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

可逆抽象機終止條件與其形式化證明

Formal Proof of Termination 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 一一三年四月
April 2024

可逆抽象機終止條件與其形式化證明

Formal Proof of Termination 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

April 2024

Taiwan, Republic of China

中華民國 一一三年四月

Acknowledgement

TODO: 致謝辭

可逆抽象機終止條件與其形式化證明

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 CHAO-HONG CHEN 的研究中，對緊湊封閉類別的以「反轉時間」和「反轉空間」的解釋，將其形式化應用於可逆 SAT 求解器的實作。其中，對可逆抽象機的終止證明技術顯得尤其重要。在該研究中，有兩項定理提到了終止證明，但僅使用了部分形式化的概念證明。

我們的研究具體化了這一問題，提出了一個假設：對於可逆抽象機，如果給定初始狀態，其可抵達的狀態是有限的，那麼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。我們首先針對這一假設提出了狹義的證明，即將限制條件設為「可逆抽象機的所有狀態個數為有限的」。接著，我們移除了這一限制，完成了更廣義的證明。

關鍵詞: 可逆抽象機、終止證明、Agda

Formal Proof of Termination 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 the research conducted by CHAO-HONG CHEN, the interpretation of compact closed categories in terms of ”reversing time” and ”reversing space” was formalized and applied to the implementation of reversible SAT solvers. Among these, the technique for proving termination of reversible abstract machines stands out as particularly important. In this study, two theorems are mentioned regarding termination proofs, but only partial formalization of concepts was utilized.

Our research delved into this issue specifically. We proposed a hypothesis: for reversible abstract machines, if the reachable states from a given initial state are finite, then the initial state will terminate within a finite number of steps through transitions of the reversible abstract machine. We first presented a narrow proof based on this hypothesis, with the restriction that ” the total number of states of the reversible abstract machine is finite.” Subsequently, we removed this restriction and completed a more generalized proof.

Keyword: Abstract Machines, Termination Proofs, Agda

Contents

ListofFigures

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

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 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.

"↦" 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.

"$\text{deterministic}_{\text{rev}}$", 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

Narrow Reversible Machine Termination

3.1 Formulate Statement

Before we can construct the Termination Statement, it is necessary to define certain relationships between states.

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

The is-initial code segment provides a proof that a given state has no preceding states.

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

The is-stuck code segment offers a proof that a given state has no succeeding states.

```
data _↦*_ : State → State → Set (suc ℓ) where
  ∯ : {st : State} → st ↦* st
  _∷_ : {st₁ st₂ st₃ : State} → st₁ ↦ st₂ → st₂ ↦* st₃ → st₁ ↦* st₃
```

The ↦* symbol code segment establishes a trace between two specified states, demonstrating that one state can be reached from the other through a finite number of transitions.

```
data _↦[_]_ : State → ℕ → State → Set (L.suc ℓ) where
  ∯ : ∀ {st} → st ↦[ 0 ] st
  _∷_ : ∀ {st₁ st₂ st₃ n} → st₁ ↦ st₂ → st₂ ↦[ n ] st₃ → st₁ ↦[ suc n ] st₃
```

Similar to the ↦* code, the ↦[n] segment specifies a definite trace length n, establishing a fixed number of transitions from one state to another.

First of all, we describe in natural language the proof content expected for narrow reversible termination: In a reversible machine m, if the number of states in the state set is finite, then each initial state should reach a stuck state after a finite number of transitions and terminate.

Here is the formal definition of a narrow reversible termination statement:

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

In the first two statements, we describe the necessary conditions for the termination of a reversible machine in narrow cases:

1. ”For every state, the existence of a subsequent state is decidable.” Imagine if the machine cannot determine whether a state can continue to progress; in such cases, the state would not be able to advance.

2. There is a bijection between the set of States and Fin N. This statement restricts the total number of states by establishing a correspondence with Fin N.

And in the former 2 statement, we 描述了 Termination 這件事本身：

Given an initial state, we can determine a reachable stuck state, ensuring that the machine will eventually terminate.

3.2 Informal Logic Proof

The proof of Narrow RevTermination using informal logic is straightforward and will serve as our guide for the formal proof:

1. Start from the initial state st_0 we know st_0 is reachable with 0 steps.

2. If st_0 cannot transition to another state, then st_0 is the target stuck state.

3. If st_0 can transition to st_1 we construct a trace from st_0 to st_1 denoted as $st_0 \mapsto^* st_1$ with length of 1.

4. Continue checking whether st_1 has a subsequent state to transition to until we find a stuck state or construct a trace $st_0 \mapsto^* st_n$ with length of n.

5. Upon reaching st_n it is confirmed that all states have been traversed. With no-repeating principle, we know it is impossible for st_n having next state.

