ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN PHAN TÌNH

TÍNH CẬN TRÊN BỘ NHỚ LOG CỦA CHƯƠNG TRÌNH SỬ DỤNG GIAO DỊCH

Ngành: Công Nghệ Thông Tin
Chuyên ngành: Kỹ thuật Phần Mềm
Mã số: 60480103

TÓM TẮT LUẬN VĂN THẠC SỸ CÔNG NGHỆ THÔNG TIN

Hà Nội - 2016

MỤC LỤC

MỞ ĐẦU

Tính cấp thiết của đề tài

Cùng với sự phát triển như vũ bão của khoa học công nghệ, các vi xử lý hiện đại ngày càng thể hiện sức mạnh qua nhiều nhân (core) với tốc độ xử lý ngày càng cao. Có được như vậy là do bên trong các vi xử lý này được thiết kế các luồng (thread) có khả năng chạy và xử lý song song. Trước đây để lập trình đa luồng, người ta sử dụng cơ chế đồng bộ (synchronization) dựa trên khóa (lock) để áp đặt giới hạn về quyền truy cập tài nguyên trong một môi trường khi có nhiều luồng thực thi.Tuy nhiên, khi áp dụng phương pháp này thường nảy sinh các vấn đề như khóa chết (deadlock) hoặc các lỗi tiềm tàng…

Software Transactional Memory (STM- bộ nhớ giao dịch phần mềm) [8] là một giải pháp đơn giản hơn, nhưng vô cùng mạnh mẽ mà có thể giải quyết được hầu hết các vấn đề trên. Nó đã thay thế hoàn toàn giải pháp cũ trong việc truy cập bộ nhớ dùng chung. STM giao tiếp với bộ nhớ thông qua các giao dịch. Các giao dịch này cho phép tự do đọc, ghi để chia sẻ các biến và một vùng nhớ gọi là log sẽ được sử dụng để ghi lại các hoạt động này cho tới khi kết thúc giao dịch.

Một trong những mô hình giao dịch phức tạp sử dụng STM là mô hình giao dịch lồng và đa luồng (nested and multi-threaded transaction) [5]. Trong quá trình thực thi của các chương trình giao dịch lồng và đa luồng, khi các luồng mới được sinh ra hoặc một giao dịch được bắt đầu, các vùng bộ nhớ gọi là log sẽ được cấp phát. Các log này dùng để lưu lại bản sao của các biến dùng chung, nhờ vậy mà các luồng trên có thể sử dụng các biến này một cách độc lập.

Vấn đề đặt ra ở đây là tại thời điểm chương trình chạy liệu lượng bộ nhớ cần cấp phát cho các log có vượt quá tài nguyên bộ nhớ của máy, hay chương trình có thể chạy một cách trơn tru mà không gặp phải bất kỳ lỗi nào như hết bộ nhớ. Chính vì vậy, việc xác định cận trên của bộ nhớ ở thời điểm chạy chương trìnhcủa chương trình giao dịch là một vấn đề then chốt, có ý nghĩa hết sức quan trọng.

Chính vì lí do đó, trong luận văn thực hiện ở đây, một nghiên cứu sử dụng phương pháp phân tích tĩnh để giải quyết bài toán tính cận trên bộ nhớ log của chương trình có giao dịch sẽ được trình bày, dựa trên bài báo đã được các tác giả công bố trong [1].

Mục tiêu nghiên cứu

Luận văn được thực hiện dựa trên nghiên cứu trong bài báo [1]. Do vậy để có thể hiểu được giải pháp các tác giả đã đề xuất trong [1], trong luận văn này tập trung nghiên cứu các lý thuyết về hệ thống kiểu; Các khái niệm cơ bản cũng như tính chất của giao dịch; Nghiên cứu cú pháp và ngữ nghĩa của ngôn ngữ TM (Transactional Memory) – Một ngôn ngữ để viết các chương trình giao dịch. Từ việc nắm được giải pháp xây dựng hệ thống kiểu đề cập trong bài báo, một công cụ sẽ được cài đặt dựa trên ngôn ngữ C#.

Phương pháp nghiên cứu

Để thực hiện được mục tiêu đã đề ra trong luận văn, các phương pháp nghiên cứu sau đây đã được kết hợp:

- Phương pháp nghiên cứu lý thuyết: bao gồm phân tích, tổng hợp các tài liệu, bài báo có liên quan về lý thuyết hệ thống kiểu đặc biệt là hệ thống kiểu cho các chương trình TM, tài liệu về các thuật toán dựa trên hệ thống kiểu..

- Phương pháp thực nghiệm: Cài đặt thuật toán đã đề xuất, chạy thử để kiểm tra tính đúng đắn của chương trình.

Cấu trúc của luận văn

Luận văn được trình bày trong các phần, với nội dung chính của mỗi phần như sau:

Mở đầu: Nêu ra tính cấp thiết của đề tài, nêu ra các mục tiêu cần nghiên cứu, các phương pháp được sử dụng khi nghiên cứu và cấu trúc các phần của luận văn.

Chương 1: Giới thiệu bài toán

Trình bày nội dung cụ thể của bài toán tính cận trên bộ nhớ log của chương trình có sử dụng giao dịch, các vấn đề cần giải quyết trong bài toán này và hướng tiếp cận để giải quyết bài toán. Trong phần này, các điểm cải tiến của phương pháp giải quyết bài toán ở đây so với các phương pháp trước kia cũng được nêu ra. Ví dụ được trình bày trong mục 1.3 sẽ minh họa rõ ràng cho bài toán và hướng tiếp cận đã đưa ra.

Chương 2: Một số kiến thức cơ sở

Trình bày các lý thuyết cơ bản được sử dụng làm cơ sở để giải quyết bài toán, bao gồm: Lý thuyết về hệ thống kiểu như khái niệm, các thuộc tính hay ứng dụng của hệ thống kiểu trong thực tế; Lý thuyết về giao dịch, bộ nhớ giao dịch phần mềm gồm các khái niệm, tính chất, ứng dụng…

Chương 3: Ngôn ngữ giao dịch

Giới thiệu ngôn ngữ giao dịch TM (Transactional memory)- Một ngôn ngữ được dùng để viết các chương trình giao dịch. Trong chương này, cú pháp và ngữ nghĩa của ngôn ngữ TM sẽ được trình bày cụ thể.

