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 (Wen-Lung Tsai)
指導教授：陳穎平 (Ying-ping Chen)

中華民國 一一三年五月
May 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
指導教授：陳穎平

Student: Wen-Lung Tsai
Advisor: Dr. Ying-ping Chen

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

May 2024

Taiwan, Republic of China

中華民國 一一三年五月

# Acknowledgement

TODO: 致謝辭

# 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

陽明交大
NYCU

# A Formal Termination Proof for Reversible Abstract Machines

Student: Wen-Lung Tsai Advisor: Dr. Ying-ping Chen

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

陽明交大
NYCU

# Contents

# ListofFigures

陽明交大
NYCU

# ListofTables

# 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 **??** introduces thesis.cls document class. Chapter **??** introduces section ordering. Chapter **??** and **??** 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. 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.



```
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. Mathematical Induction on Natural Numbers: 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 $\ell$

$\_\mapsto\_$ : Rel State $\ell$

deterministic : $\forall\ \{st\ st_1\ st_2\} \longrightarrow st \mapsto st_1 \longrightarrow st \mapsto st_2 \longrightarrow st_1 \equiv st_2$

deterministic$_{rev}$ : $\forall\ \{st\ st_1\ st_2\} \longrightarrow st_1 \mapsto st \longrightarrow st_2 \mapsto st \longrightarrow st_1 \equiv st_2$



"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$_{rev}$", 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

# Narrow Reversible Machine Termination

## 3.1 Formulate Statement

First of all, 先定義以下名詞：“initial state” means that in the machine, there is no other state reaching it. “the termination of machine” means we reach a state in the machine, and there is no next state can be reached. The last state also be called the “stuck state” When we said “the machine will terminate from an initial state” means that starting from initial state, 經過 machine 有限次的推移，we will reach an stuck state.

We have a reversible abstract machine, and 指定 one of the state as initial state. It’s known that there are finite number of reachable states from initial state. With the forward deterministic of reversible machine, we could make the assumption that the initial state will terminate from the initial state.



Here is our 完整的 statement: “Given a reversible abstract machine, it will inevitably terminate from any initial state with a finite number of reachable states.” We call it “Broad RevTerminate Principle”.

In the statement, the number of total state in the machine may be infinite. For the informal logic proof, we can easily ignore those unreachable states. However, it’s 相對困難 for the formal proof. Therefore, we make an 較狹義的 statement as below: “Given a reversible abstract machine with a finite number of total states, it will inevitably terminate from any initial state.” We call it “Narrow RevTerminate Principle”.

In the statement, we could say “we have walked through all the states, so it should be terminate.” in both informal and formal proof easily. We will make the 更詳細的證明 in the next section.

