ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN NGỌC KHẢI

HỆ THỐNG KIỂU ĐỂ ƯỚC LƯỢNG TĨNH TÀI NGUYÊN SỬ DỤNG CỦA CHƯƠNG TRÌNH GIAO DỊCH

TÓM TẮT LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

Hà Nội - 2021

Công trình được hoàn thành tại: Trường Đại học Công Nghệ, Đại học Quốc gia Hà Nội

Người hướng dẫn khoa học: PGS. TS. Trương Anh Hoàng

Phản biện: TS. Nguyễn Hữu Đức

Phản biện: TS. Đỗ Thị Bích Ngọc

Luận án sẽ được bảo vệ trước Hội đồng cấp Đại học Quốc gia chấm luận án tiến sĩ tại phòng 212, nhà E3, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội vào hồi ... giờ ... ngày ... tháng ... năm 2021

Có thể tìm hiểu luận án tại:

- Thư viện Quốc gia Việt Nam

- Trung tâm Thông tin - Thư viện, Đại học Quốc gia Hà Nội

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN NGỌC KHẢI

HỆ THỐNG KIỂU ĐỂ ƯỚC LƯỢNG TĨNH TÀI NGUYÊN SỬ DỤNG CỦA CHƯƠNG TRÌNH GIAO DỊCH

Chuyên ngành: Kỹ thuật phần mềm
Mã số: 9480103.01

TÓM TẮT LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS. TS. TRƯƠNG ANH HOÀNG

Hà Nội - 2021

Mục lục

Chương 1

Giới thiệu

1.1 Đặt vấn đề

Ngày nay, những thiết bị được trang bị bộ vi xử lý đa lõi trở nên phổ biến, vì vậy việc phát triển những phần mềm tương tranh để khai thác hiệu quả các hệ thống phần cứng này cũng đang rất phát triển. Tuy nhiên, việc phát triển các phần mềm tương tranh đòi hỏi nhiều kỹ thuật phức tạp hơn so với phát triển các phần mềm hoạt động theo cơ chế tuần tự, trong đó việc đồng bộ giữa các luồng là một kỹ thuật quan trọng.

Thông thường người lập trình thường sử dụng cơ chế dựa trên khóa (lock based) để đồng bộ các luồng. Tuy nhiên, cơ chế đồng bộ dựa trên khóa có thể gây ra nhiều lỗi nghiêm trọng như lỗi khóa sống (livelock), lỗi khóa chết (deadlock), và những lỗi tiềm ẩn khác về khóa. Ngoài ra, người lập trình còn phải quan tâm tới việc quản lý các khóa. Điều này gây ra nhiều khó khăn, phức tạp cho người lập trình.

Để giúp người lập trình viết chương trình đơn giản hơn, các nhà nghiên cứu đã đưa ra các giải pháp thay thế cho cơ chế dựa trên khóa, trong đó cơ chế bộ nhớ giao dịch phần mềm (Software Transactional Memory - STM) là một giải pháp nhiều hứa hẹn. Gần đây, cơ chế này đã được trang bị cho một số ngôn ngữ lập trình tương tranh hiện đại như ngôn ngữ Scala, Haskell, Java, C♯,...

Điểm nổi bật trong các ngôn ngữ này đó là được trang bị những tính năng giao dịch đa luồng và lồng nhau. Đây là những tính năng tiên tiến, tạo nhiều thuận lợi cho người lập trình. Giao dịch đa luồng có nghĩa là ở trong một giao dịch chúng ta có thể khởi tạo nhiều luồng. Giao dịch lồng nhau ở đây đặc biệt ở chỗ một giao dịch có thể chứa một hoặc nhiều giao dịch ở trong một luồng con, và luồng con phải đồng bộ với giao dịch của luồng cha khi kết thúc giao dịch. Khi một giao dịch commit, các luồng con ở trong giao dịch đó cũng phải cùng commit.

Để các luồng chạy được độc lập, không phải chờ đợi nhau, mỗi luồng/giao dịch sẽ đọc hoặc ghi trên các bản sao của các bộ nhớ dùng chung, gọi là các log. Sau khi hoàn thành giao dịch, chúng sẽ đồng bộ các log và các giá trị gốc của nó. Nếu không có xung đột, giao dịch sẽ được hoàn thành, ngược lại giao dịch sẽ phải thực hiện lại (rollback), hoặc bị hủy bỏ (abort). Các log này là một trong những yếu tố làm cho chương trình STM tiêu thụ nhiều tài nguyên bộ nhớ hơn các chương trình truyền thống khác. Điều này có thể dẫn đến các lỗi thiếu tài nguyên bộ nhớ dành cho chương trình.

Mục đích nghiên cứu của chúng tôi là đưa ra được phương pháp để giúp người lập trình ước lượng tĩnh tài nguyên tối đa cần sử dụng của các chương trình giao dịch đa luồng sử dụng cơ chế STM. Từ đó, người lập trình có thể tối ưu chương trình của mình để sử dụng tài nguyên hiệu quả hơn, nhằm tiết kiệm tài nguyên, hạn chế các lỗi do thiếu tài nguyên khi thực thi chương trình.

Một ứng dụng khác đó là dựa trên số lượng log được tạo ra đồng thời chúng ta có thể suy đoán được mức độ hiệu quả của chương trình. Nếu quá nhiều logs được tạo ra đồng thời thì khả năng xung đột sẽ cao hơn và nhiều luồng hoặc giao dịch sẽ phải chạy lại hoặc bị hủy bỏ.

Do sự phức tạp của bài toán đặt ra nên chúng tôi chia bài toán thành nhiều bước để giải quyết. Đầu tiên, chúng tôi giải quyết bài toán xác định số lượng log lớn nhất cùng tồn tại trong một thời điểm. Từ đây chúng ta đã có thông tin sơ bộ về bộ nhớ log sử dụng. Làm mịn thêm kích thước của các log này dựa trên thông tin về số lượng và kiểu của các biến hoặc các đối tượng trên các log, chúng ta có được thông tin cụ thể về bộ nhớ sử dụng của chương trình.

Với nhiều ưu điểm vượt trội như đã phân tích ở trên, cơ chế STM sẽ là một lựa chọn tốt cho người lập trình. Tuy nhiên, cơ chế bộ nhớ giao dịch cũng có những hạn chế, đó là việc tiêu thụ nhiều tài nguyên, nguy cơ phải thực hiện lại hoặc hủy bỏ giao dịch cao làm giảm hiệu suất của chương trình. Vì vậy nếu không kiểm soát tốt số log chương trình tạo ra và tài nguyên chúng sử dụng thì người lập trình không sẵn sàng để sử dụng chúng.

1.2 Những đóng góp chính, ý nghĩa khoa học và thực tiễn

Những đóng góp chính của luận án bao gồm hai hệ thống kiểu với mục đích ước lượng tĩnh biên tài nguyên cần sử dụng của các chương trình STM:

- Hệ thống kiểu thứ nhất được xây dựng với các quy tắc đơn giản hơn, thuận lợi để mở rộng và cài đặt. Hệ thống kiểu này yêu cầu một chương trình hoàn thiện (nghĩa là một chương trình đã được viết xong và hợp lệ) thì mới có thể định kiểu được. Chúng phù hợp để định kiểu cho các chương trình nhỏ, được viết bởi số ít người.

- Hệ thống kiểu thứ hai có khả năng định kiểu linh hoạt hơn. Chúng có thể định kiểu cho những thành phần bất kỳ trong chương trình mà không cần phải là một chương trình hoàn chỉnh, sau đó tích hợp lại để được kiểu của toàn bộ chương trình. Tính chất này được gọi là tính tích hợp, một trong những đặc điểm thường thấy của các hệ thống kiểu. Hệ thống kiểu này phù hợp để định kiểu cho những chương trình lớn, được viết bởi nhiều người.

Ngoài những đóng góp chính ở trên, luận án đã đưa ra một số cải tiến cho cơ chế STM để sử dụng bộ nhớ giao dịch hiệu quả hơn. Đồng thời, luận án cũng đưa ra thuật toán suy diễn kiểu để định kiểu cho chương trình. Thuật toán đã được cài đặt bằng phương pháp lập trình hàm với ngôn ngữ lập trình F♯.

Ý nghĩa của luận án không chỉ ở việc giải quyết bài toán ước lượng tài nguyên bộ nhớ của chương trình, mà luận án còn có ý nghĩa trong việc góp phần phát triển lĩnh vực nghiên cứu về lý thuyết kiểu trong phương pháp hình thức trong công nghệ phần mềm. Cụ thể, luận án đã đóng góp cho cộng đồng nghiên cứu những bộ quy tắc kiểu với mục đích chính để giúp người lập trình kiểm soát được tài nguyên chương trình cần sử dụng. Bộ quy tắc này là nền tảng để chúng ta có thể mở rộng, phát triển cho các mục đích khác chẳng hạn như xác định tài nguyên CPU, độ phức tạp thời gian, số gas

tiêu thụ trong các hợp đồng thông minh trong Ethereum ... Ngoài ra, luận án cũng góp phần thay đổi tư duy của người lập trình trong việc tối ưu thuật toán để sử dụng tiết kiệm tài nguyên trong quá trình lập trình.

Những nghiên cứu hiện tại chủ yếu giải quyết cho chương trình tuần tự hoặc song song nhưng các luồng hoạt động độc lập. Chúng chỉ là những trường hợp cụ thể trong bài toán của chúng tôi. Những nghiên cứu cho các chương trình đa luồng sử dụng cơ chế STM chủ yếu được thực hiện theo phương pháp động, nghĩa là thực thi chương trình trong những ngôn ngữ cụ thể để thực hiện việc so sánh, đánh giá. Những phương pháp này chỉ cho kết quả gần đúng và yêu cầu phải có chương trình hoàn thiện để thực hiện thử nghiệm.

Hệ thống kiểu chúng tôi đang xây dựng và phát triển có cấu trúc giàu thông tin và có thể áp dụng được cho nhiều chương trình tương tranh có cách thức đồng bộ tương tự. Giải pháp sử dụng hệ thống kiểu của chúng tôi cũng giúp việc xác định tài nguyên đảm bảo tính đúng mà không cần chạy chương trình.

Tính thời sự và sự cần thiết của nghiên cứu này có thể thấy rõ trong ngữ cảnh phát triển phần mềm trên các thiết bị đa nhân ngày nay. Ngày càng có nhiều thiết bị sử dụng bộ xử lý đa nhân. Việc viết chương trình đa luồng để khai thác các thiết bị này đang là thực tế đặt ra. Cơ chế STM là một cơ chế mạnh, giúp người lập trình viết các chương trình giao dịch đa luồng đơn giản hơn. Tuy nhiên, như đã trình bày ở trên, với cơ chế này người lập trình chỉ yên tâm khi kiểm soát được tài nguyên sử dụng và số log tối đa được tạo ra bởi chương trình. Đặc biệt, điều này rất quan trọng đối với lập trình nhúng, lập trình cho các thiết bị IOT, khi mà tài nguyên bộ nhớ hạn chế, cũng như sự tiêu tốn tài nguyên liên quan trực tiếp đến tuổi thọ của pin. Vì vậy, kết quả của luận án là một giải pháp giúp cơ chế bộ nhớ giao dịch trở nên thực tế và hiệu quả hơn.

1.3 Bố cục của luận án

Luận án được bố cục bao gồm 6 chương: Chương 1 giới thiệu tổng quan về bài toán đặt ra, những đóng góp chính của luận án; Chương 2 trình bày những kiến thức cơ bản, nền tảng, và những nghiên cứu liên quan; Chương 3 xây dựng ngôn ngữ giao dịch tối giản và hệ thống kiểu để ước lượng biên tài nguyên tiêu thụ của chương trình; Chương 4 mở rộng ngôn ngữ của chương 3 với việc bổ sung các lệnh cơ bản đối với ngôn ngữ lập trình hướng cấu trúc. Từ đó, chúng tôi xây dựng một hệ thống kiểu để xác định biên bộ nhớ cấp phát cho các biến dùng chung. Dựa trên các quy tắc kiểu này, chúng tôi đã cài đặt một công cụ để suy ra kiểu của chương trình. Tiếp theo, chúng tôi cải tiến hệ thống kiểu để được một hệ thống kiểu có khả năng định kiểu linh hoạt hơn (gọi là hệ thống kiểu tích hợp); Chương 5 tiếp tục mở rộng ngôn ngữ đã xây dựng ở chương 3 với cấu trúc hướng đối tượng, đồng thời cải tiến cơ chế STM để chương trình sử dụng bộ nhớ hiệu quả hơn. Tài nguyên tiêu thụ trong chương này được xác định là bộ nhớ cấp phát cho các đối tượng trong các giao dịch. Cùng với đó, chúng tôi xây dựng một hệ thống kiểu để xác định biên bộ nhớ giao dịch cho chương trình; Chương 6 luận án tổng kết lại những vấn đề luận án đã giải quyết được, những vấn đề còn tồn tại cần tiếp tục nghiên cứu để có thể áp dụng kết quả vào thực tế một cách thuận tiện hơn.