Chương 4: Hệ thống kiểu cho chương trình giao dịch

Nghiên cứu hệ thống kiểu để giải quyết bài toán tính cận trên bộ nhớ log cho chương trình có giao dịch đã được trình bày trong bài báo [1]. Lý thuyết hệ thống kiểu được phát triển ở đây bao gồm các kiểu và các quy tắc kiểu.

Chương 5: Xây dựng công cụ và thực nghiệm

Cài đặt các thuật toán kiểu dựa trên hệ thống kiểu đã được trình bày ở chương 4. Từ các thuật toán đó, xây dựng công cụ để giải quyết bài toán tính cận trên bộ nhớ log và thực nghiệm để kiểm tra tính đúng đắn của công cụ.

Kết luận:

Đánh giá các kết quả đã đạt được, nêu ra tồn tại và hướng phát triển.

CHƯƠNG 1. GIỚI THIỆU BÀI TOÁN

1.1. Giới thiệu

Như chúng ta đã đề cập ở phần mở đầu, STM là giải pháp được sử dụng trong việc chia sẻ bộ nhớ dùng chung và một trong những mô hình giao dịch phức tạp sử dụng STM là mô hình giao dịch lồng và đa luồng (nested and multi-threaded transaction)

Ở đây, một giao dịch được gọi là lồng khi nó chứa một số giao dịch khác. Chúng ta gọi giao dịch cũ là giao dịch cha và gọi các giao dịch mà sinh ra trong giao dịch cha là giao dịch con. Các giao dịch con này phải được đóng trước giao dịch cha. Hơn nữa, giao dịch được gọi là đa luồng (multi-threaded) khi các luồng con sinh ra được chạy bên trong giao dịch đồng thời chạy song song với luồng cha đang thực thi giao dịch. Khi một giao dịch được bắt đầu một vùng bộ nhớ gọi là log được cấp phát dùng để lưu lại bản sao của các biến dùng chung. Một luồng mới hay luồng con, khi được sinh ra cũng sẽ tạo một bản sao các log giao dịch của luồng cha. Khi luồng cha thực hiện đóng (commit) giao dịch, tất cả các luồng con được tạo bên trong luồng cha phải cùng đóng với luồng cha. Chúng ta gọi kiểu đóng này là *join commit*, và thời điểm khi những commit này xảy ra được gọi là thời điểm joint commit. Ở thời điểm join commit bộ nhớ được cấp phát cho các log cũng được giải phóng. Join commit đóng vai trò như sự đồng bộ ngầm của các luồng song song. Chính vì hình thức đồng bộ này mà các luồng song song bên trong một giao dịch không hoàn toàn chạy độc lập.

Và vấn đề cần trả lời ở đây là liệu ở thời điểm chạy chương trình, liệu lượng bộ nhớ cần cấp phát cho các log có vượt quá tài nguyên bộ nhớ của máy, hay chương trình có thể chạy một cách trơn tru mà không gặp phải bất kỳ lỗi nào như hết bộ nhớ. Để trả lời cho câu hỏi này, chúng ta cần phải xác định được biên bộ nhớ của chương trình giao dịch hay chính là cận trên bộ nhớ được cấp phát cho các log ở thời điểm biên dịch.

Ở các nghiên cứu trước đây [2, 11], một hệ thống kiểu được phát triển để đếm số lượng log lớn nhất mà cùng tồn tại ở một thời điểm khi chương trình đang chạy. Con số này chỉ cho thông tin thô về bộ nhớ được sử dụng bởi các log giao dịch. Để quyết định thêm chính xác lượng bộ nhớ lớn nhất mà các log giao dịch có thể sử dụng, trong nghiên cứu [1] các tác giả đã đề xuất phương pháp giải quyết vấn đề với việc tính đến kích thước của mỗi log. Đây cũng là điểm cải tiến của hướng tiếp cận mới này so với các hướng tiếp cận trước đó.

Như vậy, bài toán cần giải quyết ở đây có thể phát biểu như sau: Tính toán lượng bộ nhớ yêu cầu lớn nhất cho toàn bộ chương trình giao dịch khi biết kích thước của các log.

1.2. Hướng tiếp cận

Để giải quyết bài toán đặt ra, trước hết chúng ta sẽ viết các chương trình giao dịch bằng một ngôn ngữ dành riêng cho nó, cụ thể là ngôn ngữ TM sẽ được trình bày trong chương 3.

Để thêm thông tin về kích thước của mỗi log, chúng ta sẽ mở rộng lệnh bắt đầu giao dịch trong các nghiên cứu trước để chứa thông tin này. Sau đó chúng ta phát triển một hệ thống kiểu để đánh giá tài nguyên bộ nhớ log mà các giao dịch có thể yêu cầu.

So với các nghiên cứu trước [2,11] thì ý tưởng về các cấu trúc kiểu trong nghiên cứu [1] không có gì thay đổi. Tuy nhiên, các ngữ nghĩa kiểu và các quy tắc kiểu là mới và khác so với các nghiên cứu trước đây.

1.3. Ví dụ minh họa

CHƯƠNG 2. MỘT SỐ KIẾN THỨC CƠ SỞ

2.1. Hệ thống kiểu

2.1.1. Giới thiệu về hệ thống kiểu

Về định nghĩa hệ thống kiểu, có rất nhiều quan điểm được đưa ra. Trong các ngôn ngữ lập trình, hệ thống kiểu được định nghĩa là tập các quy tắc để gán thuộc tính được gọi là kiểu cho các cấu trúc của một chương trình máy tính bao gồm các biến, biểu thức, các hàm, hoặc module...Theo lý thuyết ngôn ngữ, một hệ thống kiểu là một tập các quy tắc quy định cấu trúc và lập luận cho ngôn ngữ. Trong lập trình, hệ thống kiểu được định nghĩa là một cơ chế cú pháp ràng buộc cấu trúc của chương trình bằng việc kết hợp các thông tin ngữ nghĩa với các thành phần trong chương trình và giới hạn phạm vi của các thành phần đó.

