syntax of the proof terms. Building on a clever observation made by Crary[2]: If a variable occurs once, then the typing assumption will be only used once. In other words, occurrences of variables in the syntax correspond to how often typing assumptions are consumed. This is to say, Crary noticed that to check linearity of a term one only has to look at the syntax directly, allowing the mechanizer to evade the context splitting entirely! This idea of a predicate to check linearity of a term by its syntax has been used in mechanizing other linear systems such as type preservation for processes[10], but our application of the linearity predicate is slightly different because a mechanization of contextual equivalence requires not only straight forward syntactic inductive reasoning, but also coinductive proofs.

The local linearity checks are given by the judgment `linear` $x\ M$ which read "x is used linearly in M". Its on paper definition is provided below.

$$\frac{}{\texttt{linear}\ x\ x}\ \text{lin\_var} \qquad \frac{\texttt{linear}\ x\ (M\ x)}{\texttt{linear}\ x\ ((M\ x)\ N)}\ \text{lin\_app1}$$

$$\frac{\Pi x : \text{term}.\texttt{linear}\ y\ (M\ y\ x)}{\texttt{linear}\ y\ (\text{lam}\ \lambda x.M\ x\ y)}\ \text{lin\_lam} \qquad \frac{\texttt{linear}\ x\ (N\ x)}{\texttt{linear}\ x\ (M\ (N\ x))}\ \text{lin\_app2}$$

The rule `app_1` enforces that x does not appear in N and similarly `app_2` enforces that x does not appear in M.

Using this linearity predicate, we will the mechanization from [8] in Beluga to mechanize a proof of bisimilarity for our linear language. The first step of this adaptation is to enforce linearity in the bisimilarity relation. We will do this by making some changes to the Howe relation definition.

Red text represents any changes made to the initial implementation. These will be discussed further in the following section on mechanization3.1.

$$\frac{\Gamma \vdash \langle\rangle \preccurlyeq^{\circ}_{\top} N}{\Gamma \vdash \langle\rangle \preccurlyeq^{\mathcal{H}}_{\top} N}\text{howe\_unit} \qquad \frac{\Gamma, x:\tau \vdash x \preccurlyeq^{\circ}_{\tau} N}{\Gamma, x:\tau \vdash x \preccurlyeq^{\mathcal{H}}_{\tau} N}\text{howe\_var}$$

$$\frac{\Gamma \vdash M_1 \preccurlyeq^{\mathcal{H}}_{\tau\to\tau'} M_1' \qquad \Gamma \vdash M_2 \preccurlyeq^{\mathcal{H}}_{\tau} M_2' \qquad \Gamma \vdash M_1'\ M_2' \preccurlyeq^{\circ}_{\tau'} N}{\Gamma \vdash M_1\ M_2 \preccurlyeq^{\mathcal{H}}_{\tau'} N}\text{howe\_app}$$

$$\frac{\Gamma, x:\tau \vdash M \preccurlyeq^{\mathcal{H}}_{\tau'} M' \qquad \Gamma \vdash \text{lam}\ x.M' \preccurlyeq^{\circ}_{\tau\to\tau'} N \qquad \Gamma \vdash \texttt{linear}\ x\ M}{\Gamma \vdash \text{lam}\ x.M \preccurlyeq^{\mathcal{H}}_{\tau\to\tau'} N}\text{howe\_lam}$$

3 Realization in Mechanization

In this section, we will explore how using a linearity predicate let us leverage a previous mechanization of bisimilarity.

The mechanization we chose to build off of uses the Beluga proof environment. It was chosen by the original authors for a few reasons, one being it's support for representing open terms and simultaneous substitutions. In this project we can adapt the built in simultaneous substitution using an inductive definition to get a substitution that preserves linearity. This allows us to encode the notoriously painful proof of the substitutivity of the Howe relation with a less difficulty. Another reason is that Beluga supports coinductive reasoning, which is an essential in a mechanization of bisimilarity.