## 3.2 Informal Logic Proof

### 3.2.1 No Repeat

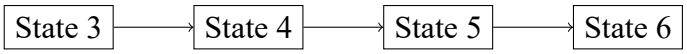
"No Repeat Principle" is formally proved in the paper "A Computational Interpretation of Compact Closed Categories: Reversible Programming with Negative and Fractional Types." Here is the definition of the principle: Given a reversible abstract machine and an initial state, given two different natural numbers m and n, and the initial state reach $st_m$ and $st_n$ by walk through m and n steps. Then $st_m$ should not equal to $st_n$.

### 3.2.2 狹義的 statement

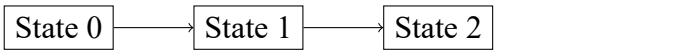
重述一次 the statement we have to proof: "Given a reversible abstract machine with a finite number of total states, it will inevitably terminate from any initial state."

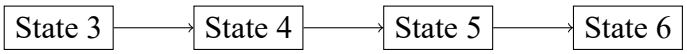


Here is the graph of the machine we expected. It seems monotonous. The initial state will 不重複地經過 all the other states and then terminate.

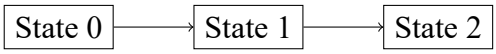

This is the other case for the machine with finite states. In the graph, the initial state also walk through finite state and terminate. However, not all the states are traversed.



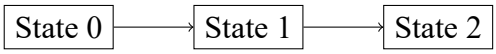

Assume number of all states is N. That is, we have $st_0$, $st_1$, ..., and $st_{N-1}$. Assume $st_0$ is the initial state, we can check whether $st_0$ has next state. Once current state doesn't have next state, the machine will terminate. Therefore, we just have to consider the states starting from $st_0$ always have its next state.

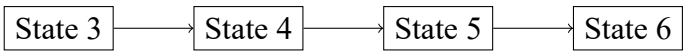
當 trace 推移了 N 次, it should be like the graph below:





這裡，我們不特別提及已遍歷的前 n 個 state 應完全不重複，即使顯而易見地他們重複時會引發矛盾。我們將重點關注於 the whole trace with lenth of N+1. The reversible machine have only N states, but there are N+1 state we have traversed. According to piegonhole principle, we can find two same states in the trace.




With No-Repeat principle, we know it's impossible for an initial state, walk through two different steps but reach same state. 到這邊，我們成功找出矛盾。The contradiction shows the length of trace is always less or equal than N, and then terminate.



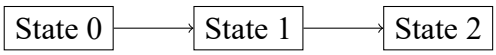
### 3.2.3 廣義的 statement

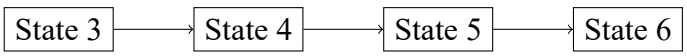
重述一次 the statement we have to proof: "Given a reversible abstract machine, it will inevitably terminate from any initial state with a finite number of reachable states."

Similar to Narrow RevTerminate Principle, we assume $st_0$ is the

$initial state, and$關注於$there are always states after each state we walk through. The Broad Rev Termi$

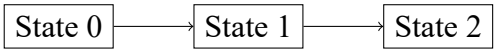
$st_0$ be able to reach. Assume the number of finite states $st_0$ be able to reach is N. 我們與先前一樣走 N 步以讓 the length of trace be N+1.

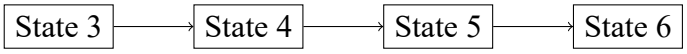



There are N+1 state we have traversed. And now, there are only N states $st_0$ be able to

reachable. According to piegonhole principle, we can find two same states in the trace.




