國立陽明交通大學
資訊科學與工程研究所
碩士論文

Institute of Computer Science and Engineering
National Yang Ming Chiao Tung University
Master Thesis

可逆抽象機終止的形式化證明

A Formal Termination Proof for Reversible Abstract Machines

研究生 : 蔡文龍 (Tsai, Wen-Lung)
指導教授 : 陳穎平 (Chen, Ying-Ping)

中華民國 一一三年六月
June 2024

可逆抽象機終止的形式化證明

A Formal Termination Proof for Reversible Abstract Machines

研 究 生：蔡文龍　　Student: Tsai, Wen-Lung
指導教授：陳穎平　　Advisor: Dr. Chen, Ying-Ping

國立陽明交通大學
資訊科學與工程研究所
碩士論文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Science
National Yang Ming Chiao Tung University
in partial Fulfilment of the Requirements
for the Degree of
Master
in
Science

June 2024

Taiwan, Republic of China

中華民國 一一三年六月

TODO

可逆抽象機終止的形式化證明

學生：蔡文龍　　　　　　　　　　　　　　　　指導教授：陳穎平 博士

國立陽明交通大學 資訊科學與工程研究所

摘　　要

在陳昭宏 (Chao-Hong Chen) 的論文《A Computational Interpretation of Compact Closed Categories: Reversible Programming with Negative and Fractional Types》中，他探討了緊湊封閉類別透過「反轉時間」與「反轉空間」的解釋，並將其形式化應用於可逆 SAT 求解器的開發上。該論文中，對可逆抽象機的終止證明尤為關鍵，涉及兩項重要定理的形式化證明。儘管這些證明顯然正確，在該論文中尚僅提供了部分的形式化證明。

我們的研究深入探討了論文中的這一陳述。首先，明確重申了陳述的具體內容：對於可逆抽象機，給予一初始狀態，並且已知該初始狀態可抵達的狀態是有限的，則該初始狀態經由可逆抽象機的推移，將會在有限的步數內終止。隨後，我們為此定理附加了一項條件並完成了相關的形式化證明，即所有狀態的個數有限的情況下，對該陳述的形式化證明。最後，我們移除了有限狀態的假設，並以類似方法完成了無限狀態下的形式化證明，從而補充並完善了論文中該部分的證明。

關鍵詞: 可逆抽象機、終止證明、Agda

A Formal Termination Proof for Reversible Abstract Machines

Student: Tsai, Wen-Lung Advisor: Dr. Chen, Ying-Ping

Institute of Computer Science and Engineering
National Yang Ming Chiao Tung University

Abstract

In Chao-Hong Chen's paper, "A Computational Interpretation of Compact Closed Categories: Reversible Programming with Negative and Fractional Types," he explores an interpretation of compact closed categories through "reversing time and space," formalizing its application in the development of a reversible SAT solver. The termination proof of a reversible abstract machine, an essential component of the paper, involved the formalization of two crucial theorems. Although these proofs are clearly correct, only partial formal proofs are provided in the paper.

Our research delves deeply into this statement from Chen's work. Initially, we rearticulate the specifics of the statement: for a reversible abstract machine, given an initial state with a known finite number of reachable states, this initial state will terminate within a finite number of transitions through the reversible abstract machine. Subsequently, we added a condition and completed a formal proof under the assumption that all states are finite. Finally, we removed the assumption of finite states and employed similar methods to complete a formal proof under infinite states, thus complementing and enhancing the proofs in the original paper.

Keyword: Abstract Machines, Termination Proofs, Agda

Contents

ListofFigures

Chapter 1

Introduction

1.1 Motivation

In CHAO-HONG CHEN's study [1], three significant contributions are meticulously elaborated. Initially, the research formalizes the concepts of "reversing time" and "reversing space," showcased through the compact closed categories (N,+, 0) and (N, *, 1) respectively; secondly, it further implements a reversible SAT solver, proving the feasibility of high-level control abstractions described in the first contribution within reversible programming languages; the third contribution focuses on proving the termination conditions for a large class of reversible abstract machines.

Specifically, the paper presents two combinatorial proofs in Chapters 5 and 7, regarding the property that a class of reversible abstract machines will inevitably terminate for initial states with a finite number of reachable states. This proof, although intuitively obvious —that is, on a path of unique states, the number of reachable states decreases as the state progresses, until no further states can be reached, or a termination state is achieved. However, the original paper only partially formalized this proof, failing to fully substantiate this concept.