Chương 2

Kiến thức nền tảng và nghiên cứu liên quan

Trong chương này, luận án trình bày những khái niệm, nguyên tắc cơ bản của lập trình tương tranh, cơ chế STM và hệ thống kiểu. Đây là những khối kiến thức cơ bản liên quan trực tiếp đến bài toán đặt ra của luận án và giải pháp để giải quyết chúng. Phần cuối chương, luận án trình bày về những hướng nghiên cứu liên quan tới bài toán và hướng tiếp cận để giải quyết bài toán của luận án.

2.1 Cơ chế bộ nhớ giao dịch phần mềm

Để đồng bộ giữa các luồng, người lập trình thường sử dụng cơ chế đồng bộ dựa trên khóa. Tuy nhiên, cơ chế này tiềm ẩn những lỗi nghiêm trọng về khóa như lỗi khóa sống, lỗi khóa chết, và người lập trình phải quan tâm xử lý các khóa.

Để giải quyết vấn đề về khóa và giúp người lập trình viết chương trình đơn giản hơn, một số giải pháp thay thế cho cơ chế này đã được đề xuất, trong đó cơ chế STM là một giải pháp nhiều hứa hẹn.

Trong khoa học máy tính, STM là một cơ chế điều khiển tương tranh tương tự như các giao dịch trong cơ sở dữ liệu để kiểm soát quyền truy cập vào bộ nhớ dùng chung trong các tính toán diễn ra một cách đồng thời. Chúng là một giải pháp thay thế cho cơ chế đồng bộ dựa trên khóa.

Từ năm 2005, STM đã trở thành trọng tâm của các nghiên cứu mạnh mẽ và hỗ trợ cho việc triển khai thực tế ngày càng tăng. Chúng đã được triển khai trong nhiều ngôn ngữ lập trình thực tế hiện đại như Haskell, Java, C♯, Scala,...

2.1.1 Đặc điểm của chương trình STM

Một chương trình STM được cấu tạo bởi các luồng hoạt động song song nhưng không độc lập. Từ một luồng (luồng cha) chúng có thể sinh ra các luồng khác (luồng con). Trong các luồng có các giao dịch, và trong các giao dịch có thể sinh ra các luồng.

Mỗi giao dịch có một vùng bộ nhớ cục bộ riêng biệt cho từng luồng, được gọi là *log*, để lưu trữ các đối tượng dùng chung (shared objects) để các luồng truy cập độc lập các bản sao của đối tượng dùng chung trong quá trình thực hiện. Mỗi luồng có thể tạo một số giao dịch lồng nhau nên sẽ có tương ứng một số lượng log được tạo ra cho mỗi giao dịch. Hơn nữa, một luồng con khi tạo ra trong một giao dịch cũng sẽ có các bản sao như cha của nó. Khi đồng kết thúc, các log được so sánh với nhau và nếu không có mâu thuẫn trong việc cập nhật các đối tượng thì giao dịch được hoàn tất. Ngược lại nếu có mâu thuẫn trong việc cập nhật các đối tượng của các luồng tham gia vào quá trình đồng kết thúc thì giao dịch có thể được thực hiện lại (rollback) hoặc hủy bỏ (abort). Tại các

điểm đồng kết thúc, các log được giải phóng, tức là tài nguyên bộ nhớ tương ứng được giải phóng.

Đối với một số ngôn ngữ lập trình thực tế, người lập trình không phải quan tâm về việc đồng bộ giữa các luồng, các giao dịch mà các trình biên dịch sẽ tự động thực thi công việc này. Đặc điểm này gọi là đồng bộ ngầm. Chúng tạo thuận tiện cho người lập trình, tuy nhiên khi phân tích chương trình, chúng ta cần phải phân tích ở mức ngữ nghĩa của ngôn ngữ thay vì ở mức mã nguồn của ngôn ngữ.

Những đặc điểm trên làm cho chương trình STM trở nên đơn giản, rõ ràng hơn, thuận lợi cho người lập trình và công tác bảo trì sau này. Người lập trình có thể tạo ra các chương trình với các giao dịch và các luồng đan xen lồng nhau một cách linh hoạt, đáp ứng được nhu cầu của các bài toán thực tế. Tuy nhiên, cũng vì có các giao dịch lồng nhau phức tạp, các luồng chạy song song nhưng không độc lập, các nút đồng bộ ngầm ta không nhìn thấy khi phân tích ở mức mã nguồn, tạo nên những khó khăn thách thức khi ta xây dựng các hệ thống kiểu để tổng quát chương trình một cách chính xác.

Mặc dù cơ chế STM có nhiều ưu điểm, tuy nhiên do chúng tiêu tốn nhiều tài nguyên bộ nhớ hơn các chương trình truyền thống nên người lập trình chỉ yên tâm khi kiểm soát được tài nguyên chương trình cần sử dụng.

2.2 Hệ thống kiểu

Trong lĩnh vực phương pháp hình thức, hệ thống kiểu đã được quan tâm nghiên cứu từ lâu bởi nhiều nhà khoa học. Sau đây chúng tôi trình bày về một số khái niệm cơ bản và những đặc điểm, những ứng dụng phổ biến của hệ thống kiểu.

2.2.1 Khái niệm hệ thống kiểu

Hiện nay, nhiều tác giả đã đưa ra những khái niệm về hệ thống kiểu dưới các góc độ khác nhau, trong đó một số khái niệm đã được công nhận rộng rãi:

- Hệ thống kiểu là một phương pháp kiểm soát cú pháp của chương trình để cho thấy sự vắng mặt hành vi nào đó của chương trình bằng cách phân loại các thành phần theo kiểu cho các giá trị nó tính toán.

- Hệ thống kiểu là một phương pháp chính thức ngăn chặn các lỗi trong quá trình thực thi chương trình.

- Hệ thống kiểu là một tập các chú thích kiểu và cơ chế kiểm tra trợ giúp cho người lập trình; với người viết trình dịch, một hệ thống kiểu là nguồn thông tin có thể được sử dụng để tối ưu mã máy sinh ra; theo lý thuyết ngôn ngữ, một hệ thống kiểu là một bộ quy tắc để quy định cấu trúc và lập luận về ngôn ngữ.

Như vậy, ta có thể khái quát hệ thống kiểu là một cơ chế cú pháp ràng buộc cấu trúc của một 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.

2.2.2 Ứng dụng của hệ thống kiểu

Hệ thống kiểu có một vai trò quan trọng trong phát triển phần mềm, trong đó có hai hướng ứng dụng chính của hệ thống kiểu:

- Thứ nhất, nó được sử dụng như một công cụ phân tích hình thức cho một ngôn ngữ lập trình. Hệ thống kiểu cung cấp một mô hình toán học hình thức để mô tả và phân tích ngữ nghĩa của một ngôn ngữ;

- Thứ hai, nó được sử dụng như một phương pháp hình thức để phát triển một ngôn ngữ một cách chặt chẽ và chính xác. Hệ thống kiểu được định nghĩa hợp lệ là một định nghĩa cụ thể chặt chẽ của một ngôn ngữ. Nó là điều kiện tiên quyết để sử dụng và triển khai một ngôn ngữ. Ngày nay, hệ thống kiểu là một thành phần cốt lõi của nhiều ngôn ngữ mới.

Đặc trưng của một ngôn ngữ được thiết kế trên hệ thống kiểu đòi hỏi mỗi biến trong chương trình phải có miền giá trị cụ thể mà nó có thể tham chiếu. Mỗi khai báo hàm cũng phải thao tác trên các đối số cụ thể và trả về giá trị của kiểu cụ thể. Thông tin này có thể được khai báo rõ ràng bằng cách sử dụng các chú thích kiểu hoặc suy ra bởi các bộ kiểm tra kiểu. Khi một chương trình được biên dịch, bộ kiểm tra kiểu này đảm bảo mỗi biến chỉ được gán đúng kiểu của giá trị và các hàm chỉ được gọi với các tham số đúng kiểu.

Các ngôn ngữ hạn chế phạm vi của các biến được gọi là ngôn ngữ kiểu, ngược lại gọi là ngôn ngữ phi kiểu. Một hệ thống kiểu là thành phần của một ngôn ngữ kiểu, và nó có vai trò theo dõi các kiểu của các biến và các kiểu của tất cả các biểu thức trong chương trình.

Trong một ngôn ngữ mà có hệ thống kiểu giữ cho tất cả các chương trình chạy được trên ngôn ngữ lập trình cụ thể ta gọi là ngôn ngữ có kiểu tốt. Nó chỉ ra rằng, một phân tích cẩn thận là cần thiết để tránh những tuyên bố sai cho các ngôn ngữ lập trình. Một hệ thống kiểu được sử dụng để xác định liệu chương trình có được coi là có hành vi đúng (well-behaved) không?

Khi phát triển một cách đúng, các hệ thống kiểu cung cấp các công cụ để đánh giá một cách đầy đủ các khía cạnh quan trọng của các định nghĩa ngôn ngữ. Mô tả ngôn ngữ thường không xác định cấu trúc của một ngôn ngữ đầy đủ chi tiết để cho phép thực thi rõ ràng. Thường xảy ra với các trình biên dịch khác nhau cho cùng một ngôn ngữ khi cài đặt các hệ thống kiểu khác nhau. Nhiều định nghĩa ngôn ngữ đã được tìm thấy là kiểu không đúng, theo đó một chương trình có thể sụp đổ mặc dù các đánh giá được chấp nhận bởi bộ kiểm tra kiểu. Lý tưởng nhất hệ thống kiểu chính thức nên là một phần định nghĩa của tất cả các ngôn ngữ lập trình được định kiểu. Bằng cách này các thuật toán kiểm tra kiểu có thể được đo đạc một cách rõ ràng dựa vào các thông số kỹ thuật chính xác nếu tất cả có thể và khả thi, ngôn ngữ có thể được thể hiện là kiểu đúng.

Trong ngôn ngữ lập trình, một hệ thống kiểu là một tập hợp các quy tắc gán cho một thuộc tính được gọi là kiểu để xây dựng một chương trình máy tính bao gồm như các biến, biểu thức, hàm hoặc các mô đun. Mục đích chính của một hệ thống kiểu là để

giảm thiểu khả năng xảy ra lỗi trong các chương trình máy tính, bằng cách định nghĩa các giao diện giữa các phần khác nhau của một chương trình máy tính, và sau đó kiểm tra các phần đã được kết nối với nhau một cách nhất quán chưa. Việc kiểm tra này có thể xảy ra tĩnh (tại thời gian biên dịch), động (tại thời gian chạy), hoặc là một sự kết hợp của kiểm tra tĩnh và động.

Ngoài ra, một hệ thống kiểu còn có các mục đích khác, chẳng hạn như cho phép tối ưu hóa trình biên dịch, cung cấp một hình thức tài liệu cho người lập trình. Một hệ thống kiểu là thành phần của một ngôn ngữ kiểu, có vai trò theo dõi các kiểu của các biến và các kiểu của tất cả các biểu thức trong chương trình. Trong một ngôn ngữ có hệ thống kiểu, mà tất cả chương trình viết đúng kiểu đều chạy một cách hoàn toàn xác định, thì chúng ta nói rằng ngôn ngữ là kiểu tốt (well typed).

