ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN VĂN ĐỒNG

XÂY DỰNG HỆ THỐNG ĐẠI SỐ MÁY TÍNH XỬ LÝ BIỂU THỨC TOÁN HỌC

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà nội – 2016

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN VĂN ĐỒNG

XÂY DỰNG HỆ THỐNG ĐẠI SỐ MÁY TÍNH XỬ LÝ BIỂU THỨC TOÁN HỌC

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã số: 60480103

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

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

Hà nội- 2016

LỜI CẢM ƠN

Trước tiên em xin chân thành cảm ơn PGS.TS.Trương Anh Hoàng đã tận tình hướng dẫn, giúp đỡ em trong suốt quá trình thực hiện luận văn tốt nghiệp này.

Em xin chân thành cảm ơn các thầy cô giáo khoa Công nghệ Thông tin, trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, những người đã tận tình truyền đạt các kiến thức, quan tâm, động viên trong suốt thời gian tôi học tập và nghiên cứu tại Trường.

Nhân đây cho phép em gửi lời cảm ơn tới gia đình, bạn bè đặc biệt là nhóm các bạn học cùng lớp K20CNPM, lớp chuyên ngành công nghệ phần mềm đã thường xuyên quan tâm, giúp đỡ, chia sẻ kinh nghiệm, cung cấp các tài liệu hữu ích trong suốt thời gian học tập tại trường.

Hà Nội, tháng 06 năm 2016

Tác giả luận văn

Nguyễn Văn Đồng

LỜI CAM ĐOAN

Tôi xin cam đoan bản luận văn "Xây dựng hệ thống đại số máy tính xử lý biểu thức toán học" là công trình nghiên cứu của tôi dưới sự hướng dẫn khoa học của PGS.TS.Trương Anh Hoàng, tham khảo các nguồn tài liệu đã chỉ rõ trong trích dẫn và danh mục tài liệu tham khảo. Các nội dung công bố và kết quả trình bày trong luận văn này là trung thực và chưa từng được ai công bố trong bất cứ công trình nào.

Hà Nội, tháng 06 năm 2016

Tác giả luận văn

Nguyễn Văn Đồng

MỤC LỤC

Danh mục hình ảnh

Danh mục bảng

Danh mục chữ viết tắt

Thuật ngữ/ Từ viết tắt	Mô tả
BAE	Basic algebraic expression
GRE	General rational expression
SAE	Simpily algebraic expression
RNE	Rational number expression
gcd	Greatest common divisor

Mở đầu

Ngày nay các nhà khoa học mô hình hóa các hiện tượng tự nhiên bằng cách dịch các kết quả thực nghiệm và khái niệm lý thuyết vào những biểu thức toán học chứa số, biến, hàm số và các toán tử. Sau đó dựa vào các định lý đã được chứng minh để biến đổi hoặc chuyển thành các biểu thức khác để khám phá các hiện tượng đang được nghiên cứu. Cách tiếp cận toán học như vậy là một thành phần quan trọng của phương pháp nghiên cứu khoa học trong các ngành khoa học hiện nay.

Trong hơn nửa thế kỉ qua máy tính đã trở thành thiết bị không thể thiếu giúp giải quyết các vấn đề toán học. Các nhà toán học thường xuyên sử dụng máy tính để tìm lời giải cho các vấn đề khó khăn hoặc những vấn đề không thể thực hiện được bằng phương pháp thủ công. Trên thực tế máy tính chỉ thao tác với hai kí hiệu 0 - 1 thông qua các luật được thiết lập sẵn nên không thể mong đợi nó tạo ra tiên đề, lý thuyết.... Tuy nhiên một phần của lý luận toán học như các thao tác máy móc, phân tích biểu thức… thì có thể thực hiện bằng các thuật toán. Hiện nay có các chương trình máy tính có khả năng rút gọn biểu thức, tích hợp các chức năng phức tạp, giải chính xác phương trình… Các lĩnh vực toán học và khoa học máy tính có liên quan đến vấn đề này thì được gọi là đại số máy tính.

Đại số máy là tính là một lĩnh vực khoa học đề cập tới việc nghiên cứu và phát triển các thuật toán và phần mềm ứng dụng trong tính toán các biểu thức toán học và các đối tượng toán học khác. Trong đó hệ thống đại số máy tính là một phần của đại số máy tính, một chương trình phần mềm cho phép tính toán các biểu thức toán học bằng cách tương tự như tính toán bằng phương pháp thủ công mà các nhà toán học và khoa học thường sử dụng.

Hệ thống đại số máy tính là gì?

Hệ thống đại số máy tính là chương trình phần mềm thực hiện biến đổi các biểu thức toán học trong đó các yếu tố toán học như rút gọn, giai thừa, lũy thừa… được kết hợp với các cấu trúc điều khiển như vòng lặp, cấu trúc rẽ nhánh và các chương trình con để tạo ra các chương trình có thể giải quyết các vấn đề toán học.[23]

Hệ thống đại số máy tính đặc biệt hữu ích cho các nhà toán học, khoa học vì chúng có nhiều chức năng như tính toán biểu thức, xử lý biểu tượng (symbolic manipulation), giải phương trình…

Tại sao lại cần một hệ thống đại số máy tính?

- Trên thực tế có những bài toán hoặc vấn đề không thể giải quyết được bằng phương pháp thủ công.
- Các đáp án đưa ra bằng phương pháp đại số thường ngắn gọn và cung cấp thông tin về mối liên hệ giữa các biến.
- Từ biểu thức đại số có thể suy ra các thay đổi của tham số có thể ảnh hưởng đến kết quả tính toán.

- Kết quả của tính toán đại số thì luôn chính xác còn tính toán số học thường tồn tại giá trị xấp xỉ có thể dẫn đến các sai lệch trong kết quả.
- Trong một số trường hợp hệ thống đại số máy tính sẽ rút gọn thời gian tính toán hơn là các phương pháp tính toán truyền thống.

Hệ thống SMC [14]

Đếm mẫu là vấn đề cổ điển trong tính toán số lượng giải pháp thỏa mãn một tập các ràng buộc. Nó có nhiều ứng dụng trong lĩnh vực khoa học máy tính như trí tuệ nhận tạo, tối ưu hóa chương trình, phân tích lưu lượng thông tin.

Đếm mẫu là kỹ thuật có thể áp dụng cho số nguyên, giá trị logic nhưng không thể áp dụng trực tiếp cho dữ liệu phức tạp như một chuỗi kí tự, để giải quyết vấn đề này nhóm tác giả Loi Luu, Shweta Shinde, Prateek Saxena của trường đại học quốc gia Singapore (National University of Singapore) đã đưa ra giải pháp trong đó có trình bày một công cụ gọi là SMC (string model-counting).

Cho một tập chuỗi kí tự và ràng buộc của chúng, SMC có thể tính biên dựa trên số lượng phần tử của tập chuỗi thỏa mãn ràng buộc với độ chính xác và hiệu quả cao. Nhóm tác giả sử dụng hàm sinh (*generating functions* - GFs) một công cụ toán học quan trọng cho lý luận về chuỗi vô hạn, nó cung cấp cơ chế cho phép xác định số lượng phần tử của một tập chuỗi ràng buộc. Ý tưởng đằng sau hàm sinh (GFs) là mã hóa số lượng các chuỗi có độ dài k như là hệ số thứ k của một đa thức. Các đa thức có thể biểu diễn được dưới dạng các biểu thức hữu hạn, khi đó biểu thức hữu hạn này sẽ có khả năng biểu diễn tập vô hạn các chuỗi.

Trong công cụ SMC có sử dụng hệ thống Mathematica (một hệ thống đại số máy tính) để xử lý các biểu thức đại số, xử lý đa thức và một số các tính toán khác.

Mục tiêu của luận văn

Mục tiêu của luận văn là dựa vào nền tảng lý thuyết về toán học và các khái niệm thuật toán cơ bản để xây dựng các thuật toán và thể hiện của nó bằng các toán tử và cấu trúc điều khiển có trong ngôn ngữ lập trình để giải quyết các vấn đề trong hệ thống đại số máy tính để từ đó phát triển một hệ thống đại số máy tính miễn phí cho phép thực hiện các thao tác tính toán từ cơ bản đến phức tạp như tính giá trị biểu thức, tối giản phân số, tính toán đa thức …Trong đó mục tiêu chính của luận văn là phát triển các hàm xử lý đa thức nhằm thay thế hoàn toàn Mathematica trong công cụ SMC.

Các vấn đề được nêu ra và xử lý trong phạm vi luận văn:

- Xử lý biểu thức
    - Phân tích chuỗi đầu vào để nhận biết biểu thức.
    - Tính giá trị biểu thức.
    - Rút gọn biểu thức.
- Xử lý đa thức
    - Đa thức một biến, nhiều biến.
    - Các phép toán cơ bản trên đa thức.

    - Khai triển đa thức.
- Xây dựng các hàm xử lý cho hệ thống SMC
    - Tìm chuỗi taylor tại một giá trị bất kỳ, đến một hệ số bất kỳ.
    - Xây dựng hàm MAXF, MINF, DEDUP.

1 Kiến thức nền tảng

1.1 Ngôn ngữ giả mã

Là một ngôn ngữ biểu tượng được sử dụng trong luận văn để mô tả các khái niệm, định lý, ví dụ, đặc biệt là các thuật toán và các thủ tục thực hiện thuật toán. Ngôn ngữ giả mã tương tự như một ngôn ngữ đại số máy tính nhưng nó mang tính hình thức vì sử dụng cả biểu tượng toán học, tiếng anh và tiếng việt trong đó.

Để sử dụng một hệ thống đại số máy tính hiệu quả thì điều quan trọng là phải hiểu rõ ràng về cấu trúc và ý nghĩa của các biểu thức toán học. Về cơ bản biểu thức toán học trong ngôn ngữ giả mã cũng giống như các biểu thức toán học thông thường nhưng có một số thừa nhận để thích hợp trong môi trường tính toán. Các biểu thức được mô tả bằng cấu trúc sử dụng các toán tử và các ký hiệu sau:

- **Số nguyên và phân số**

  Một phần mềm thực hiện chính xác các thao tác trên biểu thức toán học phải có khả năng thực hiện chính xác các tính toán số học. Trong các ngôn ngữ lập trình thông thường việc tính toán với số thực dấu phẩy động thường có liên quan đến làm tròn số nên sẽ không phù hợp với hầu hết các hệ thống đại số máy tính. Thay vào đó các hệ thống sử dụng số hữu tỉ để đảm bảo thu được kết quả chính xác.

  Ví dụ:

  $$f = (x^2 - 1)/(x - 1)$$
  $$g = (x^2 - 0.99)/(x - 1)$$

  Mặc dù giá trị của f và g là gần như nhau với mỗi giá trị của x nhưng đặc điểm toán học của hai biểu thức là hoàn toàn khác nhau.Với $x \neq 1$ thì f có thể rút gọn thành $(x + 1)$ trong khi đó g thì không thể rút gọn được.

- **Số thực**

  Trong ngôn ngữ giả mã thì số thực là một số hữu hạn bao gồm dấu phẩy thập phân và có thể có số mũ của 10

  Ví dụ: 467.22, 0.33333333, 6.02.10^23.

  Trong toán học một số thực không có dạng hữu tỉ thì gọi là số vô tỉ. Do khó có thể thực hiện các thao tác tính toán biểu tượng với số vô tỉ nên số vô tỉ sẽ được thay bằng các ký hiệu $(e, ln...)$ hoặc các biểu thức đại số $(2^{\frac{1}{2}}...)$

- **Định danh**

  Trong ngôn ngữ giả mã định danh là một chuỗi các chữ cái tiếng anh, tiếng hy lạp, chữ số và dấu gạch dưới. Định danh được sử dụng trong ngôn ngữ giả mã như là biến lập trình tương ứng với kết quả của một phép tính, như một hàm, một toán tử, tên của thủ tục, ký hiệu toán học hoặc các ký tự đặc biệt.

- **Toán tử đại số và dấu ngoặc**

  Các toán tử đại số được trình bày trong bảng dưới. Dấu ngoặc được sử dụng để thay đổi cấu trúc của biểu thức.

Toán tử toán học	Toán tử trong ngôn ngữ giả mã
Cộng, trừ	$+, -$
Nhân, chia	$*, /$
Lũy thừa	^
Giai thừa	!

Bảng 1.1 Các toán tử đại số

- **Hàm số**

  Trong ngôn ngữ giả mã hàm số dùng để biểu diễn các hàm toán học như ($sin(x)$, $exp(x)$, $arctan(x)$...), các toán tử toán học ($Expand(u)$, $Factor\ (u)$, $Integral(u, x\ )$...), và các hàm như ($f(x)$, $g(x, y)$...).Trong hệ thống đại số máy tính các hàm toán học được định nghĩa thông qua các hành động của các luật biến đổi trong hệ thống. Hàm dùng để thao tác và phân tích biểu thức toán học được gọi là toán tử toán học.

  Một dạng quan trọng của hàm là dạng không xác định, trong dạng này biểu thức được ký hiệu ($f(x), g(x, y)$...). Các hàm dạng này không có luật biến đổi, không có thuộc tính mà chỉ có sự phụ thuộc của tên hàm vào biểu thức bên trong dấu ngoặc đơn.

- **Các toán tử logic và toán tử quan hệ**
  - Các toán tử quan hệ được sử dụng trong ngôn ngữ giả mã là:

    $$=, \neq, <, \leq, >, \geq$$

  - Các tán tử logic:

    $$and, or, not, true, false$$

- **Tập hợp và danh sách**
  - **Tập hợp**

    Trong ngôn ngữ giả mã một tập hợp là một tập bao gồm hữu hạn các biểu thức toán học được bao quanh bởi cặp dấu ngoặc '{}' và thỏa mãn hai tính chất sau:

    1. Nội dung của một tập hợp không phụ thuộc vào thứ tự của các phần tử trong tập hợp.
    2. Các phần tử trong tập hợp phải là duy nhất.

    Các toán tử của tập hợp: Cho A và B là hai tập hợp các toán tử của tập hợp được định nghĩa như sau:

    - ***Hợp*** (A ∪ B): là một tập mới chứa tất cả các phần tử của A và B. Ví dụ: $\{a, b, c, d\} \cup \{c, d, e, f\} \rightarrow \{a, b, c, d, e, f\}$
    - ***Giao*** (A ∩ B): là một tập mới chứa các phần tử có trong cả A và B. Ví dụ: $\{a, b, c, d\} \cap \{c, d, e, f\} \rightarrow \{c, d\}$
  - **Danh sách**

    Trong ngôn ngữ giả mã một danh sách bao gồm một số hữu hạn các biểu thức toán học và được bao quanh bởi cặp dấu ngoặc '[]'. Một danh sách rỗng thì không chứa biểu thức nào và được ký hiệu là [].

Danh sách có các tích chất sau:

1. Thứ tự của các phần tử trong danh sách là có ý nghĩa. Ví dụ [a, b] khác [b, a].
2. Các phần tử trong danh sách có thể giống nhau. Ví dụ [a,b] khác [a,b,b].

Cho L, M, N tương ứng là các danh sách và biểu thức x. Các toán tử của danh sách được định nghĩa như sau:

- **First**(L): toán tử sẽ trả về biểu thức đầu tiên trong L. Nếu L = [] thì toán tử trả về Undefined.
  - Ví dụ: First([a, b, c]) → a
- **Rest**(L): toán tử trả về một danh sách bao gồm tất cả các biểu thức có trong L ngoại trừ biểu thức đầu tiên. Nếu L = [] thì toán tử trả về Undefined.
  - Ví dụ: Rest([a, b, c]) → [b, c]
- **Adjoin**(x, L): Toán tử trả về một danh sách mới chứa toán tử đầu tiên là biểu thức x và theo sau là các biểu thức của L.
  - Ví dụ: Adjoin(d, [a, b, c]) → [d, a, b, c].
- **Join**(L, M,..., N): toán tử sẽ trả về một danh sách mới chứa các biểu thức của L và các biểu thức có trong các danh sách còn lại.
  - Ví dụ: Join([a, b], [b, c], [c, d, e]) → [a, b, b, c, c, d, e]

- **Biểu thức toán học trong ngôn ngữ giả mã**

  Trong ngôn ngữ giả mã biểu thức toán học là bất kỳ biểu thức nào được tạo thành bằng cách sử dụng số nguyên, phân số, số thực, định danh, hàm số, tập hợp, danh sách và các toán tử đại số, toán tử logic, toán tử quan hệ được mô tả ở trên.

1.2 Tính toán biểu thức và chương trình toán học

Tính toán biểu thức

Thuật ngữ tính toán biểu thức liên quan đến các hành động trong hệ thống đại số máy tính để đáp ứng một biểu thức đầu vào. Các hành động bao gồm: [13]

1. Phân tích cấu trúc của biểu thức và biến đổi sang cấu trúc của hệ thống.
2. Tính giá trị của các biến được gán và toán tử toán học xuất hiện trong biểu thức.
3. Áp dụng một số các quy tắc rút gọn cơ bản của đại số và lượng giác.

Chương trình toán học

Một chương trình toán học hay còn gọi là một thuật toán toán học là một chuỗi các câu lệnh để thực hiện các toán tử và cấu trúc điều khiển trong lập trình đại số máy tính. Cấu trúc của chương trình thường có các tính chất sau: [13]

1. Các câu lệnh trong chương trình được xem như một đơn vị được nhập vào tại một dấu nhắc đơn hoặc vùng đầu vào trong chế độ tương tác cho các chương trình lớn hơn.

2. Các câu lệnh bao gồm các biểu thức toán học, các câu lệnh gán, câu lệnh quyết định, câu lệnh lặp, hàm và các thủ tục được định nghĩa.
3. Như với các chương trình thông thường một số câu lệnh có vai trò như là các câu lệnh đầu vào, một số câu lệnh là tính toán trung gian với đầu ra không được hiển thị, một số câu lệnh thì để hiển thị dữ liệu là kết quả của quá trình tính toán.
4. Chương trình được thiết kế tổng quát để có thể thực hiện một lớp các vấn đề thay vì một vấn đề duy nhất.

1.3 Khái niệm toán học cơ bản

1.3.1 Số nguyên

Trong phần này sẽ đưa ra các tính chất cơ bản của số nguyên và mô tả một số thuật toán quan trọng để thao tác với số nguyên trong đại số máy tính.

$$Z = \{\ldots -2, -1, 0, 1, 2, \ldots\}.$$

Định nghĩa 1.1: Cho số nguyên a và b khác 0, có số nguyên q và r là duy nhất sao cho:

$$a = q * b + r$$

Số nguyên q là thương tương ứng với toán tử $iquot(a, b)$ và r là phần dư tương ứng toán tử $irem(a, b)$.

Định nghĩa 1.2:

- Số nguyên $b \neq 0$ là ước của số nguyên a nếu có một số nguyên q sao cho $a = q * b$
- Một ước chung của hai số nguyên a và b là một số nguyên c sao cho c là ước của a và b.

Định nghĩa 1.3: Hai số nguyên a và b là số nguyên tố cùng nhau nếu chúng chỉ có ước chung là 1 và -1.

Ước chung lớn nhất

Ước chung lớn nhất của hai số nguyên a và b là số nguyên d thỏa mãn:

1. d là ước chung của a và b
2. Nếu e là ước chung khác của a và b thì e chia hết cho d
3. d > 0

Chú ý: Nếu cả a và b bằng 0 thì định nghĩa trên không được áp dụng. Trong trường hợp này quy định $gcd(0,0) = 0$.

Thuật toán Euclid để tìm ước chung lớn nhất

Định nghĩa 1.4: Cho a và b là các số nguyên khác 0 và cho $r = irem(a, b)$ thì $gcd(a, b) = gcd(b, r)$.

- Thủ tục thực hiện toán tử tìm ước chung lớn nhất của hai số nguyên a và b

```
Procedure IntegerGCD(a, b);
Input
        a, b : là các số nguyên;
Output
        Ước chung lớn nhất của a và b;
Local Variables
        A, B, R;
Begin
        A := a; B := b;
        while B = 0 do
                R := Irem(A, B);
                A := B;
                B := R;
        Return(AbsoluteValue(A))
End
```

Hình 1.1 Thủ tục tìm ước chung lớn nhất của hai số nguyên a và b

Chú ý:

- $gcd(a, b)$ *là ước chung lớn nhất của a và b (Greatest Common Divisor).*
- *AbsoluteValue(A): giá trị tuyệt đối của A*

1.3.2 Số hữu tỉ

Trong ngôn ngữ giả mã một số hữu tỉ là một phân số a/b với a và $b \neq 0$ là các số nguyên. Với định nghĩa trên thì số hữu tỉ có thể biểu diễn dưới nhiều dạng ($1/2$, $(-2)/(-4)$...). Quá trình rút gọn sẽ biến đổi số hữu tỉ về dạng chuẩn.

Ví dụ: $2/4 \rightarrow 1/2, 2/(-4) \rightarrow (-1)/2\ (-2)/(-4) \rightarrow 1/2, 4/1 \rightarrow 4$

Phép biến đổi sẽ thu được bởi luật sau:

Định nghĩa 1.5: Cho a và b là các số nguyên. Phân số a/b là ở dạng chuẩn nếu thỏa mãn hai điều kiện sau:

1. $b > 1$
2. $gcd(a, b) = 1$

Định nghĩa 1.6: Một biểu thức là một số hữu tỉ ở dạng chuẩn nếu biểu thức đó là phân số ở dạng chuẩn hoặc là một số nguyên.