Mục đích cơ bản của hệ thống kiểu là ngăn chặn các sự cố do các lỗi thực thi trong quá trình chương trình chạy [3, 6]. Nó được thực hiện bằng cách định nghĩa các giao diện giữa các phần khác nhau của một chương trình máy tính, và sau đó kiểm tra xem các thành phần đã được ghép nối nhất quán hay chưa. Việc kiểm tra này có thể xảy ra tĩnh (tại thời gian biên dịch), hoặc động (tại thời gian chạy), hoặc kết hợp cả kiểm tra tĩnh và động. Ngoài ra hệ thống kiểu còn được sử dụng với nhiều mục đích khác, chẳng hạn như cho phép tối ưu hóa trình biên dịch nhất định, cung cấp một hình thức tài liệu…

2.1.2. Các thuộc tính của hệ thống kiểu

Một hệ thống kiểu có một số thuộc tính sau:

Khả năng kiểm chứng: Hệ thống kiểu phải có thuật toán kiểm tra kiểu để phân loại các chương trình. Một hệ thống kiểu phải chủ động nắm bắt lỗi thực thi trước khi chúng xảy ra.

Tường minh: Các lập trình viên có thể dự đoán nếu một chương trình vượt qua bộ kiểm tra kiểu. Nếu nó lỗi khi kiểm tra, nên tìm được lí do một cách dễ dàng.

Khả năng thực thi: Các biến, biểu thức nên được kiểm tra tĩnh càng nhiều càng tốt. Mặt khác, chúng cũng cần được kiểm tra động. Sự nhất quán cần được kiểm chứng một cách thường xuyên.

2.1.3. Các ứng dụng của hệ thống kiểu

Hệ thống kiểu đóng vai trò quan trọng trong công nghệ phần mềm và trong lĩnh vực bảo mật mạng.

Đối với công nghệ phần mềm, nó được sử dụng trong trình biên dịch của các ngôn ngữ lập trình, tối ưu hóa, trong cơ sở dữ liệu và thậm chí là mô hình các ngôn ngữ tự nhiên… Trong ngôn ngữ lập trình, hệ thống kiểu có các chức năng chính sau :

a. **Phát hiện lỗi**
b. **Trừu tượng hóa**
c. **Làm tài liệu**

d. Tăng hiệu quả

2.2. Giao dịch và bộ nhớ giao dịch phần mềm (Software Transactional Memory- STM)

2.2.1. Giao dịch (Transaction)

Một giao dịch là một luồng điều khiển mà áp dụng một chuỗi hữu hạn các thao tác nguyên thủy (primitive) vào bộ nhớ [8]. Hay nói cách khác một giao dịch là một thực thi của một chương trình người dùng.

2.2.2. Bộ nhớ giao dịch phần mềm (Software Transactional Memory- STM)

Từ năm 1986, ý tưởng cung cấp hỗ trợ phần cứng cho các giao dịch đã ra đời. Cho đến 1995 Nir Shavit và Dan Touitou đã mở rộng ý tưởng này cho bộ nhớ giao dịch phần mềm. Kể từ đó, nó đã trở thành trọng tâm của các các lý thuyết nghiên cứu chuyên sâu và các ứng dụng thực tế.

Trong khoa học máy tính, bộ nhớ phần mềm giao dịch (STM) là một cơ chế kiểm soát đồng thời tương tự như các giao dịch cơ sở dữ liệu cho việc kiểm soát quyền truy cập vào bộ nhớ dùng chung trong tính toán song song. Đây là một phương pháp thay thế cho cơ chế đồng bộ dựa trên khóa. STM là một chiến lược thực hiện trong phần mềm, chứ không phải là một thành phần phần cứng.

CHƯƠNG 3. NGÔN NGỮ GIAO DỊCH

Trong chương này chúng ta sẽ nghiên cứu về cú pháp và ngữ nghĩa của một ngôn ngữ giao dịch được gọi là TM (Transactional Memory).

Một chương trình TM bắt đầu bằng lệnh onacid(n) (với n biểu diễn kích thước bộ nhớ được cấp phát cho log khi mở một giao dịch mới)và kết thúc bằng lệnh commit.

Dưới đây, chúng ta sẽ tìm hiểu cú pháp và ngữ nghĩa của TM. Trong đó, cú pháp nhằm mô tả các thành phần của một ngôn ngữ. Và các công thức thể hiện hoạt động của chương trình ở các mức cục bộ (bên trong một luồng), ở mức toàn cục (trong các luồng song song). Ngữ nghĩa thể hiện cách thức một chương trình được thực hiện như thế nào.

3.1. Cú pháp của TM [1]

Bảng 3.1 Bảng cú pháp của TM

$$
\begin{aligned}
P &::= 0 \quad | \quad P \parallel P \quad | \quad p(e) \\
e &::= \alpha \quad | \quad \mathbf{onacid}(n) \quad | \quad \mathbf{commit} \quad | \\
&\quad e_1; e_2 \quad | \quad e_1 + e_2 \quad | \quad \mathbf{spawn}(e)
\end{aligned}
$$

3.2. Các ngữ nghĩa động

Ngữ nghĩa của TM được đưa ra bởi 2 mức tập hợp của các quy tắc hoạt động, tương ứng với các ngữ nghĩa cục bộ và toàn cục.

Môi trường thực thi (toàn cục) được cấu trúc như là một tập của các môi trường cục bộ. Mỗi môi trường cục bộ là một chuỗi các log cùng với kích thước của nó.

Môi trường cục bộ và môi trường toàn cục được định nghĩa như sau:

3.2.1. Ngữ nghĩa cục bộ

Các ngữ nghĩa cục bộ liên quan tới việc đánh giá một luồng đơn và các giao dịch cục bộ ở dạng $E, e \rightarrow E', e'$. E và E' ở đây là các môi trường cục bộ, trong khi e và e' là các biểu thức sẽ được thực thi bởi luồng, có nghĩa là một biểu thức e được đánh giá trong môi trường cục bộ E thì nó sẽ được chuyển thành một biểu thức e', tương ứng với nó môi trường cục bộ E sẽ chuyển thành môi trường cục bộ E'.