Our contribution starts from constructing a similar class of reversible abstract machines. Begin with a given initial state, demonstrating that when all possible states of a reversible abstract machine are finite, it will ultimately reach a termination state. Subsequently, we modified the constraints to align with the objective of "finite reachable states" mentioned in the original paper, thereby accomplishing a more generalized proof.

This paper is accompanied by approximately 400 lines of Agda code, serving to complement the proofs of the two theorems mentioned in the original paper as being incomplete.

1.2 Research Objectives

1.3 Road Map

TODO Chapter 2 introduces thesis.cls document class. Chapter 3 introduces section ordering. Chapter 4 and 5 explain how to load citation, figures and tables (not completed).

Chapter 2

Background Knowledge

In this chapter, we introduce the target language used in the research: Agda. As a dependently typed functional programming language, we provide a foundational explanation of its usage.

2.1 Agda

Agda is a dependently typed functional programming language [3]. Writing in the Agda language is akin to organizing a mathematical proof. It begins with self-evident datatypes, proposes assumptions to be proven, and provides detailed evidence for their validity thereafter. The following is an example of a datatype concerning natural numbers[5].

```
data ℕ : Set where
  zero  : ℕ
  suc   : ℕ → ℕ
```

This definition captures the first two axioms corresponding to Peano's axioms, and the other axioms within Peano's axioms can also be easily proven. The function suc can be considered as a mapping operation. We can describe this operation in natural language as follows: ”Given a natural number n, suc(n) is also a natural number.” Such a function from ℕ to ℕ will be utilized in Chapter TODO, where it will constitute a critical part of the proof process.

And the following demonstrates two examples of the proof process:

```
axioms-3 : ∄[ n ] (suc n ≡ zero)
axioms-3 ()
```

Here, we take the third Peano axiom as an example, which is to prove that there does not exist a natural number such that suc(n) is zero. In Agda, a proof of non-existence typically begins by assuming the existence of such an entity, and then deriving a contradiction (denoted as ⊥). In this case, the proof process is merely an empty parenthesis. When Agda attempts to find n among the two possible kinds of natural numbers (zero and suc(n)), it immediately encounters an obvious contradiction. Therefore, it concludes that no further cases need to be proven.

```
axioms-5 : ∀ {f : ℕ → Set}
  → f 0
  → (∀ n → f n → f (suc n))
  → (∀ n → f n)
```

This statement corresponds to the fifth Peano axiom. Agda uses the right arrow to sequentially assign the necessary conditions, with the conclusion appearing to the right of the last right arrow. The fifth Peano axiom describes that for any function f, if:

1. f(0) holds true,

2. f(n) being true implies that f(suc(n)) is also true

then for all natural numbers n, f(n) holds true.

```
axioms-5 f0 fn-sucn zero = f0
axioms-5 f0 fn-sucn (suc n)
  with axioms-5 f0 fn-sucn n
... | fn = fn-sucn n fn
```

This illustrates the proof of the fifth Peano axiom. This straightforward proof highlights a few commonly used proof techniques in Agda:

1. Pattern Matching on Variables: The example splits the natural number n into two cases: zero and suc n, for separate discussion. This technique allows for detailed examination of different scenarios directly related to the structure of natural numbers.

2. Well-founded recursion in Agda [6]: Theorems involving natural numbers often leverage induction. In Agda, we typically prove a case for n to infer the case for suc n. Agda ensures that there is a corresponding proof for the base case (usually zero), which acts as the termination condition for a comprehensive proof.

3. Simplifying Variables Using 'with': In the example, the proof for f(n) is obtained using axiom-5, and it is named fn. This approach helps to avoid verbose variable expressions, streamlining the proof.

2.2 Reversible Abstract Machine

The Reversible Abstract Machine, abbreviated as RevMachine, is defined as follows.

```
field
  State : Set ℓ
  _↦_ : Rel State ℓ
  deterministic : ∀ {st st₁ st₂} → st ↦ st₁ → st ↦ st₂ → st₁ ≡ st₂
  deterministic_rev : ∀ {st st₁ st₂} → st₁ ↦ st → st₂ ↦ st → st₁ ≡ st₂
```

"State" is the set of all states.

"$\mapsto$" is used to record state transitions, for example, $st_0 \mapsto st_1$ indicates that st_0 transitions to st_1.