Khi phát triển một cách đúng, các hệ thống kiểu cung cấp các công cụ khái niệm để đánh giá một cách đầy đủ các khía cạnh quan trọng của các định nghĩa ngôn ngữ. Mô tả ngôn ngữ hình thức thường không xác định cấu trúc của một ngôn ngữ đầy đủ chi tiết để cho phép thực thi rõ ràng. Thường xảy ra với các trình biên dịch khác nhau cho cùng một ngôn ngữ cài đặt các hệ thống kiểu khác nhau. Hơn nữa, nhiều định nghĩa ngôn ngữ đã được tìm thấy là kiểu không đúng đắn, theo đó một chương trình có thể sụp đổ mặc dù các đánh giá được chấp nhận bởi bộ kiểm tra kiểu. Lý tưởng nhất, hệ thống kiểu hình thức nên là một phần định nghĩa tất cả các ngôn ngữ lập trình được định kiểu. Bằng cách này, các thuật toán kiểm tra kiểu có thể được đo đạc một cách rõ ràng dựa vào các thông số kỹ thuật chính xác nếu tất cả có thể và khả thi, ngôn ngữ có thể được thể hiện là kiểu đúng đắn.

2.3 Nghiên cứu liên quan

Ước lượng tài nguyên sử dụng của các chương trình phần mềm đã được nghiên cứu trong nhiều ngữ cảnh khác nhau, trong đó các tác giả chủ yếu tập trung phân tích các chương trình xây dựng theo phương pháp lập trình tuần tự hoặc lập trình tương tranh nhưng các luồng chạy độc lập. Những nghiên cứu về sự tiêu thụ tài nguyên của các chương trình tương tranh sử dụng cơ chế STM còn ít và chủ yếu sử dụng các phương pháp phân tích động cho những ngôn ngữ, những thư viện cụ thể. Trong phần này luận án trình bày về những nghiên cứu có mục đích nghiên cứu hoặc phương pháp tiếp cận gần với phương pháp của luận án, trong đó tập trung vào ba hướng nghiên cứu chính đó là những nghiên cứu liên quan tới ước lượng tài nguyên tiêu thụ trong chương trình tuần tự, ước lượng tài nguyên tiêu thụ trong các chương trình tương tranh, và ước lượng tài nguyên tiêu thụ trong các chương trình STM.

2.4 Tổng kết chương

Trong chương này luận án đã trình bày những vấn đề cơ bản về cơ chế điều khiển tương tranh dựa trên khóa, cơ chế STM, hệ thống kiểu, và những nghiên cứu liên quan đến bài toán đặt ra. Đây là những kiến thức cơ bản để làm rõ bài toán đặt ra và là nền tảng để xây dựng những ngôn ngữ và hệ thống kiểu trong các chương tiếp theo.

Chương 3

Ước lượng biên tài nguyên chương trình của ngôn ngữ tối giản

Trong chương này, chúng tôi đưa ra một ngôn ngữ giao dịch với các câu lệnh cốt lõi nhất đặc trưng cho cơ chế STM và liên quan đến cơ chế tiêu thụ tài nguyên của chương trình (gọi là ngôn ngữ giao dịch tối giản). Từ đó, chúng tôi xây dựng một hệ thống kiểu để xác định số giao dịch tối đa của chương trình viết bởi ngôn ngữ này. Tiếp theo, chúng tôi cải tiến ngôn ngữ bằng cách bổ sung thêm mỗi giao dịch một tham số đại diện cho lượng tài nguyên tiêu thụ của giao dịch. Cùng với đó, chúng tôi cải tiến hệ thống kiểu để xác định được biên tài nguyên tiêu thụ của chương trình dựa trên các tham số trên. Mục đích của việc chia nhỏ bài toán để giải quyết từng bước nhằm đơn giản hóa bài toán, đồng thời giải quyết bài toán một cách tổng quát hơn.

3.1 Xác định số giao dịch tối đa

Mục đích của phần này nhằm xây dựng một hệ thống kiểu để xác định số giao dịch tối đa cùng tồn tại của một chương trình STM. Hệ thống kiểu này cần có các quy tắc kiểu đơn giản thuận lợi để chứng minh tính đúng. Kết quả trong phần này sẽ làm nền tảng để các nghiên cứu sau dễ dàng mở rộng, phát triển một cách thuận lợi, chính xác.

Để đảm bảo tính tổng quát và đơn giản hóa bài toán, chúng tôi không thực hiện trên một ngôn ngữ cụ thể, đầy đủ. Thay vào đó, chúng tôi đưa ra một ngôn ngữ lõi hoạt động theo cơ chế STM. Sau đó, chúng tôi phân tích các chương trình trên ngôn ngữ này để xây dựng một hệ thống kiểu với mục đích nêu trên.

3.1.1 Ngôn ngữ giao dịch tối giản

Trong phần này luận án đưa ra một ngôn ngữ giao dịch tối giản. Các tính năng của ngôn ngữ được tổng quát hóa vì vậy các chương trình giao dịch viết bởi các ngôn ngữ đầy đủ khác có thể được dịch sang ngôn ngữ của chúng tôi để tính toán.

Đặc điểm chính của ngôn ngữ này là cho phép người lập trình tạo ra các luồng, các giao dịch đan xen lẫn nhau. Khi một giao dịch được lồng trong một giao dịch khác, chúng ta gọi giao dịch ngoài là giao dịch cha, giao dịch trong là giao dịch con. Một giao dịch con phải kết thúc trước khi giao dịch cha của nó kết thúc. Khi một giao dịch được tạo ra, một vùng bộ nhớ gọi là *log* được cấp phát để lưu trữ các biến, các đối tượng dùng chung. Một giao dịch được khởi tạo mà chưa đóng thì được gọi là giao dịch dịch đang mở. Bên trong một giao dịch đang mở, chương trình có thể sinh ra các luồng mới.

3.1.1.1 Cú pháp

Cú pháp của ngôn ngữ được thể hiện trong Bảng 3.1. Dòng thứ nhất để biểu diễn các luồng/tiến trình. Dòng thứ 2 xác định các lệnh cơ bản tạo thành chương trình. e có thể là e_1 nối tiếp bởi e_2 hoặc nhận một trong hai giá trị e_1 hoặc e_2. Dòng sau cùng là ba lệnh để tạo một luồng, bắt đầu và kết thúc một giao dịch. Hai lệnh cuối này chính là các lệnh sử dụng (cấp phát) và giải phóng tài nguyên.

Bảng 3.1: Cú pháp của ngôn ngữ STM tối giản

1.	P	::=	$\mathbf{0} \mid P \parallel P \mid \text{p(e)}$
2.	e	::=	$e_1; e_2 \mid e_1 + e_2 \mid$
3.			$\mathbf{spawn}(e) \mid \mathbf{onacid} \mid \mathbf{commit}$

3.1.1.2 Ngữ nghĩa

Ngữ nghĩa của ngôn ngữ STM tối giản được thể hiện bởi hai tập quy tắc hoạt động: tập quy tắc cho ngữ nghĩa mệnh lệnh hay ngữ nghĩa cục bộ trong một luồng (Bảng 3.2) và tập quy tắc cho ngữ nghĩa giao dịch và đa luồng (Bảng 3.3).

Môi trường lúc chạy (toàn cục) là một tập hợp các luồng. Mỗi luồng có một môi trường cục bộ là một chuỗi các log. Để quản lý các luồng và log này chúng ta gán cho mỗi luồng và mỗi log một định danh. Phần tiếp theo chúng tôi sẽ định nghĩa một cách hình thức về môi trường cục bộ và môi trường toàn cục.

a. Ngữ nghĩa mệnh lệnh

Ngữ nghĩa mệnh lệnh là các quy tắc chuẩn tương tự như các ngôn ngữ thông dụng khác. Ở đây, chúng tôi chỉ trình bày những điểm chính liên quan đến việc tạo và hủy các log của các giao dịch và các luồng.

Để mô tả sự thay đổi trạng thái của chương trình sau khi thực thi các câu lệnh, sau đây chúng tôi đưa ra định nghĩa về môi trường cục bộ của một luồng và môi trường toàn cục của chương trình.

Định nghĩa 3.1 (Môi trường cục bộ) *Một môi trường cục bộ E là một dãy hữu hạn các log được gắn nhãn $l_1 : log_1; \ldots; l_k : log_k$, trong đó phần tử thứ i của dãy bao gồm một nhãn l_i (một tên giao dịch) và một* log *log_i.*

Cho $E = l_1 : log_1; \ldots; l_k : log_k$, ta gọi k là kích cỡ của E, và được ký hiệu là $|E|$. Kích cỡ $|E|$ là độ sâu của các giao dịch lồng nhau (l_1 là giao dịch ngoài cùng, l_k là giao dịch trong cùng). Môi trường rỗng/trống ký hiệu là ϵ.

Bảng 3.2: Ngữ nghĩa cục bộ của ngôn ngữ STM tối giản

$E, (e_1 + e_2); e \to E, (e_1; e)$	L-COND$_1$
$E, (e_1 + e_2); e \to E, (e_2; e)$	L-COND$_2$

Ngữ nghĩa ở mức cục bộ được biểu diễn dạng $E, e \to E', e'$. Trong đó E là môi trường cục bộ ở trạng thái ban đầu, E' là môi trường cục bộ ở trạng thái sau khi thực thi câu lệnh; e là biểu thức được đánh giá trong môi trường E, e' là biểu thức thay đổi từ biểu thức e sau khi thực thi câu lệnh (được đánh giá trong các môi trường E').

Hai quy tắc L-COND$_1$ và L-COND$_2$ để thực hiện các câu lệnh điều kiện. Chúng được hiểu như việc lựa chọn thực thi một trong hai biểu thức e_1 hoặc e_2 tùy thuộc vào giá trị các biểu thức điều kiện.

b. Ngữ nghĩa giao dịch và đa luồng

Các quy tắc cho các ngữ nghĩa giao dịch và đa luồng được thể hiện dưới dạng $\Gamma, P \Rightarrow \Gamma', P'$ hoặc $\Gamma, P \Rightarrow error$, trong đó Γ, P thể hiện trạng thái ban đầu của chương trình, Γ', P' thể hiện trạng thái sau khi thực thi một câu lệnh của chương trình. Γ là một môi trường toàn cục và P là một tập các luồng/tiến trình có dạng $p(e)$. Một môi trường toàn cục bao gồm các môi trường cục bộ của các luồng và được định nghĩa như sau.

Định nghĩa 3.2 (Môi trường toàn cục) *Một môi trường toàn cục Γ là một ánh xạ xác định từ định danh của luồng tới môi trường cục bộ của nó, $\Gamma = p_1 : E_1, \ldots, p_k : E_k$, trong đó p_i là một định danh của luồng và E_i là môi trường cục bộ của luồng p_i.*

Bảng 3.3: Ngữ nghĩa giao dịch và luồng của ngôn ngữ STM tối giản

$$\frac{E, e \to E', e' \quad p : E \in \Gamma \quad reflect(p, E', \Gamma) = \Gamma'}{\Gamma, P \parallel p(e) \Rightarrow \Gamma', P \parallel p(e')} \text{ G-PLAIN}$$

$$\frac{p' \; fresh \quad spawn(p, p', \Gamma) = \Gamma'}{\Gamma, P \parallel p(\mathbf{spawn}(e_1); e_2) \Rightarrow \Gamma', P \parallel p(e_2) \parallel p'(e_1)} \text{ G-SPAWN} \qquad \frac{l \; fresh \quad start(l, p, \Gamma) = \Gamma'}{\Gamma, P \parallel p(\mathbf{onacid}; e) \Rightarrow \Gamma', P \parallel p(e)} \text{ G-TRANS}$$

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

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

Các quy tắc trong Bảng 3.3 có nghĩa như sau:

- Quy tắc G-PLAIN để cập nhật các thay đổi ở mức cục bộ lên mức toàn cục. Giả sử $p : E \in \Gamma$, khi luồng p tạo ra biến đổi $E, e \to E', e'$ trên môi trường cục bộ E của nó. Hàm $reflect$ sẽ cập nhật những biến đổi này lên môi trường toàn cục, chuyển từ môi trường Γ thành Γ'.