3.1 Relation Definitions

Returning to the above adapted definition of the Howe relation, we notice that very little was changed. The only modification was to add a local predicate as a premise in the lambda case. This is adequate for enforcing that the term on the left of the relation is linear because of Crary's observation that terms are enough to track the use of restricted assumptions. Any time a variable binding construct occurs, there is a corresponding check using the local linearity predicate that the new variable(s) introduced appear linearly in the scope in which they are defined. One might worry that the linearity predicate enforces that only the left argument in a Howe relation must be linear, and it does, but there is no need to worry. We must keep in mind the current goal we are trying to reach is not similarity for linear terms, but bisimilarity. We informally argue that for this reason enforcing only the term on the left of the relation to be linear (as opposed to both the left and right) does not change the fact that once we use symmetry to arrive at bisimilarity, bisimilarity will be a relationship strictly between linear terms.

In a similar vein, if we adapt the definition of coincidence between the Howe relation and open similarity as below to enforce that the left term is globally linear, we do not need to enforce linearity in the similarity or open similarity definitions.

$$\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau} N \text{ iff } \Gamma \vdash M \preccurlyeq^{\circ}_{\tau} N \text{ and } \texttt{lin}\ \Gamma\ M$$

This way we avoid touching the coinductively defined similarity and open similarity with our predicate.

3.2 Global Linearity Predicate

We saw in the definition of the Howe relation that the global linearity of a term can be enforced by a term syntax traversal using only to the local linearity predicate at variable binders. In fact, many of the lemmata we must prove require a predicate to enforce the global linearity of a term. In lemmata where we need to construct a Howe relation (such as `howe_refl` 3.3), an additional premise stating that the term that is to become the left operand of the Howe relation must be globally linear. It allows us extract the proof of local linearity that is demanded in the lambda case of the Howe relation definition `howe_lam` whenever a variable binding constructs is encountered during the traversal of the term.

The global linearity predicate definition began as an LF level judgment that traverses the term and enforces linearity of the new binder at each instance of variable binding. Later on during mechanization, the utility of an inductive version of the global linearity predicate became apparent. Since the schema of the context does not know that all variables in the context are linear (i.e $x \in \Gamma$, and `lin` x), a solution that limits changes to the Beluga file (thus simplify development) is to lift the definition of global linearity from the LF level to an inductive type. This gives us the lemma that all variables in the context are linear `all_vars_lin` for free from the lifted predicate's constructor for variables `L_var`. The Beluga encoding of the local and global linearity predicates used are provided below.

```
LF linear:
 (term A → term B) → type =
 | lin_app1 : linear M
   → linear (\x. app (M x) N)
 | lin_app2 : linear N
   → linear (\x. app M (N x))
 | lin_var  : linear (\x.x)
 | lin_lam  :
   ({x:term _ } linear (\y. M y x))
   → linear (\y. lam \x. M y x)
;
```

Local Linearity Predicate

```
inductive Lin :
 {Γ : ctx} [Γ ⊢ term T[]] → ctype =
 | L_var : Lin [Γ] [Γ ⊢ #p]
 | L_unit : Lin [Γ] [Γ ⊢ unit]
 | L_app : Lin [Γ] [Γ ⊢ M]
   → Lin [Γ] [Γ ⊢ N]
   → Lin [Γ] [Γ ⊢ app M N]
 | L_lam : [Γ ⊢ linear \y.M]
   → Lin [Γ,y:term _] [Γ,y:term _ ⊢ M]
   → Lin [Γ] [Γ ⊢ lam \y.M]
;
```

Global Linearity Predicate

3.3 Changes to Lemmata

We will look closer at two on paper proofs of lemmas from the mechanization and discuss how they work and what changes were made to adapt them for linearity. First we will examine the reflectivity of the Howe relation, an example that demonstrates the need of the global linearity predicate. Then we will look at the more complicated example of the Howe relation being closed under linear renaming, which illustrate many interesting things about how the mechanization was adapted.

Example 1: the Howe Relation is Reflexive

Theorem. If $\Gamma \vdash M : \tau$ and $\texttt{lin}\ \Gamma\ M$ then $\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau} M$

The proof of reflexivity for the Howe relation is the first step towards proving it is a pre-congruence. It proceeds by induction on the typing derivation of M. We adapt this proof for our linear language by adding the assumption that M is globally linear in the context Γ. The cases for unit and variables are unchanged and omitted from the on-paper proof found in the Appendix5.