"Deterministic" refers to forward determinism, meaning that identical states will transition to the same next state.

"deterministic$_{\text{rev}}$", on the other hand, is the opposite of "deterministic", indicating that for each state, all possible transitions to it are the same.

Based on the definitions above, given a RevMachine and one of its states, we can determine an invariant trace composed of states.

Chapter 3

Informal Proof

3.1 Formulate Statement

First of all, let' s define the following terms: "Initial state" means a state that no other state can reach in the machine. "Termination of the machine" means reaching a state with no subsequent state to transition to. This final state is also called the "stuck state." When we said "the machine will terminate from an initial state," it means that starting from the initial state, through a finite number of transitions, we will reach a stuck state.

We have a reversible abstract machine, and designate one of the state as initial state. It's known that there are finite number of reachable states from the initial state. Given the forward determinism of the reversible machines, we can assume that the machine will terminate from the initial state.

Here is our complete statement: "Given a reversible abstract machine, it will inevitably terminate from any initial state with a finite number of reachable states." We call this the "Broad RevTerminate Principle."

In the statement, the total number of states in the machine may be infinite. For the informal logic proof, we can easily ignore those unreachable states. However, it is relatively difficult for formal proof. Therefore, we make a narrower statement as follows:

"Given a reversible abstract machine with a finite number of total states, it will inevitably terminate from any initial state." We call this the "Narrow RevTerminate Principle".

In the statement, we could say "we have walked through all the states, so it should terminate." in both informal and formal proofs easily. We will provide a detailed proof in the next section.

3.2 Informal Logic Proof

3.2.1 No Repeat

The “No Repeat Principle” is formally proved by Chao-Hong Chen[2]. Here is the definition of the principle: Given a reversible abstract machine and an initial state, for two different natural numbers m and n, if the initial state reach st_m and st_n by walking through m and n steps respectively, then st_m should not beequal to st_n.

3.2.2 Narrow RevTerminate Principle

Restating the statement of Narrow RevTerminate Principle: “Given a reversible abstract machine with a finite number of total states, it will inevitably terminate from any initial state.”

Here is the graph of the expected machine. It seems monotonous: the initial state will traverse all the other states without repetition and then terminate.

$$\begin{tikzcd}
\boxed{\text{Initial State}} \arrow[r] & \boxed{\text{State A}} \arrow[r] & \boxed{\text{State B}} \arrow[r] & \boxed{\cdots} \arrow[r] & \boxed{\text{Stuck State}}
\end{tikzcd}$$

This is the other case for the machine with finite states. In the graph, the initial state also traverses a finite number of states and terminates. However, not all the states are traversed.

$$\begin{tikzcd}
\boxed{\text{Initial State}} \arrow[r] & \boxed{\text{State A}} \arrow[r] & \boxed{\text{State B}} \arrow[r] & \boxed{\cdots} \arrow[r] & \boxed{\text{Stuck State}} \\
\boxed{\cdots} \arrow[r] & \boxed{\text{State X}} \arrow[r] & \boxed{\text{State Y}} \arrow[r] & \boxed{\text{State Z}} \arrow[r] & \boxed{\cdots}
\end{tikzcd}$$

Assume the number of all states is $\mathbb{N}$. That is, we have st_0, st_1, ..., and stN-1. Assume st_0 is the initial state. We can check whether st_0 has a next state. Once the current state doesn’t have next state, the machine will terminate. Therefore, we only need to consider the states starting from st_0 always have its next state.

When the trace progresses $\mathbb{N}$ times, it should look like the graph below:

$$\begin{tikzcd}
\boxed{\text{Initial State(Step 0)}} \arrow[r] & \boxed{\text{State A(Step 1)}} \arrow[r] & \boxed{\text{State B(Step 2)}} \arrow[r] & \boxed{\cdots} \arrow[dlll] \\
\boxed{\text{State X(Step N-1)}} \arrow[r] & \boxed{\text{Stuck State(Step N)}} & &
\end{tikzcd}$$

Here, we don't specifically mention that the first $\mathbb{N}$ states traversed should be entirely non-repetitive, even though it's evident that a repetition would lead to a contradiction. Instead, we focus on the whole trace with a lenth of $\mathbb{N}$ + 1. The reversible machine has only $\mathbb{N}$ states, but there are $\mathbb{N}$ + 1 state we have traversed. According to piegonhole principle, we can find two identical states in the trace.