- Quy tắc G-SPAWN dùng cho trường hợp tạo luồng mới với lệnh **spawn**. Lệnh **spawn**(e_1) tạo ra luồng mới p' thực thi e_1 song song với luồng cha p, biến đổi môi trường từ Γ sang Γ'.

- Quy tắc G-TRANS dùng trong trường hợp luồng p tạo một giao dịch mới với lệnh **onacid**. Giao dịch mới với nhãn l được sinh ra, môi trường được biến đổi từ Γ sang Γ'.

- Quy tắc G-COMM để thực hiện việc kết thúc các giao dịch. Giao dịch hiện tại của luồng p là l. Tất cả các luồng trong giao dịch l phải cùng kết thúc (commit) khi giao dịch l kết thúc.

- Quy tắc G-ERROR dùng trong trường hợp việc kết thúc một giao dịch không thành công. Trong trường hợp luồng p thực hiện việc kết thúc giao dịch bên ngoài môi trường của nó nghĩa là $|E| = 0$ thì sẽ trả lại kết quả lỗi ($error$).

3.1.2 Hệ thống kiểu tính số giao dịch tối đa

Trong hệ thống kiểu này, chúng tôi đề xuất kiểu của một thành phần là một dãy số được gắn dấu. Các dãy số này là trừu tượng các hành vi giao dịch của thành phần được định kiểu.

3.1.2.1 Kiểu

Để mô tả hành vi giao dịch của một thành phần, chúng tôi sử dụng một tập các ký hiệu $\{+, -, \neg, \sharp\}$. Ký hiệu + và − lần lượt mô tả cho sự bắt đầu và kết thúc một giao dịch. Ký hiệu ¬ mô tả các điểm đồng kết thúc của các giao dịch trong các luồng song song và ♯ mô tả số log tối đa được tạo ra. Để tiện lợi hơn cho việc tính toán về sau trên các dãy này, chúng tôi gán một ký hiệu với một số tự nhiên không âm $n \in \mathbb{N}^+ = \{0, 1, 2, ..\}$ thành dạng *số có dấu*. Vì vậy, các kiểu này sử dụng dãy hữu hạn của tập số có dấu ${}^{T}\mathbb{N} = \{ {}^{+}n, {}^{-}n, {}^{\sharp}n, {}^{\neg}n \mid n \in \mathbb{N}^+\}$. Chúng tôi sẽ đưa ra các quy tắc để mô tả một thành phần (biểu thức) của chương trình STM tối giản bằng một dãy số có dấu.

Một đoạn chương trình có kiểu ${}^{+}n$ $({}^{-}n)$ được hiểu là nó có liên tiếp n lệnh **onacid** (**commit**). Kiểu ${}^{\neg}n$ của đoạn chương trình được hiểu là trong đoạn chương trình này có n luồng cần phải đồng bộ với một **onacid** nào đó để hoàn tất một giao dịch. Ký hiệu ${}^{\sharp}n$ được hiểu là thành phần này tạo ra tối đa n log.

3.1.2.2 Quy tắc kiểu

Cú pháp ngôn ngữ kiểu T được định nghĩa như sau:

$$T = S \mid S^{\rho}$$

Ký hiệu S^{ρ} được sử dụng cho biểu thức **spawn** để đánh dấu chúng cần được đồng bộ với luồng cha của chúng. Chúng ta định nghĩa hàm $kind(T)$ trả về trống cho kiểu thông thường S hoặc ρ nếu T có dạng S^{ρ}.

Môi trường kiểu cung cấp thông tin ngữ cảnh cho biểu thức đang được tính kiểu. Các phát biểu về kiểu có dạng sau:

$$n \vdash e : T$$

Trong đó $n \in \mathbb{N}$ là môi trường kiểu. Khi n là dương nó thể hiện số giao dịch đang mở mà e sẽ đóng bởi kết thúc thường hoặc đồng kết thúc trong e.

Các quy tắc kiểu được mô tả trong Bảng 3.4. Quy tắc T-SPAWN chuyển đổi S thành chuỗi đồng bộ và tạo ra kiểu mới bởi ρ để có thể hợp các luồng chạy song song với nó bằng quy tắc T-MERGE. Quy tắc T-PREP cho phép chúng ta tạo một kiểu phù hợp cho e_2 để áp dụng T-MERGE. Quy tắc T-JC để thực hiện kết hợp kiểu của luồng cha và luồng con. Các quy tắc còn lại thì khá tự nhiên theo ngữ nghĩa của chương trình.

Định lý 3.1 (Tính chất của một phát biểu kiểu) *Nếu* $n \vdash e : T$ *và* $n \geq 0$ *thì* $\mathrm{sim}({}^{+}n, T) = {}^{\sharp}m$ *và* $m \geq n$ *trong đó* $\mathrm{sim}(T_1, T_2) = \mathrm{seq}(\mathrm{jc}(S_1, S_2))$ *với* S_i *là* T_i *được bỏ* ρ.

Chứng minh: Việc chứng minh được thực hiện bằng phương pháp quy nạp dựa trên các quy tắc xác định kiểu được mô tả trong Bảng 3.4 ■

Bảng 3.4: Quy tắc định kiểu cho chương trình STM tối giản

$$\frac{}{-1 \vdash \mathbf{onacid} : {}^{+}1}\ \text{T-ONACID} \qquad \frac{}{1 \vdash \mathbf{commit} : {}^{-}1}\ \text{T-COMMIT} \qquad \frac{n \vdash e : S}{n \vdash \mathbf{spawn}(e) : \mathrm{join}(S)^{\rho}}\ \text{T-SPAWN}$$

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

$$\frac{n \vdash e_1 : S_1^{\rho} \quad n \vdash e_2 : S_2^{\rho} \quad S = \mathrm{merge}(S_1, S_2)}{n \vdash e_1; e_2 : S^{\rho}}\ \text{T-MERGE} \qquad \frac{n \vdash e : S}{n \vdash e : \mathrm{join}(S)^{\rho}}\ \text{T-PREP}$$

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

3.1.2.3 Tính đúng của hệ thống kiểu

Để chứng minh tính đúng của hệ thống kiểu, ta cần chứng minh một chương trình có kiểu hợp lệ sẽ không có số log (tại bất kỳ thời điểm nào trong quá trình thực hiện của chương trình) lớn hơn số thể hiện trong kiểu của nó. Ta gọi chương trình có kiểu hợp lệ là e và kiểu của nó là $\natural n$. Ta cần chứng minh khi thực thi e theo ngữ nghĩa trong Phần 3.1.1.2, số log trong môi trường toàn cục luôn nhỏ hơn n.

Một trạng thái của chương trình là một cặp Γ, P trong đó $\Gamma = p_1 : E_1; \ldots; p_k : E_k$ và $P = \coprod_1^k p_i(e_i)$. Ta nói rằng Γ thỏa mãn P, ký hiệu là $\Gamma \vDash P$, nếu tồn tại $S_1, \ldots, S_k$ mà $|E_i| \vdash e_i : S_i$ với mọi $i = 1, \ldots, k$. Với mỗi i, E_i mô tả số log đã được tạo ra hoặc sao chép trong luồng p_i và S_i mô tả số log sẽ được tạo ra khi thực thi e_i. Vì vậy, luồng p_i có hành vi về log được mô tả bởi $\mathrm{sim}({}^{+}|E_i|, S_i)$, trong đó hàm sim được định nghĩa trong Định lý 3.1. Ta chứng minh rằng $\mathrm{sim}({}^{+}|E_i|, S) = \natural n$ với một số n nào đó. Ta ký hiệu giá trị n này là $|E_i, S_i|$. Tổng số log của một trạng thái chương trình, bao gồm trong Γ và các log sẽ được tạo ra khi thực thi phần còn lại của chương trình, được ký hiệu là $|\Gamma, P|$, được định nghĩa như sau:

$$|\Gamma, P| = \sum_{i=1}^{k} |E_i, S_i|$$

Bổ đề 3.1 (Trạng thái) *Nếu $\Gamma \vDash P$ thì $|\Gamma, P| \geq |\Gamma|$.*

Chứng minh: Theo định nghĩa của $|\Gamma, P|$ và $|\Gamma|$, ta chỉ cần chứng minh $|E_i, S_i| \geq |E_i|$ với mọi i. Điều này đúng theo Định lý 3.1. ■

Bổ đề 3.2 (Bất biến cục bộ) *Nếu $E, e \to E', e'$, và $|E| \vdash e : S$ thì tồn tại S' sao cho $|E'| \vdash e' : S'$ và $|E, S| \geq |E', S'|$.*

Chứng minh: Việc chứng minh được thực hiện bằng cách kiểm tra trực tiếp các luật ngữ nghĩa đối tượng trong Bảng 3.2. ■

Bổ đề 3.3 (Bất biến toàn cục) *Nếu $\Gamma \vDash P$ và $\Gamma, P \Rightarrow \Gamma', P'$ thì $\Gamma' \vDash P'$ và $|\Gamma, P| \geq |\Gamma', P'|$.*

Chứng minh: Việc chứng minh được thực hiện bằng cách kiểm tra từng quy tắc một tất cả các quy tắc ngữ nghĩa trong Bảng 3.3. Với mỗi quy tắc, ta cần chứng minh hai phần: (i) $\Gamma' \vDash P'$ và (ii) $|\Gamma, P| \geq |\Gamma', P'|$. ■

Định lý 3.2 (Tính đúng) *Giả sử* $0 \vdash e : \natural n$ *và* $p_1 : \epsilon, p_1(e) \Rightarrow^* \Gamma, P$ *khi đó* $|\Gamma| < n$.

Chứng minh: Với môi trường ban đầu ta có $|p_1 : \epsilon, p_1(e)| = \mathrm{sim}(0, \natural n) = \natural n$. Từ Bổ đề 3.2, 3.3 và Định lý 3.1, định lý được chứng minh bằng quy nạp theo độ dài của dẫn xuất. ■

Định lý cuối cùng này đã khẳng định rằng nếu chương trình có kiểu hợp lệ, thì chương trình đó khi thực hiện sẽ không bao giờ tạo ra số log cùng tồn tại lớn hơn giá trị thể hiện trong kiểu của chương trình.

3.2 Xác định biên tài nguyên tiêu thụ của chương trình

Với ngôn ngữ giao dịch tối giản trong phần 3.1 chúng tôi xây dựng được một hệ thống kiểu để xác định số giao dịch tối đa của chương trình. Điều này không chỉ nhằm mục đích xác định số giao dịch tối đa, mà chúng giúp chúng tôi đưa ra được các quy tắc kiểu đơn giản, thuận lợi để chứng minh, kiểm soát tính đúng của các quy tắc, làm nền tảng cho các nghiên cứu tiếp theo.

Tuy nhiên, với ngôn ngữ trên, chúng ta chưa có thông tin để xác định tài nguyên tiêu thụ của chương trình mà mới chỉ dừng lại ở việc xác định số giao dịch tối đa của chương trình. Vì vậy, trong phần này chúng tôi cải tiến ngôn ngữ 3.1.1 bằng việc bổ sung mỗi giao dịch một tham số thể hiện lượng tài nguyên mà giao dịch cần sử dụng. Từ đó, chúng tôi xây dựng một hệ thống kiểu để xác định biên tài nguyên mà chương trình cần sử dụng dựa trên tham số của các giao dịch.

Các tham số của các giao dịch không phải do người dùng đưa vào mà chúng sẽ được tổng hợp từ các câu lệnh, các khai báo trong các giao dịch. Điều này giúp chúng tôi giải quyết bài toán một cách tổng quát hơn. Tài nguyên ở đây là các tham số, chưa phải một loại tài nguyên cụ thể, vì vậy tùy từng trường hợp trong thực tế mà chúng ta xác định loại tài nguyên cụ thể cần tính (có thể là bộ nhớ cấp phát cho các biến, hoặc các đối tượng dùng chung, hoặc bộ nhớ cho các tập dữ liệu,....).

Ngôn ngữ và hệ thống kiểu trong phần này có một số quy tắc và công thức được thay đổi để đạt được mục đích xác định biên tài nguyên của chương trình dựa trên các tham số của các giao dịch. Về mặt ý nghĩa, mục đích của chúng tương tự với ngôn ngữ và hệ thống kiểu đã trình bày ở trên, vì vậy chúng tôi không mô tả chi tiết ở đây.