Toán tử rút gọn số hữu tỉ

Cho u là một số nguyên hoặc phân số có mẫu khác không. Toán tử $SymplifyRationalNumber$ sẽ biến đổi u thành số hữu tỉ ở dạng chuẩn.

(Chú ý: Ký hiệu hàm $FracOp(a, b)$ sẽ được sử dụng để biểu diễn phân số a/b).

- Thủ tục rút gọn số hữu tỉ

```
Procedure SimplifyRationalNumber(u);
Input
        u : là một phân số biểu diễn bằng ký hiệu hàm FracOp (với mẫu số
        khác 0) hoặc là một số nguyên;
Output
        là một phân số ở dạng chuẩn biểu diễn bằng ký hiệu hàm FracOp
        hoặc là một số nguyên;
Local Variables
        n, d, g;
Begin
        if Kind(u) = integer then Return(u)
        elseif Kind(u) = FracOp then
                n = Operand(u, 1);
                d = Operand(u, 2);
                if Irem(n, d)=0 then Return(Iquot(n, d))
                else
                        g := IntegerGCD(n, d);
                        if d > 0 then
                                Return(FracOp(Iquot(n, g),Iquot(d, g)))
                        elseif d < 0 then
                                Return(FracOp(Iquot(−n, g),Iquot(−d, g)))
End
```

Hình 1.2 Thủ tục rút gọn số hữu tỉ

2 Cấu trúc của biểu thức đại số

Do biểu thức toán học là các đối tượng dữ liệu trong chương trình đại số máy tính nên hiểu về mối quan hệ giữa các toàn tử và các toán hạng của biểu thức là rất cần thiết. Trong phần này luận văn sẽ mô tả về cấu trúc thông thường của một biểu thức toán học.

Định nghĩa 2.1: Một biểu thức toán học là sự kết hợp của các ký hiệu thông qua các luật. Các ký hiệu toán học có thể là số (hằng số), các biến số, các toán tử, các hàm số hoặc các ký hiệu nhóm. Trong đó:

- Hằng số là các giá trị đã được xác định.
- Biến số là ký hiệu tương ứng cho một giá trị chưa xác định hoặc có thể thay đổi trong biểu thức toán học.
- Toán tử : cộng, trừ, nhân, chia, lũy thừa và giai thừa.
- Các loại ngoặc: ngoặc đơn (), ngoặc nhọn {} hoặc ngoặc vuông [].

Biểu thức toán học bao gồm các biểu thức số học, đa thức, biểu thức đại số, biểu thức giải tích. Trong phạm vi luận văn các mô tả và định nghĩa sẽ tập trung vào biểu thức đại số.

Phân loại toán tử trong biểu thức

Các thuật ngữ sau áp dụng cho các toán tử trong biểu thức đại số, chúng được sử dụng để mô tả cấu trúc biểu thức.

Định nghĩa 2.2: Hai toán tử trong một biểu thức đại số có cấp độ ngoặc khác nhau nếu một toán tử ở bên trong và toán tử còn lại ở bên ngoài ngoặc. Khi hai toán tử không khác nhau về cấp độ ngoặc thì chúng được xem như có cùng cấp độ ngoặc.

Ví dụ: Trong biểu thức $a * (b + c)$ thì hai toán tử * và + không cùng cấp. Trong biểu thức $a * (b + c)/d$ toán tử * và / có cùng cấp độ.

Các toán tử trong một biểu thức được phân loại bởi số lượng các toán hạng và vị trí của các toán hạng liên quan đến toán tử. Các thuộc tính này mô tả bởi các thuật ngữ sau đây:

- Một toán tử hậu tố đơn phân là toán tử chỉ có một toán hạng ở ngay phía trước toán tử đó. Ví dụ $n!$
- Một toán tử tiền tố đơn phân là toán tử chỉ có một toán hạng ở ngay sau toán tử đó. Ví dụ: $-x$
- Toán tử tiền tố hàm là một biểu thức có ký hiệu hàm với một hoặc nhiều toán hạng. Ví dụ: $f(x, y)$
- Toán tử nhị phân trung tố là một toán tử có hai toán hạng trong đó một toán hạng ở ngay trước và một toán hạng ở ngay sau của toán tử. Ví dụ: $(a + b)$
- Toán tử trung tố n-ary là một toán tử có hai hoặc nhiều toán hạng ở cùng cấp ngoặc. Ví dụ: $(a + b + c + d + e)$

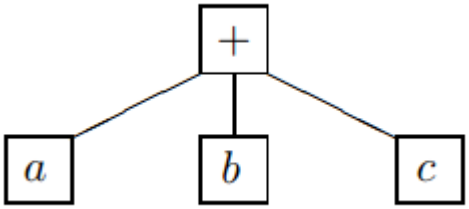
2.1 Cây biểu thức

Cây là một tập hợp hữu hạn các nút trong đó có một nút đặc biệt được gọi là gốc. Giữa các nút có quan hệ phân cấp gọi là quan hệ cha con. Định nghĩa đệ quy của cây [1]:

1. Một nút là một cây. Nút đó cũng là gốc của cây.
2. Nếu n là một nút và $T_1, T_2, \dots, T_k$ là các cây với $n_1, n_2 \dots n_k$ lần lượt là gốc thì một cây T mới sẽ được tạo ra bằng cách cho n là nút cha của các nút $n_1, n_2 \dots n_k$. Nghĩa là trên cây T lúc này n là gốc còn các cây $T_1, T_2, \dots, T_k$ là cây con của T, $n_1, n_2 \dots n_k$ là con của nút n

Cấu trúc của một biểu thức bao gồm các mối quan hệ giữa các toán tử và các toán hạng. Một cây biểu thức là một sơ đồ biểu diễn cấu trúc này.

Ví dụ: Biểu thức $(a + b + c)$ sẽ tương ứng với cây biểu thức sau



Mỗi toán tử và toán hạng trong biểu thức tương ứng bởi vị trí của nút trên cây. Nội dung của nút và mối liên hệ giữa các nút được xác định bởi luật ưu tiên các toán tử. Toán tử có độ ưu tiên thấp nhất của biểu thức xuất hiện trên cùng của cây. Nút này gọi là nút gốc của cây. Phần tương ứng với một toán hạng được gọi là nhánh của cây hoặc gọi là cây con.

2.2 Cấu trúc đệ quy của biểu thức đại số

Lý do đệ quy quan trọng trong hệ thống đại số máy tính là do cấu trúc đệ quy của biểu thức. Cấu trúc này được mô tả bởi Định nghĩa 2.3 dưới đây.

Định nghĩa 2.3: Các biểu thức được phân vào một trong hai loại sau [13]:

1. Biểu thức nguyên tử: là một số nguyên, số thực, ký hiệu hoặc các ký hiệu được quy ước $(e, true, \ldots)$. Biểu thức nguyên tử là thành phần cơ bản để xây dựng các biểu thức phức tạp hơn.
2. Biểu thức phức hợp bao gồm một toán tử với các toán hạng. Các toán tử có thể là một toán tử đại số $(+, -, *\ldots)$, một toán tử quan hệ $(<, >, =\ldots)$, một toán tử logic $(and, or, not\ldots)$, một toán tử tập hợp $(\cup, \cap \ldots)$, hoặc một hàm số. Một toán hạng của một toán tử có thể là biểu thức nguyên tử hoặc là biểu thức phức hợp khác. Phụ thuộc vào mỗi toán tử mà sẽ có số toán hạng tương ứng.

2.3 Cấu trúc thông thường của biểu thức đại số

Cấu trúc thông thường của một biểu thức đại số trong hệ thống đại số máy tính tương tự như cấu trúc của một biểu thức trong toán học và trong các ngôn ngữ lập trình phổ biến.

Định nghĩa 2.4: Cho u là một biểu thức đại số. Các toán tử trong u phải thỏa mãn:

1. Các toán tử $+$ và $-$ là toán tử tiền tố đơn phân hoặc toán tử trung tố nhị phân.
2. Các toán tử *, / , ^ là các toán tử trung tố nhị phân
3. Toán tử ! là toán tử hậu tố đơn phân

Định nghĩa 2.5: (Thứ tự ưu tiên của các toán tử)

1. Cho u là một biểu thức đại số thì thứ tự ưu tiên của các toán tử có cùng cấp độ ngoặc là:

Toán tử hàm (f, g…)
!
^

$*$,/
$+,-$

Bảng 2.1 Thứ tự ưu tiên của các toán tử

Nếu hai toán tử trong u cùng độ ưu tiên ở trên thì thứ tự ưu tiên sẽ được xác định bởi các luật sau:

a) Nếu các toán tử là $(+,-)$ hoặc $(*,/)$ hoặc (!) thì toán tử bên phải sẽ có độ ưu tiên thấp hơn.
b) Nếu toán tử là (^) thì toán tử bên trái có độ ưu tiên thấp hơn.

2. Hai toán tử ở cấp độ ngoặc khác nhau thì toán toán tử trong ngoặc có độ ưu tiên cao hơn còn toán tử bên ngoài ngoặc có độ ưu tiên thấp hơn.

2.4 Cấu trúc rút gọn của biểu thức đại số

Cấu trúc rút gọn của biểu thức giúp đơn giản hóa quá trình lập trình bằng cách loại bỏ các toán tử không cần thiết và cung cấp cơ chế truy cập dễ dàng hơn tới các toán hạng của biểu thức. Các giả thiết về cấu trúc và quy tắc ưu tiên của biểu thức đại số rút gọn được nêu ra ở hai định nghĩa Định nghĩa 2.6, Định nghĩa 2.7 dưới đây.

Định nghĩa 2.6: Cho u là một biểu thức đại số rút gọn. Các toán tử trong u thỏa mãn các cấu trúc sau [13]:

1. Toán tử + là một toán tử trung tố có hai hoặc nhiều toán hạng trong đó không có toán hạng nào của nó là một tổng và nhiều nhất một toán hạng là một số nguyên hoặc phân số.
2. Toán tử * là một toán tử trung tố với hai hoặc nhiều toán hạng trong đó không có toán tử nào của nó là một tích và có nhiều nhất một toán hạng là một số nguyên hoặc phân số. Nếu một số nguyên hoặc phân số là toán hạng của một tích thí nó là toán hạng đầu tiên.
3. Toán tử đơn phân $-$ và toán tử nhị phân $-$ không xuất hiện trong biểu thức rút gọn.
4. Toán tử nhị phân / không xuất hiện trong biểu thức rút gọn.
5. Phân số phải thỏa mãn các các luật sau:
   - Một phân số c/d với c ≠ 0 và d ≠ 0 là các số nguyên tương ứng với cây biểu thức có toán hạng ký hiệu là **fraction,** toán tử đầu tiên là c, toán tử thứ hai là d.
   - Một phân số âm sẽ có phần tử số là số âm và phần mẫu số là số dương.
6. Toán tử ^ là toán tử nhị phân. Nếu $u = v^n$ với n là số nguyên thì v không thể là số nguyên, phân số, tích hoặc lũy thừa.
7. Toán tử ! là một toán tử hậu tố đơn phân có toán hạng là số nguyên không âm

Định nghĩa 2.7: (Luật ưu tiên): Cho u là một biểu thức đại số rút gọn

1. Thứ tự ưu tiên của các toán tử trong cùng cấp ngoặc là:

Tên hàm (f, g…)

!
^
*
+

Bảng 2.2 Thứ tự ưu tiên của các toán tử cùng cấp độ ngoặc

Nếu hai toán tử ! ở cùng một cấp ngoặc thì toán tử bên phải có độ ưu tiên thấp hơn. Nếu hai toán tử ^ ở cùng một cấp ngoặc thì toán tửu bên trái có độ ưu tiên thấp hơn.

2. Các toán tử có cấp độ ngoặc khác nhau thì toán tử bên ngoài có độ ưu tiên thấp hơn, toán tử bên trong có độ ưu tiên cao hơn.

Dựa vào cấu trúc giả định và luật ưu tiên ở trên định nghĩa của một biểu thức đại số có thể biến đổi thành

Định nghĩa 2.8: Một biểu thức đại số u là dạng rút gọn nếu thỏa mãn các tính chất sau:

1. u là một số nguyên
2. u là phân c/d với c, d là các số nguyên khác 0
3. u là một ký hiệu
4. u là tổng, tích, lũy thừa, giai thừa hoặc hàm số với mỗi toán hạng của u là một biểu thức đại số rút gọn.

Chú ý: Mặc dù cấu trúc giả định và các luật ưu tiên được nêu trên là các thuộc tính rất quan trọng của biểu thức rút gọn nhưng chúng chưa đầy đủ, một định nghĩa đầy đủ sẽ được nêu ra ở phần sau của luận văn. Các mô tả trên chỉ đưa ra cái nhìn tổng quát giúp việc định nghĩa trong các phần tiếp theo rõ ràng hơn.

2.5 Các toán tử cơ bản của biểu thức đại số rút gọn

Để phân tích và vận dụng một biểu thức đại số yêu cầu phải truy cập vào các toán tử và toán hạng của biểu thức. Các toán tử sau sẽ thực hiện các việc đó.[12]

2.5.1 Định nghĩa toán tử $Kind(u)$

1. Nếu u là một biểu thức nguyên tử thì trả về kiểu của u (số nguyên, số hữu tỉ hoặc ký hiệu...)
2. Nếu u là một biểu thức phức tạp thì *Kind(u)* trả về toán tử nằm ở gốc của biểu thức.

Ví dụ:

$$Kind(3) \rightarrow integer$$

$$Kind(m * x + b) \rightarrow +$$

$$Kind((a+b)*sin(x^2)) \rightarrow *$$

$$Kind(2/3) \rightarrow fraction$$

$$Kind(sin(x)) \rightarrow sin$$

2.5.2 Định nghĩa toán tử $NumberOfOperands(u)$

Nếu u là một biểu thức phức hợp thì $NumberOfOperands(u)$ trả về số toán hạng của toán tử gốc của biểu thức. Nếu u không phải là biểu thức phức hợp thì toán tử sẽ trả về 0.

Ví dụ:

$$NumberOfOperands(m*x+b) \rightarrow 2$$

$$NumberOfOperands(n!) \rightarrow 1$$

$$NumberOfOperands(x) \rightarrow 0$$

2.5.3 Định nghĩa toán tử $Operand(u,i)$

Nếu u là một biểu thức phức hợp thì toán tử sẽ trả về toán hạng thứ i của u. Nếu u không phải là biểu thức phức tạp thì toán tử trả về **Undefined.**

Ví dụ:

$$Operand(m*x+b,2) \rightarrow b$$

$$Operand(x^2,1) \rightarrow x$$

$$Operand(x-x,1) \rightarrow Undefined$$

$$Operand(2/(-3),2) \rightarrow 3$$

2.6 Các toán tử dựa trên cấu trúc của biểu thức

2.6.1 Định nghĩa toán tử $CompleteSubExpression(u)$

Một biểu thức con đầy đủ của u là chính nó hoặc là một toán hạng của các toán tử trong u. Cho u là một biểu thức rút gọn toán tử $CompleteSubExpression(u)$ trả về một danh sách các biểu thức con đầy đủ của u.

Ví dụ: Cho $u = sin(a)*(1+b+c^2)$

$$CompleteSubExpression(u) \rightarrow sin(a)*(1+b+c^2), sin(a), a, 1+b+c^2, 1, b, c^2, c, 2.$$

2.6.2 Định nghĩa toán tử $FreeOf(u,t)$

Toán tử $FreeOf(u,t)$ sẽ xác định nếu biểu thức u không phụ thuộc vào biểu thức t (u không chứa t).

Cho u và t là các biểu thức toán học. Toán tử $FreeOf(u,t)$ trả về false nếu t là một biểu thức con đầy đủ của u và trả về True nếu t không phải là biểu thức con đầy đủ của u.

Ví dụ:

$$FreeOf(a + b, b) \rightarrow false,$$

$$FreeOf(a + b, c) \rightarrow true,$$

$$FreeOf((a + b + c) * d, a + b) \rightarrow true,$$

3 Thuật toán

3.1 Thuật toán toán học

Một thuật toán toán học nói chung là quá trình từng bước để giải quyết các vấn đề toán học, quá trình này có thể thực hiện bởi các chương trình máy tính.

Toán tử toán học trong thuật toán

Có một số toán tử được sử dụng trong các thuật toán là:

- Toán tử đại số : $+, -, *, /, \hat{}$ và !
- Toán tử quan hệ và toán tử logic: $=, <=, >=, <, >, and, or, not, \neq$.
- Toán tử tập hợp: $\cup$, $\cap, \sim$ và $\in$
- Toán tử danh sách: $First, Rest, Adjoin, Join$
- Toán tử cấu trúc cơ bản: $Kind, Operand, NumbeOfOpreand$
- Toán tử dựa trên cấu trúc: $FreeOf, CompleteSubExpression, Substitute$
- Toán tử cấu trúc của đa thức: $Degree, Coefficient$
- Toán tử thao tác đại số của đa thức: $Expand$
- Toán tử cấu trúc của biểu thức hữu tỉ: $Numerator, Denominator$
- Toán tử $Simplify$

3.2 Thuật toán đệ quy

Phần này sẽ tìm hiểu thuật toán đệ quy được sử dụng như thế nào trong đại số máy tính.

Định nghĩa đệ quy của toán tử giai thừa như sau:

$$n! = \begin{cases} 1, & nếu\ n = 0 \\ 1\ .\ 2\ \dots\ (n-1).\ n, & nếu\ n > 0 \end{cases}$$

Với n=4 quá trình tính toán dựa trên định nghĩa này sẽ diễn ra như sau:

$$4! = 4(3!) = 4(3(2!)) = 4(3(2(1!))) = 4(3(2(1(0!)))) = 4(3(2(1(1)))) = 24.$$

- Sau đây là thủ tục thực hiện thuật toán:

```
Procedure RecFact(n);
Input
        n : là số nguyên không âm;
Output
        n!;
Local Variables f;
Begin
        if n = 0 then
                f := 1
        else
                f := n ∗ RecFact(n − 1)
        Return(f)
End
```

Hình 3.1 Thuật toán đệ quy tìm giai thừa của một số nguyên không âm

Với n > 0 thủ tục sẽ gọi chính nó để thực hiện một phiên bản đơn giản của phép tính. Thủ tục gọi chính nó một cách trực tiếp hoặc thông qua một chuỗi các thủ tục được gọi là thủ tục đệ quy. Trường hợp $n = 0$ gọi là điều kiện kết thúc của thủ tục. Mỗi thủ tục đệ quy phải có một hoặc nhiều điều kiện kết thúc.

3.3 Thủ tục đệ quy

Phần này sẽ đưa ra một số ví dụ minh họa về việc sử dụng thuật toán đệ quy trong hệ thống đại số máy tính.

3.3.1 Toán tử $CompleteSubExpression$

- Thủ tục đệ quy thực hiện toán tử $CompleteSubExpression$

```
Procedure CompleteSubExpression(u);
Input
    u : một biểu thức toán học;
Output
    tập các biểu thức con đầy đủ của u;
Local Variables
    s, i;
Begin
    if Kind(u) ∈ {integer, ký hiệu} then
        Return({u})
    else
        s := {u};
        for i := 1 to NumberOfOperands(u) do
```

```
            s := s ∪ CompleteSubExpression(Operand(u, i));
        Return(s)
End
```

Hình 3.2 Thủ tục thực hiện toán tử $CompleteSubExpression$

3.3.2 Toán tử $FreeOf$

- Thủ tục thực thiện toán tử $FreeOf$

```
Procedure FreeOf (u, t);
Input
    u, t : là các biểu thức toán học;
Output
    true hoặc false;
Local Variables
    i;
Begin
    if u = t then
        Return(false)
    else if Kind(u) ∈ {symbol, integer} then
        Return(true)
    else
        i := 1;
        while i ≤ NumberOfOperands(u) do
            if not FreeOf(Operand(u, i), t) then
                Return(false);
            i := i + 1;
        Return(true)
End
```

Hình 3.3 Thủ tục thực hiện toán tử $FreeOf$

4 Rút gọn biểu thức

Quá trình rút gọn là một phần của quá trình tính toán giá trị được định nghĩa như một tập hợp các phép biến đổi rút gọn đại số và biến đổi lượng giác áp dụng cho biểu thức đại số.

4.1 Các phép biến đổi sử dụng trong quá trình rút gọn biểu thức

Các luật biến đổi trong quá trình rút gọn được xác định bởi các tiên đề và những hệ quả biến đổi logic của các tiên đề. Phần này sẽ trình bày các tiên đề cơ bản và vai trò của chúng trong quá trình rút gọn.

Phép phân phối: Trong rút gọn phép phân phối liên quan tới việc rút gọn phần hệ số là số nguyên hoặc phân số của các số hạng trong một tổng.