Định nghĩa 1(Local environment – Môi trường cục bộ) *Một môi trường cục bộ E là một chuỗi tuần tự của các log và kích thước của nó:* $l_1{:}n_1;\ldots l_k{:}n_k$. *Môi trường không có phần tử nào được gọi là môi trường rỗng và ký hiêu bởi* ε *[1].*

3.2.2. Ngữ nghĩa toàn cục

Ở mức toàn cục, ngữ nghĩa sẽ có dạng: $\Gamma, P \Rightarrow \Gamma', P'$ hoặc $\Gamma, P \Rightarrow error$ trong đó: Γ là môi trường toàn cục và P là tập các tiến trình có dạng (e) . Môi trường toàn cục là một tập các môi trường cục bộ mà không rỗng.

Định nghĩa 2 ***(Global environment – Môi trường toàn cục)***

Một môi trường toàn cục Γ là một tập các luồng và môi trường cục bộ của nó, được viết là: $\Gamma = \{ p_1 : E_1 ; \ldots ; p_k : E_k \}$, với p_i là tên luồng và E_i là môi trường cục bộ của luồng [1].

Bảng 3.2. Bảng ngữ nghĩa động của TM

$$\frac{p' \; fresh \quad spawn(p, p', \Gamma) = \Gamma'}{\Gamma, P \parallel p(\mathbf{spawn}(e_1); e_2) \Rightarrow \Gamma', P \parallel p(e_2) \parallel p'(e_1)} \text{ S-SPAWN}$$

$$\frac{l \; fresh \quad start(l{:}n, p, \Gamma) = \Gamma'}{\Gamma, P \parallel p(\mathbf{onacid}(n); e) \Rightarrow \Gamma', P \parallel p(e)} \text{ S-TRANS}$$

$$\frac{intranse(\Gamma, l : n) = \mathbf{p} = \{p_1, .., p_k\} \quad commit(\mathbf{p}, \Gamma) = \Gamma'}{\Gamma, P \parallel \coprod_1^k p_i(\mathbf{commit}; e_i) \Rightarrow \Gamma', P \parallel \coprod_1^k p_i(e_i)} \text{ S-COMM}$$

$$\frac{i = 1, 2}{\Gamma, P \parallel p(e_1 + e_2) \Rightarrow \Gamma, P \parallel p(e_i)} \text{ S-COND}$$

$$\frac{}{\Gamma, P \parallel p(\alpha; e) \Rightarrow \Gamma, P \parallel p(e)} \text{ S-SKIP}$$

$$\frac{\Gamma = \Gamma' \cup \{p : E\} \quad |E| = 0}{\Gamma, P \parallel p(\mathbf{commit}; e) \Rightarrow error} \text{ S-ERROR-C} \qquad \frac{\Gamma = \Gamma' \cup \{p : E\} \quad |E| > 0}{\Gamma, P \parallel p() \Rightarrow error} \text{ S-ERROR-O}$$

CHƯƠNG 4. HỆ THỐNG KIỂU CHO CHƯƠNG TRÌNH GIAO DỊCH

Mục đích chính của hệ thống kiểu là để xác định lượng bộ nhớ lớn nhất mà một chương trình TFJ có thể yêu cầu.

Kiểu của một thành phần (term) trong hệ thống được tính toán từ chuỗi các số có dấu, là một biểu diễn trừu tượng của thành phần hành vi giao dịch liên quan tới bộ nhớ log.

4.1. Các kiểu

Theo [1], các kiểu của chúng ta là các chuỗi giới hạn trên tập được gọi là chuỗi số có dấu. Một số có dấu là một cặp của các dấu và các số tự nhiên không âm N^+. Chúng ta sử dụng 4 dấu {+, −, ¬, #} cho việc ký hiệu tương ứng mở, đóng, joint commit, và bộ nhớ tích lũy lớn nhất mà các log sử dụng.

Tập tất cả các chuỗi số có dấu được ký hiệu là ${}^{T}N$.

Do đó ${}^{T}N= \{{}^{+}n, {}^{-}n, {}^{\neg}n, {}^{\#}n\}$

Ý nghĩa của những số có dấu này được mô tả như sau :

- Số có dấu ${}^{+}n$ chỉ ra rằng mở giao dịch có kích thước của log là n đơn vị bộ nhớ. Lưu ý là ngữ nghĩa này khác so với các nghiên cứu [5,6], ở đó nó ký hiệu cho n lệnh mở giao dịch onacid liên tiếp.
- Số có dấu ${}^{-}n$ có nghĩa là có n lệnh commit liên tiếp.
- Số có dấu ${}^{\neg}n$ có nghĩa là n luồng yêu cầu sự đồng bộ ở thời điểm Join commit.
- Số có dấu ${}^{\#}n$ chỉ ra số đơn vị bộ nhớ lớn nhất hiện tại mà thành phần sử dụng là n.

Định nghĩa 5 (Chuỗi chuẩn tắc): *Một chuỗi gọi là chuẩn tắc nếu tag(S) không chứa '−−', '##', '+ −', '+ ¬', '+ #¬' hoặc '+# −' và $|S(i)|>0$ với mọi i [1].*

Định nghĩa 6 (Rút gọn):

Hàm rút gọn seq được định nghĩa đệ quy như sau:

$seq(S) = S$ *khi S là chuẩn tắc*

$seq(S\,{}^{\#}m\,{}^{\#}n\,S') = seq(S\,{}^{\#}\max(m,n)S')$

$seq(S\,{}^{-}m\,{}^{-}n\,S') = seq(S\,{}^{-}(m+n)S')$

$seq(S\,{}^{+}k\,{}^{\#}l\,{}^{-}n\,S') = seq(S\,{}^{\#}(l+k)\,{}^{-}(n-1)\,S')$ *[1]*

Định nghĩa 7 (Cộng):

Cho $S= s_1...s_k$ là một chuỗi chuẩn tắc mà + không nằm trong {S } và giả sử i=first(S, −) . Thì hàm cộng join(S) định nghĩa đệ quy thay thế − trong S bởi ¬ như sau:

$join(S)= S$ *nếu i=0;*

$join(S) = s_1...s_{i-1}\ {}^{\neg}{}_1 join(\,{}^{-}(|s_i|-1)\,s_{i+1}...s_k)$ *ngược lại [1]*