$$\begin{tikzcd}
\boxed{\text{Initial State(Step 0)}} \arrow[r] & \boxed{\text{State A(Step 1)}} \arrow[r] & \boxed{\text{State B(Step 2)}} \arrow[r] & \boxed{\cdots} \arrow[dlll] & \\
\boxed{\text{State C(Step m)}} \arrow[r] & \boxed{\cdots} \arrow[r] & \boxed{\text{State C(Step n)}} \arrow[r] & \boxed{\cdots} \arrow[r] & \boxed{\text{Stuck State(Step N)}}
\end{tikzcd}$$

With No-Repeat Principle, we know it's impossible for an initial state to walk through two different steps but reach same state. At this point, we have successfully identified a contradiction. The contradiction shows that the length of trace is always less or equal than N, and thus the machine will terminate.

3.2.3 Broad RevTerminate Principle

Restating the statement of Broad RevTerminate Principle: "Given a reversible abstract machine, it will inevitably terminate from any initial state with a finite number of reachable states."

Similar to Narrow RevTerminate Principle, we assume st_0 is the initial state, and focus on the fact that there are always subsequent states after each state we traverse. The Broad RevTerminate Principle asserts that there are a finite number of states st_0 can reach. Assume the number of reachable states from st_0 is $\mathbb{N}$. As before, we walk $\mathbb{N}$ steps, making the length of the trace $\mathbb{N}$ + 1.

$$\begin{tikzcd}
\boxed{\text{Initial State(Step 0)}} \arrow[r] & \boxed{\text{State A(Step 1)}} \arrow[r] & \boxed{\text{State B(Step 2)}} \arrow[r] & \boxed{\cdots} \arrow[dlll] \\
\boxed{\text{State X(Step N-1)}} \arrow[r] & \boxed{\text{Stuck State(Step N)}} & &
\end{tikzcd}$$

There are $\mathbb{N} + 1$ states we have traversed. However, there are only N states st_0 can reach. According to piegonhole principle, we can find two identical states in the trace.

$$\begin{tikzcd}
\boxed{\text{Initial State(Step 0)}} \arrow[r] & \boxed{\text{State A(Step 1)}} \arrow[r] & \boxed{\text{State B(Step 2)}} \arrow[r] & \boxed{\cdots} \arrow[dlll] & \\
\boxed{\text{State C(Step m)}} \arrow[r] & \boxed{\cdots} \arrow[r] & \boxed{\text{State C(Step n)}} \arrow[r] & \boxed{\cdots} \arrow[r] & \boxed{\text{Stuck State(Step N)}}
\end{tikzcd}$$

With No-Repeat principle, we know it's impossible for an initial state to traverse two different steps and reach the same state. At this point, we have successfully identified a contradiction. The contradiction shows the length of trace is always less or equal than N, leading to termination.

Chapter 4

Formal Proof

4.1 Formulate Statement

4.1.1 Redefinition of Narrow RevTerminate Principle

Compared to informal logic proofs, our formal logic proof requires stricter definitions. Here is the Narrow RevTerminate Principle: “Given a reversible abstract machine with a finite number of total states, it will inevitably terminate from any initial state.”

First of all, we define the initial state. Given a state, if it is not the next state of any other state, we call it an initial state.

```
is-initial : State → Set
is-initial st = ∄[ st' ] (st' ↦ st)
```

Secondly, we define the terminate of a reversible machine. Starting from the initial state st_0, we use $st_0 \mapsto^* st_n$ to indicate that st_0 can reach st_n by traversing a finite number of steps.

```
data ↦* : State → State → Set (L.suc ℓ) where
  ◾ : {st : State} → st ↦* st
  :: : {st₁ st₂ st₃ : State} → st₁ ↦ st₂ → st₂ ↦* st₃ → st₁ ↦* st₃
```

If a state st_0 have no next state, we call it a stuck state.

```
is-stuck : State → Set
is-stuck st = ∄[ st' ] (st ↦ st')
```

The termination of a reversible machine means that given an initial state, it will reach a stuck state.

```
∃[ stₙ ] (st₀ ↦* stₙ × is-stuck stₙ)∃[ stₙ ] (st₀ ↦* stₙ × is-stuck stₙ)∃[ stₙ ] (st₀ ↦* stₙ ×
is-stuck stₙ)
```