Tính chất phân phối có dạng: $(a + b) * c = a * c + b * c$

Phép kết hợp:

$$a + (b + c) = (a + b) + c$$

$$(a * b) * c = a * (b * c)$$

Tính chất kết hợp liên quan tới phép biến đổi làm thay đổi cấu trúc của biểu thức u bằng các cách sau:

1. Giả sử u là một tổng. Nếu s là toán hạng của u và cũng là một tổng thì toán tử của s sẽ bị loại bỏ khỏi cây biểu thức và s trở thành toán hạng chính của u. Phép chuyển đổi này sẽ được thực hiện trước phép phân phối.
2. Giả sử u là một tích. Nếu p là một toán tử của u và cũng là một tích thì toán tử của p sẽ bị loại bỏ khỏi cây biểu thức và p trở thành toán hạng chính của u. Phép biến đổi sẽ được thực hiện trước khi biến đổi lũy thừa.

Phép giao hoán:

$$a + b = b + a$$

$$a * b = b * a$$

Phép giao hoán liên quan đến phép biến đổi dựa trên các thuộc tính giao hoán của phép nhân và phép cộng. Phép biến đổi này sẽ sắp xếp lại các toán hạng trong một tổng hoặc một tích thành dạng chuẩn.

Biến đổi lũy thừa:

Các phép biến đổi sau sẽ được áp dụng trong quá trình rút gọn.

$$u^v * u^w \rightarrow u^{v+w}$$

$$(u^v)^n \rightarrow u^{v*n}$$

$$(u * v)^n \rightarrow u^n + v^n$$

Các phép biến đổi cơ bản khác:

Mỗi hiệu đơn phân được thay thế bởi tích: $-u = (-1) * u$

Mỗi hiệu nhị phân được thay thế bởi tổng: $u - v = u + (-1) * v$

Phép biến đổi thương cơ bản: $u/v = u * v^{(-1)}$

Các phép đồng nhất cơ bản

Các phép biến đổi cơ bản sau được áp dụng trong quá trình rút gọn:

$$u + 0 \rightarrow u,$$

$$u * 0 \rightarrow 0,$$

$$u * 1 \rightarrow u,$$

$$0^w \rightarrow \begin{cases} 0 \, nếu \, w \, là \, số \, nguyên \, dương \, hoặc \, phân \, số \\ Undefined \, trong \, các \, trường \, hợp \, còn \, lại \end{cases}$$

$$1^w \rightarrow 1,$$

$$v^0 \rightarrow \begin{cases} 1 \, nếu \, v \neq 0 \\ Undefined \, nếu \, v = 0 \end{cases}$$

$$v^1 \rightarrow \text{v}$$

$$\frac{u}{0} \rightarrow Undefined$$

$$\frac{0}{u} \rightarrow \begin{cases} 0 \, nếu \, u \neq 0 \\ Undefined \, nếu \, u = 0 \end{cases}$$

$$\frac{u}{1} \rightarrow u$$

Các biểu thức sau khi rút gọn phải thỏa mãn các thuộc tính sau:

1. Một tổng không thể có toán hạng 0.
2. Một tích không thể có toán hạng 0 hoặc 1.
3. Một lũy thừa không thể có cơ số hoặc số mũ là 0 hoặc 1.

Phép biến đổi đơn phân cơ bản: Các phép biến đổi đơn phân sau được áp dụng trong quá trình rút gọn:

$$* x \rightarrow x$$

$$+x \rightarrow x$$

Phép biến đổi Undefined: Nếu u là một biểu thức phức hợp với một toán hạng là ký hiệu Undefined thì sẽ được rút gọn thành Undefined.

4.1.1 Biểu thức đại số cơ bản và biểu thức đại số rút gọn

Định nghĩa 4.1: Biểu thức đại số cơ bản (Basic algebraic expression - BAE): u là một biểu thức đại số cơ bản nếu thỏa mãn một trong số các luật sau:

1. u là số nguyên.
2. u là phân số.
3. u là một ký hiệu.
4. u là một tích mà các toán hạng là các biểu thức đại số cơ bản.
5. u là một tổng mà các toán hạng là các biểu thức đại số cơ bản.
6. u là một thương mà hai toán hạng là các biểu thức đại số cơ bản.

7. u là hiệu đơn phân hoặc nhị phân mà các toán hạng là các biểu thức đại số cơ bản.
8. u là lũy thừa với hai toán hạng là các biểu thức đại số cơ bản.
9. u là giai thừa với toán hạng là biểu thức đại số cơ bản.
10. u là một hàm mà các toán hạng là các biểu thức đại số cơ bản.

Định nghĩa 4.2: Biểu thức đại số rút gọn: Một biểu thức rút gọn u (Simplified algebraic expression - SAE) được định nghĩa đệ quy như sau:

1. u là một một số nguyên.
2. u là phân số ở dạng rút gọn.
3. u là ký hiệu ngoại trừ Undefined.

Luật số 4 tiếp theo sẽ định nghĩa dạng rút gọn của một tích, trong đó sử dụng hai toán tử $base(u)$ và $exponent(u)$ để tìm cơ số và số mũ của một biểu thức rút gọn.

$$base(u) = \begin{cases} u \text{ nếu } u \text{ là một ký hiệu, tích, tổng, giai thừa, hàm số} \\ Operand(u,1) \text{ nếu } u \text{ là một lũy thừa} \\ Undifined \text{ nếu } u \text{ là một số nguyên hoặc phân số} \end{cases}$$

$$exponent(u) = \begin{cases} 1 \text{ nếu } u \text{ là một ký hiệu, tích, tổng, giai thừa, hàm số} \\ Operand(u,2) \text{ nếu } u \text{ là một lũy thừa} \\ Undifined \text{ nếu } u \text{ là một số nguyên hoặc phân số} \end{cases}$$

4. u là tích với hai hoặc nhiều toán hạng $u_1 u_2 \dots u_n$ thỏa mãn các thuộc tính sau:
   a. Mỗi toán hạng u_i là một biểu thức đại số rút gọn. Nó có thể là một số nguyên (khác 0 và 1), phân số, ký hiệu (khác undifined), tổng, lũy thừa, giai thừa hoặc hàm (toán hạng của một tích không thể là một tích).
   b. Tối đa một toán hạng là hằng số (số nguyên hoặc phân số).
   c. Nếu $i \neq j$ thì $base(u_i) \neq base(u_j)$.
   d. Nếu $i < j$ thì $u_i \lhd u_j$ (Toán tử $\lhd$ sẽ được nêu ra ở Định nghĩa 4.3).

Luật số 5 tiếp theo mô tả dạng rút gọn của một tổng trong đó sử dụng hai toán tử $term(u)$ và $const(u)$ để xác định phần số hạng và phần hằng số của một tích.

$$term(u) = \begin{cases} * \ u \text{ nếu } u \text{ là một ký hiệu, tích, tổng, giai thừa, hàm số.} \\ u \text{ nếu } u = u_1 \dots u_n \text{ và } u_1 \text{ không là hằng số.} \\ u_2 \dots u_n \text{ nếu } u = u_1 \dots u_n \text{ là tích và } u_1 \text{ là một hằng số.} \\ Undifined \text{ nếu } u \text{ là một số nguyên hoặc phân số.} \end{cases}$$

$$const(u) = \begin{cases} 1 \text{ nếu } u \text{ là một ký hiệu, tích, tổng, giai thừa, hàm số} \\ u_1 \text{ nếu } u = u_1 \dots u_n \text{ là tích và } u_2 \dots u_n \text{ không là hằng số} \\ 1 \text{ nếu } u \text{ là một số nguyên hoặc phân số} \end{cases}$$

5. u là một tổng của 2 hoặc nhiều toán hạng $u_1, u_2, \dots, u_n$ thỏa mãn một trong các thuộc tính sau:

   a. Mỗi toán hạng của u_i là một biểu thức đại số rút gọn có thể là số nguyên khác 0, phân số, ký hiệu (khác undefined), tích, lũy thừa, giai thừa hoặc hàm số.
   b. Nhiều nhất một toán hạng của u là hằng số.
   c. Nếu $i \neq j$ thì $term(u_i) \neq term\ (u_j)$.
   d. $i < j$ thì $u_i \lhd u_j$.
6. u là lũy thừa v^w thỏa mãn:
   a. v và w là các biểu thức đại số rút gọn.
   b. Số mũ w khác 0 và 1.
   c. Nếu w là số nguyên thì cơ số v là một biểu thức đại số rút gọn (ký hiệu, tổng, giai thừa, hoặc hàm số).
   d. Nếu w không là số nguyên thì cơ số v là bất kỳ biểu thức đại số rút gọn nào khác 0 và 1.
7. u là giai thừa với một toán hạng là một biểu thức đại số rút gọn bất kỳ ngoại trừ số nguyên âm.
8. u là một hàm với một hoặc nhiều toán hạng là các biểu thức đại số rút gọn.

Định nghĩa 4.3: Quan hệ thứ tự giữa các toán hạng trong một toán tử

Quan hệ thứ tự định nghĩa hành động của phép giao hoán trong quá trình rút gọn tổng hoặc tích của biểu thức, các toán hạng sẽ được sắp xếp thông qua các quan hệ này. Do toán hạng của các biểu thức được rút gọn đệ quy nên quan hệ thứ tự giữa chúng được xác định theo các luật dưới đây:

Có u và v là hai biểu thức đại số rút gọn khác nhau. Quan hệ thứ tự giữa hai biểu thức được ký hiệu là '$\lhd$' và được định nghĩa bởi các luật sau [13]:

1. Giả sử u và v là hai hằng số (số nguyên hoặc phân số)
$$u \lhd v \rightarrow u < v$$
2. Giả sử u và v là hai ký hiệu thì thứ tự được xác định theo thứ tự bảng chữ cái.
3. Giả sử cả u và v là tích hoặc cả hai là tổng với các toán hạng
$$u_1, u_2, \dots, u_m \text{ và } v_1, v_2, \dots, v_n$$
   - Nếu $u_m \neq v_n$ thì u $\lhd$ v $\rightarrow$ $u_m \lhd v_n$
   - Nếu có một số nguyên k với $1 < k < min(\{m, n\}) - 1$ và $u_{(m-j)} = v_{(n-j)}, j = 0, 1, .. k-1$ và $u_{(m-k)} \neq v_{(n-k)}$ thì
$$u \lhd n \rightarrow u_{(m-k)} \lhd v_{(n-k)}$$
   - Nếu $u_{(m-k)} = v_{(n-k)}$ với $k = 0,1, \dots, min(\{m, n\}) - 1$ thì
   $u \lhd n \rightarrow m < n$ (vì u và v là các biểu thức đại số rút gọn nên thỏa mãn $u_i \lhd u_j$ và $v_i \lhd v_j$ với $i < j$)
4. Giả sử u và v là 2 lũy thừa
   - Nếu $base(u) \neq base(v)$ thì
$$u \lhd v \rightarrow base(u) \lhd base(v)$$
   - Nếu $base(u) = base(v)$ thì
$$u \lhd v \rightarrow exponent(u) \lhd exponent(v)$$

Nói các khác nếu cơ số khác nhau thì thứ tự sẽ phụ thuộc vào cơ số. Nếu cơ số bằng nhau thì thứ tự phụ thuộc vào số mũ.

5. Nếu u và v cùng là giai thừa thì:

$$u \lhd v \rightarrow Operand(u,1) \lhd Operand(v,1)$$

6. Giả sử u và v là các hàm:
   - Nếu $Kind(u) \neq Kind(v)$ thì $u \lhd v \rightarrow Kind(u) \lhd Kind(v)$
   - Nếu $Kind(u) = Kind(v)$ và các toán hạng của hai hàm số lần lượt là $u_1, u_2 \dots u_m$ và $v_1, v_2 \dots v_n$
     - Nếu $u_1 \neq u_2$ thì $u \lhd v \rightarrow u_1 \lhd u_2$
     - Nếu có một số nguyên k với $1 \leq k \leq \min(\{m,n\})$ với $u_j = v_j$, $j = 1, \dots, k-1$ và $u_k \neq v_k$ thì $u \lhd v \rightarrow u_k \lhd v_k$
     - Nếu $u_k = v_k$ với $1 \leq k \leq min(\{m,n\})$ thì $u \lhd v \rightarrow m < n$

   Khi hai hàm có tên khác nhau thì thứ tự xác định bởi tên hàm. Cách so sánh tên hàm tương tự so sánh hai tích hoặc hai tổng

7. Nếu u là một số nguyên hoặc một phân số và v là các kiểu còn lại thì $u \lhd v$
8. Giả sử u là một tích. Nếu v là lũy thừa, tổng, giai thừa, hàm số, hoặc symbol thì:

$$u \lhd v \rightarrow u \lhd . v$$

   Nói cách khác thì luật này sẽ được xác định bằng cách xem cả hai biểu thức như là tích và áp dụng luật số 3.

9. Giả sử u là lũy thừa. Nếu *v* là một tổng, giai thừa, hàm số hoặc symbol thì thì:

$$u \lhd v \rightarrow u \lhd v^1$$

10. Giả sử *u* là một tổng. Nếu *v* là lũy thừa, hàm số hoặc symbol thì:

$$u \lhd v \rightarrow u \lhd +v$$

11. Giả sử *u* là một giai thừa. Nếu *v* là hàm số hoặc symbol thì:
    - Nếu $Operand(u,1) = v$ thì $u \lhd v \rightarrow$ false
    - Nếu $Operand(u) \neq v$ thì $u \lhd v \rightarrow u \lhd v!$
12. Giả sử u là một hàm số và v là symbol:
    - Nếu $Kind(u) = v$ thì $u \lhd v \rightarrow$ false
    - Nếu $Kind(u) \neq v$ thì $u \lhd v \rightarrow Kind(u) \lhd v$

13. Nếu u và v không thỏa mãn hết các trường hợp trên thì: $u \lhd v \rightarrow not(v \lhd u)$

4.1.2 Thể hiện của biểu thức đại số cơ bản

Lớp AnyNode

Lớp AnyNode được thiết kế dựa trên lý thuyết về cây biểu thức

- Các thuộc tính:

key : int	Toán tử của nút

leaf : ArrayList<AnyNode>	Các toán hạng của nút
value : int	Giá trị của nút
name : String	Tên nút

Bảng 4.1 Các thuộc tính của lớp AnyNode

- Các phương thức chính:

nop()	Trả về số lượng toán tử của nút
operand(int)	Trả về toán hạng thứ i của nút
numerator()	Trả về tử số của nút (nếu nút là số nguyên)
denominator()	Trả về mẫu của nút (nếu nút là số nguyên)
base()	Trả về cơ số của nút
exponent()	Trả về số mũ của nút
term()	Trả về phần số hạng của nút
constant()	Trả về phần hằng số của nút
kind(AnyNode)	Trả về loại của nút
compare(AnyNode)	Toán tử so sánh thứ tự của hai nút
equal(AnyNode)	Trả về true nếu hai nút giống nhau và ngược lại trả về false
substitute(AnyNode, AnyNode)	Phương thức thay thế

Bảng 4.2 Các phương thức chính của lớp AnyNode

Lớp Bae

Lớp Bae là lớp được thiết kế dựa trên các đặc điểm của biểu thức đại số cơ bản.

- Các thuộc tính:

list : ArrayList<AnyNode>	
node : AnyNode	Nút gốc của cây

Bảng 4.3 Các thuộc tính của lớp Bae

- Các phương thức:

Bae()	Hàm khởi tạo không tham số
Bae(ExpressionProgram)	Hàm khởi tạo với tham số là một biểu thức được lưu trữ dưới dạng mảng
Bae(String)	Hàm khởi tạo với tham số là một chuỗi ký tự
createBae(ExpressionProgram)	Hàm tạo ra nút gốc của cây biểu thức

Bảng 4.4 Các phương thức chính của lớp BAE

Phương thức createBae(ExpressionProgram) có tham số đầu vào là một đối tượng ExpressionProgram, ExpressionProgram sẽ phân tích để tạo ra một biểu thức đại số cơ bản sau đó được xử lý bằng thuật toán rút gọn thành biểu thức rút gọn.

```
private void createBae(ExpressionProgram exprog){
      int length = exprog.progCt;
      for (int i = 0; i < length; i++) {
```

```
            if (exprog.prog[i] >= 0) {
                  if (exprog.prog[i] < CEILING) {
                        // node la so nguyen
                        AnyNode temp = new AnyNode(0,(int)
exprog.constant[exprog.prog[i]]);
                        list.add(temp);
                  } else {
                        String name = "" + exprog.command[exprog.prog[i]
- CEILING];
                        list.add(new AnyNode(CEILING, name));
                  }
            } else {
                  AnyNode temp = new AnyNode();
                  temp = determineAnyNode(exprog.prog[i]);
                  temp.sort();
                  list.add(temp);
            }
      }
      node = Simplify.simplify(list.get(0));
      list.clear();
}
```

Hình 4.1 Phương thức tạo nút gốc của lớp Bae.

4.2 Thuật toán rút gọn

Thuật toán rút gọn được thực hiện dựa trên các phép biến đổi cơ bản và các định nghĩa đã nêu ở các phần trước. Thuật toán bao gồm các bước sau:

1. Nếu u là một biểu thức đại số cơ bản thuật toán sẽ trả về một biểu thức đại số rút gọn.
2. Nếu u là một biểu thức đại số rút gọn thì thuật toán trả ra u.

4.2.1 Thủ tục rút gọn chính

Procedure $Simplify$(u);
Input
    u: a là một biểu thức đại số cơ bản dưới dạng ký hiệu;
Output
    Một biểu thức đại số rút gọn dưới dạng ký hiệu hàm hoặc
    là biểu tượng Undefined
Local Variables
    v;
Begin
    if ($Kind$(u) = integer or $Kind$(u) = symbol) then
        Return(u);
    else if (Kind(u) = FracOp) then
        Return($SimplifyRationalNumber$(u))
    else
        v = Map($Simplify$, u);
    if ($Kind$(v) = PowOp) then

```
            Return(SimplifyPower(v))
        else if (Kind(v) = ProdOp) then
            Return(SimplifyProduct(v))
        else if (Kind(v) = SumOp) then
            Return(SimplifySum(v))
        else if (Kind(v) = FactOp) then
            Return(SimplifyFactorial(v))
        else
            Return(SimplifyFunction(v))
End
```

Hình 4.2 Thủ tục rút gọn chính

Toán tử Map sẽ được trình bày trong phần Phụ lục 1

4.2.2 Rút gọn biểu thức số hữu tỉ

Thuật toán trong Hình 4.3 sử dụng để tính toán biểu thức số học với số nguyên và phân số. Đầu vào của thuật toán được mô tả trong Định nghĩa 4.4 dưới đây.

Định nghĩa 4.4: Một biểu thức đại số u là một biểu thức số hữu tỉ nếu:

1. u là số nguyên.
2. u là phân số.
3. u là tổng nhị phân mà các toán hạng là các biểu thức số hữu tỉ.
4. u là tích với các toán hạng là các biểu thức số hữu tỉ.
5. u là lũy thừa với cơ số là biểu thức số hữu tỉ và số mũ là một số nguyên.

Để thuận tiện cho việc trình bày thủ tục rút gọn thì các toán tử đại số và toán tử phân số sẽ được thay thế bằng dạng ký hiệu hàm. Mỗi toán tử trung tố sẽ được thay bởi tên và các toán hạng tương ứng, mỗi tên sẽ được thêm hậu tố Op (viết tắt của Operator). Toán tử $DenominatorFun(u)$ và $NumeratorFun(u)$ sẽ trả về mẫu số và tử số của u.

- Thủ tục thực hiện toán tử $SimplifyRNE$

```
Procedure SimplifyRNE(u);
Input
    u : là một biểu thức số hữu tỉ;
Output
    là một số nguyên, một phân số dạng chuẩn hoặc ký hiệu Undefined;
Local Variables
    v, w;
Begin
    if Kind(u) = integer then Return(u)
    else if Kind(u) = FracOp then
        return SimplifyRationalNumber(u);
```

```
        else NumberOfOperands(u)=2 then
                v := SimplifyRNE(Operand(u, 1));
                w := SimplifyRNE(Operand(u, 2));
                if v = Undefined or w = Undefined then
                        return Undefined;
                else if Kind(u) = ProdOp then
                        return EvaluateProduct(v,w);
                else if Kind(u) = SumOp then
                        return EvaluateSum(v,w);
                elseif Kind(u) = PowOp then
                        return EvaluatePower(u);
End
```

Hình 4.3 Thủ tục thực hiện toán tử $SimplifyRNE$

Các thủ tục $EvaluateProduct, EvaluateSum, EvaluatePower$ sẽ được trình bày trong phần Phụ lục 3, Phụ lục 4, Phụ lục 5.

4.2.3 Rút gọn lũy thừa

Toán tử $SimplifyPower$ sẽ biến đổi v^m thành biểu thức đại số rút gọn hoặc ký hiệu Undefined.