Định nghĩa 8 (Gộp):

Cho S_1 và S_2 là các chuỗi cộng mà số các thành phần ¬ trong S_1 và S_2 là như nhau (có thể là 0). Hàm merge được định nghĩa đệ quy như sau :

Merge ($^{\#} m_1, {}^{\#} m_2$)= $^{\#} (m_1 + m_2)$

Merge ($^{\#} m_1 \neg n_1 S_1'$, $^{\#} m_2 \neg n_2 S_2'$)= $^{\#} (m_1 + m_2) \neg (n_1 + n_2)$ merge(S_1', S_2') [1]

Định nghĩa 9 (Chọn) :

Cho S_1 và S_2 là 2 chuỗi mà nếu chúng ta loại bỏ thành phần # từ chúng, thì hai chuỗi còn lại là giống hệt nhau. Hàm alt được định nghĩa đệ quy như sau :

alt ($^{\#} m_1$, $^{\#} m_2$)= $^{\#}$max(m_1,m_2)

alt ($^{\#} m_1 {}^{*} n_1 S_1'$, $^{\#} m_2 {}^{*} n_2 S_2'$)= $^{\#}$ max(m_1, m_2) *n alt (S_1', S_2') [1]

4.2. Các quy tắc kiểu

Bảng 4.1 Các quy tắc kiểu

$$\frac{}{E \vdash onacid(n): {}^{+}n} \quad T-ONACID$$

$$\frac{n \in N^{+}}{n \vdash commit: {}^{-}1} \quad T-COMMIT$$

$$\frac{n \vdash e: S}{n \vdash spawn(e): join(S)^{p}} \quad T-SPAWN$$

$$\frac{n \vdash e: S}{n \vdash e: join(S)^{p}} \quad T-PREP$$

$$\frac{n_i \vdash e_i: S_i \quad i = 1,2 \;\; S = seq(S_1\ S_2\ )}{n_1 + n_2 \vdash e_1; e_2\ : S} \quad T-SEQ$$

$$\frac{n_1 \vdash e_1: S_1 \quad n_2 \vdash e_2: S_2^{\rho} \;\; S = jc(S_1\ , S_2\ )}{n_1 + n_2 \vdash e_1; e_2\ : S} \quad T-JC$$

$$\frac{n \vdash e_i: S_i^{\rho} \;\; i = 1,2 \;\; S = merge(S_1\ , S_2\ )}{n \vdash e_1; e_2\ : S^{\rho}} \quad T-MERGE$$

$$\frac{n \vdash e_i: S_i^{\rho} \;\; i = 1,2 \;\; kind(T_1) = kind(T_2)\ T_i\ =\ S_i^{kind(T_i)}}{n \vdash e_1 + e_2\ : alt(S_1, S_2)^{kind(S_1)}} \quad T-COND$$

Trong bảng trên, quy tắc T-ONACID cho phép chuyển onacid(n) thành +n.

Quy tắc T- COMMIT cho phép chuyển commit thành -1;

Quy tắc T- SPAWN chuyển S từ chuỗi cộng và đánh dấu các kiểu mới bởi ρ do đó chúng ta có thể gộp nó với chuỗi của luồng cha trong hàm T- MERGE.

Quy tắc T- PREP cho phép chúng ta tìm kiểu phù hợp cho e trong T- MERGE.

Quy tắc T-JC giải quyết join commit giữa các luồng chạy song song và sử dụng tời công thức jc trình bày ở định nghĩa 10. Trong đó, phần tử + cuối cùng trong S_1, gọi là

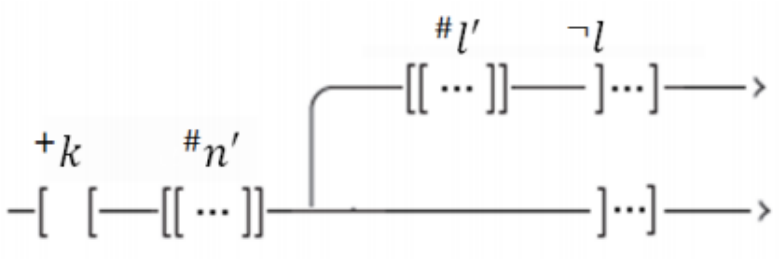
$^{+}$n sẽ được kết hợp với phần tử ¬ đầu tiên trong S_2, gọi là $^{\neg}l$ (Hình 4.1). Nhưng sau $^{+}$n, có thể có thành phần #, gọi là $^{\#}n'$, do vậy cận của các đơn vị bộ nhớ cục bộ được sử dụng bởi thành phần có kiểu $^{+}n^{\#}n'$ là n+ n'. Trước $^{\neg}l$ có thể có $^{\#}l'$, vì vậy khi thực hiện join commit các thành phần có kiểu $^{\neg}l$ với giao dịch bắt đầu của nó có kiểu $^{+}$n, kiểu của các phân đoạn sẽ là $l'+l*n$. Sau khi kết hợp $^{+}$n từ S_1 với $^{\neg}l$ từ S_2 chúng ta có thể rút gọn các chuỗi mới và lặp lại các join commit của jc. Do đó hàm jc được xác định như sau :

***Định nghĩa 10 (Join commit)**:*

Hàm jc được xác định đệ quy như sau :

$jc(S_1' +n\ ^{\#}n', ^{\#}l'^{\neg}l\ S_2') = jc(seq(\ S_1'\ ^{\#}(n+\ n'),\ seq(^{\#}(n+n'), seq(^{\#}(\ l' + l*n)\ S_2'\ ))$ *nếu l>0*

$jc(^{\#}n, ^{\#}l'\ S_2') = seq(^{\#}\ max(n', l')\ S_2')$ *ngược lại [1]*



Hình 4.1 Các luồng song song Joincommit

Định nghĩa 11 (Chương trình well- typed): *Một thành phần e được gọi là well- typed nếu tồn tại một dẫn xuất cho e mà 0 ⊢ e :$^{\#}$ n với một số n* [1].