In the application case, the new assumption gives us that both M and N are globally linear so we can appeal to the induction hypothesis, essentially passing along the information. The interesting case is the typing for lambda terms, where this assumption is

$$\texttt{lin}\ \Gamma\ \text{lam}\ x.M$$

By the definition of `lin` on this assumption, we get the global linear judgment that allows us to call the inductive hypothesis (similar to the application case) as well as the local linearity that the Howe relation definition for lambda terms requires.

$$\Gamma \vdash \texttt{linear}\ x\ M$$

This is a recurrent use of the global linearity predicate throughout our adapted mechanization.

Example 2: the Howe Relation is Closed Under Linear Renaming

First we must establish a definition of linear renaming. Variable renaming is a special case of substitution where all of the expressions being substituted in are variable names, and a linear one is a renaming that preserves linearity. Ours is simply the identity renaming which has been sufficient for our purposes. We provide our definition in Beluga.

```
inductive Lin_rename: {Γ:ctx} (Ψ:ctx) {$S : $[Ψ |-# Γ]} ctype =
| LNil_ren : Lin_rename [] $[Ψ |-# ^]
| LCons_ren :  Lin_rename [Γ] $[Ψ |-# $S]
    -> Lin_rename [Γ, x:term T[]] $[Ψ, x:term T[] |-# $S[..], x];
```

Now we proceed to state the theorem about linear renaming.

Theorem. If $\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau} N$ and R is a linear renaming from Γ to Ψ then $\Psi \vdash R(M) \preccurlyeq^{\mathcal{H}}_{\tau} R(N)$

The proof that the Howe relation is closed under renaming is essential for proving that it is closed under linear substitution. It is a good example of changes made to the mechanization because it relies on additional lemmata for proving linearity is preserved for local and global linearity respectively,

Lemma. (`linear_ren`) *Linear renaming preserves local linearity*
If $\Gamma, x : \tau \vdash M$ and $\Gamma \vdash$ `linear` $x\ M$ and R is a linear renaming from Γ to Ψ and if we extend the renaming with the identity renaming for x $R' := R, [x/x]$ then $\Psi \vdash$ `linear` $x\ R'(M)$.

Lemma. (`lin_ren`) *Linear renaming preserves global linearity*
If `lin` $\Gamma\ M$ and R is a linear renaming from Γ to Ψ then `lin` $\Psi\ R(M)$.

as well as the property of semi-transitivity for the Howe relation with open similarity.

Definition. *Semi-transitivity for the Howe relation with open similarity*
If $A \preccurlyeq^{\mathcal{H}}_{\tau} B$ and $B \preccurlyeq^{\circ}_{\tau} C$ then $A \preccurlyeq^{\mathcal{H}}_{\tau} C$.

Semi-transitivity states that the composition of the Howe relation with open similarity is contained within the Howe relation and didn't requires any changes in the current version of our mechanization.

The on-paper proof that the Howe relation is closed under linear renaming can be found in the Appendix 5.1. The proof proceeds by induction on the Howe relation. In every case of the Howe relation, the substitutivity of open similarity will be used on the premise that uses open similarity. The unit case is the simplest example of this, remaining unchanged from before the adaptation for linearity since it only needs the use the definition of the Howe relation for unit on the result of appealing to the substitutivity of open similarity. In the case for variables, reflexivity of the Howe relation is used, but it now requires an additional global linearity argument which is obtained by using the lemma `lin_ren` with the assumption that R is a linear renaming. The variable case demonstrates that changes to the mechanization of eariler lematta where they are given additional premises requre more changes later on when they are use since they need to be provided the new arguments. The other notable thing occurring in the variable case is the use of semi-transitivity.

The case for applications is straightforward. It uses the substitutivity of open similarity and appeals to the induction hypothesis for the other two premises. Something important to notice here is how the assumption that R is a linear renaming does not need to be changed like the global linearity predicate had to be in the proof of reflexivity. It just gets passed on as is.

Finally, the lambda case is similar to that of application, but also needs the new lemma `linear_ren` to provide the local linearity argument that `howe_lam` requires.

4 Related Work

We are aware of two papers that provided a proof of contextual equivalence for a linear language. Unfortunately neither of them provide mechanizations, and both managed the context explicitly throughout. We review them primarily from the angle of how their approach to linearity can be made more elegant by instead working with a non-linear functional language and making use of the linear predicate.