Định nghĩa 4.5: $u = v^w$ vậy cơ số $v = Operand(u, 1)$ và lũy thừa $w = Operand(u, 2)$ là các biểu thức đại số rút gọn hoặc ký hiệu $undefined$. Toán tử $SimplifyPower(u)$ định nghĩa bởi các luật biến đổi sau:

1. Nếu $v = undefined$ hoặc $w = undefined$ thì:
$$SimplifyPower(u) \rightarrow undefined$$
2. Giả sử $v = 0$
   - Nếu w là số nguyên dương hoặc là phân số thì:
   $$SimplifyPower(u) \rightarrow 0$$
   - Trong các trường hợp khác thì:
   $$SimplifyPower(u) \rightarrow undefined$$
3. Nếu $v = 1$ thì
$$SimplifyPower(u) \rightarrow 1$$
4. Nếu w là một số nguyên thì:
$$SimplifyPower(u) \rightarrow SimplifyIntegerPower(v, w)$$
(Toán tử $SimplifyIntegerPower$ được nêu ra ở Định nghĩa 4.6)
5. Nếu các luật trên không được áp dụng thì
$$SimplifyPower(u) \rightarrow u$$

Định nghĩa 4.6: v^n với v khác 0 và là biểu thức đại số rút gọn, n là một số nguyên. Toán tử $SimplifyIntegerPower(v, n)$ được định nghĩa bởi các luật sau:

1. Nếu $Kind(v)$ là số nguyên hoặc phân số thì:

$$SimplifyIntegerPower(v,n) \rightarrow SimplifyRNE(PowOp(v,n)).$$

2. Nếu n bằng 0 thì $SimplifyIntegerPower(v,n) \rightarrow 1$
3. Nếu n bằng 1 thì $SimplifyIntegerPower(v,n) \rightarrow v$
4. Giả sử $Kind(v) = PowOp$ với cơ số $r = Operand(v,1)$ và lũy thừa

   $s = Operand(v,2)$ khi đó cho $p = SimplifyProduct(ProdOp(s,n))$

   - Nếu p là số nguyên thì:
     $$SimplifyIntegerPower(v,n) \rightarrow SimplifyIntegerPower(r,p)$$
   - Nếu p không là số nguyên
     $$SimplifyIntegerPower(v,n) \rightarrow PowOp(r,p)$$
5. Giả sử rằng $Kind(v) = ProdOp$ và
   $r = Map(SimplifyIntegerPower, v, n)$ thì:
   $$SimplifyIntegerPower(v,n) \rightarrow SimplifyProduct\ (r).$$

Luật này áp dụng khi v là một tích (toán tử Map sẽ được trình bày trong phần Phụ lục 1)

6. Nếu không có luật nào được áp dụng thì:

$$SimplifyIntegerPower(v,n) \rightarrow PowOp(v,n)$$

4.2.4 Rút gọn tích

Toán tử $SimplifyProduct$ sẽ biến đổi một tích thành một biểu thức đại số rút gọn hoặc ký hiệu $Undefined$ dựa trên các phép biến đổi sau: biến đổi kết hợp, giao hoán, biến đổi lũy thừa, tính đồng nhất, phép biến đổi nhị phân, biến đổi cơ số, biến đổi về Undefined

Định nghĩa 4.7: Cho u là một tích với một hoặc nhiều toán hạng là các biểu thức đại số rút gọn, cho $L = [u_1, u_2, \ldots, u_n]$ là một danh sách các toán hạng của u. Toán tử $SimplifyProduct\ (u)$ được định nghĩa bởi các luật sau:

1. Nếu Undefined ∈ L thì: $SimplifyProduct(u) \rightarrow Undefined$
2. Nếu 0 ∈ L thì: $SimplifyProduct\ (u) \rightarrow 0$
3. Nếu $L = [u_1]$ thì : $SimplifyProduct\ (u) \rightarrow u_1$
4. Giả sử rằng ba luật trên không áp dụng thì cho : $v = SimplifyProductRec(L)$

Trong đó toán tử $SimplifyProductRec$ (Định nghĩa 4.8) trả về một danh sách không có hoặc có nhiều toán hạng và nếu v có hai hoặc nhiều toán hạng thì phải thỏa mãn luật số 4 trong Định nghĩa 4.2. Có ba trường hợp như sau:

- Nếu v = $[v_1]$ thì
  $$SimplifyProduct\ (u) \rightarrow v_1$$
- Nếu $v = [v_1, v_2, \ldots, v_s]$ với $2 \leq s$ thì:

$$SimplifyProduct\ (u) \rightarrow Construct(*, v)$$

- Nếu $v = [\,]$ thì:

$$SimplifyProduct\ (u)\ \rightarrow\ 1$$

(Chú ý: Toán tử Construct được mô tả ở Phụ lục 2).

Định nghĩa 4.8: Cho L = $[u_1, u_2, \ldots, u_n]$ là một danh sách không rỗng với $n \geq 2$. Toán tử $SimplifyProductRec(L)$ trả về một danh sách gồm không hoặc nhiều toán hạng. Toán tử trên định nghĩa bởi các luật sau:

1. Giả sử rằng L = $[u_1, u_2]$ và không có toán hạng nào là tích.
   1.1. Giả sử rằng cả hai toán hạng là hằng số và

$$P = SimplifyRNE(ProdOp(u_1, u_2))$$

   - Nếu $P = 1$ thì $SimplifyProductRec(L)) \rightarrow [\,]$
   - Nếu $P \neq 1$ thì $SimplifyProductRec$(L) $\rightarrow$ [P]

   1.2. Nếu u_1 = 1 thì: $SimplifyProductRec(L) \rightarrow [u_2]$
   1.3. Nếu u_2 = 1 thì: $SimplifyProductRec(L) \rightarrow [u_1]$
   1.4. Giả sử rằng base(u_1) = base(u_2) và cho

$$S = SimplifySum(SumOp(exponent(u_1), exponent(u_2)))$$

   Và $P = SimlifyPower(PowOp(base(u_1), S))$

   - Nếu $P = 1$ thì $SimplifyProductRec(L)\ ) \rightarrow [\,]$
   - Nếu $P \neq 1$ thì $SimplifyProductRec(L) \rightarrow [P]$

   (Toán tử $SimplifySum$ sẽ được trình bày ở phần 4.2.5)

   1.5. Nếu $u_2 \lhd u_1$ thì:

$$SimplifyProductRec(L) \rightarrow [u_2, u_1]$$

   1.6. Nếu năm luật trên không thể áp dụng thì:

$$SimplifyProductRec(L) \rightarrow [L]$$

2. Giả sử $L = [u_1, u_2]$ trong đó ít nhất một toán hạng là một tích
   2.1. Nếu u_1 là một tích với các toán hạng p_1, p_2,..., p_s và u_2 là một tích với các toán hạng q_1, q_2,..., q_t thì:

$$SimplifyProductRec(L) \rightarrow MergeProducts([p_1, p_2, \ldots, p_s], [q_1, q_2, \ldots, q_t])$$

   ($MergeProducts$ được nêu ra ở Định nghĩa 4.9)

   2.2. Nếu u_1 là tích với các toán hạng $p_1, p_2, \ldots, p_s$ và u_2 không là tích thì:

$$SimplifyProductRec(L) \rightarrow MergeProducts([p_1, p_2, \ldots, p_s], [u_2])$$

   2.3. Nếu u_2 là tích với các toán hạng $q_1, q_2, \ldots, q_t$ và u_1 không là tích thì:

$$SimplifyProductRec(L)\ \rightarrow\ MergeProducts([u_1], [q_1, q_2, \ldots, q_t])$$

3. Giả sử $L = [u_1, u_2, \ldots, u_n]$ với $n > 2$ và $w = SimplifyProductRec(Rest(L))$

3.1. Nếu u_1 là tích với các toán hạng $p_1, p_2, \ldots, p_s$ thì:

$$SimplifyProductRec(L) \rightarrow MergeProducts([p_1, p_2, \ldots, p_s], w)$$

3.2. Nếu u_1 không là tích thì:

$$SimplifyProductRec(L) \rightarrow MergeProducts(([u_1], w)$$

Định nghĩa 4.9: Cho p và q là các danh sách của không hoặc nhiều nhân tử đã được sắp xếp theo thứ tự $\lhd$. Nếu p hoặc q có hai hoặc nhiều toán hạng thì các toán hạng này thỏa mãn luật số 4 trong Định nghĩa 4.2. Toán tử $MergeProducts(p, q)$ được định nghĩa bởi các luật sau:

1. Nếu $q = []$ thì $MergeProducts(p, q) \rightarrow p$
2. Nếu $p = []$ thì $MergeProducts(p, q) \rightarrow q$
3. Giả sử p và q là các danh sách không rỗng, $p_1 = First(p)$ và $q_1 = First(q)$.
   $h = SimplifyProductRec([p_1, q_1])$

   Có 4 khả năng có thể sảy ra:

   3.1. Nếu $h = []$ thì:

   $$MergeProducts(p, q) \rightarrow MergeProducts(Rest(p), Rest(q))$$

   3.2. Nếu $h = [h_1]$ thì:

   $$MergeProducts(p, q) \rightarrow Adjoin(h_1, MergeProducts(Rest(p), Rest(q)))$$

   3.3. Nếu $h = [p_1, q_1]$ thì:

   $$MergeProducts(p, q) \rightarrow Adjoin(p_1, MergeProducts(Rest(p), q))$$

   3.4. Nếu $h = [q_1, p_1]$ thì:

   $$MergeProducts(p, q) \rightarrow Adjoin(q_1, MergeProducts(p, Rest(q)))$$

4.2.5 Rút gọn tổng

Toán tử $SimplifySum$ sẽ biến đổi một tổng thành một biểu thức đại số rút gọn hoặc ký hiệu $Undefined$.

Định nghĩa 4.10: Cho u là một tổng có danh sách các toán hạng $L = [u_1, u_2, \ldots, u_n]$ là các biểu thức đại số rút gọn. Rút gọn tổng u được định nghĩa bởi các luật sau:

1. Nếu $Undefined \in L$ thì:

$$SimplifySum(u) \rightarrow Undefined$$

2. Nếu L = $[u_1]$ thì:

$$SimplifySum(u) \rightarrow [u_1]$$

3. Giả sử 2 luật trên không thể áp dụng thì :

$$v = SimplifySumRec(L)$$

($SimplifySumRec$ được nêu ra ở **Định nghĩa 4.11**)

3.1. Nếu $v = [v_1]$ thì:

$$SimplifySumRec(L) \rightarrow v_1$$

3.2. Nếu $v = [v_1, v_2, \ldots, v_n]$ thì:

$$SimplifySumRec(L) \rightarrow Construct(+, v)$$

3.3. Nếu $v = []$ thì:

$$SimplifySumRec(L) \rightarrow []$$

(Chú ý: Toán tử Construct được mô tả ở Phụ lục 2).

Định nghĩa 4.11: Cho $L = [u_1, u_2, \ldots, u_n]$ là một danh sách không rỗng với $n \geq 2$. Toán tử $SimplifySumRec(L)$ trả về một danh sách gồm không hoặc nhiều toán hạng. Toán tử trên định nghĩa bởi các luật sau:

1. Giả sử $L = [u_1, u_2]$ và cả hai không là tổng
   1.1. Cả hai là hằng số và $p = SimplifyRNE(u_1, u_2)$
   - Nếu $p = 0$ thì $SimplifySumRec(L) \rightarrow []$
   - Nếu $p \neq 0$ thì $SimplifySumRec(L) \rightarrow [p]$

   1.2. Nếu $u_1 = 0$ thì $SimplifySumRec(L) \rightarrow [u_2]$
   1.3. Nếu $u_2 = 0$ thì $SimplifySumRec(L) \rightarrow [u_1]$
   1.4. Nếu $term(u_1) = term(u_2)$
   và $s = SimplifyRNE(const(u_1), const(u_2))$
   - Nếu $s = 0$ thì $SimplifySumRec(L) \rightarrow []$
   - Nếu $s \neq 0$ thì $SimplifySumRec(L) \rightarrow [s * term(u_1)]$

   1.5. Nếu $u_2 \lhd u_1$ thì: $SimplifySumRec(L) \rightarrow [u_2, u_1]$
   1.6. Nếu năm luật trên không áp dụng thì: $SimplifySumRec(L) \rightarrow L$
2. Giả sử $L = [u_1, u_2]$ và ít nhất một trong hai là tổng
   2.1. Cả hai u_1 và u_2 là tổng. Các toán hạng của u_1 là $p_1, p_2, \ldots, p_s$ và các toán hạng của u_2 là $q_1, q_2, \ldots, q_t$

$$SimplifySumRec(L) \rightarrow MergeSums([p_1, p_2, \ldots, p_s], [q_1, q_2, \ldots, q_t])$$

   2.2. u_1 là tổng và u_2 không là tổng

$$SimplifySumRec(L) \rightarrow MergeSums([p_1, p_2, \ldots, p_s], [u_2])$$

   2.3. u_2 là tổng và u_1 không là tổng

$$SimplifySumRec(L) \rightarrow MergeSums([u_1], [q_1, q_2, \ldots, q_t])$$

Định nghĩa 4.12: Cho p và q là các danh sách của không hoặc nhiều toán hạng đã được sắp xếp theo thứ tự $\lhd$. Nếu p hoặc q có hai hoặc nhiều toán hạng thì các toán hạng này thỏa mãn luật số 4 trong Định nghĩa 4.2. Toán tử $MergeSums(p, q)$ được định nghĩa bởi các luật sau:

1. Nếu $p = []$ thì $MergeSums(p, q) \rightarrow q$
2. Nếu $q = []$ thì $MergeSums(p, q) \rightarrow p$
3. Nếu p và q đều khác rỗng và $p_1 = First(p)$, $q_1 = First(q)$,

$$h = SimplifySumRec([p_1, q_1])$$

3.1. Nếu $h = []$ thì:

$$MergeSums(p, q) \rightarrow MergeSums(Rest(p), Rest(q))$$

3.2. Nếu $h = [h_1]$ thì:

$$MergeSums(p, q) \rightarrow Adjoin(h_1, MergeSums(Rest(p), Rest(q)))$$

3.3. Nếu $h = [p_1, q_1]$ thì:

$$MergeSums(p, q) \rightarrow Adjoin(p_1, MergeSums(Rest(p), q))$$

3.4. Nếu $h = [q_1, p_1]$ thì:

$$MergeSums(p, q) \rightarrow Adjoin(q_1, MergeSums(p, Rest(q)))$$

4.3 Thể hiện của thuật toán rút gọn

Lớp Simplify được thiết kế bao gồm các phương thức tĩnh sử dụng để rút gọn biểu thức.

4.3.1 Phương thức rút gọn biểu thức số hữu tỉ

Phương thức $simplifyRNE$ được thiết kế dựa trên các định nghĩa và các thủ tục giả mã được nêu ở 4.2.2 nhằm mục đích rút bọn hiểu thức số hữu tỉ.

- Phương thức $simplifyRNE$

```
public static AnyNode simplifyRNE(AnyNode anyNode) {
	if (anyNode.getKey() == 0) // node la so nguyen
		return anyNode;
	else if (anyNode.getKey() == -37) { // node la phan so
		return simplifyRationalNumber(anyNode);
	} else if (anyNode.nop() == 2) { // node la bieu thuc so huu ti
		AnyNode v = simplifyRNE(anyNode.operand(1));
		AnyNode w = simplifyRNE(anyNode.operand(2));
		if (v.getKey() == -38 || w.getKey() == -38) {
			return new AnyNode(-38);
		} else if (anyNode.getKey() == -5) { // node la luy thua
			return evaluatePower(v, w);
		} else if (anyNode.getKey() == -3) { // node la tich
			return evaluateProduct(v, w);
		} else if (anyNode.getKey() == -1) { // node la mot tong
			return evaluateSum(v, w);
		}
	}
	return anyNode;
}
```

Hình 4.4 Phương thức $simplifyRNE$

Các phương thức *simplifyRationalNumber*, *evaluateProduct*, *evaluateSum* và *evaluatePower* được trình bày trong Phụ lục 18, Phụ lục 6, Phụ lục 7, Phụ lục 8.

4.3.2 Phương thức rút gọn lũy thừa

Phương thức *simplifyPower* được thiết kế dựa trên các định nghĩa đã được nêu ở 4.2.2 để rút gọn biểu thức lũy thừa.

- Phương thức *simplifyPower*

```
private static AnyNode simplifyPower(AnyNode anyNode) {
	AnyNode base = anyNode.leaf.get(0); // co so
	AnyNode exponent = anyNode.leaf.get(1); // so mu
	if (base.getKey() == -38 || exponent.getKey() == -38)
		return new AnyNode(-38);
	else if (base.getValue() == 0 && base.getKey() == 0)// co so bang 0
		return new AnyNode(0, 0);
	else if (base.getValue() == 1 && base.getKey() == 0)// co so bang 1
		return new AnyNode(0, 1);
	else if (exponent.getKey() == 0) // neu so mu la so nguyen
		return simplifyIntegerPower(anyNode);
	else
		return anyNode;
}
```

Hình 4.5 Phương thức *simplifyPower*

Phương thức *simplifyIntegerPower* được trình bày trong Phụ lục 9.

4.3.3 Phương thức rút gọn tích

Phương thức *simplifyProduct* được thiết kế dựa trên các định nghĩa đã được nêu ở 4.2.4 để rút gọn một tích.

- Phương thức *simplifyProduct*

```
private static AnyNode simplifyProduct(AnyNode anyNode) {
	if (anyNode.nop() == 1)
		return anyNode.operand(1);
	for (int i = 1; i <= anyNode.nop(); i++) {
		if (anyNode.operand(i).getKey() == -38) {
			return new AnyNode(-38);
		} else if (anyNode.operand(i).getValue() == 0
				&& anyNode.operand(i).getKey() == 0) {
			return new AnyNode(0, 0);
		}
	}
	ArrayList<AnyNode> temp = simplifyProductRec(anyNode.leaf);
	if (temp == null || temp.size() == 0)
		return new AnyNode(0, 1);
	else if (temp.size() == 1)
		return temp.get(0);
	else {
		anyNode.leaf = temp;
```

```
            return anyNode;
      }
}
```

Hình 4.6 Phương thức *simplifyProduct*

Các phương thức *simplifyProductRec* và *mergeProducts* được trình bày trong Phụ lục 10, Phụ lục 11.

4.3.4 Phương thức rút gọn tổng

Phương thức *simplifySum* được thiết kế dựa trên các định nghĩa đã được nêu ở 4.2.5 để rút gọn một tổng.

- Phương thức *simplifySum*

```
private static AnyNode simplifySum(AnyNode anyNode) {
      if (anyNode.nop() == 1)
            return anyNode.operand(1);
      for (int i = 0; i < anyNode.nop(); i++) {
            if (anyNode.operand(i + 1).getKey() == -38)
                  return new AnyNode(-38);
      }
      ArrayList<AnyNode> temp = simplifySumRec(anyNode.leaf);
      if (temp == null || temp.size() == 0)
            return new AnyNode(0, 0);
      else if (temp.size() == 1)
            return temp.get(0);
      else {
            anyNode.leaf = temp;
            return anyNode;
      }
}
```

Hình 4.7 Phương thức *simplifySum*

Các phương thức *simplifySumRec* và *mergeSums* được trình bày trong Phụ lục 12, Phụ lục 13.

4.3.5 Phương thức rút gọn chính

Phương thức *Simplify* là phương thức chính sử dụng để rút gọn biểu thức cơ bản về dạng biểu thức rút gọn.

```
public static AnyNode simplify(AnyNode anyNode) {
      if (anyNode.getKey() == 0 || anyNode.getKey() == CEILING) {
            return anyNode;
      } else if (anyNode.getKey() == -37) {
            return simplifyRationalNumber(anyNode);
      } else {
            AnyNode temp = new AnyNode(anyNode.getKey(),
anyNode.getName());
            for (int i = 0; i < anyNode.nop(); i++) {
                  AnyNode tempLeaf = simplify(anyNode.operand(i + 1));
                  temp.leaf.add(tempLeaf);
            }
            if (temp.getKey() == -5)
```

```
                return simplifyPower(temp);
            else if (temp.getKey() == -3)
                return simplifyProduct(temp);
            else if (temp.getKey() == -1)
                return simplifySum(temp);
            else if (temp.getKey() == -16)
                return simplifyFactorial(temp);
            else if (temp.getKey() <= -17 && temp.getKey() >= -22)
                return simplifyTrigonometry(temp);
            else if (temp.getKey() == -28)
                return simplifyExp(temp);
            else if (temp.getKey() == -29)
                return simplifyLn(temp);
            else if (temp.getKey() < -22 && temp.getKey() >= -36)
                return temp;
            return anyNode;
        }
}
```

Hình 4.8 Phương thức $Simplify$

Phương thức simplifyFactorial được trình bày trong Phụ lục 16.

5 Cấu trúc của đa thức và biểu thức hữu tỉ

Phần này của luận văn mô tả cấu trúc đa thức và cấu trúc hữu tỉ của một biểu thức đại số. Phần đa thức có một số định nghĩa tổng quát về đa thức một biến (5.1), đa thức nhiều biến (5.2) và đa thức tổng quát (5.3). Mỗi định nghĩa sẽ có các thủ tục để xác định cấu trúc đa thức của một biểu thức. Phần cuối cùng (5.4) sẽ mô tả cấu trúc hữu tỉ của một biểu thức đại số và một thuật toán biến đổi biểu thức về dạng hữu tỉ.

5.1 Đa thức một biến

5.1.1 Phân tích

Định nghĩa 5.1: (Định nghĩa toán học) u là đa thức một biến x là một biểu thức có dạng:

$$u = u_n x^n + u_{n-1} x^{n-1} + \cdots + u_1 x + u_0$$

với hệ số u_i là số hữu tỉ và n là một số nguyên không âm. Nếu $u_n \neq 0$ thì u_n gọi là hệ số đầu tiên của u và n là bậc của u. Biểu thức $u = 0$ gọi là đa thức 0, nó có hệ số đầu tiên bằng 0 và bậc được quy ước là $-\infty$. Hệ số đầu tiên tương ứng với $lc(u, x)$ và bậc tương ứng $deg(u, x)$. Khi biến x được hiểu là hiển nhiên thì có thể dùng ký hiệu $lc(u)$ và $deg(u)$.