With No-Repeat principle, we know it's impossible for an initial state, walk through two different steps but reach same state. 到這邊，我們成功找出矛盾。The contradiction shows the length of trace is always less or equal than N, and then terminate.

# Chapter 4

# Narrow Reversible Machine Termination

## 4.1 Formulate Statement

### 4.1.1 Redefinition of Narrow RevTerminate Principle

相較於 informal logic proof, we need 更嚴格的定義 for our formal logic proof. 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 他不是任何 state 的 next state，we call the state is initial. is-initial : State → Set

is-initial $st$ = ∄[ $st'$ ] ($st' \mapsto st$)



Secondly, we define the terminate of a reversible machine. Starting from the initial state $st_0$, we use $st_0 \mapsto^* st_n$ presents $st_0$ can reach $st_n$ by walk through finite steps.

is-initial : State → Set

is-initial $st$ = ∄[ $st'$ ] ($st' \mapsto st$)

And if a state $st_0$ have no next state, we call the state is stuck. is-stuck : State → Set

is-stuck $st$ = ∄[ $st'$ ] ($st \mapsto st'$)

The terminate of a reversible machine means "given an initial state, it will reach a stuck state." is-stuck : State → Set

is-stuck $st$ = ∄[ $st'$ ] ($st \mapsto st'$)

At last, we define “a reversible abstract machine with finite number of total states.” Here is the definition of Fin in agda. a Fin N set have exactly N elements. is-stuck : State → Set

is-stuck $st$ = ∄[ $st'$ ] ($st \mapsto st'$)

We construct a bijection relation between state and Fin N. It seems like all states are indexed by one of element in Fin N set. is-stuck : State → Set

is-stuck $st$ = ∄[ $st'$ ] ($st \mapsto st'$)

Combine the definition above, and we have the exact definition of Narrow RevTerminate Principle: is-initial : State → Set

is-initial $st$ = ∄[ $st'$ ] ($st' \mapsto st$)



### 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 same as Narrow RevTerminate Principle, we have:

Given $st_0$. For all of the states, if their next state is not $st_0$, then $st_0$ is initial state. is-initial : State → Set

is-initial $st$ = ∄[ $st'$ ] ($st' \mapsto st$)

$st_0 \mapsto^* st_n$ presents $st_0$ can reach $st_n$ by walk through finite steps. is-initial : State → Set

is-initial $st$ = ∄[ $st'$ ] ($st' \mapsto st$)

If a state have no next state, we call the state is stuck. is-stuck : State → Set

is-stuck $st$ = ∄[ $st'$ ] ($st \mapsto st'$)

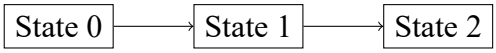
Before defining “a finite number of reachable states”, we have to dealt with the “set

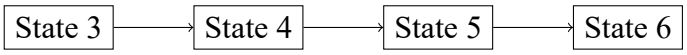
of reachable states." $st_0 \mapsto$[m] $st_m$ means $st_0$ walk exactly m steps and reach $st_m$ For all the m and 對應的 $st_m$ 滿足 $st_0 \mapsto$[m] $st_m$, it's called "set of reachable states."

is-initial : State → Set

is-initial *st* = ∄[ *st'* ] (*st'* ↦ *st*)

In agda, each elements of reachable states set seems like a tuple.



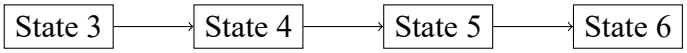


Then, we construct a bijection relation between reachable state set and Fin N. is-initial : State →

is-initial *st* = ∄[ *st'* ] (*st'* ↦ *st*)



It seems like all reachable states are indexed by one of element in Fin N set.