Do các kiểu phản ánh hành vi của một thành phần, vì vậy kiểu của một chương trình định kiểu tốt (well- typed) chứa chỉ một chuỗi $^{\#}$n ở đó n là số đơn vị bộ nhớ lớn nhất được sử dụng khi thực hiện chương trình.

Dưới đây chúng ta sẽ sử dụng các lý thuyết ở trên để giải bài toán tính cận trên cho chương trình trong hình 1.3.

Để thuận lợi cho việc tính toán, trước hết ta chia nhỏ chương trình thành các biểu thức

e_1 = onacid (1); onacid (2);

e_2 = spawn (onacid (4); commit; commit; commit; commit)

e_3 =onacid (3);

e_4 = spawn (onacid (5); commit; commit; commit; commit)

e_5 = commit;onacid (6);commit;commit;onacid (7);commit;commit;

Trước hết ta tính kiểu cho e_4;

Sử dụng các quy tắc T-ONACID, T-COMMIT và T-SEQ ta có kiểu của

Onacid (5);commit;commit;commit;commit : $^{\#}5^{\neg}1^{\neg}1^{\neg}1$

Áp dụ ng T-SPAWN, ta được

$6\vdash e_4 :(^{\#}5^{\neg}1^{\neg}1^{\neg}1)^{p}$

Tiếp theo ta tính e_5 ; cũng áp dụng các hàm T-ONACID, T-COMMIT và T-SEQ ta có

$6 \vdash e_5 : {}^{-}1^{\#}6^{-}1^{\#}7^{-}1$

Sử dụng T-PREP, chúng ta có kiểu phù hợp với S_4 , tiếp tục áp dụng hàm T- MERGE ta được

$6 \vdash e_{45} : ({}^{\#}5^{-}2^{\#}6^{-}2^{\#}7^{-}2)^{p}$

Với $-3 \vdash e_3 : {}^{+}3$, chúng ta áp dụng T-JC ta được

$3 \vdash e_{345}: {}^{\#}11^{-}2^{\#}7^{-}2$

vì $jc({}^{+}3, {}^{\#}5^{-}2^{\#}6^{-}2^{\#}7^{-}2) = jc(seq({}^{\#}3), seq({}^{\#}(5+3*2)\,{}^{\#}6^{-}2^{\#}7^{-}2))$

$=jc({}^{\#}3, {}^{\#}11^{-}2^{\#}7^{-}2)= {}^{\#}11^{-}2^{\#}7^{-}2;$

Tương tự ta tính được kiểu của e_2 : $3 \vdash e_3 : ({}^{\#}4^{-}1^{-}1)^{p}$. Kiểu của e_3 phù hợp với e_{345} , do vậy ta lại sử dụng hàm T-MERGE và được kiểu của e_{2345}

$3 \vdash e_{2345} : {}^{\#}15^{-}3^{\#}7^{-}3$

Áp dụng T-JC cho e_1 và e_2 ta đươc

$0 \vdash e_{12345} : {}^{\#}24$

vì $jc({}^{+}1^{+}2, {}^{\#}15^{-}3^{\#}7^{-}3) = jc(seq({}^{+}1^{\#}2), seq({}^{\#}21, {}^{\#}7^{-}3) = jc({}^{+}1^{\#}2, {}^{\#}21^{-}3) = jc(seq({}^{\#}3), seq({}^{\#}24)) = {}^{\#}24$

Vậy chương trình này là well-typed và lượng bộ nhớ lớn nhất mà chương trình cần là 24 đơn vị bộ nhớ.

CHƯƠNG 5. XÂY DỰNG CÔNG CỤ VÀ THỰC NGHIỆM

5.1. Giới thiệu ngôn ngữ lập trình/ nền tảng

Ngôn ngữ lập trình được sử dụng để xây dựng công cụ trong luận văn này là ngôn ngữ C#, trên nền .NET. Đây một ngôn ngữ lập trình ứng dụng, ngôn ngữ biên dịch, ngôn ngữ đa năng được phát triển bởi Microsoft. Ngôn ngữ C# là một ngôn ngữ được phát triển từ C, C++ và Java, nhưng nó được tạo từ nền tảng phát triển hơn. C# được thêm vào những đặc tính mới để giúp cho nó uyển chuyển và dễ sử dụng hơn. Nhiều đặc tính trong ngôn ngữ C# khá giống với ngôn ngữ Java. Cụ thể, C# có những đặc tính cơ bản sau :

- Đơn giản, dễ học : Chỉ có khoảng hơn 80 từ khóa và mười mấy kiểu dữ liệu được dựng sẵn
- Gần gũi với các ngôn ngữ lập trình thông dụng như C, C++, Java
- Xây dựng dựa trên nền tảng của những ngôn ngữ lập trình mạnh nên thừa hưởng được các ưu điểm của các ngôn ngữ đó.
- Hướng đối tượng
- Mạnh mẽ và mềm dẻo
- Cung cấp những đặc tính hướng thành phần như property, event…
- C# có bộ Garbage Collector sẽ tự động thu gom vùng nhớ khi không sử dụng nữa.
- Hỗ trợ khái niệm giao diện

Song hành với ngôn ngữ C# là nền tảng .NET Framework. Đây là một nền tảng lập trình và cũng là một nền tảng thực thi ứng dụng chủ yếu trên hệ điều Windows. Nó cũng được phát triển bởi Microsoft và bao gồm 2 thành phần chính:

- CLR: Các chương trình được viết trên nền.NET sẽ được triển khai trong môi trường được gọi là CLR (Common Language Runtime). Môi trường phần mềm này đóng vai trò là một máy ảo cung cấp các dịch vụ như bảo mật, quản lý ngoại lệ hay bộ nhớ.
- Thư viện các lớp: .NET framework gồm nhiều lớp thư viện, những thư viện này sẽ hỗ trợ các lập trình viên trong việc xây dựng giao diện, kết nối cơ sở dữ liệu, giao tiếp mạng…

5.2. Xây dựng công cụ và thực nghiệm

Trong phần này của luận văn, dựa vào các quy tắc kiểu đã được đề cập ở chương 4, tôi sẽ xây dựng thuật toán tính kiểu đề giải quyết bài toán đã nêu trong chương 2. Bước tiếp theo, dựa trên thuật toán có được viết một chương trình bằng ngôn ngữ C# để tính toán cận trên tài nguyên cho một chương trình giao dịch