3.3 Tổng kết chương

Trong chương này, chúng tôi đã xây dựng một ngôn ngữ giao dịch với các lệnh cốt lõi và liên quan tới việc tiêu thụ tài nguyên của chương trình hoạt động theo cơ chế STM. Cùng với đó, chúng tôi xây dựng một hệ thống kiểu để tìm ra số giao dịch và tài nguyên tiêu thụ tối đa của một chương trình giao dịch hợp lệ. Hệ thống kiểu của chúng tôi đã được chứng minh để đảm bảo tính đúng.

Kết quả nghiên cứu được công bố tại *tạp chí Khoa học, Đại học Sư Phạm Hà Nội* (Công trình khoa học số (1)), Hội thảo quốc tế *Theoretical Aspects of Computing (ICTAC) (Công trình khoa học số (2))*

Chương 4

Tính bộ nhớ tối đa cho chương trình của ngôn ngữ mệnh lệnh

Trong chương này chúng tôi đưa ra một ngôn ngữ giao dịch được mở rộng từ ngôn ngữ trong chương 3 với việc bổ sung các câu lệnh khai báo, khởi tạo các biến dùng chung, các câu lệnh điều kiện, câu lệnh vòng lặp, và các phép toán. Tiếp theo chúng tôi cải tiến hệ thống kiểu trong chương 3 để được một hệ thống kiểu với mục đích xác định biên tài nguyên bộ nhớ cấp phát cho các biến dùng chung (các biến sẽ bị nhân bản bởi cơ chế STM). Sau đó, chúng tôi cải tiến hệ thống kiểu để được một hệ thống kiểu mới có tính tích hợp. Đây là một tính năng thường thấy trong các hệ thống kiểu. Cuối cùng, chúng tôi cài đặt thuật toán suy diễn kiểu để định kiểu cho một thành phần của chương trình.

4.1 Giới thiệu

Trong chương 3, chúng tôi đã trình bày một ngôn ngữ giao dịch tối giản và một hệ thống kiểu để tính tài nguyên tối đa cần sử dụng của chương trình. Tài nguyên này được tính toán dựa trên các tham số là tài nguyên mà mỗi giao dịch cần sử dụng. Kết quả chính của chương 3 đó là một hệ thống kiểu với các quy tắc kiểu đơn giản thuận lợi để đảm bảo tính đúng. Đây là một kết quả quan trọng, làm nền tảng cho các nghiên cứu tiếp theo.

Để áp dụng hệ thống kiểu này người dùng cần tính toán thủ công tài nguyên tiêu thụ của mỗi giao dịch. Như vậy, công việc xác định tài nguyên tiêu thụ của chương trình vẫn là bán tự động. Vì vậy, kết quả này vẫn mang tính phương pháp là chủ yếu.

Để giải quyết bài toán một cách hoàn toàn tự động và sát với thực tế hơn, trong chương này chúng tôi cải tiến ngôn ngữ bằng việc bổ sung các câu lệnh khai báo, khởi tạo các biến, các câu lệnh điều kiện, câu lệnh lặp, các phép toán số học, phép toán logic. Cùng với đó, chúng tôi cải tiến hệ thống kiểu để xác định biên tài nguyên tiêu thụ của các chương trình.

Trong ngôn ngữ này, các biến dùng chung giữa các luồng sẽ là các biến có thể bị nhân bản khi một luồng mới được sinh ra. Vì vậy, tài nguyên trong nghiên cứu này được cụ thể hóa là bộ nhớ cấp phát cho các biến dùng chung của các luồng.

4.2 Ngôn ngữ giao dịch với cấu trúc mệnh lệnh

Cú pháp của ngôn ngữ được mô tả trong Bảng 4.1. Ngữ nghĩa của ngôn ngữ được mô tả trong Bảng 4.2. Cú pháp và ngữ nghĩa giao dịch của ngôn ngữ tương tự với ngôn ngữ được trình bày trong Chương 3, tuy nhiên chúng được làm mịn hơn. Cú pháp và

ngữ nghĩa của các câu lệnh khác tương tự như các ngôn ngữ mệnh lệnh thông thường.

Bảng 4.1: Cú pháp của ngôn ngữ giao dịch cấu trúc mệnh lệnh

1.	P	::=	$\phi \mid P \parallel P \mid p(e)$	luồng/tiến trình
2.	T	::=	$\mathbf{int} \mid \mathbf{bool}$	kiểu
3.	D	::=	$\mathbf{shared}\ T\ \vec{x} := \vec{v} \mid\ T\ \vec{x} := \vec{v}$	khởi tạo các biến
4.	O	::=	$\bullet \mid \blacksquare \mid \blacklozenge \mid \blacktriangle$	phép toán
5.	v_i	::=	n	giá trị số nguyên
6.	v_b	::=	$\mathbf{true} \mid \mathbf{false}$	giá trị logic
7.	v	::=	$v_i \mid v_b$	giá trị
8.	e_i	::=	$e_i \bullet e_i \mid v_i$	biểu thức số nguyên
9.	e_b	::=	$e_i \blacksquare e_i \mid e_b \blacklozenge e_b \mid \blacktriangle\ e_b \mid v_b$	phép toán logic
10.	e	::=	$D \mid e; e \mid x := e_i \mid x := e_b$	biểu thức
11.			$\mid \mathbf{if}(e_b)\ \mathbf{then}\{\ e\ \}\ \ \mathbf{else}\{\ e\ \}$	điều kiện
12.			$\mid \mathbf{while}(e_b)\{\ e\ \}$	lặp
13.			$\mid \mathbf{spawn}\{\ e\ \}$	tạo luồng
14.			$\mid \mathbf{onacid} \mid \mathbf{commit}$	mở/đóng giao dịch

Bảng 4.2: Ngữ nghĩa của ngôn ngữ giao dịch cấu trúc mệnh lệnh

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

$$\frac{l\ fresh \quad start(l{:}n, p, \Gamma) = \Gamma'}{\Gamma, P \parallel p(\mathbf{onacid}(e_1); e_2) \Rightarrow \Gamma', P \parallel p(e_2)}\ \text{S-TRANS}$$

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

$$\frac{x, v : \mathcal{T}}{\Gamma, P \parallel p(shared\ \mathcal{T}\ x := v; e) \Rightarrow \Gamma', P \parallel p(e)}\ \text{S-INIT}$$

$$\frac{e_b \downarrow \mathbf{true}}{\Gamma, P \parallel p(\mathbf{if}(e_b)\mathbf{then}\{e_1\}\mathbf{else}\{e_2\}) \Rightarrow \Gamma, P \parallel p(e_1)}\ \text{S-COND}_1$$

$$\frac{e_b \downarrow \mathbf{false}}{\Gamma, P \parallel p(\mathbf{if}(e_b)\mathbf{then}\{e_1\}\mathbf{else}\{e_2\}) \Rightarrow \Gamma, P \parallel p(e_2)}\ \text{S-COND}_2$$

$$\frac{e_b \downarrow true \quad e}{\Gamma, P \parallel p(\mathbf{while}(e_b)\{e\}) \Rightarrow \Gamma, P \parallel p(e; \mathbf{while}(e_b)\{e\})}\ \text{S-WHILE}$$

$$\frac{e_b \downarrow false \quad e_1}{\Gamma, P \parallel p(\mathbf{while}(e_b)\{e_1\}; e_2) \Rightarrow \Gamma, P \parallel p(e_2)}\ \text{S-NO WHILE}$$

$$\frac{x, e_1, e_2 : \mathcal{T}}{\Gamma, P \parallel p(x := e_1; e_2) \Rightarrow \Gamma, P \parallel p(e_2)}\ \text{S-ASSIGN} \qquad \frac{}{\Gamma, P \parallel p(\alpha; e) \Rightarrow \Gamma, P \parallel p(e)}\ \text{S-SKIP}$$

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

4.3 Hệ thống kiểu tìm biên bộ nhớ cho các biến dùng chung

Hệ thống kiểu ở đây được phát triển từ Hệ thống kiểu trong chương 3 với những cải tiến để tính được bộ nhớ *log* tối đa của chương trình giao dịch một cách tự động.

Bởi vì hệ thống kiểu ở đây được phát triển dựa trên hệ thống kiểu trong chương 3, nên các quy tắc có mục đích và ý nghĩa tương tự, chúng chỉ khác nhau các phép toán chi tiết. Vì vậy ở đây, chúng tôi không trình bày chi tiết hệ thống kiểu này.

4.3.1 Suy diễn kiểu

Dựa trên các quy tắc kiểu trình bày ở trên, chúng tôi đã cài đặt một công cụ để suy ra kiểu cho những chương trình hợp lệ. Công cụ này được xây dựng bằng phương pháp lập trình hàm và sử dụng ngôn ngữ lập trình F♯. Công cụ này đã được kiểm thử với một số ca kiểm thử do chúng tôi đề xuất và chúng đã vượt qua các ca kiểm thử này.

4.4 Hệ thống kiểu tích hợp xác định biên bộ nhớ cấp phát cho các biến dùng chung

Hệ thống kiểu trong chương 3 và sau đó được phát triển, mở rộng trong phần 4.3 của chương 4 đã xác định được biên bộ nhớ dành cho các biến dùng chung giữa các luồng của chương trình. Hệ thống kiểu này có các quy tắc kiểu ngắn gọn, dễ hiểu, dễ phát triển và cài đặt. Tuy nhiên hệ thống kiểu này đòi hỏi một chương trình hoàn chỉnh mới có thể định kiểu được. Vì vậy chúng không thuận tiện cho các chương trình lớn, được viết bởi nhiều người.

Trong phần này luận án sẽ xây dựng một hệ thống kiểu với mục đích tương tự nhưng chúng có khả năng tích hợp. Nghĩa là chúng có thể định kiểu được cho các đoạn chương trình bất kỳ sau đó tích hợp chúng lại với nhau để được kiểu của cả chương trình. Đây là một đặc điểm thường thấy trong các hệ thống kiểu.

Định nghĩa 4.1 (Kiểu của một thành phần) *Kiểu T của một thành phần được định nghĩa như sau:*

$$T = S \mid TT \mid T \otimes T \mid T^{\rho} \mid T \oslash T \mid T \parallel^{k} T$$

Kiểu của một thành phần có thể là một chuỗi số được gắn ký hiệu S như các hệ thống kiểu ở trên, hoặc được tổng hợp từ các kiểu của các thành phần khác.

Trong định nghĩa này, TT có nghĩa rằng kiểu của một thành phần được suy ra từ kiểu của hai thành phần tuần tự. T^{ρ} có nghĩa rằng, một thành phần có kiểu T sẽ được thực thi trong một luồng chạy song song với cha của nó. Các phép toán $T \otimes T$, $T \oslash T$, và $T \parallel^{k} T$ là các phép toán hợp, chọn, song song tương ứng. Chúng tạo ra một thành phần mới từ những thành phần đang tồn tại. Quy tắc định kiểu được mô tả trong Bảng 4.3.