Combine the definition above, and we have the exact definition of Broad RevTermi-nate Principle: is-initial : State → Set

is-initial *st* = ∄[ *st'* ] (*st'* ↦ *st*)

## 4.2 Formal Logic Proof for Narrow RevTerminate Principle

### 4.2.1 Step

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

Similar to informal logic proof, we proof the principle in such steps below:

1. Starting from initial state, keep steping into the next state.
2. If the next state doesn't exist, then it terminate.
3. Upon walked N steps, show contradiction by Piegonhole Principle and No-Repeat Principle.
4. 至此，we have proved the case in N steps and it's impossible for the case over N steps.



### 4.2.2 Countdown Rule

To realize the step of "keep walking into the next step", we add a step m and 對應的 state $st_m$.

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

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

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

For the 證明方式 above, agda will give an "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 proof the case zero at last. The

“countdown” 恰好與 m 相對. We keep providing the “cd + m ≡ N.” In this way, the proof will terminate at m exactly equals to N, 就如原先的預想一樣。

Here is the new principle with countdown Finite-State-Termination-With-Countdown : ∀ {N s

→ State Fin $N$

→ is-initial $st_0$

→ ∀ $cd\ m\ st_m$ → $cd + m \equiv N$ → $st_0 \mapsto[\ m\ ]\ st_m$

→ ∃[ $st_n$ ] ($st_0 \mapsto^*\ st_n$ × is-stuck $st_n$)

### 4.2.3 The case at N

Here is the statement of Narrow revTermination after n steps, we have to proof the case is impossible:

Finite-State-Termination-At-N : ∀ {$N\ st_0$}

→ State Fin $N$

→ is-initial $st_0$

→ ∃[ $st_n$ ] ($st_0 \mapsto[\ N\ ]\ st_n$) → ⊥



The proof of 「當抵達 n-th state 會終止」can be simply 分為 the two steps below：

1. 假定 n+1-th state 存在，with Piegonhole principle ，可以在 n 個 state 映射到 n + 1 個 state，而找到重複經過的兩個相同 state
2. No-repeat 告訴我們找到的兩個 state 不該是相同的，因為他們的 step 不同，得到矛盾。

.

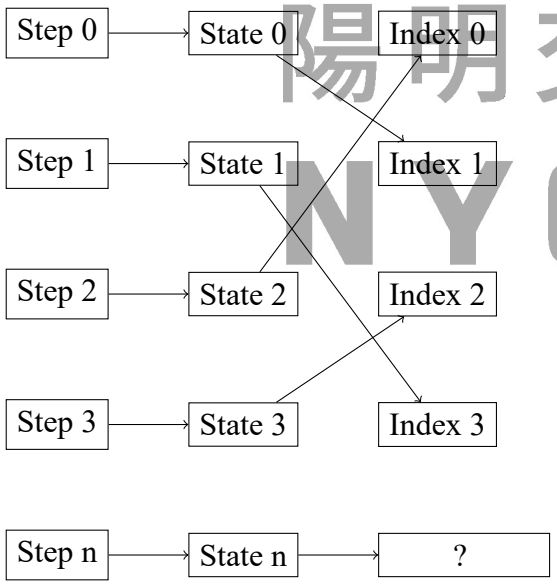
Before introducing Piegonhole principle and No-Repeat Principle, we will discussing mapping rules.

### 4.2.4 mapping rules

// TODO: Fin 相關的說明應該都改為 Index 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 定義域 of step is from 0 to N(included both sides)
2. bijection between State and Index. because of the limitation of States, we should have n indexes in total.
3. we can also construct an injection from Step to Index easily. Just Through the two injection and bijection above.

. Here is an 示意圖 for the relations






Note again that there are only n indexes in the relations. With the relation, we can 更清楚地描述 the Termination proof at n-th state:

1. first of all, we have n+1 states (from $st_0$ to $st_n$)
2. with injection from Step to index, we get two different steps mapping to same index using Piegonhole Principle
3. with bijection from index back to State, the same indexes should be mapped to same states.

4. No-repeat principle told us that the two different steps should be mapped to different states, and the contradiction occurs.

.

### 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)
```