Ví dụ: $u = 3x^6 + 2x^4 - 5/2, \qquad deg(u) = 6, lc(u) = 3$

Định nghĩa 5.2: Một đơn thức một biến x là một biểu thức đại số u thỏa mãn một trong các tính chất sau:

1. u là một số nguyên hoặc phân số.
2. $u = x$.

3. $u = x^n$ với n là số nguyên và $n > 1$.
4. u là một tích với hai toán hạng thỏa mãn các tính chất 1, 2, 3.

Định nghĩa 5.3: Một đa thức một biến x là một biểu thức thỏa mãn một trong các tính chất sau:

1. u là một đơn thức một biến x.
2. u là một tổng và mỗi toán hạng trong u là một đơn thức một biến x.

Các toán tử cơ bản của đa thức một biến

Định nghĩa 5.4: Cho u là một biểu thức đại số

- Toán tử $MonomialSV(u, x)$ trả về true nếu u là một đơn thức một biến x và trả về false nếu ngược lại.
- Toán tử $PolynomialSV(u, x)$ trả về true nếu u là một đa thức một biến x và trả về false nếu ngược lại.

Giả mã các thủ tục thực hiện hai toán tử:

- $MonomialSV$

```
Procedure MonomialSV(u, x)
Input
        u : là một biểu thức đại số;
        x : là một biến;
Output
        true or false;
Local Variables
        base, exponent;
Begin
        if Kind(u) ∈ {integer, fraction} then
                Return(true);
        else if u = x then
                Return(true);
        else if Kind(u)=" ∧ " then
                base := Operand(u, 1);
                exponent := Operand(u, 2);
                if (base = x and Kind(exponent) = integer
                                and exponent > 1) then
                        Return(true);
                else if (Kind(u)=" ∗ ") then
                        Return(NumberOfOperands(u)=2 and
                                        MonomialSV(Operand(u, 1), x)
                                        and MonomialSV(Operand(u, 2), x));
                Return(false);
```

```
End
```

Hình 5.1 Thủ tục thực hiện toán tử $MonomialSV$

- $PolynomialSV$

```
Procedure PolynomialSV(u, x);
Input
        u : là một biểu thức đại số;
        x : là một biến;
Output
        true or false;
Local Variables i;
Begin
        if MonomialSV (u, x) then
                Return(true);
        else if Kind(u)="+" then
                for i := 1 to NumberOfOperands(u) do
                        if MonomialSV(Operand(u, i), x)= false then
                                Return(false);
                        Return(true);
                Return(false);
End
```

Hình 5.2 Thủ tục thực hiện toán tử $PolynomialSV$

Toán tử DegreeSV

Định nghĩa 5.5: Cho u là một biểu thức đại số. Nếu u là một đa thức một biến x thì $DegreeSV(u, x)$ sẽ trả về bậc của u theo biến x. Nếu u không là đa thức theo biến x thì toán tử trả về ký hiệu $Undefined$.

Ví dụ:

$$DegreeSV(3x^2 + 4x + 5, x) \rightarrow 2,$$

$$DegreeSV(2x^3, x) \rightarrow 3,$$

$$DegreeSV((x + 1)(x + 3), x) \rightarrow Undefined,$$

$$DegreeSV(3, x) \rightarrow 0$$

- Thủ tục thực hiện toán tử $DegreeMonomialSV(u, x)$ sẽ trả về bậc của đơn thức u theo biến x

```
Procedure DegreeMonomialSV(u, x);
Input
        u : là một biểu thức đại số;
        x : là một biến;
```

```
Output
        Bậc của u theo x hoặc ký hiệu Undefined;
Local Variables
        base, exponent, s, t;
Begin
        if u = 0 then
                Return(Undefined)
        else if Kind(u) ϵ {integer,fraction} then
                Return(0);
        else if u = x then
                Return(1);
        Else if Kind(u)=”^” then
                base := Operand(u, 1);
                exponent := Operand(u, 2);
                if base = x and Kind(exponent) = integer and exponent > 1 then
                        Return(exponent);
                elseif Kind(u)=” * ” then
                        if NumberOfOperands(u)=2 then
                                s := DegreeMonomialSV(Operand(u, 1), x);
                                t := DegreeMonomialSV(Operand(u, 2), x);
                                if s≠Undefined and t ≠ Undefined then
                                        Return(t);
        Return(Undefined);
End
```

Hình 5.3 Thủ tục thực thực hiện toán tử $DegreeMonomialSV$

- Thủ tục thực hiện toán tử $DegreeSV(u, x)$

```
Procedure DegreeSV(u, x);
Input
        u : là một biểu thức đại số;
        x : ký hiệu;
Output
        bậc của u theo biến hoặc ký hiệu undefined;
Local Variables
        d, i, f;
Begin
        d := DegreeMonomialSV(u, x);
        if d ≠ Undefined then
                Return(d);
        elseif Kind(u)=”+” then
                d := 0;
```

```
            for i := 1 to NumberOfOperands(u) do
                f := DegreeMonomialSV(Operand(u, i), x);
                if f = Undefined then
                    Return(Undefined);
                else
                    d := Max({d, f})
                    Return(d);
        Return(Undefined);
End
```

Hình 5.4 Thủ tục thực thực hiện toán tử $DegreeSV$

Toán tử CoefficientSV

Định nghĩa 5.6: Cho u là một biểu thức đại số. Nếu u là một đa thức một biến x, toán tử $CoefficientSV(u, x, j)$ trả về hệ số u_j của x^j. Nếu $j > deg(u, x)$ thì $CoefficientSV$ trả về 0. Nếu u không là đa thức biến x thì toán tử trả về ký hiệu Undefined.

Ví dụ:

$$CoefficientSV(x^2 + 3x + 5, x, 1) \rightarrow 3$$

$$CoefficientSV(2x^3 + 3x, x, 4) \rightarrow 0$$

$$CoefficientSV(3, x, 0) \rightarrow 3$$

$$Coefficient\ sv((x + 1)(x + 3), x, 2) \rightarrow Undefined$$

Toán tử $CoefficientMonomialSV$ trả về hệ số của biến x^m của đơn thức u.

- Thủ tục thực hiện toán tử $CoefficientMonomialSV$

```
Procedure CoefficientMonomialSV(u, x);
Input
        u : một biểu thức toán học;
        x : một biến;
Output
        [c,m] với m là bậc của u theo x còn c là hệ số của của x^m;
Local Variables
        c, m, t, base, exponent;
Begin
   if(MonomialSV(u, x)) then
                m:=DegreeMonomialSV(u,x)
                if Kind(u) ϵ {integer,fraction} then
                    c:=u;
                else if u = x then
                    c:=1;
```

```
            ese if Kind(u)="^" then
                    base := Operand(u, 1);
                    exponent := Operand(u, 2);
                    if base = x and Kind(exponent) = integer and exponent > 1 then
                            c:=1;
                    else if
                            c:=Undefined;
            else if Kind(u)=" * " then
                    t:=Operand(u,1);
                    if t ∈{integer, fraction} then
                            c:=t;
                    else if
                            c:=1;
            return [c,m];
    else
            return [undefined, undefined];
End
```

Hình 5.5 Thủ tục thực hiện toán tử $CoefficientMonomialSV$

- Thủ tục thực hiện toán tử $CoefficientSV$

```
Procedure CoefficientSV(u, x, j);
Input
    u : một biểu thức đại số;
    x : một biến;
Output
    Hệ số của x^j trong đa thức;
Local Variables
    t;
Begin
    if PolynomialSV(u,x)=True then
        for i:= 1 to NumberOfOperands(u) do
            t:= DegreeMonomialSV(Operand(u,i),x);
            if t=j then
                Return(CoefficientMonomialSV(Operand(u,i), x));
        Return 0;
    else
        Return(Undefined);
End
```

Hình 5.6 Thủ tục thực hiện toán tử $CoefficientSV$

Toán tử LeadingCoefficientSV

Định nghĩa 5.7: Cho u là biểu thức đại số. Nếu u là một đa thức một biến x, toán tử $LeadingCoefficientSV(u,x)$ trả về $lc(u,x)$. Nếu u không phải đa thức biến x thì toán tử trả về ký hiệu Undefined.

Ví dụ:

$$LeadingCoefficientSV(x^2+3x+5, \quad x) \to 1$$

$$LeadingCoefficientSV(3, \quad x) \to 3.$$

5.1.2 Các thể hiện của đơn thức và đa thức một biến

5.1.2.1 Lớp đơn thức một biến MonomialSV

- Các thuộc tính

coeffi : AnyNode	Hệ số của đơn thức
deg : AnyNode	Bậc của đơn thức
mono : AnyNode	Đơn thức
isMono : boolean	Là đơn thức sẽ có giá trị True và ngược lại sẽ là False
var : AnyNode	Phần biến của đơn thức

Bảng 5.1 Các thuộc tính của lớp MonomialSV

- Các phương thức

MonomialSV()	Phương thức khởi tạo không tham số
getVar()	
MonomialSV(AnyNode, AnyNode)	Phương thức khởi tạo với hai tham số là đơn thức và biến
coefficientMonomialSV(AnyNode, AnyNode)	Phương thức tìm phần hệ số của đơn thức
degreeMonomialSV(AnyNode, AnyNode)	Phương thức tìm bậc của đơn thức
monomialSV(AnyNode, AnyNode)	Phương thức trả về true nếu là đơn thức, ngược lại trả về false

Bảng 5.2 Các phương thức của lớp MonomialSV

- Phương thức *monomialSV*

```
private boolean monomialSV(AnyNode u, AnyNode x) {
      if (x.getKey() != CEILING) {
            return false;
      } else {
            if (u.getKey() == 0 || u.getKey() == -37)
                  return true;
            else if (u.equal(x)) {
                  return true;
            } else if (u.getKey() == -5) {
                  AnyNode base = u.base();
                  AnyNode exponent = u.exponent();
```

```
                    if (base.equal(x)
                            && (exponent.getKey() == 0 && exponent.getValue()
> 1)) {
                            return true;
                    }
              } else if (u.getKey() == -3) {
                    if (u.nop() == 2 && monomialSV(u.operand(1), x)
                                && monomialSV(u.operand(2), x)) {
                            return true;
                    }
              }
              return false;
        }
}
```

Hình 5.7 Phương thức *monomialSV*

- Phương thức khởi tạo *MonomialSV*

```
public MonomialSV(AnyNode u, AnyNode x) {
      u = Simplify.simplify(u);
      if (monomialSV(u, x)) {
            this.mono = u;
            this.var = x;
            this.isMono = true;
            this.deg = degreeMonomialSV(u, x);
            ArrayList<AnyNode> temp = coefficientMonomialSV(u, x);
            this.coeffi = temp.get(0);
      } else {
            // Don thuc "u" khong phu thuoc vao bien x
            this.mono = new AnyNode(-38);
            this.var = new AnyNode(-38);
            this.coeffi = new AnyNode(-38);
            this.deg = new AnyNode(-38);
            this.isMono = false;
      }
}
```

Hình 5.8 Phương thức khởi tạo *MonomialSV*

Các phương thức *degreeMonomialSV*, *coefficientMonomialSV* sẽ được trình bày trong Phụ lục 20, Phụ lục 21.

5.1.2.2 Đa thức một biến PolynomialSV

- Các thuộc tính

poly : AnyNode	Đa thức
isPoly : boolean	Là true nếu đa thức phụ thuộc biến var, ngược lại có giá trị là false
var : AnyNode	Phần biến của đa thức
deg : AnyNode	Bậc của đa thức

Bảng 5.3 Các thuộc tính của lớp PolynomialSV

- Các phương thức

PolynomialSV()	Phương thức khởi tạo không tham số
PolynomialSV(AnyNode, AnyNode)	Phương thức khởi tạo hai tham số lần lượt là biểu thức và biến
degreeSV(AnyNode, AnyNode)	Phương thức tìm bậc đa thức
coefficientSV(int)	Phương thức tìm phần hệ số của biến có số mũ là j
polynomialSV(AnyNode, AnyNode)	Trả về true nếu biểu thức là đa thức trong biến x, ngược lại trả về false

Bảng 5.4 Các phương thức của lớp PolynomialSV

- Phương thức *polynomialSV*

```
private boolean polynomialSV(AnyNode u, AnyNode x) {
	if (new MonomialSV(u, x).isMonomial())
		// u la don thuc phu thuoc bien "x"
		return true;
	else if (u.getKey() == -1) {
		for (int i = 1; i <= u.nop(); i++) {
			MonomialSV temp = new MonomialSV(u.operand(i), x);
			if (!temp.isMonomial())
				return false;
		}
		return true;
	}
	return false;
}
```

Hình 5.9 Phương thức *polynomialSV*

- Phước thức khởi tạo *PolynomialSV*

```
public PolynomialSV(AnyNode u, AnyNode x) {
	if (polynomialSV(u, x)) {
		// neu u la da thuc phu thuoc x
		this.poly = u;
		this.var = x;
		this.isPoly = true;
		this.deg = degreeSV(u, x);
		ArrayList<AnyNode> s = new ArrayList<AnyNode>();
		s.add(x);
	} else {
		this.poly = new AnyNode(-38);
			this.var = new AnyNode(-38);
		this.deg = new AnyNode(-38);
		this.isPoly = false;
	}
}
```

Hình 5.10 Phương thức khởi tạo *PolynomialSV*

Các phương thức *coefficientSV*, *degreeSV* sẽ được trình bày trong Phụ lục 22, Phụ lục 23.

5.2 Đa thức nhiều biến

Một đa thức chứa một hoặc nhiều biến được gọi là đa thức nhiều biến

Định nghĩa 5.8: (Định nghĩa toán học) một đa thức nhiều biến u trong tập biến $\{x_1, x_2, \dots, x_m\}$ là một tổng hữu hạn của các đơn thức có dạng

$$c x_1^{n_1} x_2^{n_2} \dots x_m^{n_m}$$

với hệ số c là một số hữu tỉ và số mũ n_j là số nguyên không âm.

Ví dụ:

$$p + {}^1\!/_2\, pv^2 + gpy,\ ax^2 + 2bc + 3c$$

5.3 Đa thức tổng quát

Có nhiều biểu thức là đa thức trong ngữ cảnh tính toán nhưng không phải là đa thức trong định nghĩa của phần 5.1

Ví dụ: Biểu thức $u = \frac{a}{(a+1)} x^2 + bx + \frac{1}{a}$

Có thể xem u như là một đa thức một biến x với phần hệ số là $\frac{a}{(a+1)}$, $b, \frac{1}{a}$

Định nghĩa 5.9: (Định nghĩa toán học) Cho $c_1, c_2, \dots, c_r$ là các biểu thức đại số và $x_1, x_2, \dots, x_m$ là các biểu thức đại số không phải là số nguyên hoặc phân số.

- Một đơn thức tổng quát trong tập biến $\{x_1, x_2 \dots, x_m\}$ là một biểu thức có dạng

  $$c_1 c_2 \dots c_r x_1^{n_1} x_2^{n_2} \dots x_m^{n_m}$$

  với số mũ n_j là số nguyên không âm và c_i thỏa mãn điều kiện sau

  $$FreeOf(c_i, x_j) \rightarrow true \text{ với } j = 1,2, \dots, m$$

  Biểu thức x_j là biến tổng quát và c_i gọi là hệ số tổng quát. Biểu thức $x_1^{n_1} x_2^{n_2} \dots x_m^{n_m}$ gọi là phần biến của đơn thức, nếu không có biến tổng quát trong đơn thức thì phần biến có giá trị là 1. Biểu thức $c_1 c_2 \dots c_r$ gọi là phần hệ số của đơn thức, hệ số bằng 1 nếu không có phần hệ số.

- Một biểu thức u là một đa thức tổng quát nếu nó là một đơn thức tổng quát hoặc là tổng của các đơn thức tổng quát trong tập biến $\{x_1, x_2, \dots, x_m\}$

Ví dụ:

$$x^2 - x + 1, \qquad (x_1 = x)$$

$$\sin^3(x) + 2\sin^2(x) + 3, \qquad (x_1 = \sin(x))$$

$$(x+1)^3 + 2(x+1)^2 + 3\,, \qquad (x_1 = x + 1)$$

Định nghĩa 5.10: (Định nghĩa tính toán) Một đơn thức tổng quát trong tập biến tổng quát $S = \{x_1, x_2 \dots, x_m\}$ là một biểu thức toán học u thỏa mãn các luật sau:

1. $FreeOf(u, x_j) \rightarrow true$ với $j = 1, \dots, m$
2. $u \in S$
3. $u = x^n$ với $x \in S$ và n là số nguyên lớn hơn 1
4. u là một tích và mỗi toán hạng của u là một đơn thức tổng quát trong S

Định nghĩa 5.11: (Định nghĩa tính toán) Một đa thức tổng quát trong tập $S = \{x_1, x_2 \dots, x_m\}$ là một biểu thức toán học u thỏa mãn các luật sau:

1. u là một đơn thức tổng quát trong S.
2. u là tổng mà mỗi toán hạng của nó là một đơn thức tổng quát trong S.

5.3.1 Các toán tử cơ bản của đơn thức tổng quát

5.3.1.1 Toán tử MonomialGPE

Định nghĩa 5.12: Cho u là một biểu thức đại số và cho v là biến x hoặc tập biến tổng quát S. Toán tử $MonomialGPE(u, v)$ trả về true nếu u là một đơn thức tổng quát trong S và ngược lại trả về false.

Ví dụ:

$$MonomialGPE(ax^2y^2, \{x, y\}) \rightarrow true$$

$$MonomialGPE(x^2 + y^2, \{x, y\}) \rightarrow false$$

- Thủ tục thực hiện toán tử $MonomialGPE$

```
Procedure MonomialGPE(u, v);
Input
        u : là một biểu thức đại số;
        v : là một biến hoặc một tập biến tổng quát;
Output
        true hoặc false;
Local Variables
        i, S, base, exponent;
Begin
        if Kind(v) = set then S := {v} else S := v;
        if u ∈ S then
                Return(true)
        elseif Kind(u) = " ∧ " then
                                base := Operand(u, 1);
                            exponent := Operand(u, 2);
                if base ∈ S and Kind(exponent) = integer and exponent >1 then
                        Return(true)
        elseif Kind(u) = " * " then
```

```
            for i := 1 to NumberOfOperands(u) do
                if MonomialGPE(Operand(u, i), S) = false then
                    Return(false);
            Return(true);
        Return(SetFreeOf(u, S));
End
```

Hình 5.11 Thủ tục thực hiện toán tử $MonomialGPE$

5.3.1.2 Toán tử CoefficientGME

Toán tử trả về một danh sách gồm hai phần tử c và m trong đó m là bậc của đơn thức và c là hệ số của x^m hoặc trả về ký hiệu $Undefined$.

Định nghĩa 5.13: Cho u là một biểu thức đại số toán tử $CoefficitentGME(u, x)$ được định nghĩa bởi các luật sau ([c, m]):

1. Nếu $u = x$ thì: $CoefficitentGME(u, x) \rightarrow [1, 1]$
2. Nếu u là một lũy thừa và :

$$base = Operand(u, 1)$$

$$exponent = Operand(u, 2)$$

   nếu $u = x$ và $exponent$ là số nguyên lớn hơn 1 thì trả về [1, exponent].

3. Nếu u là một tích thì toán tử trả về Undifined nếu không có toán hạng nào trong u là đơn thức phụ thuộc x. Nếu trong u có toán hạng phụ thuộc x và toán hạng đó có bậc m là số dương thì trả về [u/x^m, m].
4. Nếu u không là đơn thức phụ thuộc x thì toán tử trả về Undefined.

- Thủ tục thực hiện toán tử $CoefficitentGME$:

```
Procedure CoefficitentGME(u, x);
Input
        u : một biểu thức đại số;
        x : một biến tổng quát;
Output
        Danh sách [c, m] với m là bậc của đơn thức và c là hệ số của x^m
        hoặc là ký hiệu Undefined;
Local Variables
        base, exponent, i, c, m, f;
Begin
        if u = x then
                Return([1, 1]);
        else if Kind(u)=" ∧ " then
                base := Operand(u, 1);
                exponent := Operand(u, 2);
```

```
            if base = x and Kind(exponent)= integer and exponent > 1 then
                Return([1, exponent]);
        else if Kind(u)=" * " then
            m := 0;
            c := u;
            for i := 1 to NumberOfOperands(u) do
                f := CoefficientGME(Operand(u, i), x);
                if f = Undefined then
                    Return(Undefined);
                else if Operand(f, 2) ≠ 0 then
                    m := Operand(f, 2);
                    c := u/x^m;
                Return([c, m]);
        If FreeOf(u, x) then
            Return([u, 0]);
        else
            Return(Undefined);
End
```

Hình 5.12 Thủ tục thực hiện toán tử $CoefficientGME$

5.3.1.3 Toán tử DegreeGME

Toán tử tìm bậc của đơn thức theo tập biến tổng quát s.