Finally, we define "a reversible abstract machine with a finite number of total states." Here is the definition of Fin in agda[4]. A Fin ℕ set have exactly ℕ elements.

```
data Fin : ℕ → Set where
  zero : {n : ℕ} → Fin (suc n)
  suc  : {n : ℕ} (i : Fin n) → Fin (suc n)
```

We construct a bijection between the set of states and Fin ℕ. It is as if all states are indexed by an element in the Fin ℕ set.

```
State  Fin NState  Fin NState  Fin N
```

Combining the definition above, we have the exact definition of Narrow RevTerminate Principle:

```
Finite-State-Termination : ∀ {N st₀}
  → State  Fin N
  → (has-next : ∀ (st : State) → Dec (∃[ st' ] (st ↦ st')))
  → is-initial st₀
  → ∃[ stₙ ] (st₀ ↦* stₙ × is-stuck stₙ)
```

4.1.2 Redefinition of Broad RevTerminate Principle

Here is the Broad RevTerminate Principle: "Given a reversible abstract machine, it will inevitably terminate from any initial state with a finite number of reachable states."

Most of the definitions are the same as those in the Narrow RevTerminate Principle:

For a state to be initial, no other state should have it as their next state.

```
is-initial : State → Set
is-initial st = ∄[ st' ] (st' ↦ st)
```

$st_0 \mapsto^* st_n$ indicates that st_0 can reach st_n by traversing a finite number of steps.

```
data ↦* : State → State → Set (L.suc ℓ) where
  ◾ : {st : State} → st ↦* st
  :: : {st₁ st₂ st₃ : State} → st₁ ↦ st₂ → st₂ ↦* st₃ → st₁ ↦* st₃
```

If a state have no next state, we call it a stuck state.

```
is-stuck : State → Set
is-stuck st = ∄[ st' ] (st ↦ st')
```

Before defining “a finite number of reachable states”, we need to define the “set of reachable states.” st_0 ↦[m] st_m means st_0 reaches st_m exactly after m steps. For all the m and corresponding st_m that satisfy st_0 ↦[m] st_m, this is called “set of reachable states.”

```
∃[ m ] ∃[ stₘ ] (st₀ ↦[ m ] stₘ) ∃[ m ] ∃[ stₘ ] (st₀ ↦[ m ] stₘ) ∃[ m ] ∃[ stₘ ] (st₀ ↦[ m ] stₘ)
```

In Agda, each element of reachable states set is represented as a tuple.

$$\begin{tikzcd}
\boxed{0, st_0, st_0 \mapsto^0 st_0} & \boxed{1, st_1, st_0 \mapsto^1 st_1} & \boxed{2, st_2, st_0 \mapsto^2 st_2} \\
\boxed{\text{State 0}} \arrow[u, leftrightarrow] \arrow[r] & \boxed{\text{State 1}} \arrow[u, leftrightarrow] \arrow[r] & \boxed{\text{State 2}} \arrow[u, leftrightarrow]
\end{tikzcd}$$

Then, we construct a bijection between the set of reachable states and Fin ℕ.

```
St-Fin : ∃[ m ] ∃[ stₘ ] (st₀ ↦[ m ] stₘ)  Fin NSt-Fin : ∃[ m ] ∃[ stₘ ] (st₀ ↦[ m ] stₘ)  Fin N
```

Combining the definition above, we have the exact definition of the Broad RevTerminate Principle:

```
Finite-Reachable-State-Termination : ∀ {N st₀}
```

$\rightarrow (\mathit{St\text{-}Fin} : \exists[\ m\ ]\ \exists[\ st_m\ ]\ (st_0 \mapsto [\ m\ ]\ st_m)\ \ \mathsf{Fin}\ N)$

$\rightarrow (\mathit{has\text{-}next} : \forall\ (st : \mathsf{State}) \rightarrow \mathsf{Dec}\ (\exists[\ st'\ ]\ (st \mapsto st')))$

$\rightarrow \mathsf{is\text{-}initial}\ st_0$

$\rightarrow \exists[\ st_n\ ]\ (st_0 \mapsto^*\ st_n \times \mathsf{is\text{-}stuck}\ st_n)$