Trước tiên để xây dựng thuật toán ta quy ước một số ký hiệu được sử dụng để biểu diễn một chuỗi số có dấu khi lập trình như sau:

{+n, −n, #n,! n } (trong đó *n* là số tự nhiên) tương ứng với các thành phần dạng

{ ^{+}n, ^{-}n, $^{\#}n$, $^{\neg}n$ } đã đề cập trong phần 4.1

Chúng ta sẽ chuyển đổi mã của chương trình giao dịch thành một chuỗi gồm các dấu, các số và các dấu đóng mở ngoặc tương ứng với các lệnh sinh một luồng mới và đóng luồng. Cụ thể:

onacid (n): Tương ứng với chuỗi "+n";

commit: Tương ứng với "-1";

spawn: Tương ứng với "( )" – Khởi tạo một luồng

Qua bước chuyển đổi các thành phần trên, ta sẽ kết xuất được một xâu là đầu vào để thực hiện tính toán.

Chương trình được xây dựng bao gồm các phương thức cơ bản sau:

- CalculResourses (string Input): Phương thức thực hiện việc tính cận trên bộ nhớ log của chương trình sử dụng giao dịch. Phương thức này sẽ gọi tới các phương thức Seq, Joint, Merge, JointCommit được trình bày ở bên dưới.
- Seq (string InputString): Phương thức rút gọn một chuỗi gồm dấu và số.
- Joint (string InputString): Phương thức chuyển các dấu "─ " trong chuỗi đã được rút gọn (sau khi thực hiện hàm Seq ở trên) về dấu "⌐".
- Merge (string s1, string s2): Phương thức gộp 2 chuỗi chuẩn tắc.
- JoinCommit(string s1, string s2): Phương thức jointcommit 2 chuỗi chuẩn tắc

Sau đây chúng ta sẽ đi xây dựng thuật toán cụ thể cho các phương thức đã nêu ở trên:

5.2.1. Thuật toán rút gọn (chính tắc hóa) một chuỗi

5.2.1.1. Mô tả thuật toán:

Thuật toán dựa trên các quy tắc được trinh bày trong định nghĩa 6, mục 4.1.

Đầu vào: Một chuỗi số có dấu chưa được chính tắc, cho trước các quy tắc rút gọn như sau:

seq(S) = S khi S là một chuỗi chính tắc;

seq($S^{\#}m^{\#}nS'$) = seq($S^{\#}max(m,n)S'$)

seq($S^{-}m^{-}nS'$) = seq($S^{-}(m^{+}n)S'$)

seq($S^{+}m^{\#}l^{-}nS'$) = seq($S^{\#}$ (m+ 1) $^{-}$ (n - 1)S')

Đầu ra: Chuỗi S đã được rút gọn hay chính tắc

5.2.2. Thuật toán Cộng (Joint)

5.2.2.1. Mô tả thuật toán

Thuật toán dựa trên các quy tắc được trinh bày trong định nghĩa 7, mục 4.1

Đầu vào: Chuỗi có dấu chính tắc S không chứa dấu +, và i là vị trí chứa dấu – đầu tiên trong chuỗi S, i # 0. Hàm join(S) được định nghĩa đệ quy như sau:

$join(S) = S$ nếu i=0;

$join(S) = s_1 \ldots s_{i-1} \neg 1\ join\ (-(|s_i| - 1)\ s_{i+1} \ldots s_k)$ nếu i≠0;

Đầu ra: Chuỗi S chỉ chứa dấu # và ⌐ (trong lập trình dùng ký hiệu "!" thay cho ⌐)

5.2.3. Thuật toán gộp (Merge)

5.2.3.1. Mô tả thuật toán

Thuật toán dựa trên các quy tắc được trinh bày trong định nghĩa 8, mục 4.1

Đầu vào: S_1, S_2 là 2 chuỗi có dấu đã chính tắc, với số phần tử chứa dấu *"¬"* bằng nhau (có thể bằng 0). Hàm *merge* định nghĩa đệ quy như sau:

$Merge\ (S_1, S_2) = {}^{\#}(m_1+m_2)$ khi $S_i = {}^{\#}m_i$, $i = 1, 2$

$Merge\ ({}^{\#} m_1 \neg n_1 S'_1 , {}^{\#} m_2 \neg n_2 S'_2) = {}^{\#} (m_1 + m_2) \neg (n_1 + n_2)\ Merge(S'_1, S'_2)$

Đầu ra: Chuỗi mới được gộp từ 2 chuỗi có dấu chính tắc ban đầu (chỉ chứa # và ¬).

Để kiểm tra tính đúng đắn của thuật toán, ta kiểm tra với các đầu vào khác nhau và được kết quả cho ở bảng sau:

Bảng 5.3 Bảng kết quả kiểm thử hàm gộp

Lượt	Chuỗi 1	Chuỗi 2	Kết quả
1	!1	!1	!2
2	!1	#1 !1	#1!2
3	#2!2	!1	#2!3
4	#3	#3	#6
5	#3!1#2!2	#1!1!1	#4!2#2#2!3

5.2.4. Thuật toán JoinCommit

5.2.4.1.Mô tả thuật toán:

Thuật toán dựa trên các quy tắc được trinh bày trong định nghĩa 10, mục 4.1

Đầu vào: S_1, S_2 là 2 chuỗi chính tắc.

Hàm *joint commit* được định nghĩa như sau:

Jc $({}^{\#}n_1, {}^{\#}l_1) = {}^{\#}\max(n_1, l_1)$

Jc $(S'^{+}_1 n_1 {}^{\#}n_2, {}^{\#}l_1 \neg l_2\ S'_2)$ = jc (Sequence $(S'_1{}^{\#}(n_1+n_2))$, Sequence(${}^{\#}(l_1+1_2*n)\ S'_2$)

Đầu ra: Giá trị cận trên tài nguyên hoặc thông báo chương trình thành lập không hợp lệ

5.2.5. Thuật toán tính cận trên tài nguyên của chương trình giao dịch

5.2.5.1. Mô tả thuật toán

Đầu vào: Chuỗi kết xuất được từ chương trình Featherweight Java có sử dụng giao dịch

Đầu ra: Giá trị cận trên tài nguyên bộ nhớ log của chương trình hoặc thông báo chương trình thành lập không hợp lệ

KẾT LUẬN

Qua thời gian nghiên cứu và tìm hiểu đề tài, luận văn đã được hoàn thành và đạt được những nội dung đề ra với mục tiêu chính là giải quyết bài toán tính cận trên bộ nhớ log cho các chương trình sử dụng giao dịch.