Định nghĩa 5.14: Cho u là một biểu thức đại số và tập biến tổng quát S toán tử $DegreeGME(u, x)$ được định nghĩa bởi các luật sau:

1. Nếu $u \epsilon S$ thì toán tử trả về 1.
2. Nếu u là lũy thừa thì toán tử trả về $exponent(u)$.
3. Nếu u là tích của các toán hạng $u_1, u_2, \dots, u_n$ thì bậc của u bằng tổng bậc của các toán hạng $u_1, u_2, \dots, u_n$với tập biến tổng quát S.
4. Nếu u không thuộc S và $u = 0$ toán tử trả về Undefined ngược lại trả về 0.

5.3.1.4 Thể hiện của đơn thức tổng quát

Lớp $GeneralMonomial$ được thiết kế dựa trên các phân tích về đơn thức tổng quát.

Các thuộc tính

coeffi : AnyNode	Hệ số của đơn thức
deg : AnyNode	Bậc của đơn thức
mono : AnyNode	Đơn thức
isMono : boolean	Bằng true nếu là đơn thức, ngược lại bằng false
s : ArrayList<AnyNode>	Tập biến tổng quát của đơn thức

Bảng 5.5 Các thuộc tính của lớp GeneralMonomial

Các phương thức

GeneralMonomial()	Phương thức khởi tạo không tham số
GeneralMonomial(AnyNode, ArrayList<AnyNode>)	Phương thức khởi tạo đơn thức u với tập biến S
setUndefined(GeneralMonomial)	Phương thức thiết lập các giá trị nếu u là Undefined
monomialGpe(AnyNode, ArrayList<AnyNode>)	Phương thức trả về true nếu u là một đơn thức trong S và false trong các trường còn lại.
coefficientGme(AnyNode)	Phương thức trả về danh sách gồm bậc m của đơn thức và hệ số của x^m
degreeGme(AnyNode, ArrayList<AnyNode>)	Phương thức tìm bậc của đơn thức với tập biến tổng quát S
multi(GeneralMonomial)	Phương thức nhân hai đơn thức
div(GeneralMonomial)	Phương thức chia hai đơn thức
sub(GeneralMonomial)	Phương thức trừ hai đơn thức đồng dạng
add(GeneralMonomial)	Phương thức cộng hai đơn thức đồng dạng

Bảng 5.6 Các phương thức của lớp GeneralMonomial

- Phương thức *monomialGpe*

```
private boolean monomialGpe(AnyNode u, ArrayList<AnyNode> s) {
	this.s = s;
	if (u.getKey() == -38 || s.isEmpty()) {
		setUndefined(this);
		return false;
	} else if (u.getKey() == 0 && u.getValue() == 0) {
		this.mono = u;
		this.coeffi = new AnyNode(0, 0);
		this.deg = new AnyNode(-38);
		this.var = new AnyNode(0, 1);
		return true;
	} else if (isMember(u, s)) {
		this.mono = u;
		this.coeffi = new AnyNode(0, 1);
		this.deg = new AnyNode(0, 1);
		this.var = u;
		return true;
	} else if (u.getKey() == -5) {
		AnyNode base = u.base();
		AnyNode exponent = u.exponent();
		if (isMember(base, s) && exponent.getKey() == 0
				&& exponent.getValue() > 1) {
			this.mono = u;
			this.coeffi = new AnyNode(0, 1);
			this.deg = exponent;
			this.var = base;
```

```
                        return true;
                  }
            } else if (u.getKey() == -3) {
                  AnyNode tempCoeffi = new AnyNode(-3);
                  AnyNode tempVar = new AnyNode(-3);
                  int degree = 0;
                  for (int i = 1; i <= u.nop(); i++) {
                        if (!monomialGpe(u.operand(i), s)) {
                              setUndefined(this);
                              return false;
                        }
                        if (u.operand(i).getKey() != -5) {
                              if (isMember(u.operand(i), s))
                                    degree +=
u.operand(i).exponent().getValue();
                        } else {
                              if (isMember(u.operand(i).operand(1), s))
                                    degree +=
u.operand(i).exponent().getValue();
                        }
                        if (setFreeOf(u.operand(i), s)) {
                              tempCoeffi.leaf.add(u.operand(i));
                        } else {
                              tempVar.leaf.add(u.operand(i));
                        }
                  }
                  this.deg = new AnyNode(0, degree);
                  if (tempCoeffi.leaf == null || tempCoeffi.leaf.size() == 0)
                        this.coeffi = new AnyNode(0, 1);
                  else
                        this.coeffi = Simplify.simplify(tempCoeffi);
                  if (tempVar.leaf == null || tempVar.leaf.size() == 0)
                        this.var = new AnyNode(0, 1);
                  else
                        this.var = Simplify.simplify(tempVar);
                  this.mono = u;
                  return true;
            }
            if (setFreeOf(u, s)) {
                  this.coeffi = u;
                  this.var = new AnyNode(0, 1);
                  this.deg = new AnyNode(0, 0);
                  this.mono = u;
                  return true;
            } else {
                  setUndefined(this);
                  return false;
            }
}
```

Hình 5.13 Phương thức *monomialGpe*

Các phương thức *coefficientGme*, *degreeGme*, *add*, *sub*, *multi*, *div* sẽ được trình bày ở phần Phụ lục 24, Phụ lục 25, Phụ lục 29, Phụ lục 28, Phụ lục 26, Phụ lục 27.

5.3.2 Các toán tử cơ bản của đa thức tổng quát

5.3.2.1 Toán tử PolynomialGPE

Định nghĩa 5.15: Toán tử $PolynomialGPE(u, v)$ trả về true nếu u là một đa thức với biến tổng quát trong $\{x\}$ hoặc S ngược lại trả về false.

Ví dụ:

$$PolynomialGPE(x^2 + y^2, \{x, y\}) \rightarrow true,$$

$$PolynomialGPE(sin^2(x) + 2\ sin(x) + 3, sin(x)) \rightarrow true,$$

$$PolynomialGPE(x/y + 2\ y, \{x, y\}) \rightarrow false$$

- Thủ tục thực hiện toán tử $PolynomialGPE$

```
Procedure PolynomialGPE(u, v);
Input
        u : là một biểu thức đại số;
        v : là một biến hoặc một tập biến tổng quát;
Output
        true or false;
Local Variables
        i, S;
Begin
        if Kind(v) = set then S := {v} else S := v;
                Return(MonomialGPE(u, S));
        else
                if u ∈ S then Return(true);
                        for i := 1 to NumberOfOperands(u) do
                                if MonomialGPE(Operand(u, i), S) = false then
                                        Return(false);
                        Return(true);
End
```

Hình 5.14 Thủ tục thực hiện toán tử $PolynomialGPE$

5.3.2.2 Toán tử Variables

Cấu trúc đa thức của một đa thức tổng quát phụ thuộc và biểu thức mà nó chọn làm biến tổng quát. Toán tử này sẽ xác định tập biến tổng quát tự nhiên của một biểu thức.

Định nghĩa 5.16: Cho u là một biểu thúc đại số. Toán tử $Variables(u)$ được định nghĩa bởi các luật sau đây:

1. Nếu u là một số nguyên hoặc phân số thì

$$Variable(u) \rightarrow \emptyset$$

2. Giả sử u là một lũy thừa. Nếu số mũ của u là một số nguyên lớn hơn 1 thì

$$Variables(u) \to \{Operand(u,1)\}$$

Trong các trườngh hợp còn lại

$$Variables(u) \to \{u\}$$

3. Giả sử u là một tổng. Toán tử $Variable(u)$ là hợp của các biến tổng quát của mỗi toán hạng trong u được xác định bởi các luật 1, 2, 4, 5.
4. Giả sử u là một tích. Toán tử $Variable(u)$ chứa hợp của các biến tổng quát của mỗi toán hạng trong u được xác định bởi các luật 1, 2, 5.
5. Nếu u không thỏa mãn các trường hợp trên thì

$$Variables(u) \to \{u\}$$

Ví dụ:

$$Variables(x^3 + 3x^2y + 3xy^2 + y^3) \to \{x, y\}$$

$$Variables(3x(x+1)y^2z^n) \to \{x, x+1, z^n\}$$

$$Variables(1/2) \to \emptyset$$

5.3.2.3 Toán tử DegreeGPE

Định nghĩa 5.17:

- Cho $S = \{x_1, x_2, \dots, x_m\}$ là tập các đơn thức tổng quát. Cho $u = c_1 \dots c_r . x_1^{n_1} \dots x_m^{n_m}$ là một đơn thức với phần hệ số khác 0. Bậc của u trong S là tổng số mũ của các biến tổng quát:

$$\deg(u, S) = n_1 + n_2 \dots + n_m$$

Theo quy ước toán học bậc của đơn thức 0 là $-\infty$

- Nếu u là một đa thức tổng quát và là tổng của các đơn thức thì $\deg(u, S)$ là giá trị lớn nhất của các bậc của các đơn thức trong u. Nếu u chứa một biến x thì bậc của u là $\deg(u, x)$.

Ví dụ:

$$\deg(3wx^2y^3z^4, \{x, z\}) = 6$$

$$\deg(ax^2 + bx + c, \{x\}) = 2$$

$$\deg(a\,sin^2(x) + bsin(x) + c\,, \{sin(x)\}) = 2$$

Định nghĩa 5.18: Cho u là một biểu thức đai số và v là một biến tổng quát x hoặc tập biến tổng quát S. Toán tử DegreeGPE có dạng

$$DegreeGPE(u, v)$$

Khi u là một đa thức tổng quát trong v thì toán tử trả về bậc của u. Nếu u không là đa thức tổng quát trong v thì toán tử trả về ký hiệu Undefined.

Định nghĩa 5.19: Cho u là một biểu thức toán học và

$$S = Variables(u)$$

Toán tử $\deg(u, S)$ gọi là tổng bậc của biểu thức u

Ví dụ: $u = ax^2 + bx + c$ và $S = \{a, b, c, x\}$ tổng bậc của u $\deg(u, \{a, b, c, x\}) = 3$

5.3.2.4 Toán tử CoefficientGPE

Định nghĩa 5.20: Cho u cho một biểu thức toán học. Nếu u là đa thức tổng quát với biến tổng quát x và $j \geq 0$ là một số nguyên thì toán tử $CoefficientGPE(u, x, j)$ trả về tổng phần hệ số của tất cả các đơn thức của u mà có phần biến là x^j. Nếu không có đơn thức nào có phần biến là x^j thì toán tử trả về 0. Nếu u không phải là đa thức biến x thì toán tử trả về ký hiệu Undefined.

Ví dụ:

$$CoefficientGPE((3\sin(x))x^2 + \big(2\ln(x)\big)x + 4, x, 2) \rightarrow Undefined$$

$$CoefficientGPE(ax^2 + bx + c, x, 2) \rightarrow a$$

$$CoefficientGPE(3xy^2 + 5x^2y + 7x + 9, x, 1) \rightarrow 3y^2 + 7$$

5.3.2.5 Toán tử LeadingCoefficientGPE

Định nghĩa 5.21: Cho u là một biểu thức đại số. Nếu u là một đa thức tổng quát trong x thì $LeadingCoefficientGPE$(u) với x được định nghĩa như là tổng của phần hệ số của tất cả các đơn thức với phần biến $x^{\text{degreeGpe}(u,x)}$. Hệ số đầu tiên sẽ được biểu diễn bởi $lc(u, x)$.

Ví dụ:

$$lc(3xy^2 + 5x^2y + 7x^2y^3 + 9, x) = 5y + 7y^3$$

Định nghĩa 5.22: Cho u là một đa thức tổng quát trong x. Toán tử

$$LeadingCoefficientGPE(u, x)$$

trả về $lc(u, x)$. Nếu u không là đa thức tổng quát trong x thì toán tử trả về ký hiệu Undefined.

5.3.2.6 Thiết kế lớp tương ứng của đa thức tổng quát

- Các thuộc tính

listMono: ArrayList<GeneralMonomial>	Danh sách đơn thức
poly : AnyNode	Nút chứa biểu thức
isPoly : boolean	True nếu là đa thức và Flase nếu ngược lại
s : ArrayList<AnyNode>	Tập biến tổng quát của đa thức

Bảng 5.7 Các thuộc tính của lớp GeneralPolynomial

- Các phương thức

getS()	
getListMono()	
setListMono(ArrayList<GeneralMonomial>)	
GeneralPolynomial()	
GeneralPolynomial(AnyNode, ArrayList<AnyNode>)	
polynomialGpe(AnyNode, ArrayList<AnyNode>)	
polynomialExpansion(GeneralPolynomial, AnyNode, AnyNode)	
leadingCoefficientGPE(AnyNode)	
coefficientGPE(AnyNode, int)	
add(GeneralPolynomial)	
sub(GeneralPolynomial)	
mul(GeneralPolynomial)	
div(GeneralPolynomial, AnyNode)	
degreeGpe()	
degreeGpe(AnyNode)	
collectTerms()	

Bảng 5.8 Các phương thức của lớp GeneralPolynomial

- Phương thức *polynomialGpe*

```
private boolean polynomialGpe(AnyNode u, ArrayList<AnyNode> s) {
	this.s = s;
	if (u.getKey() != -1) {
		// u khong phai la tong
		if (new GeneralMonomial(u, s).isMono) {
			// u la don thuc thuoc s
			this.poly = u;
			this.listMono.add(new GeneralMonomial(u, s));
			return true;
		} else {
			// u khong phai la don thuc trong s
			this.poly = new AnyNode(-38);
			this.listMono = null;
			return false;
		}
	} else {
		if (isMember(u, s)) {
			// u thuoc s (u la don thuc trong s)
			this.poly = u;
			this.isPoly = true;
			this.listMono.add(new GeneralMonomial(u, s));
			return true;
		} else {
			// u la tong
			for (int i = 1; i <= u.nop(); i++) {
```

```
                        GeneralMonomial temp = new
GeneralMonomial(u.operand(i), s);
                        if (!temp.isMono) {
                              // ui khong phai la don thuc trong s
                              this.poly = new AnyNode(-38);
                              this.listMono = null;
                              return false;
                        } else
                              this.listMono.add(temp);
                  }
                  this.poly = u;
                  this.isPoly = true;
                  return true;
            }
      }
}
```

Hình 5.15 Phương thức $polynomialGpe$

Các phương thức $polynomialExpansion$, add, sub, $multi$, div, $degreeGpe$,$leadingCoefficientGPE$, $coefficientGPE$, $degreeGpe(AnyNode)$, $collectTerms$ được trình bày ở phần Phụ lục 30.

5.3.3 Các toán tử thao tác với đa thức tổng quát

Trong phần này mô tả hai toán tử sử dụng để thao tác trong khi tính toán đa thức tổng quát. Cả hai toán tử dựa trên hai tính chất phân phối vào giao hoán của phép cộng

$$a(a+b) = ab + ac$$

$$(a+b)c = ac + bc$$

5.3.3.1 Toán tử $CollectTerms$

Việc rút gọn hệ số của đa thức thường xuyên được áp dụng trong quá trình tính toán đại số. Trong suốt quá trình rút gọn toán tử này chỉ áp dụng cho các đơn thức có phần hệ số là số hữu tỉ (số nguyên hoặc phân số). Việc rút gọn các hệ số được thực hiện bởi toán tử $CollectTerms$.

Định nghĩa 5.23: Một đa thức tổng quát u có phần hệ số thu gọn trong tập biến tổng quát S nếu thỏa mãn một trong số các thuộc tính sau:

1. U là một đơn thức tổng quát trong S
2. U là tổng của các đơn thức tổng quát trong S với phần biến rõ ràng (distinct) (Định nghĩa này tương tự định nghĩa Định nghĩa 5.9 ngoại trừ trong luật số hai yêu cầu phần biến phải được xác định rõ ràng.)

Ví dụ: $(2a+3b)xy + (4a+5b)x$

là đa thức có các phần hệ số được thu gọn trong tập biến tổng quát $S = \{x, y\}$ (Vì trong trường hợp này tập biến tổng quát của u được hiểu bao gồm các biến $\{x, y, xy\}$. Các biến này được tìm bởi toán tử $variable$). Nhưng đa thức

$$2axy + 3bxy + 4ax + 5bx$$

thì không phải là đa thức có phần hệ số được thu gọn vì trong đó có hai đơn thức có phần biến là xy và hai đơn thức có phần biến là x.

Dạng thu gọn hệ số của đa thức phụ thuộc và biểu thức được chọn làm phần biến tổng quát. Ví dụ với tập biến tổng quát $S = \{a, b\}$ thì đa thức $2axy + 3bxy + 4ax + 5bx$ có dạng thu gọn là $(2xy + 4x)a + (3xy + 5x)b$.

- Thủ tục giả mã thực hiện toán tử $CollectTerms$

```
Procedure CollectTerms(u, S);
Input
        u : la biểu thức đại số;
        S: một tập biến tổng quát không rỗng;
Output
       Dạng thu gọn hệ số của u hoặc là ký hiệu Undefined nếu u không phải
        Là đa thức tổng quát trong tập S
Local Variables
        f, combined, i, j, N, T, v;
Begin
        if Kind(u) =”+” then
                if CoeffVarMonomial(u, S) = Undefined then
                        Return(Undefined)
                else
                        Return(u)
        else
                if u ∈ S then
                        Return(u);
                N := 0;
                for i := 1 to NumberOfOperands(u) do
                        f := CoeffVarMonomial(Operand(u, i), S);
                        if f = Undefined then
                                Return(Undefined)
                        else
                                j := 1;
                                combined := false;
                                while not combined and j ≤ N do
                                        if Operand(f, 2) = Operand(T[j], 2) then
                                                T[j] := [Operand(f, 1) + Operand(T[j],
1), Operand(f, 2)];
                                                combined := true;
                                        j := j + 1;
```

```
                        if not combined then
                            T[N + 1] := f;
                            N := N + 1;
        v := 0;
        for j := 1 to N do
            v := v + Operand(T[j], 1) * Operand(T[j], 2);
        Return(v)
End
```

Hình 5.16 Thủ tục thực hiện toán tử $CollectTerm$

5.3.3.2 Toán tử $Expand$

Toán tử $Expand$ áp dụng hai phép biến đổi phân phối tới tích và lũy thừa để thu được một tổng.

Ví dụ:

$$(x+2)(x+3)(x+4) \to x^3 + 9x^2 + 26x + 24$$

$$(x+1)^2 + (y+1)^2 \to x^2 + 2x + y^2 + 2y + 2$$

$$((x+2)^2 + 3)^2 \to x^4 + 8x^3 + 30x^2 + 56x + 49$$

Định nghĩa 5.24 sẽ mô tả dạng đầu ra của toán tử $Expand$

Định nghĩa 5.24: Một biểu thức đại số u có dạng khai triển nếu tập các biến tổng quát được tính bởi toán tử $Variable(u)$ không chứa một tổng.

Định nghĩa 5.24 phía trên chỉ có ý nghĩa trong phạm vi của hệ thống rút gọn biểu thức. Một số biểu thức không ở dạng khai triển nếu không trong phạm vi này.

Ví dụ: biểu thức

$$((x+1)^2)^2$$

sẽ được biến đổi: $((x+1)^2)^2 \to (x+1)^4$ khi đó $Variable(u) = \{x+1\}$, tập biến của u chứa một tổng nên u không ở dạng khai triển theo Định nghĩa 5.24.

Trường hợp số mũ nguyên: trường hợp đơn giản đầu tiên của thuật toán là áp dụng với biểu thức đại số u có lũy thừa là số nguyên. Thủ tục $Expand(u)$ thực hiện khai triển đệ quy lần đầu với các toán hạng tổng, tích và lũy thừa với số mũ nguyên dương và gọi tới hai thủ tục $ExpandProduct$, $ExpandPower$ để áp dụng luật phân phối tổng và tích.

Thủ tục $ExpandProduct(r, s)$ khai triển tích của hai biểu thức đã được khai triển. Sử dụng đệ quy để áp dụng luật phân phối tới cả về trái và phải của biểu thức.

Thủ tục $ExpandPower(u, n)$ khai triển một biểu thức u đã được khai triển với lũy thừa $n \geq 2$. Khi u là một tổng dạng khai triển sẽ thu được bởi

$$f = Operand(u, 1), \quad r = u - f$$

và áp dụng công thức

$$u^n = (f + r)^n = \sum_{k=0}^{n} \left(\frac{n!}{k!\,(n-k)!} f^{n-k} \right) r^k$$

- Thủ tục $Expand$

```
Procedure Expand(u);
Input
        u : là một biểu thức đại số mà tất cả các số mũ của các lũy thừa là số nguyên
Output
        Dạng khai triển của u
Local Variables
        v, base, exponent;
Begin
        if Kind(u)="+" then
                v := Operand(u, 1);
                Return(Expand(v) + Expand(u − v));
        Else if Kind(u) = " * "  then
                v := Operand(u, 1);
                Return(ExpandProduct(Expand(v), Expand(u/v)));
        Else if Kind(u) = " ∧ " then
                base := Operand(u, 1);
                exponent := Operand(u, 2);
                if Kind(exponent) = integer and exponent ≥ 2 then
                        Return(ExpandPower(Expand(base), exponent));
        Return(u)
End
```

Hình 5.17 Thủ tục $Expand$

Thủ tục $ExpandProduct$, $ExpandPower$ được trình bày ở Phụ lục 31, Phụ lục 32.