The first is "Observations on a linear PCF" [1] whose language of choice is the simple linear functional language called LinPCF, which contains both additive and multiplicative pairs. In terms of evaluation, function application and multiplicative pairs are eager, and additive pairs are lazy. The proof of contextual equivalence manages the linear context explicitly, and mechanization is never mentioned. We would expect such a mechanization would be inelegant as it provides no alternatives to managing the context splits explicitly throughout.

The second is "Completeness of Bisimilarity for Contextual Equivalence in Linear Theories"[3], concerning a language more similar to ours. It has function space, multiplicative pairs, and a divergent term with CBN evaluation. Like Bierman, the context is managed explicitly and although Crole does not provide a mechanization he claims his definition of the linear context, despite requiring additional lemmata that Bierman's does not, is more amenable to mechanization. Regardless, there is no mention of mechanization concerning Howe's method on linear terms, and as with Bierman we expect such a mechanization to be inelegant. One can imagine that structural inductive proofs on the Howe relation would struggle in the multiplicative cases like function application where context splitting is needed. In contrast with Bierman, Crole uses a linearized version of pre-congruence, where the strengthening and weakening requirements are dropped. He also modified substitution properties accordingly.

5 Conclusion

Since many substructural languages are useful to programmers and there already exist mechanizations of useful properties for functional languages, it is desirable to have a proof technique that leverages this previous work and is applicable to a variety of situations. Furthermore, managing a linear context explicitly can make mechanization tricky and inelegant, so a technique that allows one to avoid it altogether is desirable. Although the application of a linearity predicate for enforcing linearity in a non-linear language seen in this paper is not new, we have shown it can be employed in a setting where coinductive reasoning is necessary.

Ongoing
This project was developed as part of a single-semester research course, and consequently due to these imposed time constraints some parts of the mechanization of bisimilarity remains incomplete. This includes two lemmata that are used in the proof of substitutivity of the Howe relation that did not exist in the previous non-linear mechanization. They are proofs that renaming preserves local and global linearity named `linear_ren` and `lin_ren` respectively. Additionally, `subst_wkn` remains incomplete because it does not have enough information to be proven as it currently stands, and we haven't yet figured out exactly what it requires. Once that is complete, the proof of substitutivity of the Howe relation may be completed, and following it so will `down_closed`. At this point we make any additional changes to supply the new arguments demanded by the newly mechanized lemmata and we have reached the end, proving the coincidence of simulation and the Howe relation. On top of this, we would have liked to find a proof that enforcing linearity on the left operand of the Howe relation implies our new bisimilarity can only be between linear terms, but we expect this requires a predicate to be spliced in to the open similarity definition as well.

Future work
The attentive reader might notice that we have not showed bisimilarity coincides with contextual equivalence. Given more time, We would have liked to complete the proof all the way to contextual equivalence, perhaps considering Crole's linear context definition to see if his way of getting coincidence of bisimilarity and contextual equivalence is more amenable to mechanization than Bierman is as he claims, but that work is for another time. It would also be nice to extend the language to a full linear version of PCFL, one with both additive and multiplicative pairs, lazy lists, and a divergent term.

References

[1] G. M. Bierman (2000): *Program equivalence in a linear functional language. Journal of Functional Programming* 10(2), pp. 167–190, doi:10.1017/S0956796899003639.

[2] Karl Crary (2010): *Higher-order representation of substructural logics. SIGPLAN Not.* 45(9), p. 131–142, doi:10.1145/1932681.1863565. Available at `https://doi.org/10.1145/1932681.1863565`.

[3] Roy L. Crole (2001): *Completeness of bisimilarity for contextual equivalence in linear theories. Logic Journal of the IGPL* 9(1), pp. 27–51, doi:10.1093/jigpal/9.1.27. arXiv:https://academic.oup.com/jigpal/article-pdf/9/1/27/1866364/090027.pdf.

[4] D.J. Howe (1989): *Equality in lazy computation systems.* In: *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, IEEE Comput. Soc. Press, Pacific Grove, CA, USA, pp. 198–203, doi:10.1109/LICS.1989.39174.