Về lý thuyết, luận văn đã trình bày được các kiến thức cơ sở về hệ thống kiểu nói chung bao gồm định nghĩa hệ thống kiểu, các thuộc tính cơ bản của hệ thống kiểu và ứng dụng của hệ thống kiểu trong thực tế. Ngoài ra, luận văn còn trình bày các khái niệm cơ bản về giao dịch và bộ nhớ giao dịch phần mềm. Tiếp theo, cú pháp và ngữ nghĩa của ngôn ngữ giao dịch TM cũng được giới thiệu trong luận văn. Từ cú pháp và ngữ nghĩa của ngôn ngữ TM, luận văn đã trình bày phương pháp xây dựng hệ thống kiểu để xác định cận trên bộ nhớ log của chương trình sử dụng giao dịch, dựa trên nghiên cứu được các tác giả thực hiện trong bài báo [1]. Một chương trình có giao dịch được cấu thành từ các thành phần cơ bản, mỗi thành phần thể hiện hành vi giao dịch và được định kiểu thông qua một dạng chuỗi số đặc biệt, chuỗi số có dấu. Hệ thống kiểu được trình bày ở đây bao gồm các kiểu, các quy tắc kiểu trong đó chứa định nghĩa các phép toán được sử dụng để định kiểu cho từng thành phần trong chương trình sử dụng giao dịch.

Về thực nghiệm, một công cụ được viết bằng ngôn ngữ C# đã được cài đặt để tính cận trên bộ nhớ log của chương trình sử dụng giao dịch. Chương trình bao gồm các phương thức được xây dựng để thực hiện các phép toán như rút gọn một chuỗi số có dấu, gộp 2 chuỗi số có dấu, Joincommit…Và đặc biệt là phương thức để tính cận trên bộ nhớ log. Chương trình đã được thực nghiệm với nhiều chuỗi được kết xuất từ các chương trình giao dịch khác nhau và cho kết quả tương đối chính xác.

Tuy nhiên, do thời gian có hạn và tài liệu nghiên cứu liên quan chưa nhiều. Hơn nữa, đây là một đề tài khó, đòi hỏi sự đầu tư nhiều về thời gian và công sức nên trong luận văn này không tránh khỏi những hạn chế. Trong quá trình nghiên cứu về đề tài, chúng tôi cũng nhận thấy các kết quả nghiên cứu mới chỉ dừng ở mức độ thực hiện và kiểm chứng về mặt lý thuyết mà chưa hề được kiểm chứng ở thực tế. Do vậy trong tương lai, hi vọng đề tài có thể được nghiên cứu và kiểm chứng ở thực tế. Nếu thành công, các kết quả đạt được này sẽ đóng góp đáng kể vào việc tối ưu các chương trình phần mềm và làm tăng hiệu quả sử dụng tài nguyên bộ nhớ.

TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] Anh-Hoang Truong, Ngoc-Khai Nguyen, Dang Van Hung, and Dang Duc Hanh (2016), "Calculate statically maximum log memory used by multi-threaded transactional programs", *Theoretical Aspects of Computing – ICTAC 2016,* pp. 82-99

[2] Anh-Hoang Truong, Dang Van Hung, Duc-Hanh Dang, and Xuan-Tung Vu, "A type system for counting logs of multi-threaded nested transactional programs", In Nikolaj Bjørner, Sanjiva Prasad, and Laxmi Parida, editors, *Distributed Computing and Internet Technology - 12th International Conference, ICDCIT 2016, Proceedings*, volume 9581 of *LNCS*, pp. 157-168

[3] Hoang Truong (2006), *Type Systems for Guaranteeing Resource Bounds of Component Software*, Dissertation for the degree Philosophiae Doctor (PhD) University of Bergen, Norway

[4] Igarashi, Atsushi, Pierce Benjamin C, Wadler, Philip (2001), "Featherweight Java: a minimal core calculus for Java and GJ", *Journal ACM Transactions on Programming Languages and Systems,* Volume 23, pp. 396- 450

[5] Jiang Hui,Lin Dong, Zhang Xingyuan, Xie Xiren(2001), "Type System in Programming Languages", *Journal of Computer Science and Technology*, Volume 16, pp. 286-292

[6] Luca Cardelli (1996), "Type system", *ACM Computing Surveys (CSUR),*Volume 28 Issuse1, pp. 263-264

[7] Martin Steffen and Thi Mai Thuong Tran (2009), "Safe commits for Transactional Featherweight Java", *Integrated Formal Methods*, pp. 290-304

[8] N.Shavit, and D.Touitou (1995), "Software Transational Memory", Proceeding PODC *'95 Proceedings of the fourteenth annual ACM symposium on Principles distributed computing*, pp 204-213

[9] Thi Mai Thuong Tran, O.Owe, and Martin Steffen (2010), "Safe typing for transactional vs. lock-based concurrency in multi-threaded Java", *KSE '10 Proceedings of the 2010 Second International Conference on Knowledge and Systems Engineering*, pp.188-193

[10] Thi Mai Thuong Tran, Martin Steffen, and Hoang Truong (2011), "Estimating Resource Bounds for Software Transactions", *SEFM 2013 Proceedings of the 11th International Conference on Software Engineering and Formal Methods*, Volume 8137, pp. 212-228

[11] Xuan-Tung Vu, Thi Mai Thuong Tran, Anh-Hoang Truong, and Martin Steffen. "A type system for finding upper resource bounds of multi-threaded programs with nested transactions", In *Symposium on Information and Communication Technologys 2012, SoICT '12, Halong City, Quang Ninh, Viet Nam, August 23-24, 2012*, pp. 21-30