Bảng 4.3: Quy tắc kiểu của hệ thống kiểu tích hợp

$$\frac{}{0 \vdash \mathbf{onacid} : {}^{+}0}\ \text{T-ONACID} \qquad \frac{n \in \mathbb{N}^{*}}{n \vdash \mathbf{commit} : {}^{-}1}\ \text{T-COMMIT} \qquad \frac{n \vdash e : T}{n \vdash \mathbf{spawn}\{\ e\ \} : T^{\rho}}\ \text{T-SPAWN}$$

$$\frac{x : T \quad v : T \quad n = size(x)}{-n \vdash \mathbf{shared}\ T\ x := v : {}^{\star}n}\ \text{T-NEW} \qquad \frac{n = isclone(x)\ ?\ 0\ :\ size(x)}{-n \vdash x := e_1 : {}^{\star}n}\ \text{T-ASSIGN} \qquad \frac{n_k \vdash e_k : T_k \quad k = 1,2}{n_1 + n_2 \vdash e_1; e_2 : T_1 T_2}\ \text{T-SEQ}$$

$$\frac{e_b : \mathbf{bool} \quad n_k \vdash e_k : T_k \quad k = 1,2}{n \vdash \mathbf{if}(e_b)\ \mathbf{then}\{\ e_1\ \}\ \mathbf{else}\{\ e_2\ \} : T_1 \oslash T_2}\ \text{T-COND} \qquad \frac{e_b : \mathbf{bool} \quad n \vdash e : T \quad m = maxloop()}{n' \vdash \mathbf{while}(e_b)\{\ e\ \} : T^{m}}\ \text{T-WHILE}$$

$$\frac{n \vdash e : T}{n \vdash p(e) : T}\ \text{T-THREAD} \qquad \frac{n \vdash e_1 : T_1^{\rho} \quad n \vdash e_2 : T_2}{n \vdash e_1; e_2 : T_1^{\rho} T_2}\ \text{T-MERGE}$$

$$\frac{\Gamma_k, P_k \quad n_k \vdash P_k : T_k \quad \Gamma = \Gamma_1, \Gamma_2 \quad common(\Gamma_1, \Gamma_2) = n \quad k = 1,2}{n_1 + n_2 \vdash P_1 \parallel P_2 : T_1 \parallel^{n} T_2}\ \text{T-PAR}$$

Định nghĩa 4.2 (Kiểu hợp lệ) *Một chương trình được xem là* hợp lệ *nếu nó có kiểu T và $T \Rightarrow^{*} s^{\sharp}$.*

Định nghĩa 4.3 (Tài nguyên tiêu thụ) *Nếu* Γ, P *là một trạng thái đang chạy của chương trình và* $P : T$*, thì tài nguyên tiêu thụ tối đa trong khi thực thi* P *là:*

$$[\![\Gamma, P]\!] = \begin{cases} [\![s^\sharp]\!] & \text{nếu } ST \Rightarrow^* s^\sharp \\ error & \text{các trường hợp khác} \end{cases}$$

trong đó S có dạng ${}^+n_1\ {}^+n_2 \ldots {}^+n_k$ và $[\![S]\!] = [\![\Gamma]\!]$.

Định lý 4.1 (Tính chất của một phát biểu kiểu) *Giả sử* $e{:}T$*, nếu* $ST \Rightarrow^* s^\sharp$ *thì* $[\![s^\sharp]\!] \geq [\![S]\!]$*, trong đó* $S = {}^+n_1\ {}^+n_2 \ldots {}^+n_k$*, và* $[\![S]\!] = [\![\Gamma]\!]$.

Chứng minh: Bằng phương pháp quy nạp trên các quy tắc định kiểu trong Bảng 4.3. ■

Bổ đề 4.1 (Tính bất biến) *Nếu* $\Gamma, P \Longrightarrow \Gamma', P'$ *bởi quy tắc* R *và* $[\![\Gamma, P]\!] = n$ *thì* $[\![\Gamma', P']\!] = n'$ *và* $n \geq n'$ *với* $\forall\ R$.

Chứng minh: Việc chứng minh được thực hiện bằng cách kiểm tra từng quy tắc một tất cả các quy tắc ngữ nghĩa trong Bảng 4.2. ■

Bổ đề 4.2 (Trạng thái) *Cho một chương trình hợp lệ* P_0*, kiểu của nó là* T *và* $T \Rightarrow^* s^\sharp$*. Đối với trạng thái bất kỳ* Γ, P *của chương trình, ta có* $[\![\Gamma, P]\!] \leq [\![s^\sharp]\!]$.

Chứng minh: Chứng minh bằng phương pháp quy nạp trên các quy tắc ngữ nghĩa.

- Trạng thái khởi tạo: $[\![\phi, P]\!] = [\![s^\sharp]\!] \leq [\![s^\sharp]\!]$.
- Nếu $\Gamma, P \Longrightarrow T', P'$, giả sử $[\![\Gamma, P]\!] \leq [\![s^\sharp]\!]$, theo Bổ đề 4.1, ta có $[\![\Gamma', P']\!] \leq [\![\Gamma, P]\!] \leq [\![s^\sharp]\!]$.

■

Định lý 4.2 (Tính đúng) *Cho một chương trình hợp lệ* P_0*, kiểu của nó là* T*, và* $T \Rightarrow^* s^\sharp$ *thì tài nguyên tiêu thụ của chương trình trong khi thực thi không thể vượt quá* $[\![s^\sharp]\!]$.

Chứng minh: Giả sử Γ, P là một trạng thái của chương trình, theo Bổ đề 4.2 ta có $[\![\Gamma, P]\!] \leq [\![s^\sharp]\!]$. Theo Định nghĩa $[\![\Gamma, P]\!]$ và Định lý 4.1 ta suy ra $[\![\Gamma]\!] \leq [\![s^\sharp]\!]$. ■

Định lý này khẳng định rằng, nếu một chương trình là hợp lệ thì bộ nhớ lớn nhất sử dụng cho các biến dùng chung sẽ không vượt quá giá trị được mô tả trong kiểu của nó.

4.5 Tổng kết chương

Trong chương này, chúng tôi đã mở rộng ngôn ngữ từ chương 3 để gần với ngôn ngữ thực tế hơn. Sau đó, chúng tôi xây dựng hai hệ thống kiểu để xác định tài nguyên bộ nhớ tối đa cần cấp phát cho các biến dùng chung của chương trình.

Kết quả nghiên cứu trong chương này đã được công bố (hoặc đã gửi đăng) tại các tạp chí, hội thảo khoa học: Hội thảo quốc tế *The Eighth International Symposium on Information and Communication Technology (SOICT)* (Công trình khoa học số (3)), tạp chí *Khoa học và Kỹ thuật, Học viện Kỹ thuật quân sự* (Công trình khoa học số (4)), tạp chí *VNU Journal of Science* (công trình khoa học số (5)) (Đang chờ phản biện).

Chương 5

Xác định bộ nhớ giao dịch tối đa cho chương trình của ngôn ngữ hướng đối tượng

Trong chương này chúng tôi trình bày về một ngôn ngữ giao dịch với cấu trúc hướng đối tượng, trong đó cơ chế STM được cải tiến so với ngôn ngữ trong các chương 3, chương 4 để chúng sử dụng bộ nhớ hiệu quả hơn. Cùng với đó, chúng tôi trình bày một hệ thống kiểu được cải tiến, mở rộng từ hệ thống kiểu trong chương 4 để tìm biên bộ nhớ cấp phát cho các đối tượng dùng chung giữa các giao dịch trong chương trình.

5.1 Giới thiệu

Trong chương 4, luận án đã đưa ra một ngôn ngữ giao dịch với cấu trúc mệnh lệnh, sau đó đề xuất hai hệ thống kiểu để ước lượng biên tài nguyên tiêu thụ của chương trình. Ngôn ngữ này có hai đặc điểm nổi bật đó là: Thứ nhất, chúng là ngôn ngữ đa luồng với cấu trúc mệnh lệnh; Thứ hai, cơ chế STM của ngôn ngữ thực hiện sao chép toàn bộ các biến dùng chung của luồng cha vào luồng con (khi luồng con bắt đầu được sinh ra). Như vậy, chương 4 đã giải quyết được bài toán ước lượng tài nguyên của chương trình cho lớp ngôn ngữ giao dịch với cấu trúc mệnh lệnh gần với ngôn ngữ thực tế. Cơ chế STM của ngôn ngữ cũng đã khắc phục được những hạn chế của cơ chế dựa trên khóa và hoạt động phù hợp với cơ chế hoạt động của một số ngôn ngữ giao dịch thực tế.

Tuy nhiên, hiện nay nhiều ngôn ngữ được xây dựng theo cấu trúc hướng đối tượng. Như vậy, kết quả trong chương 4 chưa giải quyết được bài toán đặt ra đối với lớp ngôn ngữ này. Thêm nữa, chúng ta nhận thấy rằng, cơ chế sao chép toàn bộ các biến dùng chung cho luồng con từ luồng cha tạo ra sự thuận lợi cho người cài đặt ngôn ngữ. Tuy nhiên, dưới góc độ sử dụng bộ nhớ của chương trình thì điều này chưa tối ưu. Bởi vì, những biến dùng chung của luồng cha mà luồng con không cần sử dụng thì việc sao chép chúng là không cần thiết.

Để giải quyết các vấn đề trên, trong chương này, chúng tôi đưa ra một ngôn ngữ giao dịch với cấu trúc hướng đối tượng (gọi tắt là ngôn ngữ giao dịch cấu trúc hướng đối tượng). Cơ chế STM của ngôn ngữ được cải tiến để sử dụng bộ nhớ hiệu quả hơn. Cụ thể, khi luồng con được sinh ra, chúng sẽ không sao chép toàn bộ các đối tượng dùng chung của luồng cha, mà chỉ khi giao dịch cần sử dụng một đối tượng dùng chung nào đó chúng mới sao chép đối tượng đó từ giao dịch gần với nó nhất để tạo thành bản sao để chúng sử dụng riêng. Tiếp theo, chúng tôi cải tiến hệ thống kiểu tích hợp đã trình bày trong chương 4 để xác định biên tài nguyên bộ nhớ cấp phát cho các đối tượng dùng chung của các giao dịch (gọi tắt là bộ nhớ giao dịch).

5.2 Ngôn ngữ giao dịch với cấu trúc hướng đối tượng

5.2.1 Cú pháp

Cú pháp của ngôn ngữ được trình bày trong Bảng 5.1, trong đó chúng tôi sử dụng véc tơ ngang để mô tả một chuỗi thứ tự hữu hạn, ví dụ như $\overrightarrow{\texttt{f}}$ mô tả cho `f1`, `f2`, ..., `fn`.

Tại dòng 1, `P` mô tả các luồng/tiến trình, chúng có thể là một luồng trống (ký hiệu bởi ϕ), hoặc các luồng song song `P`‖`P`, hoặc một luồng `p` thực thi biểu thức `e`.

Tại dòng 2, `L` mô tả các lớp, trong đó `I` mô tả các lớp số nguyên đã được định nghĩa sẵn, và `class C{`$\overrightarrow{\texttt{f}}$`;` $\overrightarrow{\texttt{M}}$`}` mô tả việc khai báo các lớp. Dòng 3 mô tả các phương thức của lớp. Ký hiệu `m(`$\overrightarrow{\texttt{C}}$ $\overrightarrow{\texttt{x}}$`){ e }` nghĩa là phương thức `m` có các tham số $x_1, x_2, ..., x_n$, chúng là các biến đối tượng của các lớp $C_1, C_2, \ldots, C_n$ tương ứng.

Từ dòng 4 đến dòng 7, một biểu thức `e` có thể là: một biến `x`; các phép toán của lớp số nguyên, ví dụ như cộng, trừ, nhân, và chia (ký hiệu bởi $\bullet$); các câu lệnh để khởi tạo đối tượng mới, truy cập vào các trường của đối tượng, gọi các phương thức của đối tượng và gán một biểu thức cho một trường của đối tượng; các câu lệnh còn lại tương tự như các ngôn ngữ trước đã trình bày.

Tại dòng 8, `v` mô tả các giá trị, chúng có thể là các đối tượng, các giá trị của các trường của các giá trị khác, lời gọi các phương thức của các đối tượng, phép gán một giá trị vào một trường của đối tượng, hoặc một giá trị `null`.

Trong các ngôn ngữ trước đây của chúng tôi, khi một luồng được tạo bởi câu lệnh `spawn`, nó sao chép tất cả các đối tượng của luồng cha của nó. Trong ngôn ngữ này, một giao dịch chỉ sao chép các đối tượng từ giao dịch cha của nó khi giao dịch cần sử dụng (đọc, ghi) chúng. Điều này làm cho việc sử dụng bộ nhớ của chương trình hiệu quả hơn.