[5] Douglas J. Howe (1996): *Proving Congruence of Bisimulation in Functional Programming Languages. Information and Computation* 124(2), pp. 103–112, doi:https://doi.org/10.1006/inco.1996.0008. Available at `https://www.sciencedirect.com/science/article/pii/S0890540196900085`.

[6] Dexter Kozen & Alexandra Silva (2017): *Practical coinduction. Mathematical Structures in Computer Science* 27(7), p. 1132–1152, doi:10.1017/S0960129515000493.

[7] Giorgio Marabelli & Alberto Momigliano (2019): *Formalizing Program Equivalences in Dependent Type Theory.* Available at `https://ceur-ws.org/Vol-2504/paper23.pdf`.

[8] Alberto Momigliano, Brigitte Pientka & David Thibodeau (2019): *A case study in programming coinductive proofs: Howe's method. Mathematical Structures in Computer Science* 29(8), p. 1309–1343, doi:10.1017/S0960129518000415.

[9] Andrew Pitts (1997): *Operationally-Based Theories of Program Equivalence*, p. 241–298. Publications of the Newton Institute, Cambridge University Press.

[10] Chuta Sano, Ryan Kavanagh & Brigitte Pientka (2023): *Mechanizing Session-Types using a Structural View: Enforcing Linearity without Linearity. Proc. ACM Program. Lang.* 7(OOPSLA2), doi:10.1145/3622810. Available at `https://doi.org/10.1145/3622810`.

Appendix

The Howe relation is reflexive

Theorem. If $\Gamma \vdash M : \tau$ and $\texttt{lin}\ \Gamma\ M$ then $\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau} M$

Proof. by structural induction on typing

Case $$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash M : \tau \to \tau'} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash N : \tau}}{\Gamma \vdash M\ N : \tau'}\ \text{app}$$

$a:$	$\texttt{lin}\ \Gamma\ M$	
$b:$	$\texttt{lin}\ \Gamma\ N$	by def of $\texttt{lin}$ on ass
$c:$	$\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau' \to \tau} M$	by IH on $\mathcal{D}_1$ with a
$d:$	$\Gamma \vdash N \preccurlyeq^{\mathcal{H}}_{\tau'} N$	by IH on $\mathcal{D}_2$ with b
$e:$	$\Gamma \vdash M\ N \preccurlyeq^{\circ}_{\tau} M\ N$	by reflexivity of open similarity
	$\Gamma \vdash M\ N \preccurlyeq^{\mathcal{H}}_{\tau'} M\ N$	by def of Howe for application (howe_app) on c, d, e

Case $$\dfrac{\overset{\mathcal{D}}{\Gamma, x : \tau \vdash M : \tau'}}{\Gamma \vdash \texttt{lam}\, x.M : \tau \to \tau'}\ \text{lam}$$

$a:$	$\Gamma \vdash \texttt{linear}\ x\ M$	
$b:$	$\texttt{lin}\ (\Gamma, x : \tau)\ M$	by def of $\texttt{lin}$ on ass
$c:$	$\Gamma, x : \tau \vdash M \preccurlyeq^{\mathcal{H}}_{\tau' \to \tau} M$	by IH on $\mathcal{D}_1$ with b
$d:$	$\Gamma \vdash \texttt{lam}\ x.M \preccurlyeq^{\circ}_{\tau_1 \to \tau_2} \texttt{lam}\ x.M$	by reflexivity of open similarity
	$\Gamma \vdash \texttt{lam}\ x.M \preccurlyeq^{\mathcal{H}}_{\tau_1 \to \tau_2} \texttt{lam}\ x.M$	by def of Howe for lambdas (howe_lam) on c, d, a

□

5.1 The Howe Relation is Closed Under Linear Renaming

Theorem. If $\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau} N$ and R is a linear renaming from Γ to Ψ
then $\Psi \vdash R(M) \preccurlyeq^{\mathcal{H}}_{\tau} R(N)$

Proof. by induction on the derivations of $\Gamma \vdash M \preccurlyeq^{\mathcal{H}}_{\tau} N$

Case $$\dfrac{\Gamma \vdash \langle\rangle \preccurlyeq^{\circ}_{\top} N}{\Gamma \vdash \langle\rangle \preccurlyeq^{\mathcal{H}}_{\top} N}\ \text{howe\_unit}$$