4.2 Formal Logic Proof for the Narrow RevTerminate Principle

4.2.1 Steps

$\mathsf{Finite\text{-}State\text{-}Termination} : \forall\ \{N\ st_0\}$

$\rightarrow \mathsf{State}\ \ \mathsf{Fin}\ N$

$\rightarrow (\mathit{has\text{-}next} : \forall\ (st : \mathsf{State}) \rightarrow \mathsf{Dec}\ (\exists[\ st'\ ]\ (st \mapsto st')))$

$\rightarrow \mathsf{is\text{-}initial}\ st_0$

$\rightarrow \exists[\ st_n\ ]\ (st_0 \mapsto^*\ st_n \times \mathsf{is\text{-}stuck}\ st_n)$

Similar to the informal logic proof, we prove the principle using the following steps:

1. Starting from the initial state, continue steping into the next state.
2. If the next state doesn't exist, then the machine terminates.
3. Upon taking $\mathbb{N}$ steps, demonstrate a contradiction using Piegonhole Principle and No-Repeat Principle.
4. At this point, we have proven the case within $\mathbb{N}$ steps and shown that it's impossible for the case to extend beyond $\mathbb{N}$ steps.

4.2.2 Countdown Rule

To realize the step of continuing to the next step, we add a step m and the corresponding state st_m.

```
is-initial : State → Set
is-initial st = ∄[ st' ] (st' ↦ st)
```

When m is less than ℕ, we try stepping into the next state. If there isn't a next state, we have reached the target stuck state. Otherwise, we make a recursion with the case of m increasing. Until m equals ℕ, we have to prove that the case is impossible.

```
is-initial : State → Set
is-initial st = ∄[ st' ] (st' ↦ st)
```

For the method above, Agda will give a "Termination checking failed" error. To make the proof terminate, we have to create a variable "countdown" in the statement. The "countdown" should keep decreasing until it equals to zero. And we prove the case zero at last. The "countdown" is precisely the counterpart to m. We keep ensuring "cd + m ≡ N." In this way, the proof will terminate exactly when m equals to ℕ, as originally intended.

Here is the new principle with a countdown

```
Finite-State-Termination-With-Countdown : ∀ {N sto}
  → State  Fin N
  → is-initial sto
  → ∀ cd m st_m → cd + m ≡ N → sto ↦[ m ] st_m
  → ∃[ st_n ] (sto ↦* st_n × is-stuck st_n)
```

4.2.3 Case at N

Here is the statement of Narrow RevTerminate Principle after n steps; we have to prove this case is impossible:

$$
\begin{aligned}
&\text{Finite-State-Termination-At-N} : \forall\ \{N\ st_0\}\\
&\quad \to \text{State}\ \ \text{Fin}\ N\\
&\quad \to \text{is-initial}\ st_0\\
&\quad \to \exists[\ st_n\ ]\ (st_0 \mapsto[\ N\ ]\ st_n) \to \bot
\end{aligned}
$$

The proof of "reaching the n-th state will terminate" can be simply divided into the two steps below:

1. Assume the n+1-th state exists. Using the Piegonhole Principle, we map n states to n + 1 states and find two identical states that are traversed.

2. The No-Repeat Principle tells us that the two identical states should not exist, as their steps differ, resulting in a contradiction.

.

Before introducing the Piegonhole Principle and the No-Repeat Principle, we will discussing the mapping rules.

4.2.4 Mapping Rules

In our proof of n-th Termination case, a critical part is the mapping rules. We have:

1. An injection from Step to State. Note that the domain of Step is from 0 to $\mathbb{N}$ (inclusive).

2. A bijection between State and Index. Due to the limitation of States, we should have $\mathbb{N}$ indices in total.

3. An injection from Step to Index, constructed through the injection and bijection mentioned above.

.

Here is a diagram illustrating the relationships:

$$\begin{tikzcd}
\boxed{\text{Step 0}} \arrow[r] & \boxed{\text{State 0}} \arrow[dr] & \boxed{\text{Index 0}} \\
\boxed{\text{Step 1}} \arrow[r] & \boxed{\text{State 1}} \arrow[ddr] & \boxed{\text{Index 1}} \\
\boxed{\text{Step 2}} \arrow[r] & \boxed{\text{State 2}} \arrow[uur] & \boxed{\text{Index 2}} \\
\boxed{\text{Step 3}} \arrow[r] & \boxed{\text{State 3}} \arrow[ur] & \boxed{\text{Index 3}} \\
\boxed{\text{Step n}} \arrow[r] & \boxed{\text{State n}} \arrow[r] & \boxed{\text{?}}
\end{tikzcd}$$

Note again that there are only ℕ indices in these relationships. With this relationships, we can clearly describe the Termination proof at the n-th state:

1. First, we have ℕ+1 states (from st_0 to st_n)
2. Using the injection from Step to Index, we find two different steps mapping to the identical indices using Piegonhole Principle.
3. With the bijection from index back to State, the identical indices should map to the same states.
4. The No-repeat Principle tells us that the two different steps should map to different states, leading to a contradiction.