Trường hợp số mũ không phải số nguyên

Nếu u chứa lũy thừa với số mũ không phải là số nguyên, toán tử $Expand$ có thể trả về một biểu thức không ở dạng khai triển.

Ví dụ: $Expand(x(y+1)^{3/2} + 1)\,(x(y+1)^{3/2} - 1) \rightarrow x^2(y+1)^3 - 1$

Các phương thức thực hiện toán tử $CollectTerms$, $Expand$ đước trình bày ở phụ lục

5.4 Biểu thức hữu tỉ tổng quát.

Trong ngữ cảnh toán học một biểu thức hữu tỉ được định nghĩa như là thương của hai đa thức. Trong phần này của luận văn sẽ trình bày về cấu trúc biểu thức hữu tỉ của một biểu thức đại số và mô tả thuật toán biến đổi một biểu thức về dạng hữu tỉ.

Định nghĩa 5.25: cho $S = \{x_1, x_2, \dots, x_m\}$ là một tập biến tổng quát. Một biểu thức đại số u là một biểu thức hữu tỉ GRE (general rational expression) trong S nếu $u = p/q$ với p và q là các đa thức tổng quát trong S.

Ví dụ:

$$\frac{x^2 - x + y}{x + 4}, \qquad S = \{x\}$$

$$\frac{x^2 sin(y) - x sin^2(y) + 2(z + 1)}{x + sin(y)}, \qquad S = \{x, sin(y)\}$$

$$x^2 + bx + c, \qquad S = \{x\}$$

5.4.1 Toán tử $Numerator$ và $Denominator$

Để có thể xác định một biểu thức đã ở dạng hữu tỉ hay không thì cần định nghĩa chính xác tử số và mẫu số của biểu thức. Toán tử $Numerator$ và $Denominator$ sử dụng để làm việc này. Hai toán tử được định nghĩa bởi các luật biến đổi dưới đây.

Định nghĩa 5.26: Cho u là một biểu thức đại số

1. Nếu u là một phân số thì

$$Numerator(u) \rightarrow Operand(u, 1)$$

$$Denominator\ (u) \rightarrow Operand(u, 2).$$

2. Giả sử u là lũy thừa. Nếu số mũ của u là số nguyên âm hoặc phân số âm thì

$$Numerator(u) \rightarrow 1, \qquad Denominator(u) \rightarrow u^{-1}$$

nếu không thì

$$Numerator(u) \rightarrow u, \qquad Denominator\ (u) \rightarrow 1$$

3. Giả sử u là tích và $v = Operand(u, 1)$ thì:

$$Numerator(u) \rightarrow Numerator(v) * Numerator(u/v)$$

$$Denominator(u) \rightarrow Denominator(v) * Denominator(u/v)$$

4. Nếu không thỏa mãn các luật trên thì:

$$Numerator(u) \rightarrow u, \qquad Denominator(u) \rightarrow 1$$

Ví dụ: Xem xét biểu thức đại số sau

$$u = (2/3)\frac{x(x+1)}{x+2}y^n$$

$$Numerator(u) \to 2x(x+1)y^n, \qquad Denominator(u) \to 3(x+2)$$

Định nghĩa 5.27: Cho $S = \{x_1, x_2, \dots, x_m\}$ là một tập biến tổng quát. Một biểu thức u là biểu thức hữu tỉ trong S nếu $Numerator(u)$ và $Denominator(u)$ là các đa thức tổng quát trong S.

5.4.2 Toán tử RationalGPE

Định nghĩa 5.28: Cho u là một biểu thức đại số và cho v là một biến tổng quát x hoặc tập biến tổng quát S. Toán tử $RationalGPE(u, v)$ trả ra $True$ nếu u là một biểu thức hữu tỉ tổng quát trong x hoặc S và trả về $False$ trong trường hợp còn lại.

Toán tử được định nghĩa bởi luật biến đổi sau:

$$RationalGPE(u, v)$$
$$\to PolynomialGPE(Numerator(u), v)\ and\ PolynomialGPE(Denominator(u), v)$$

Ví dụ:

$$RationalGPE\left(\frac{x^2+1}{2x+3}, x\right) \to true$$

$$RationalGPE\left(\frac{1}{x}+\frac{1}{y}, \{x, y\}\right) \to false.$$

5.4.3 Toán tử RationalVariables

Toán tử này định nghĩa một tập biến tự nhiên của biểu thức hữu tỉ tổng quát.

Định nghĩa 5.29: Cho u là một biểu thức đại số. Toán tử $RationalVariables(u)$ được định nghĩa bởi các luật sau:

$$RationalVariables(u)$$
$$\to Variables(Numerator(u)) \cup Variables(Denominator(u))$$

với toán tử $Variable$ được định nghĩa ở Định nghĩa 5.16.

Ví dụ:

$$RationalVariables\left(\frac{2x+3y}{z+4}\right) \to \{x, y, z\}$$

$$RationalVariables\left(\frac{1}{x}+\frac{1}{y}\right) \to \left\{\frac{1}{x}+\frac{1}{y}\right\}$$

5.4.4 Hữu tỉ hóa một biểu thức đại số

Quá trình hữu tỉ hóa dựa trên việc biến đổi kết hợp các toán hạng của một tổng trên một mẫu số chung.

Ví dụ: $1 + \frac{1}{1+1/x} \rightarrow \frac{2x+1}{x+1}$

Định nghĩa 5.30: Một biểu thức đại số u ở dạng hữu tỉ hóa nếu nó thỏa mãn các luật sau:

1. Nếu u là số nguyên, phân số, ký hiệu, giai thừa hoặc dạng hàm.
2. u là các loại khác và xem u như là một biểu thức hữu tỉ trong tập $S = RationalVariables(u)$ thì
    a. Mỗi biểu thức v trong S đều ở dạng hữu tỉ với $Denominator(v) = 1$
    b. Phần hệ số của mỗi đơn thức trong $Numerator(u)$ và $Denominator(u)$ là số nguyên

Ví dụ: biểu thức ($a/b + c/d$) không phải ở dạng hữu tỉ vì

$$RationalVariables(a/b + c/d) \rightarrow \{a, 1/b, c, 1/d\}$$

không thỏa mãn thuộc tính số 2. Nhưng biểu thức $(ad + bc)/(bd)$ thì ở dạng hữu tỉ vì

$$RationalVariables\ ((ad + bc)/bd) \rightarrow \{a, b, c, d\}$$

và phần hệ số của mỗi đơn thức $ad,\ bc$ và bd là 1.

Toán tử RationalizeExpression

Toán tử này sẽ biến đổi một biểu thức đại số u thành một biểu thức tương đương ở dạng hữu tỉ. Toán tử được hiểu trong ngữ cảnh rút gọn bao gồm các phép biến đổi lũy thừa sau:

$$u^v u^w \rightarrow u^{v+w}$$

$$(u^v)^n \rightarrow u^{vn}$$

$$(u\ v)^n \rightarrow u^n v^n$$

Với u, v, w là các biểu thức đại số và n là số nguyên.

- Thủ tục thực hiện toán tử $RationalizeExpression$

```
Procedure RationalizeExpression(u);
Input
        u :một biểu thức đại số;
Output
        Dạng hữu tỉ hóa của u;
Local Variables f, g,r;
Begin
        if Kind(u)=" ∧ " then
                Return RationalizeExpression(Operand(u,1))^Operand(u,2);
```

```
        else if Kind(u)=" ∗ " then
                f := Operand(u, 1);
                Return(RationalizeExpression(f) ∗ RationalizeExpression(u/
f))
        else if Kind(u)="+" then
                f := Operand(u, 1);
                g := RationalizeExpression(f);
                r := RationalizeExpression(u − f);
                Return(RationalizeSum(g, r))
        else
                Return(u)
End
```

Hình 5.18 Thủ tục thực hiện toán tử $RationalizeExpression$

Thủ tục $RationalizeSum$ được trình bày ở Phụ lục 33.

5.4.5 Thể hiện của biểu thức hữu tỉ

Lớp $GenneralRationalExpression$ là lớp thể hiện được thiết kế dựa trên các đặc điểm của biểu thức hữu tỉ tổng quát.

- Các thuộc tính

numerator : GeneralPolynomial	
denominator : GeneralPolynomial	
s : ArrayList<AnyNode>	
u : AnyNode	

Bảng 5.9 Các thuộc tính của lớp GenneralRationalExpression

- Các phương thức

getNumerator()	
getDenominator()	
getS()	
getU()	
GeneralRationalExpression()	Hàm khởi tạo không tham số
GeneralRationalExpression(AnyNode, ArrayList<AnyNode>)	Hàm khởi tạo biểu thức hữu tỉ với tập biến tổng quát
rationalGre(AnyNode, ArrayList<AnyNode>)	Hàm trả về true nếu u là biểu thức hữu tỉ trong ,ngược lại trả về false.
rationalVariables(AnyNode)	Hàm tìm tập biến tự nhiên của biểu thức
numerator(AnyNode)	Hàm tìm tử số của biểu thức
denominator(AnyNode)	Hàm tìm mẫu số của biểu thức

rationalExpand(AnyNode)	Hàm biến đổi một biểu thức hữu tỉ u về dạng biểu thức hữu tỉ đã được khai triển
rationalizeExpression(AnyNode)	Hàm hữu tỉ hóa một biểu thức
rationalizeSum(AnyNode, AnyNode)	Hàm hữu tỉ hóa một tổng

Bảng 5.10 Các phương thức của lớp GenneralRationalExpression

Các phương thức $numerator$, $denominator$, $rationalExpand$, $rationalizeSum$ $rationalizeExpression$, $rationalGRE$ được trình bày ở Phụ lục 34.

6 Các toán tử trong hệ thống SMC

6.1 Khai triển Taylor

6.1.1 Toán tử $Derivative$

Định nghĩa 6.1: Cho biểu thức đại số u. Toán tử $Derivative(u, x)$ trả về đạo hàm bậc một của u tính theo biến x được định nghĩa bởi các luật sau:

1. Nếu $u = x$ thì
$$Derivative(u, x) \to 1$$
2. Nếu $u = x^w$ thì
$Derivative(u, x) \to w * v^{w-1} * Derivative(v, x) + Derivative(w, x) * v^w * \ln(v)$
3. Giả sử u là một tổng và cho $v = Operand(u, 1)$ và $w = u - v$ khi đó
$$Derivative(u, x) \to Derivative(v, x) + Derivative(w, x)$$
4. Giả sử u là một tích và cho $v = Operand(u, 1)$ và $w = u/v$ khi đó
$$Derivative(u, x) \to w * Derivative(v, x) + v * Derivative(w, x)$$
5. Nếu $u = \sin(v)$
$$Derivative(u, x) \to \cos(v) * Derivative(v, x)$$
6. Nếu $u = \cos(v)$
$$Derivative(u, x) \to -\sin(x) * Derivative(v, x)$$
7. Nếu $u = \tan(v)$
$$Derivative(u, x) \to \sec^2(v) * Derivative(v, x)$$
8. Nếu $u = \cot(v)$
$$Derivative(u, x) \to -\csc^2(v) * Derivative(v, x)$$
9. Nếu $u = sec(v)$
$$Derivative(u, x) \to \sec(v) * \tan(v) * Derivative(v, x)$$
10. Nếu $u = csc(v)$
$$Derivative(u, x) \to -\csc(v) * cot(v) * Derivative(v, x)$$
11. Nếu $FreeOf(u, x) = true$ thì
$$Derivative(u, x) \to 0$$

- Thủ tục thực hiện toán tử $Derivative$

Procedure $Derivative$(u, x); Input

```
        u : một biểu thức đại số;
        x : một biến tổng quát
Output
        Đạo hàm bậc một của u với biến x;
Local Variables r;
Begin
        if u=x then
                Return 1;
        else if Kind(u)=" ^ " then
                r := DerivativePower(u, x);
        else if Kind(u)=" * " then
                r := DerivativeProduct(u, x);
        else if Kind(u)=" + " then
                r := DerivativeSum(u, x);
        else if Kind(u)=" Trigonometric " then
                r := DerivativeTrigonometric(u, x);
        else if Kind(u)=" exp " then
                r := u * Derivative(u, x);
        else if Kind(u)=" ln " then
                r := Derivative(u, x)/u;
        else if FreeOf(u, x) = false
                r := 0;
        return r;
End
```

Hình 6.1 Thủ tục thực hiện toán tử $Derivative$

6.1.2 Toán tử $HigherDerivative$

Cho u là một biểu thức đại số, toán tử $HigherDerivative$ sẽ trả về đạo hàm bậc n của biểu thức u với biến x

- Thủ tục thực hiện toán tử $HigherDerivative$

```
Procedure HigherDerivative(u, x, n);
Input
        u : một biểu thức đại số;
        x : một biến tổng quát
        n : bậc đạo hàm
Output
        Đạo hàm bậc n của u với biến x;
Local Variables r, t;
Begin
        while n ≠ 0 do
                u := Derivative(u, x);
                n := n-1;
        return u;
End
```

Hình 6.2 Thủ tục thực hiện toán tử $HigherDerivative$

6.1.3 Toán tử $Taylor$Series

Định nghĩa 6.2: Cho f là một hàm số có đạo hàm riêng liên tục tới cấp $n+1$ trong khoảng nào đó chứa điểm a. Chuỗi taylor được tạo bởi hàm f tại điểm $x = a$ có dạng

$$f(a) + f'(0)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n + \cdots$$
$$= \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Cho u là biểu thức đại số toán tử $TaylorSeries$ sẽ trả về chuỗi taylor cấp n của u theo biến x tại điểm a.

- Thủ tục thực hiện toán tử $TaylorSeries$

```
Procedure TaylorSeries(u, x, n, a);
Input
        u : một biểu thức đại số;
        x : một biến tổng quát
        n : cấp khai triển
        a : giá trị khai triển x=a
Output
        Chuỗi taylor cấp n của u theo biến x tại điểm a;
Local Variables r, k, d, f;
Begin
        for k := 0 to n do
                d := HigherDerivative(u, x, n);
                d := Substitute(a, x);
                f := Factorial(k);
                r := (d * (x - a)^k)/f;
        return Simplify(r);
End
```

Hình 6.3 Thủ tục thực hiện toán tử $TaylorSeries$

6.2 Các toán tử khác

6.2.1 Toán tử $MINF$

Định nghĩa 6.3: [15] Cho $F_1(z),\ F_2(z)$ là các biểu thức đại số với biến tổng quát z, $F(z) = MINF(F_1(z), F_2(z))$ sẽ được tính bởi công thức sau:

$$[z^i]F(z) = MIN\left([z^i]F_1(z), [z^i]F_2(z)\right) \quad \forall i \geq 0$$

Toán tử $MINF$ trả về a_i là hệ số nhỏ nhất trong các hệ số của z^i trong F_1 và F_2.

Ví dụ:

$$\mathrm{MINF}(1 + \mathrm{z} + \mathrm{z}^2, 1 + 2\mathrm{z}^2) = 1 + \mathrm{z}^2$$

- Thủ tục thực hiện toán tử $MINF$

```
Procedure MINF(a,b,x)
Input
        a, b : là các đa thức tổng quát;
        x : một biến tổng quát
Output
        a_i là hệ số nhỏ nhất trong các hệ số của x^i trong a và b;
Local Variables m, da, db, ca, cb, r;
Begin
        da := DegreeGPE(a, x);
        db := DegreeGPE(b, x);
        if da < db then
                m := db;
        else
                m := da;
        for i:= 0 to m do
                ca := CoefficientGPE(a, x, i);
                cb := CoefficientGPE(b, x, i);
                if ca < cb
                        r := r + ca * x^i;
                else
                        r := r + cb * x^i;
        return Simplify(r);
End
```

Hình 6.4 Thủ tục thực hiện toán tử $MINF$

6.2.2 Toán tử $MAXF$

Định nghĩa 6.4: [15] Cho $F_1(z)$, $F_2(z)$ là các biểu thức đại số với biến tổng quát $z, F(z) = MAXF(F_1(z), F_2(z))$ sẽ được tính bởi:

$$[z^i]F(z) = MAX\left([z^i]F_1(z), [z^i]F_2(z)\right) \quad \forall i \geq 0$$

Toán tử $MAXF$ trả về a_i là hệ số lớn nhất trong các hệ số của z^i trong F_1 và F_2.

Ví dụ:

$$\mathrm{MAXF}(1 + \mathrm{z} + \mathrm{z}^2\ , 1 + 2\mathrm{z}^2) = 1 + \mathrm{z} + 2\mathrm{z}^2$$

Thủ tục thực hiện toán tử $MAXF$

```
Procedure MAXF(a,b,x)
Input
        a, b : là các đa thức tổng quát;
        x : một biến tổng quát
Output
        a_i là hệ số nhỏ nhất trong các hệ số của x^i trong a và b;
Local Variables m, da, db, ca, cb, r;
Begin
        da := DegreeGPE(a, x);
```

```
        db := DegreeGPE(b, x);
        if da < db then
                m := db;
        else
                m := da;
        for i:= 0 to m do
                ca := CoefficientGPE(a, x, i);
                cb := CoefficientGPE(b, x, i);
                if ca < cb
                        r: = r + cb * x^i;
                else
                        r: = r + ca * x^i;
        return Simplify(r);
End
```

Hình 6.5 Thủ tục thực hiện toán tử $MAXF$

6.2.3 Toán tử $DEUP$

Định nghĩa 6.5: Cho F(z) là một hàm số thì

$$DEDUP\big(F(z)\big) = \sum_{[z^i]F(z)>0} z^i$$

Toán tử DEDUP trả về hàm sinh mới với các hệ số của z^i được gán bằng 1 khi $[z^i]F(z) > 0$. [15]

Ví dụ: $DEDUP\big(F(z)\big) = 1 + z + z^2 + z^3 + .. = 1/(1 - z)$

- Thủ tục thực hiện toán tử $DEDUP$

```
Procedure DEDUP(u, x, n, a);
Input
        u : một biểu thức đại số;
        x : một biến tổng quát
        n : cấp khai triển
        a : giá trị khai triển x = a
Output
        Chuỗi taylor rút gọn cấp n của u theo biến x tại điểm a theo Định nghĩa 6.5;
Local Variables r, k, d, f;
Begin
        for k := 0 to n do
                d := HigherDerivative(u, x, n);
                d := Substitute(a, x);
                f := Factorial(k);
                if(d/f > 0) then
                        r := (x - a)^k;
                else
                        r := (d * (x - a)^k)/f;
        return Simplify(r);
```

End

Hình 6.6 Thủ tục thực hiện toán tử $DEDUP$

Các phương thức thực hiện các thủ tục trên được trình trong Phụ lục 35.