$\Psi \vdash R(\langle\rangle) \preccurlyeq^{\circ}_{\top} R(N)$	by substitutivity of open similarity
$\Psi \vdash R(\langle\rangle) \preccurlyeq^{\mathcal{H}}_{\top} R(N)$	by def of howe relation for unit

Case $\dfrac{\Gamma', x : \tau \vdash x \preccurlyeq^{\circ}_{\tau} N}{\Gamma', x : \tau \vdash x \preccurlyeq^{\mathcal{H}}_{\tau} N}$ howe_var where $\Gamma := \Gamma', x : \tau$

$a :$ $\Psi \vdash R(x) \preccurlyeq^{\circ}_{\tau} R(N)$ by substitutivity of open similarity

$b :$ `lin` Ψ $R(x)$ by `lin_ren` on $\Gamma, \Psi, (\Gamma', x : \tau \vdash x)$, `L_var` (variable case for global linearity), and R is linear renaming (by assumption)

$c :$ $\Psi \vdash R(x) \preccurlyeq^{\mathcal{H}}_{\tau} R(x)$ by reflexivity of Howe relation on $\Psi \vdash R(x)$ and b

$a :$ $\Psi \vdash R(x) \preccurlyeq^{\mathcal{H}}_{\tau} R(N)$ by semi-transitivity of Howe with osim on c and a

Case $\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash M_1 \preccurlyeq^{\mathcal{H}}_{\tau\to\tau'} M_1'} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash M_2 \preccurlyeq^{\mathcal{H}}_{\tau} M_2'} \qquad \Gamma \vdash M_1'\ M_2' \preccurlyeq^{\circ}_{\tau'} N}{\Gamma \vdash M_1\ M_2 \preccurlyeq^{\mathcal{H}}_{\tau'} N}$ howe_app

$a :$ $\Psi \vdash R(M_1\ M_2) \preccurlyeq^{\circ}_{\tau'} R(N)$ by substitutivity of open similarity

$b :$ $\Psi \vdash R(M_1) \preccurlyeq^{\mathcal{H}}_{\tau\to\tau'} R(M_1')$ by IH on $\mathcal{D}_1$ with R is linear renaming (by assumption)

$c :$ $\Psi \vdash R(M_2) \preccurlyeq^{\mathcal{H}}_{\tau} R(M_2')$ by IH on $\mathcal{D}_2$ with R is linear renaming (by assumption)

$\Psi \vdash R(M_1\ M_2) \preccurlyeq^{\mathcal{H}}_{\tau'} R(N)$ by def of Howe for application (howe_app) on b, c, a

Case $\dfrac{\overset{\mathcal{D}}{\Gamma, x : \tau \vdash M \preccurlyeq^{\mathcal{H}}_{\tau'} M'} \qquad \Gamma \vdash \text{lam } x.M' \preccurlyeq^{\circ}_{\tau\to\tau'} N \qquad \Gamma \vdash \texttt{linear}\ x\ M}{\Gamma \vdash \text{lam } x.M \preccurlyeq^{\mathcal{H}}_{\tau\to\tau'} N}$ howe_lam

$a :$ $\Psi \vdash R(\text{lam } x.M) \preccurlyeq^{\circ}_{\tau\to\tau'} R(N)$ by substitutivity of open similarity

$\Gamma' := \Gamma, x : \tau$

$\Psi' := \Psi, x : \tau$

$R' := R, [x/x]$

$b :$ R' is a linear renaming from Γ' to Ψ' by def of linear renaming with assumption that R is a linear renaming

$c :$ $\Psi' \vdash R'(M) \preccurlyeq^{\mathcal{H}}_{\tau'} R'(M')$ by IH on $\mathcal{D}$ with Γ', Ψ', R', b

$d :$ $\Psi \vdash \texttt{linear}\ R(x)\ R(M)$ by linear_ren on $\Gamma, \Psi, R, \Gamma' \vdash x, \Gamma \vdash \texttt{linear}\ x\ M$, and R is linear renaming (by assumption)

$\Psi \vdash R(\text{lam } x.M) \preccurlyeq^{\mathcal{H}}_{\tau\to\tau'} R(N)$ by def of Howe for application (howe_app) on c, a, d

□