.

4.2.5 Piegonhole Principle

The definition of Piegonhole Principle in Agda is shown below:

```
pigeonhole : ∀ N → (f : ℕ → ℕ)
```

```
    → (∀ n → n ≤ N → f n < N)
    → ∃[ m ] ∃[ n ] (m < n × n ≤ N × f m ≡ f n)
```

This principle requires two key inputs:

1. A injection from natural number to another natural number. In this case, it is the injection from Step to Index.
2. All mapped number should be less than ℕ. The proof is inherent in the definition of Fin. That is, a natrual number value of Fin ℕ is always less than ℕ.

.

With this, we can find two different steps that inject to the same index.

4.2.6 No-Repeat Principle

The definition of No-Repeat Principle in Agda is shown below:

The No-Repeat Principle is proved in Chao-Hong Chen's paper [1]. In the principle, we know that two states reached from the initial state by different steps should be different states.

```
NoRepeat : ∀ {st₀ stₙ stₘ n m}
    → is-initial st₀
    → n < m
    → st₀ ↦[ n ] stₙ
    → st₀ ↦[ m ] stₘ
    → stₙ ≢  stₘ
```

At this point, we have supplemented the remaining part of the Narrow RevTermination Principle, thus completing the overall proof.

4.3 Formal Logic Proof for the Broad RevTerminate Principle

4.3.1 Step

```
Finite-Reachable-State-Termination : ∀ {N st₀}
  → (St-Fin : ∃[ m ] ∃[ stₘ ] (st₀ ↦[ m ] stₘ)  Fin N)
  → (has-next : ∀ (st : State) → Dec (∃[ st' ] (st ↦ st')))
  → is-initial st₀
  → ∃[ stₙ ] (st₀ ↦* stₙ × is-stuck stₙ)
```

Similar to informal logic proof, we prove the principle using the following steps:

1. Starting from the initial state, continue steping into the next state.
2. If the next state doesn't exist, then the machine terminates.
3. After walking ℕ steps, demonstrate a contradiction using the Piegonhole Principle and Mapping Rules.
4. At this point, we have proven the case within ℕ steps and shown that it's impossible for the case to extend beyond ℕ steps.

4.3.2 Countdown Rule

Here is the Countdown Rule for Broad RevTerminate Principle. Same as the narrow version, we introduce the variable "m" to represent stepping in, and create a corresponding variable "countdown" to ensure the proof terminates.

```
is-initial : State → Set
is-initial st = ∄[ st' ] (st' ↦ st)
```

4.3.3 Case at N

Here is the statement of Broad revTermination after ℕ steps, where we have to prove the case is impossible:

```
Finite-State-Termination-At-N : ∀ {N st₀}
  → State  Fin N
  → is-initial st₀
  → ∃[ stₙ ] (st₀ ↦[ N ] stₙ) → ⊥
```

The proof of "termination at the n-th state" can be simplified into the following two steps:

1. Assume that n+1-th state exists. With the Piegonhole principle, we can map n states to n+1 states, thus finding two identical reachable states.

2. In fact, we don't need No-Repeat Principle here because the definition of a reachable state itself is sufficient to get generate a contradiction.

.

We will first discuss the mapping rules for the Broad RevTerminate Principle.