Bảng 5.1: Cú pháp của ngôn ngữ STM hướng đối tượng

1.	P ::= ϕ | P‖P | p(e)	luồng/tiến trình
2.	L ::= I | **class** C{ $\overrightarrow{\text{f}}$; $\overrightarrow{\text{M}}$ }	lớp
3.	M ::= m($\overrightarrow{\text{C}}$ $\overrightarrow{\text{x}}$){ e }	phương thức
4.	e ::= x | v | e $\bullet$ e | **new** C($\overrightarrow{\text{e}}$) | e.f | e.f:=e | e.m($\overrightarrow{\text{e}}$) | e;e	biểu thức
5.	| **spawn**{ e } | **onacid** | **commit**	
6.	| **if**(e) **then**{ e } **else**{ e }	
7.	| **while**(e){ e }	
8.	v ::= n | r | v.f | v.m($\overrightarrow{\text{v}}$) | v.f:=v | null	giá trị

5.2.2 Ngữ nghĩa

Một môi trường *(đơn vị)* E là một vùng bộ nhớ, được xác định bởi địa chỉ l của nó, bao gồm một danh sách các cặp l' và các biến tham chiếu r (được ký hiệu bởi $\mathbf{r} = (l'_1{:}r_1, l'_2{:}r_2, \ldots, l'_n{:}r_n)$), trong đó l' là địa chỉ nơi mà biến r được khởi tạo. Chúng tôi ký hiệu ϵ là một danh sách trống, danh sách mà không chứa phần tử nào. Mối biến tham chiếu có đối tượng, được ký hiệu bởi $r \mapsto O^C(\overrightarrow{v})$, trong đó O là một đối tượng của lớp C, được khởi tạo với các tham số $\overrightarrow{v}$.

Ký hiệu $[\![r]\!]$ là kích thước bộ nhớ của đối tượng được trỏ đến bởi r. Thực tế, kích thước bộ nhớ của một đối tượng có thể phụ thuộc vào giá trị của các trường thuộc tính

của đối tượng. Tuy nhiên, trong nghiên cứu này, chúng tôi giới hạn kiểu thuộc tính của đối tượng chỉ là số, vì vậy các đối tượng của một lớp có cùng kích thước bộ nhớ.

Ký hiệu $[\![E]\!]$ là kích thước của môi trường đơn vị E, và $[\![E]\!] = \sum_{r \in E} [\![r]\!]$, trong đó $r \in E$ nghĩa là tất cả các biến r trong E.

Một môi trường đơn vị E được phân thành các lớp E_α và E_l, trong đó E_α chứa các biến của luồng mà không thuộc bất cứ giao dịch nào, E_l bao gồm các biến của giao dịch l trong luồng.

Một môi trường đầy đủ của luồng p được ký hiệu là $\mathcal{E}_p$, bao gồm hai thành phần: $l{:}E_\alpha$ trong đó l là định danh của môi trường đơn vị E_α của luồng p và một chuỗi $(l{:}E_l \mid p'{:}\mathcal{E}_{p'})^*$ trong đó l là định danh của môi trường đơn vị E_l, p' là định danh của luồng con của luồng p.

Trong nghiên cứu này, cơ chế STM chỉ nhân bản các đối tượng trong giao dịch, các đối tượng không thuộc giao dịch thì không được nhân bản. Vì vậy, ước tính bộ nhớ được cấp phát cho các đối tượng không trong giao dịch (các đối tượng thuộc E_α) là không cần thiết. Vì vậy, trong các định nghĩa sau, chúng tôi bỏ qua môi trường đơn vị E_α.

Định nghĩa 5.1 (Môi trường luồng) *Một* môi trường luồng $\mathcal{E}_p$ *của một luồng* p *được định nghĩa đệ quy bởi* $\mathcal{E}_p = (l{:}E_l \mid p'{:}\mathcal{E}_{p'})^*$.

Môi trường của luồng chính là môi trường toàn cục của chương trình, được ký hiệu là $\mathcal{E}$. Tổng bộ nhớ giao dịch được chương trình sử dụng tại một thời điểm là tổng bộ nhớ được cấp phát cho các đối tượng của các giao dịch đang hoạt động của chương trình tại thời điểm đó. Chúng ta có định nghĩa sau về bộ nhớ giao dịch.

Định nghĩa 5.2 (Bộ nhớ giao dịch) Bộ nhớ giao dịch *mà chương trình cần sử dụng là tổng bộ nhớ cấp phát cho các đối tượng trong các giao dịch của chương trình, được ký hiệu bởi* $[\![\mathcal{E}]\!]$, *được định nghĩa bởi* $[\![\mathcal{E}]\!] = \sum_{E_l \in \mathcal{E}} [\![E_l]\!] + \sum_{\mathcal{E}' \in \mathcal{E}} [\![\mathcal{E}']\!]$, *trong đó* $E_l \in \mathcal{E}$, *và* $\mathcal{E}' \in \mathcal{E}$ *có nghĩa là tất cả các môi trường đơn vị* E_l, *và* $\mathcal{E}'$ *trong danh sách môi trường* $\mathcal{E}$.

Bảng 5.2: Ngữ nghĩa của ngôn ngữ STM hướng đối tượng

$$\frac{p' \ fresh \quad spawn(p', \mathcal{E}) = \mathcal{E}'}{\mathcal{E}, P \parallel p(\textbf{spawn}\{\ e_1\ };\ e_2) \Longrightarrow \mathcal{E}', P \parallel p(e_2) \parallel p'(e_1)}\ \text{S-SPAWN}$$

$$\frac{l \ fresh \quad onacid(l, \mathcal{E}) = \mathcal{E}'}{\mathcal{E}, P \parallel p(\textbf{onacid};\ e) \Longrightarrow \mathcal{E}', P \parallel p(e)}\ \text{S-ONACID} \qquad \frac{r \ fresh \quad new(r, \mathcal{E}) = \mathcal{E}'}{\mathcal{E}, P \parallel p(\textbf{new}\ C(\overrightarrow{v});\ e) \Longrightarrow \mathcal{E}', P \parallel p(e)}\ \text{S-NEW}$$

$$\frac{l = txn(p) \quad intrans(l, \mathcal{E}_p) = \phi \quad commit(l, \mathcal{E}) = \mathcal{E}'}{\mathcal{E}, P \parallel p(\textbf{commit};\ e) \Longrightarrow \mathcal{E}', P \parallel p(e)}\ \text{S-COMM} \qquad \frac{read(r, \mathcal{E}) = O^C(\overrightarrow{v}), \mathcal{E}' \quad fields(C) = \overrightarrow{f}}{\mathcal{E}, P \parallel p(r.f_i;) \Longrightarrow \mathcal{E}', P \parallel p(v_i)}\ \text{S-FIELD}$$

$$\frac{read(r, \mathcal{E}) = O^C(\overrightarrow{v}), \mathcal{E}' \quad write(r \downarrow_i^{r'}, \mathcal{E}') = \mathcal{E}''}{\mathcal{E}, P \parallel p(r.f_i := r';\ e) \Longrightarrow \mathcal{E}', P \parallel p(e)}\ \text{S-UPD} \qquad \frac{mbody(C, m) = \overrightarrow{x}, e}{\mathcal{E}, P \parallel p(r.m(\overrightarrow{v});) \Longrightarrow \mathcal{E}, P \parallel p(e|_{\overrightarrow{x} = \overrightarrow{v}})}\ \text{S-CALL}$$

$$\frac{i = (val(e) \neq 0)\ ?\ 1 : 2}{\mathcal{E}, P \parallel p(\textbf{if}(e)\ \textbf{then}\ \{\ e_1\ \}\ \textbf{else}\ \{\ e_2\ \}) \Longrightarrow \mathcal{E}, P \parallel p(e_i)}\ \text{S-COND}$$

$$\frac{e' = (val(e) \neq 0)\ ?\ e_1;\ \textbf{while}(e)\{\ e_1\ \};\ e_2 : e_2}{\mathcal{E}, P \parallel p(\textbf{while}(e)\{\ e_1\ \};\ e_2) \Longrightarrow \mathcal{E}, P \parallel p(e')}\ \text{S-WHILE}$$

$$\frac{}{\mathcal{E}, P \parallel p(\alpha;\ e) \Longrightarrow \mathcal{E}', P \parallel p(\alpha';\ e)}\ \text{S-SKIP} \qquad \frac{|\mathcal{E}_p| = 0}{\mathcal{E}, P \parallel p(\textbf{commit};\ e) \Longrightarrow error}\ \text{S-ERROR}$$

5.3 Hệ thống kiểu tích hợp tìm biên bộ nhớ giao dịch

Hệ thống kiểu của chúng tôi nhằm xác định cận trên của bộ nhớ giao dịch mà các chương trình STM cần sử dụng. Mỗi thành phần trong chương trình được định kiểu thông qua một chuỗi đặc biệt, tổng quát hành vi giao dịch của chương trình.

Định nghĩa 5.3 (Kiểu của một thành phần) Kiểu T của một thành phần *được định nghĩa đệ quy như sau:*

$$T = S \mid TT \mid T \otimes T \mid T^{\rho} \mid T \oslash T \mid T \parallel T$$

5.3.1 Quy tắc kiểu

Các quy tắc định kiểu được trình bày trong Bảng 5.3., trong đó kiểu của một thành phần có dạng $n \vdash e : T$, và được đọc là e có kiểu T. Ở đây, T nắm giữ thông tin về tài nguyên bộ nhớ của biểu thức e. n là môi trường kiểu của biểu thức e, chúng cung cấp ngữ cảnh cho biểu thức. Môi trường kiểu n được định nghĩa như sau.

Định nghĩa 5.4 (Môi trường kiểu) Môi trường kiểu *của một thành phần là lượng bộ nhớ được cấp phát hay được giải phóng khi thực thi thành phần đó. Nó thể hiện ngữ cảnh giao dịch cho thành phần đang định kiểu.*

Khi kiểu môi trường n là dương, có nghĩa là khi thực thi biểu thức, nó giải phóng n đơn vị nhớ; khi n âm, nghĩa là khi thực hiện biểu thức, nó tiêu tốn n đơn vị nhớ.

Bảng 5.3: Quy tắc định kiểu

$$\frac{}{0 \vdash \mathbf{onacid} : {}^{+}0}\ \text{T-ONACID} \qquad \frac{n \in \mathbb{N}}{n \vdash \mathbf{commit} : {}^{-}1}\ \text{T-COMMIT} \qquad \frac{r \mapsto O^{C}(\overrightarrow{v}) \quad n = size(r)}{-n \vdash \mathbf{new}\ C(\overrightarrow{v}) : {}^{\star}n}\ \text{T-NEW}$$

$$\frac{n = r \in setvar(l)\ ?\ 0 : size(r)}{-n \vdash r.f_i : {}^{\star}n}\ \text{T-ACCESS} \qquad \frac{n = r \in setvar(l)\ ?\ 0 : size(r)}{-2 * n \vdash r.f_i = v : {}^{\star}(2 * n)}\ \text{T-UPDATE} \qquad \frac{n \vdash e : T}{n \vdash \mathbf{spawn}\{\ e\ \} : T^{\rho}}\ \text{T-SPAWN}$$

$$\frac{n_1 \vdash e_1 : T_1^{\rho} \quad n_2 \vdash e_2 : T_2}{n_1 + n_2 \vdash e_1; e_2 : T_1^{\rho} T_2}\ \text{T-MERGE} \qquad \frac{n_1 \vdash e_1 : T_1 \quad n_2 \vdash e_2 : T_2}{n_1 + n_2 \vdash e_1; e_2 : T_1 T_2}\ \text{T-SEQ} \qquad \frac{mtype(C, m) : n \to T}{n \vdash r.m(\overrightarrow{v}) : T}\ \text{T-CALL}$$

$$\frac{n \vdash e_2 : T}{n' \vdash \mathbf{while}\ (e_1)\{\ e_2\ \} : T'}\ \text{T-WHILE} \qquad \frac{n_1 \vdash e_1 : T_1 \quad n_2 \vdash e_2 : T_2}{\max(n_1, n_2) \vdash \mathbf{if}\ (e)\ \mathbf{then}\{\ e_1\ \}\ \mathbf{else}\{\ e_2\ \} : T_1 \oslash T_2}\ \text{T-COND}$$

$$\frac{n \vdash e : T}{n \vdash p(e) : T}\ \text{T-THREAD} \qquad \frac{n_1 \vdash P_1 : T_1 \quad n_2 \vdash P_2 : T_2}{n_1 + n_2 \vdash P_1 \parallel P_2 : T_1 \parallel T_2}\ \text{T-PAR}$$

Các phép toán kết hợp các kiểu thành phần đã được chúng tôi định nghĩa chi tiết trong luận án. Ngoài ra, trong chương này chúng tôi đã đưa ra những chứng minh (ở dạng rút gọn) để đảm bảo tính đúng cho các quy tắc kiểu mà chúng tôi đề xuất.