The two necessary inputs are

1. N to N function. In this case, it is the injection from Step to Index.
2. all mapped N should be less than N. The proof is 隱含在 Fin 的定義裡面. That is, a natrual number value of Fin N is always less than N.



.

And we could get two different steps which will injection to the same index.

### 4.2.6 No Repeat

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

No-Repeat-Principle is proved by TODO: 學長論文. In the principle, we know that the two states with different steps from initial state, they should be different states.

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

$\longrightarrow st_0 \mapsto[\ m\ ]\ st_m$

$\longrightarrow st_n \not\equiv st_m$

Until now, we have 補足了 the 剩餘的部分 of Narrow revTermination Principle, 從而補足了整體的證明。

## 4.3 Formal Logic Proof for Broad RevTerminate Principle

### 4.3.1 Step

is-initial : State → Set

is-initial $st = \nexists[\ st'\ ]\ (st' \mapsto st)$



Similar to informal logic proof, we proof the principle in such steps below:

1. Starting from initial state, keep steping into the next state.
2. If the next state doesn't exist, then it terminate.
3. Upon walked N steps, show contradiction by Piegonhole Principle and Mapping Rules.
4. 至此，we have proved the case in N steps and it's impossible for the case over N steps.

### 4.3.2 Countdown Rule

Here is the Countdown Rule for Broad RevTerminate Principle. Same as the narrow one, we add the variable "m" to 呈現 a step-in, and make a 相對的 variable "countdown" to terminate the proof. is-initial : State → Set

is-initial $st = \nexists[\ st'\ ]\ (st' \mapsto st)$

### 4.3.3 The case at N

Here is the statement of Broad revTermination after n steps, we have to proof the case is impossible:

Finite-State-Termination-At-N : $\forall$ $\{N\ sto\}$

$\rightarrow$ State Fin $N$

$\rightarrow$ is-initial $sto$

$\rightarrow \exists[\ st_n\ ]\ (sto \mapsto [\ N\ ]\ st_n) \rightarrow \bot$

The proof of 「當抵達 n-th state 會終止」can be simply 分為 the two steps below：

1. 假定 n+1-th state 存在，with Piegonhole principle ，可以在 n 個 state 映射到 n + 1 個 state，而找到重複經過的兩個相同 reachable state

2. In fact, we don't need no-repeat principle here because the definition of reachable state is enough to get contradiction.



.

We discussing mapping rules for broad revTerminate principle firstly.

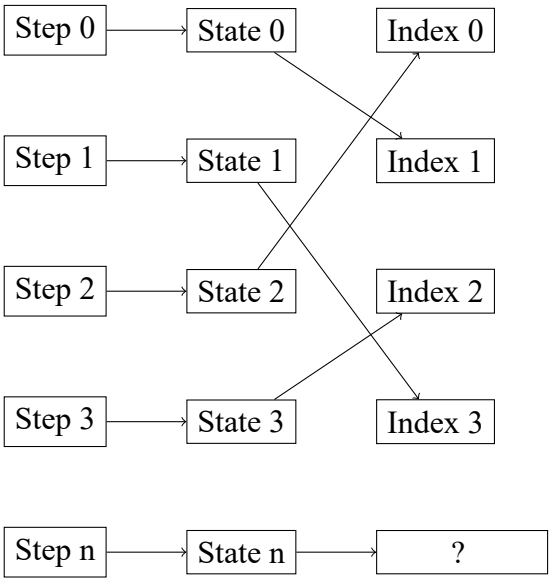
### 4.3.4 mapping rules

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

1. an injection from Step to reachable state. Note that the 定義域 of step is from 0 to N(included both sides)

2. bijection between reachable state and Index. because of the limitation of States, we should have n indexes in total.

. Here is an 示意圖 for the relations




Note again that there are only n indexes in the relations. With the relation, we can 更清楚地描述 the Termination proof at n-th state:

1. first of all, we have n+1 states (from $st_0$ to $st_n$)
2. with injection from Step to index, we get two different steps mapping to same index using Piegonhole Principle
3. with bijection from index back to State, the same indexes should be mapped to 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)
```

The two necessary inputs are

1. N to N function. In this case, it is the injection from Step to Index.

2. all mapped N should be less than N. The proof is 隱含在 Fin 的定義裡面. That is, a natrual number value of Fin N is always less than N.

.

And we could get two different steps which will injection to the same index.

### 4.3.6 contradiction

Here, we don't need the No-Repeat Principle. The piegonhole principle give us two different step and the same index, and we use bijection to make the same index back to two same reachable states. the definition of reachable states 恰好 contains the step for the trace, we can simply proof the step is same by the following codes: Finite-Reachable-State-Termin

→ (*St-Fin* : ∃[ *m* ] ∃[ $st_m$ ] ($st_0$ ↦[ *m* ] $st_m$) Fin *N*)

→ (*has-next* : ∀ (*st* : State) → Dec (∃[ *st'* ] (*st* ↦ *st'*)))

→ is-initial $st_0$

→ ∀ *cd m* $st_m$ → *cd* + *m* ≡ *N* → $st_0$ ↦[ *m* ] $st_m$

→ ∃[ $st_n$ ] ($st_0$ ↦* $st_n$ × is-stuck $st_n$)



So, we have the 結論 of the two steps are both same and different, which makes aobvious contradiction.

Until now, we have 補足了 the 剩餘的部分 of Broad revTermination Principle, 從而補足了整體的證明。

# Chapter 5

# Conclusion

# Appendix A

# 附錄標題

## A.1 Testing