. // TODO: 還可以簡化

3.3 Informal Logic Proof

3.3.1 Overall narrow proof

With the steps in informal logic proof, we can construct a Similar proof for the formal one. However, there are 許多細節需要更詳細的定義. The 大致上的 steps are shown below:

1. Starting from initial state, try looking for its next state.

2. If the next state exists, keep looking for 再下一個 state

3. Upon reaching the n-th state, proof that it's impossible for the existence of next states with no-repeat principle.

.

3.3.2 Countdown rules

The principle called Finite-State-Termination-Principle, TODO: 連結到上面 3.1 的定義, agda 無法認知 a proof with "proved n case, and any case should be 推導到 the n-th case", but

這樣的證明方式其實是數學歸納法的一個變體，我們引入 countdown variable 來轉換成 normal 數學歸納法:

$\text{Finite-State-Termination-With-Countdown} : \forall \; \{N \; st_0\}$

$\rightarrow \text{State} \;\; \text{Fin} \; N$

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

$\rightarrow \forall \; cd \; m \; st_m \rightarrow cd + m \equiv N \rightarrow st_0 \mapsto [\; m \;] \; st_m$

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

And our target Principle can be regarded as the special case when Countdown is N, 也就是剛開始的時候。

The Countdown 的運作規則 is as below:

$\text{cd-1} : \forall \{cd\} \{m\} \{N\}$

$\rightarrow \text{suc}(cd + m) \equiv N$

$\rightarrow cd + (m + 1) \equiv N$

To maintain the Countdown, there are cd and m variable. cd and m is 相對的, and their sum is always be N. Starting when m is zero and Countdown is N, after a steps, m 將會 increasing and countdown is decreasing. We use cd-1 principle to ensure the 不變 of the sum.

3.3.3 Termination rules

The 中途停止的規則：We have N different states in a RevMachine, but 並不保證 given st_0 can reach any states in RevMachine. Here is an instance:

Therefore, the principle 的證明有兩個終止規則：

1. trying to 推進 next step. However, there is no next step. In this case we directly got stuck-state.

2. trying to 推進 next step until reaching n-th step, we use no-repeat to 間接地 proof the n-th

step should be stuck.

. The former case is much simplier because it 完全地 find what we need. And the latter case, we have to keep discussing in the 之後的 sections.

3.3.4 When we are at n-th state

With the countdown system and 中途停止的規則，we can 專注於處理以下 principle: When an initial state 經過 n step and reach n-th state, then the n-th state should be stuck. The formal principle is shown below:

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

The proof of「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.

3.3.5 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 定義域 of step is from 0 to N(included both sides)

2. bijection between State and Fin N. The Fin N can be 理解成 the index of each state.

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

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.

.

3.3.6 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.

3.3.7 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
  → st₀ ↦[ m ] stₘ
  → stₙ ≢ stₘ
```

// TODO: 可能需要小結論

Chapter 4

Broad Reversible Machine Termination

4.1 Formulate Statement

The definition of broad reversible machine shows below:

$$
\begin{aligned}
&\text{Finite-Reachable-State-Termination} : \forall \{N\, st_0\} \\
&\quad \rightarrow (\mathit{St\text{-}Fin} : \exists[\, m\,]\ \exists[\, st_m\,]\ (st_0 \mapsto[\, m\,]\ st_m)\ \text{Fin}\ N) \\
&\quad \rightarrow \text{is-initial}\ st_0 \\
&\quad \rightarrow \exists[\, st_n\,]\ (st_0 \mapsto^{*} st_n \times \text{is-stuck}\ st_n)
\end{aligned}
$$

It is similar to the narrow one, except for the Bijection. In the Statement of narrow reversible machine termination, we consider the number of all states is finite. On the contrary, the Statement of broad reversible machine termination firstly given an st_0, and 限定了 the number of ”reachable states from st_0” is finite. In this restriction, the number of states could be infinite. Some initial states in machine may also not be terminated unless we know the initial state can just reach finite states.

4.2 Informal Logic Proof

4.3 Formal Logic Proof

Chapter 5
Figures and Tables

關於怎麼使用圖和表格, 網路上都可以找到許多介紹. Google it. (其實是我累了, 不想寫了 XD)

5.1 Figures

這裡介紹如何載入一張圖片, 解釋請看.tex 檔的註解吧！此外, 我個人習慣是把所有的圖檔都放在一個資料夾下, 像是 *figures/*

注意: 在呼叫圖的標籤的時候, 請寫 **Figure~\ref{label}**. (請參考.tex 檔看我是如何載入 Figure **??** 的吧).

5.2 Tables

我個人的習慣是把表格的內容放到另一個.tex 內, 再把這些.tex 檔放到另一個資料夾下 (e.g. *tables/*), 讓本文看起來不會那麼亂. 請參考 Table **??** 裡面的註解 (檔案位置 *tables/table-classopt.tex*), 看看要怎寫一個簡單的 table 吧.

Figure 5.1: The history of NCTU days back to 1896 ...

Appendix A

附錄標題

A.1 Testing