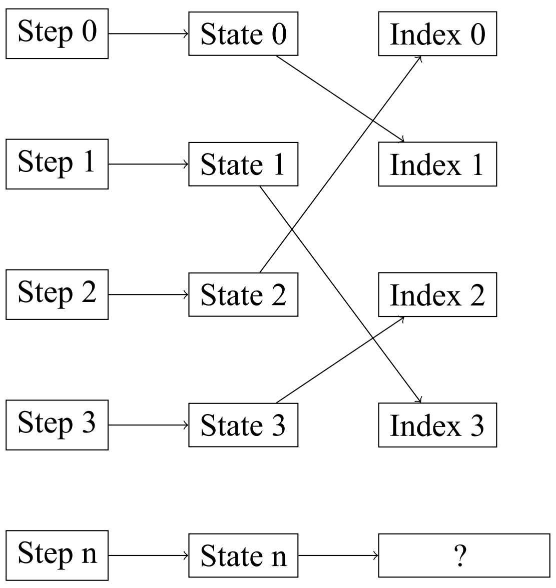
4.3.4 Mapping Rules

In our proof of termination as the n-th case, a critical component is the mapping rules. We establish:

1. An injection from Step to reachable state. Note that the domain of Step is ranges from 0 to ℕ (inclusive).

2. A bijection between reachable state and Index, due to the limited number of States, resulting in a total of ℕ indices.

. Here is a diagram illustrating these relationships:



Note again that there are only ℕ indices in the relationships. With this relationships, we can clearly describe the Termination proof at the n-th state:

1. Initially, we have n+1 states (from st_0 to st_n).
2. With the injection from Step to Index, using the Piegonhole Principle, we identify two different steps that map to the identical index.
3. With the bijection from Index back to State, the same indices must map to the same reachable states.

.

4.3.5 Piegonhole Principle

The definition of Piegonhole principle in Agda is shown below:

```
pigeonhole : ∀ N → (f : ℕ → ℕ)
  → (∀ n → n ≤ N → f n < N)
  → ∃[ m ] ∃[ n ] (m < n × n ≤ N × f m ≡ f n)
```

This principle requires two key inputs:

1. A injection from natural number to another natural number. In this case, it is the injection from Step to Index.

2. All mapped number should be less than ℕ. The proof is inherent in the definition of Fin. That is, a natrual number value of Fin ℕ is always less than ℕ.

.

Using these inputs, we can derive two different steps that map to the same index.

4.3.6 Contradiction

Here, we do not require the No-Repeat Principle. The Piegonhole Principle indicates that two different steps correspond to the same index; through bijection, this same index maps back to two identical reachable states. The definition of reachable states inherently includes the step in the trace, allowing us to straightforwardly prove that the steps are the same using the following codes:

```
proj□-eql : ∀ {sto stₘ stₙ} {m : ℕ} {n : ℕ} {pₘ : sto ↦[ m ] stₘ} {pₙ : sto ↦[ n ] stₙ}
  → (m , stₘ , pₘ) ≡ (n , stₙ , pₙ)
  → m ≡ n
proj□-eql refl = refl
```

Therefore, we arrive at the conclusion that the two steps are simultaneously the same and different, presenting an obvious contradiction.

With this, we have addressed the remaining elements of the Broad RevTerminate Principle, thereby completing the overall proof.

Chapter 5
Conclusion

In this paper, we have extended the work of Chao-Hong Chen by providing comprehensive formal proofs for the termination of reversible abstract machine. Our contributions can be summarized as follows:

1. Formalization and Proof Completion: We have successfully formalized and completed the proofs for two theorems that were only partially addressed in the original work. By doing so, we ensured a rigorous validation of the termination conditions for reversible abstract machines.

2. In the proof above, we initially focused on the scenario where all states are finite. We then extended this proof to encompass cases with infinite states but with a finite number of reachable states from given initial state, thereby broadening the applicability and robustness of our results.

3. Implementation in Agda: We provided approximately 400 lines of Agda code to support our theoretical findings. This code serves not only as a proof of concept but also as a practical tool for verifying the correctness of our formal proofs.

Reference

[1] Chao-Hong Chen. (2021). A Computational Interpretation of Compact Closed Categories: Reversible Programming with Negative and Fractional Types.

[2] Chao-Gong Chen. (2021). ALGEBRAIC INFORMATION EFFECTS. 12–13.

[3] The Agda Team. 2022. The Agda Documentation. (2022). Welcome to Agda's Documentation! https://agda.readthedocs.io/en/v2.6.2.1/.

[4] Krook, Robert and jansson, & Patrik. (2019). An Algebra of Sequential Decision Problems. 8.

[5] Natural Number Definition. (2022). Programming Language Foundations in Agda - Naturals. https://plfa.github.io/Naturals/

[6] Edward z. yang. (2010). Well-Founded Recursion in Agda. Well-Founded Recursion in Agda. http://blog.ezyang.com/2010/06/well-founded-recursion-in-agda/