5.4 Tổng kết chương

Trong chương này, chúng tôi đã trình bày một ngôn ngữ lập trình đa luồng hoạt động theo cơ chế STM và một hệ thống kiểu để xác định biên bộ nhớ cần thiết cho các đối tượng dùng chung của các giao dịch. Những tính năng mới của ngôn ngữ ở đây đó là ngôn ngữ có cấu trúc hướng đối tượng và ngữ nghĩa hoạt động tối ưu hơn.

Kết quả nghiên cứu trong chương này dự kiến gửi vào tạp chí *International Journal of Software Engineering and Knowledge Engineering Journal (IJSEKE)* (Công trình khoa học số (6)).

Chương 6

Kết luận

Hiện nay, trong các lĩnh vực lập trình nhúng, lập trình cho các thiết bị di động, các thiết bị IOT, hay các hệ thống của các trung tâm dữ liệu, nơi mà tài nguyên tiêu hao liên quan chặt chẽ đến tuổi thọ pin, hoặc chi phí duy trì hoạt động, nhu cầu xây dựng các phần mềm hiệu xuất cao, sử dụng tiết kiệm tài nguyên là một bài toán rất quan trọng. Đã có nhiều nhà khoa học, nhiều công trình nghiên cứu về vấn đề này trong các ngữ cảnh khác nhau. Hầu hết các nghiên cứu này thực hiện trên các chương trình tuần tự, hoặc các chương trình đa luồng nhưng các luồng hoạt động song song độc lập.

Một số tác giả thực hiện nghiên cứu trên các ngôn ngữ hoặc thư viện cụ thể hoạt động theo cơ chế STM bằng cách sử dụng các công cụ để đo lượng tài nguyên tiêu thụ khi chương trình chạy, sau đó lấy kết quả trung bình của các lần đo để ước lượng lượng tài nguyên tiêu thụ của chương trình. Vì vậy những phương pháp này chỉ cho kết quả gần đúng và chỉ thực hiện được khi chương trình đã hoàn thiện. Mặt khác các nghiên cứu này được thực hiện trên các ngôn ngữ hoặc các thư viện cụ thể, chúng không sử dụng được cho các ngôn ngữ khác nên chúng không mang tính tổng quát.

Nghiên cứu của chúng tôi được thực hiện trên một ngôn ngữ chung, hoạt động theo cơ chế STM vì vậy kết quả mang tính tổng quát, chúng ta có thể phát triển để ứng dụng cho các ngôn ngữ thực tế. Phương pháp của chúng tôi đề xuất là phương pháp tĩnh và kết quả được chứng minh đảm bảo tính đúng mà không cần chạy chương trình.

6.1 Những kết quả đạt được

Kết quả chính của luận án bao gồm hai hệ thống kiểu với mục đích ước lượng tĩnh tài nguyên tối đa cần sử dụng của chương trình STM:

- Hệ thống kiểu thứ nhất được xây dựng với các quy tắc đơn giản hơn, thuận lợi để phát triển, cài đặt. Hệ thống kiểu này yêu cầu một chương trình hoàn thiện (chương trình đã được viết xong và hợp lệ) thì mới có thể định kiểu được, chúng phù hợp với các chương trình nhỏ, được viết bởi số ít người.

- Hệ thống kiểu thứ hai có khả năng định kiểu linh hoạt hơn và chúng tôi gọi là hệ thống kiểu tích hợp. Chúng có thể định kiểu cho những thành phần bất kỳ trong chương trình, sau đó tích hợp lại để được kiểu của toàn bộ chương trình. Hệ thống kiểu này phù hợp với những chương trình phức tạp, được viết bởi nhiều người.

Ngoài những đóng góp chính ở trên, luận án đã đưa ra một số cải tiến cho cơ chế STM để sử dụng bộ nhớ giao dịch hiệu quả hơn.

Cuối cùng, dựa trên các quy tắc kiểu, chúng tôi đã cài đặt một công cụ để định kiểu cho chương trình. Công cụ được cài đặt bằng phương pháp lập trình hàm và đã được

kiểm thử tự động với một số ca kiểm thử do chúng tôi đề xuất, và chúng đã vượt qua các ca kiểm thử này. Công cụ này khi được hoàn thành có thể được tích hợp vào các chương trình soạn thảo mã nguồn hoặc các trình biên dịch để cung cấp thông tin cho người lập trình về lượng bộ nhớ tối đa mà chương trình cần sử dụng.

Ý nghĩa khoa học của luận án không chỉ ở việc giải quyết bài toán ước lượng tài nguyên của chương trình STM, mà còn ở phương pháp giải quyết bài toán mà chúng tôi đã xây dựng. Luận án đã đóng góp vào sự phát triển của lý thuyết kiểu những hệ thống kiểu với các bộ quy tắc kiểu giúp người lập trình kiểm soát được lượng tài nguyên tiêu thụ của chương trình. Từ các hệ thống kiểu này, chúng ta có thể tiếp tục mở rộng, phát triển để giải quyết các bài toán khác, chẳng hạn như xác định số gas yêu cầu bởi các hợp đồng thông minh trong Ethereum, ước lượng thời gian một chương trình cần để hoàn thành công việc...

Các nghiên cứu hiện tại chủ yếu áp dụng cho các chương trình tuần tự, hoặc có các luồng chạy song song độc lập. Nghiên cứu của chúng tôi áp dụng cho các chương trình có các luồng chạy song song nhưng không độc lập, nghĩa là chúng có thể có những điểm đồng bộ giữa các luồng. Như vậy, những chương trình tuần tự, hoặc có các luồng hoạt động song song độc lập là những trường hợp cụ thể trong bài toán của chúng tôi. Điều này làm cho bài toán tổng quát hơn và các tính toán cũng chính xác hơn.

Giải pháp của chúng tôi sử dụng hệ thống kiểu để ước lượng tĩnh tài nguyên sử dụng của chương trình mà không cần chạy chương trình. Các hệ thống kiểu chúng tôi đưa ra được chứng minh để đảm bảo tính đúng. Vì vậy kết quả tính toán là đáng tin cậy.

6.2 Những hạn chế và hướng nghiên cứu tiếp theo

Vấn đề về xác định biên tài nguyên cần sử dụng của các chương trình STM là một vấn đề phức tạp, trong nghiên cứu này chúng tôi đã đề xuất những giải pháp giải quyết bài toán mang tính cốt lõi nhất. Tuy nhiên, kết quả đạt được vẫn mang tính phương pháp, và còn những hạn chế cần tiếp tục cải tiến. Để có thể áp dụng kết quả nghiên cứu vào thực tế chúng tôi cần tiếp tục nghiên cứu cải tiến một số nội dung sau đây:

Ngôn ngữ giao dịch mà chúng tôi đã đề xuất đang ở dạng rút gọn với mục đích thể hiện rõ ý tưởng của bài toán đặt ra. Ngôn ngữ thực tế sẽ có nhiều cấu trúc khác nữa như hàm, thủ tục, lời gọi phương thức, truyền thông điệp, kế thừa, và nhiều tính toán chi tiết hơn. Ngoài ra, một số vấn đề khác như xử lý việc thực hiện lại các giao dịch khi xảy ra xung đột hay xác định số vòng lặp tối đa đối với các câu lệnh lặp cũng chưa được giải quyết trong nghiên cứu hiện tại. Để trở thành ngôn ngữ dùng được trong thực tế chúng tôi cần phải cải tiến, làm mịn để ngôn ngữ chi tiết hơn.

Hiện tại, các cú pháp, ngữ nghĩa của ngôn ngữ và các định nghĩa về môi trường, trạng thái của chương trình đang được thể hiện dưới dạng trừu tượng, bán hình thức. Vì vậy để cài đặt được chúng trở thành ngôn ngữ thực tế chúng tôi sẽ tiếp tục cải tiến để mô tả chúng dưới dạng hình thức hoàn toàn.

Tiếp theo, chúng tôi có kế hoạch sử dụng nền tảng $\mathbb{K}$ để cài đặt cho ngôn ngữ. Đây là một nền tảng mạnh, rất phù hợp để triển khai các ngôn ngữ dưới dạng hình thức, đặc biệt đối với các ngôn ngữ hoạt động theo cơ chế tương tranh. Nền tảng này đang

hứa hẹn trở thành một nền tảng thành công trong việc phát triển ngôn ngữ lập trình trong công nghiệp phần mềm.

Hệ thống kiểu của chúng tôi đưa ra đã đảm bảo tính đúng, tuy nhiên trong một số trường hợp chưa đảm bảo tính sắc của biên tìm được. Vì vậy trong các nghiên cứu tiếp theo chúng tôi sẽ tiếp tục cải tiến chúng để đảm bảo biên bộ nhớ tìm được là sắc. Ngoài ra, chúng tôi có kế hoạch cài đặt đầy đủ thuật toán suy diễn kiểu dựa trên hệ thống kiểu đã đề xuất để có thể tích hợp vào các trình soạn thảo, trình biên dịch của ngôn ngữ lập trình để giúp người lập trình kiểm soát chương trình của mình trong quá trình lập trình.

Trong nghiên cứu này hệ thống kiểu của chúng tôi có mục đích chính là xác định cận trên bộ nhớ của các chương trình STM. Tuy nhiên, chúng tôi đã cố gắng xây dựng chúng một cách tổng quát nhằm hướng đến việc có thể cải tiến chúng để xác định các tính chất khác của chương trình như xác định chi phí của tài nguyên CPU, băng thông mạng, hoặc số gas yêu cầu cho các hợp đồng thông minh trong Ethereum.

Bài toán của chúng tôi đang nghiên cứu trên ngôn ngữ chung với mục đích giải quyết bài toán một cách tổng quát. Ngôn ngữ của chúng tôi mang những đặc tính cốt lõi của các ngôn ngữ giao dịch thực tế. Vì vậy chúng tôi sẽ tiếp tục nghiên cứu các ngôn ngữ, thư viện cụ thể trong thực tế để xây dựng các công cụ chuyển đổi các ngôn ngữ này về ngôn ngữ của chúng tôi để áp dụng kết quả của luận án. Khi hoàn thiện công việc này, kết quả của luận án sẽ giải quyết được rất nhiều lớp ngôn ngữ thực tế vì thế khả năng ứng dụng rất rộng.

Danh mục các công trình khoa học của tác giả liên quan đến luận án

Những công trình đã công bố:

(1) Trương Anh Hoàng, **Nguyễn Ngọc Khải** (2015). "Hệ thống kiểu tính cận trên số log cho ngôn ngữ giao dịch đa luồng tối giản". Tạp chí Khoa học, Đại học Sư phạm Hà Nội, pp. 80-93.

(2) Anh-Hoang Truong, **Ngoc-Khai Nguyen**, Dang Van Hung and Duc-Hanh Dang (2016). "Calculating statically maximum log memory used by multi-threaded transactional programs". Theoretical Aspects of Computing - ICTAC - 13th International Colloquium, Taipei, Taiwan, ROC, Proceedings.

(3) **Ngoc-Khai Nguyen**, and Anh-Hoang Truong (2017). "A Compositional Type Systems for Finding Log Memory Bounds of Transactional Programs". Proceedings of the Eighth International Symposium on Information and Communication Technology - SOICT, pp. 409-416.

(4) **Nguyễn Ngọc Khải**, Trương Anh Hoàng (2018). "Hệ thống kiểu để suy ra bộ nhớ log của chương trình giao dịch từ biến dùng chung". Tạp chí Khoa học và Kỹ thuật, Học viện Kỹ thuật quân sự, pp. 18-33.

Những công trình đang gửi đăng:

(5) **Ngoc-Khai Nguyen**, Anh-Hoang Truong, and Duc-Hanh Dang. "Estimate the memory bounds required by shared variables in software transactional memory programs". VNU Journal of Science: Computer Science and Communication Engineering.

(6) **Ngoc-Khai Nguyen**, Anh-Hoang Truong, and Duc-Hanh Dang. "Finding transaction memory bound of STM programs". International Journal of Software Engineering and Knowledge Engineering Journal (IJSEKE).