7 Kiểm thử

- Kiểm thử phương thức $derivative$ (đạo hàm theo biến x)

Stt	Đầu vào	Kết quả mong đợi	Kết quả thực tế
1	x	x	x
2	x^3	$3x^2$	$3x^2$
3	$\sqrt{x}$	$\frac{1}{2\sqrt{x}}$	$\frac{1}{2}x^{-\frac{1}{2}}$
4	$\frac{1}{x}$	$\frac{-1}{x^2}$	$-1(x^{-2})$
5	$\sin(x)$	$\cos(x)$	$\cos(x)$
6	$\cos(x)$	$-\sin(x)$	$-\sin(x)$
7	$\tan(x)$	$\sec^2(x)$	$(\sec(x))^2$
8	$\cot(x)$	$-\csc^2(x)$	$-1\big(csc(x)\big)^2$
9	$ln(x)$	$\frac{1}{x}$	x^{-1}
10	$(2x+1)^3$	$6(2x+1)^2$	$6(1+2x)^2$
11	$\sqrt{2x+1}$	$\frac{-1}{(2x+1)^{\frac{3}{2}}}$	$-1(1+2x)^{-\frac{3}{2}}$
12	$\frac{1}{4x+3}$	$-\frac{4}{(4x+3)^2}$	$-4(3+4x)^{-2}$
13	$\sin(5x^2+1)$	$10xcos(5x^2+1)$	$10(\cos(1+5x^2))x$
14	$\cos(5x^2+1)$	$10xsin(5x^2+1)$	$-10(\sin(1+5x^2))x$
15	$\tan(5x^2+1)$	$10xsec^2(5x^2+1)$	$10\big(sec(1+5x^2)\big)^2x$
16	$\cot(5x^2+1)$		$-10\big(csc(1+5x^2)\big)^2x$
17	$\ln(2x+3)$	$\frac{2}{2x+3}$	$2(3+2x)^{-2}$
18	$5x^3-3x^2+10x-5$	$15x^2-6x+10$	$10+(-6)x+15x^2$

- Kiểm thử phương thức $higherOrderDerivative$ (đạo hàm bậc n theo biến x)

Stt	Đầu vào	Kết quả mong đợi	Kết quả thực tế
1	$5x^3-3x^2+10x-5,2$	$30x-6$	$-6+30x$
2	$\sqrt{x},\ 3$	$\frac{3}{8x^{5/2}}$	$\frac{3}{8}x^{-\frac{5}{2}}$
3	$2x^x, 1$	$2x^x(\ln(x)+1)$	$2(x^x+\ln(x)\,x^x)$

- Kiểm thử phương thức $taylorSeries$ (Tìm chuỗi taylor theo bậc n và biến x)

Stt	Đầu vào	Kết quả mong đợi	Kết quả thực tế
1	$\exp(x), 3$	$1 + x + \frac{x^2}{2} + \frac{x^3}{3}$	$1 + x + \frac{x^2}{2} + \frac{x^3}{3}$
2	$\exp(-(x^2)), 8$	$1 - x^2 + \frac{x^4}{2} - \frac{x^6}{6} + \frac{x^8}{24}$	$1 - x^2 + \frac{x^4}{2} - \frac{x^6}{6} + \frac{x^8}{24}$
3	$\exp(\sin(x)), 5$	$1 + x + \frac{x^2}{2} - \frac{x^4}{8} - \frac{x^5}{15}$	$1 + x + \frac{x^2}{2} - \frac{x^4}{8} - \frac{x^5}{15}$
4	$\frac{1}{x}, 3$	$1 - (x-1) + (x-1)^2 - (x-1)^3$	$1 + (-1) * (-1 + x) + (-1 + x)^2 + (-1)(-1 + x)^3$
5	$\frac{1}{1-3}, 4$	$1 + x + x^2 + x^3 + x^4$	$1 + x + x^2 + x^3 + x^4$
6	$\exp(x)\sin(x), 6$	$x + x^2 + \frac{x^3}{3} - \frac{x^5}{30} - \frac{x^6}{90}$	$x + x^2 + \frac{1}{3}x^3 + \frac{-1}{30}x^5 + \frac{-1}{90}x^6$
7	$(x+1)^{\frac{1}{2}}, 5$	$1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} - \frac{5x^4}{128} + \frac{7x^5}{256}$	$1 + \frac{1}{2}x + \frac{-1}{8}x^2 + \frac{1}{16}x^3 + \frac{-5}{128}x^4 + \frac{7}{256}x^5$

- Kiểm thử phương thức $minF$

Stt	Đầu vào	Kết quả mong đợi	Kết quả thực tế
1	$1 + x + x^2, 1 + 2x^2$	$1 + x^2$	$1 + x^2$
2	$1 + \frac{1}{2}x + \frac{-1}{8}x^2 + \frac{1}{16}x^3 + \frac{-5}{128}x^4 + \frac{7}{256}x^5,$ $x + x^2 + \frac{x^3}{3} - \frac{x^5}{30} - \frac{x^6}{90}$	$\frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 - \frac{5}{128}x^4 - \frac{1}{30}x^5 - \frac{1}{90}x^6$	$\frac{1}{2}x + \frac{-1}{8}x^2 + \frac{1}{16}x^3 + \frac{-5}{128}x^4 + \frac{-1}{30}x^5 + \frac{-1}{90}x^6$
3	$4x^5 - 15x^3 - 11x - 1,$ $-14x^5 + 14x^4 + 22x^2 + 14$	$-1 - 11x - 15x^3 - 14x^5$	$-1 + (-11)x + (-15)x^3 + (-14)x^5$

- Kiểm thử phương thức $maxF$

Stt	Đầu vào	Kết quả mong đợi	Kết quả thực tế
1	$1 + x + x^2, 1 + 2x^2$	$1 + x + 2x^2$	$1 + x + 2x^2$

2	$1+\frac{1}{2}x+\frac{-1}{8}x^2+\frac{1}{16}x^3+\frac{-5}{128}x^4+\frac{7}{256}x^5$, $x+x^2+\frac{x^3}{3}-\frac{x^5}{30}-\frac{x^6}{90}$	$1+x+x^2+\frac{1}{3}x^3+\frac{7}{256}x^5$	$1+x+x^2+\frac{1}{3}x^3+\frac{7}{256}x^5$
3	$4x^5-15x^3-11x-1$, $-14x^5+14x^4+22x^2+14$	$14+22x^2+14x^4+4x^5$	$14+22x^2+14x^4+4x^5$

- Kiểm thử phương thức $dedup$ (*Note: mặc định chuỗi taylor sẽ được tính tại giá trị x=0*)

Stt	Đầu vào	Kết quả mong đợi	Kết quả thực tế
1	$1/(1-x), 3$	$1+x+x^2+x^3$	$1+x+x^2+x^3$
2	$\frac{5}{x+1}, 4$	$1-5x+x^2-5x^3+x^4$	$1-5x+x^2-5x^3+x^4$
3	$sin(x), x=1,3$	$x+\frac{-1}{2}\sin(1)(x-1)^2+\frac{-1}{6}\cos(1)(x-1)^3$	$x+\frac{-1}{2}\sin(1)\left((-1)+x\right)^2+\frac{-1}{6}\cos(1)\left((-1)+x\right)^3$

Kết luận

Kết quả đạt được

Trong phạm vị luân văn này tôi hướng tới mục đích là tìm hiểu, nghiên cứu và phát triển một hệ thống đại số máy tính miễn phí nhằm thay thế hệ thống đại số máy tính thương mại sẵn có trong việc đáp ứng các yêu cầu của hệ thống SMC. Qua 7 chương, luận văn đã trình bày về phương pháp tiếp cận, phân tích và giải quyết các vấn đề gặp phải trong quá trình xây dựng hệ thống.

Các kết quả đặt được:

- Xây dựng được hệ thống đại số máy tính cơ bản cho phép thao tác với biểu thức đại số
    - Phân tích chuỗi đầu vào để nhận biết biểu thức.
    - Tính giá trị biểu thức.
    - Rút gọn biểu thức.
- Xử lý đa thức
    - Đa thức một biến, nhiều biến.
    - Các phép toán cơ bản trên đa thức.
    - Khai triển đa thức.
- Xây dựng các hàm xử lý cho hệ thống SMC
    - Tìm chuỗi taylor vủa một hàm số tại một giá trị bất kỳ, đến một hệ số bất kỳ.
    - Xây dựng hàm MAXF, MINF, TRUNC, DEDUP.

Hướng nghiên cứu trong lương lai

Với những gì đã làm được trọng phạm vi luận văn tôi hy vọng trong tương lai sẽ có những cải thiện giúp tăng chất lượng của hệ thống**.**

- Hoàn thiện hệ thống
    - Khả năng rút gọn biểu thức hữu tỉ.
    - Khả năng phân tích chuỗi để nhận biết biểu thức.
    - Hỗ trợ xử lý biểu thức logic.
    - Tăng hiệu xuất thực hiện bằng cách cải thiện các thuật toán, các phương thức tính toán để giảm thời gian.
    - Phát triển lại bằng ngôn ngữ C++ để có thể cạnh tranh về hiệu năng với các hệ thống đã có.
- Thêm giao diện để thân thiện với người dùng.

Tài liệu tham khảo

Tiếng việt

1. Đỗ Xuân Lôi (1999), *Cấu trúc dữ liệu và giải thuật*, Nhà xuất bản thống kê.
2. Trương Ninh Thuận – Đặng Đức Hạnh (2013), *Giáo trình phân tích và thiết kế hướng đối tượng*, Nhà xuất bản Đại Học Quốc gia Hà Nội.

Tiếng anh

3. Hazem Mohamed El-Alfy. (1997). *Computer algebraic and its applications*, B.Sc., Faculty of Engineering, Alexandria University.
4. Chee Keng Yap. (2000). *Fundamental Problems of Algorithmic Algebra*, Oxford University Press, New York.
5. David Musser. (1971). *Algorithms for Polynomial Factorization*, PhD thesis, Department of Computer Science, University of Wisconsin.
6. F. Winkler. (1996). *Polynomial Algorithms in Computer Algebra*, SpringerVerlag, New York.
7. Hans Vangheluwe, Bhama Sridharan and Indrani A.V. (2013). *An algorithm to implement a canonical representation of algebraic expression and equations in AToM*.
8. Henri Cohen. (1993). *A Course in Computational Algebraic Number Theory*, Springer-Verlag, New York.
9. J. H. Davenport, Y. Siret, and E. Tournier. (1988). *Computer Algebra, Systems and Algorithms for Algebraic Computation*, Academic Press, New York.
10. James F. Epperson. (2002). *An Introduction to Numerical Methods and Analysis*, John Wiley & Sons, New York.
11. Joachim von zur Gathen and J¨urgen Gerhard. *Modern Computer Algebra*, Cambridge University Press, New York, 1999.
12. Joel S. Cohen (2002). *Computer Algebra and Symbolic Computation: Elementary Algorithms*. A K Peters, Natick, MA
13. Joel S. Cohen (2002). *Computer Algebra and Symbolic Computation: Mathematical Methods*, A K Peters, Natick, MA.
14. John W. Gray. (1997). *Mastering Mathematica, Programming Methods and Applications*, Second Edition. Academic Press, New York.
15. Loi Luu, Shweta Shinde, Prateek Saxena. (2014). *A Model Counter For Constraints Over Unbounded Strings*, School of Computing, National University of Singapore.
16. Michael J. Wester. (1999). *Computer Algebra Systems, A Practical Guide*, John Wiley & Sons, Ltd., New York.
17. Richard Andrew Mealing. (2010). *Simplifying Numerical Expressions*, The University of Liverpool.
18. Richard J. Fateman. (1999). *Symbolic mathematics system evaluators*, In Michael J. Wester, editor, Computer Algebra Systems, A Practical Guide, pages 255–284. John Wiley & Sons, Ltd., New York.

19. Richard J. Gaylord, N. Kamin, Samuel, and Paul R. Wellin. (1996). *An Introduction to Programming with Mathematica*, Second Edition. Springer-Verlag, New York.
20. Richard Liska, Ladislav Drska, Jiri Limpouch, Milan Sinor, Michael Wester, Franz Winkler. (1999). *Computer algebraic, Algorithms, System and Applications*.
21. Richard Zippel. (1993). *Effective Polynomial Computation*, Kluwer Academic Publishers, Boston.
22. Stephen Wolfram. *The Mathematica Book. Fourth Edition*. Cambridge University Press., New York, 1999.

Website

23. https://en.wikipedia.org/wiki/Computer_algebra_system
24. https://en.wikipedia.org/wiki/Taylor_series
25. http://www.le.ac.uk/users/dsgp1/COURSES/DERIVATE/TAYLOR.PDF
26. http://mathworld.wolfram.com/TaylorSeries.html

Phụ lục

Phụ lục 1: Toán tử Map

Cho u là biểu thức phức hợp với $n = NumberOfOperator(u)$ và $F(x), G(x, y, \dots, z)$ là các toán tử. Toán tử Map có 2 dạng là:

$$Map(F, u)$$

$$Map(G, u, y, \dots z)$$

Map(F,u)

$Map(F, u)$ sẽ tạo ra một biểu thức mới với toán tử chính là $Kind(u)$ và các toán hạng là $F(Operand(u, 1)), \dots, F(Operand(u, n))$.

Ví dụ:

$$F(x) = x^2 \text{ và } u = a + b$$

Khi đó ta có toán tử chính là $Kind(u) = " + "$ và các toán hạng là:

$$F\big(Operand(u, 1)\big) = a^2, F(Operand(u, 2)) = b^2$$

Vậy $Map(F, u) = a^2 + b^2$

Map(G,u,y,…,z)

$Map(G, u, y, \dots, z)$ sẽ tạo ra một biểu thức mới với toán tử chính là $Kind(u)$ và các toán hạng là $G(Operand(u, 1), y, \dots, z), \dots\ G(Operand(u, n), y, \dots, z))$.

Ví dụ: Cho $G(x) = x^2 + y^3 + z^4$ và $u = a + b$

$$G(Operand(u, 1), c, d) = a^2 + c^3 + d^4, G(Operand(u, 2), c, d) = b^2 + c^3 + d^4$$

Vậy $Map(G, u, c, d) = a^2 + b^2 + 2c^3 + 2d^4$

Phụ lục 2: Toán tử Construct

Cho f là toán tử $(+, -, *, \text{^ } \dots)$ và danh sách các toán hạng $L = [x_1, x_2, \dots, x_n]$. Toán tử $Construct(f, L)$ tạo ra một biểu thức có toán hạng chính là f và các toán tử là $x_1, x_2, \dots, x_n$.

Ví dụ: Cho $f = ' + '$ và $L = [2, x, y]$

$$Construct(f, L) \rightarrow 2 + x + y$$

Phụ lục 3: Thủ tục $EvaluateProduct$

Procedure $EvaluateProduct$(u,v); Input u, v: là các số nguyên hoặc phân số với mẫu số khác 0 Output là một phân số ở dạng chuẩn hoặc số nguyên

```
Local Variables
        d, n;
Begin
        if v = integer and w = integer then
                return (v * w);
        else
                n := NumeratorFun(v) * NumeratorFun(w);
                d := DenominatorFun(v) * DenominatorFun(w);
                return SimplifyRationalNumber(n/d);
End
```

Phụ lục 4: Thủ tục $EvaluateSum$

```
Procedure EvaluateSum(u,v);
Input
        u, v: là số nguyên hoặc phân số với mẫu số khác 0
Output
        là một số nguyên hoặc phân số ở dạng chuẩn;
Local Variables
        d, n;
Begin
        if v = integer and w = integer then
                return (v + w);
        else
                n := NumeratorFun(v) * DenominatorFun(w)
                                + NumeratorFun(w) * DenominatorFun(v);
                d := DenominatorFun(v) * DenominatorFun(w);
                return SimplifyRationalNumber(n/d);
End
```

Phụ lục 5: Thủ tục $EvaluatePower$

```
Procedure EvaluatePower(u,n);
Input
        u : là một số nguyên hoặc một phân số với mẫu số khác 0;
        n : là một số nguyên;
Output
        là một số nguyên hoặc phân số ở dạng chuẩn;
Local Variables
        d, n;
Begin
```

```
        if n < 0 then
                n := DenominatorFun(u);
                d := NumeratorFun(u);
        else
                n := NumeratorFun(u);
                d := DenominatorFun(u);
        n := n^AbsoluteValue(n);
        d := d^AbsoluteValue(n);
        return SimplifyRNE(n/d);
End
```

Phụ lục 6: Phương thức evaluateProduct

```
public static AnyNode evaluateProduct(AnyNode v, AnyNode w) {
      if (v.getKey() == 0 && w.getKey() == 0) {
            return new AnyNode(0, v.getValue() * w.getValue());
      } else {
            int numerator = v.numerator() * w.numerator();
            int denominator = v.denominator() * w.denominator();
            return simplifyRationalNumber(new AnyNode(-37, numerator,
                        denominator));
      }
}
```

Phụ lục 7: Phương thức evaluateSum

```
public static AnyNode evaluateSum(AnyNode v, AnyNode w) {
      if (v.getKey() == 0 && w.getKey() == 0) {
            return new AnyNode(0, v.getValue() + w.getValue());
      } else {
            int   numerator   =   v.numerator()   *   w.denominator()   +
w.numerator()* v.denominator();
            int denominator = v.denominator() * w.denominator();
            return simplifyRationalNumber(new AnyNode(-37, numerator,
                              denominator));
      }
}
```

Phụ lục 8: Phương thức evaluatePower

```
public static AnyNode evaluatePower(AnyNode base, AnyNode exponent) {
      int deno, num;
      if (exponent.getValue() < 0) {
            num = base.denominator();
            deno = base.numerator();
      } else {
            num = base.numerator();
            deno = base.denominator();
      }
      num = (int) Math.pow(num, Math.abs(exponent.getValue()));
      deno = (int) Math.pow(deno, Math.abs(exponent.getValue()));
      AnyNode result = new AnyNode(-37, num, deno);
      return simplifyRationalNumber(result);
}
```

Phụ lục 9: Phương thức *simplifyIntegerPower*

```
private static AnyNode simplifyIntegerPower(AnyNode anyNode) {
	AnyNode base = anyNode.operand(1); // co so
	AnyNode exponent = anyNode.operand(2); // so mu
	if (exponent.getValue() == 0 && exponent.getKey() == 0) { // b=0
		return new AnyNode(0, 1);
	} else if (exponent.getValue() == 1) {// b=1
		return anyNode.operand(1);
	} else if (base.getKey() == 0 || base.getKey() == -37) {
		return simplifyConstant(anyNode);
	} else if (base.getKey() == -5) {
		AnyNode p = simplifyProduct(new AnyNode(-3, base.operand(2),
				exponent));
		AnyNode temp = new AnyNode(-5, base.operand(1), p);
		if (p.getKey() == 0) {
			return simplifyIntegerPower(temp);
		} else {
			return temp;
		}
	} else if (base.getKey() == -3) {
		AnyNode temp = new AnyNode(-3);
		for (int i = 0; i < base.nop(); i++) {
			temp.leaf.add(simplifyIntegerPower(new AnyNode(-5, base
					.operand(i + 1), exponent)));
		}
		return simplifyProduct(temp);
	} else {
		return anyNode;
	}
}
```

Phụ lục 10: Phương thức *simplifyProductRec*

```
private static ArrayList<AnyNode> simplifyProductRec(ArrayList<AnyNode>
list) {
	ArrayList<AnyNode> result = new ArrayList<AnyNode>();
	if (list.size() == 2) {
		AnyNode u1 = list.get(0);
		AnyNode u2 = list.get(1);
		if (u1.getKey() != -3 && u2.getKey() != -3) {
			// ca 2 khong phai la tich
			if ((u1.getKey() == 0 || u1.getKey() == -37)
					&& (u2.getKey() == 0 || u2.getKey() == -37))
{
				// 1. Ca hai la constant(integer or phan so)
				AnyNode r = simplifyConstant(new AnyNode(-3, u1,
u2));
				if (r.getValue() == 1)
					result = null;
				else
					result.add(r);
```

```
                } else if (u1.getValue() == 1) {
                        // 2.1 u1 = 1
                        result.add(u2);
                } else if (u2.getValue() == 1) {
                        // 2.2 u2=1
                        result.add(u1);
                } else if (u1.base().equal(u2.base())) {
                        // hai toan hang co co so bang nhau
                        AnyNode exponent = simplifySum(new AnyNode(-1,
                                        u1.exponent(), u2.exponent()));
                        AnyNode   r   =   simplifyPower(new   AnyNode(-5,
u1.base(),exponent));
                        if (r.getKey() == 0 && r.getValue() == 1) {
                                result = null;
                        } else {
                                result.add(r);
                        }
                } else if (!u1.compare(u2)) {
                        // neu u2 < u1
                        result.add(u2);
                        result.add(u1);
                } else {
                        // neu 4 luat tren khong phu hop
                        result = list;
                }
                return result;
        } else { // it nhat 1 trong 2 cai la tich
                if (u1.getKey() == -3 && u2.getKey() == -3) {
                        // ca hai cung la tich
                        result = mergeProducts(u1.leaf, u2.leaf);
                } else if (u1.getKey() == -3 && u2.getKey() != -3) {
                        // u1 la tich
                        ArrayList<AnyNode>        temp        =        new
ArrayList<AnyNode>();
                        temp.add(u2);
                        result = mergeProducts(u1.leaf, temp);
                } else if (u1.getKey() != -3 && u2.getKey() == -3) {
                        // u2 la tich
                        ArrayList<AnyNode>        temp        =        new
ArrayList<AnyNode>();
                        temp.add(u1);
                        result = mergeProducts(temp, u2.leaf);
                }
                return result;
        }
    } else {
        // number of operators > 2
        AnyNode a = list.get(0);
        ArrayList<AnyNode> temp = simplifyProductRec(rest(list));
        if (a.getKey() == -3) {
                result = mergeProducts(a.leaf, temp);
        } else {
                ArrayList<AnyNode> t = new ArrayList<AnyNode>();
                t.add(a);
                result = mergeProducts(t, temp);
```

```
            }
            return result;
      }
}
```

Phụ lục 11: Phương thức *mergeProducts*

```
private static ArrayList<AnyNode> mergeProducts(ArrayList<AnyNode> p,
                  ArrayList<AnyNode> q) {
      if (p == null || p.size() == 0) {
            return q;
      } else if (q == null || q.size() == 0) {
            return p;
      } else {
            AnyNode p1 = p.get(0);
            AnyNode q1 = q.get(0);
            ArrayList<AnyNode> temp = new ArrayList<AnyNode>();
            temp.add(p1);
            temp.add(q1);
            ArrayList<AnyNode> h = new ArrayList<AnyNode>();
            h = simplifyProductRec(temp);
            if (h == null) {
                  return mergeProducts(rest(p), rest(q));
            } else if (h.size() == 2
                        && ((h.get(0).equal(q1)
                                    && h.get(1).equal(p1)) || (h.get(1)
                                          .equal(q1)                    &&
h.get(0).equal(p1)))) {
                  ArrayList<AnyNode> t = new ArrayList<AnyNode>();
                  if (h.get(0).equal(q1) && h.get(1).equal(p1)) {
                        t.add(q1);
                        return adjoin(t, mergeProducts(p, rest(q)));
                  } else {
                        t.add(p1);
                        return adjoin(t, mergeProducts(rest(p), q));
                  }
            } else {
                  return adjoin(h, mergeProducts(rest(p), rest(q)));
            }
      }
}
```

Phụ lục 12: Phương thức *simplifySumRec*

```
private static ArrayList<AnyNode> simplifySumRec(ArrayList<AnyNode> list)
{
      ArrayList<AnyNode> result = new ArrayList<AnyNode>();
      if (list.size() == 2) {
            // Co 2 toan hang
            AnyNode u1 = list.get(0);
```

```
            AnyNode u2 = list.get(1);
            if (u1.getKey() != -1 && u2.getKey() != -1) {
                  // ca 2 khong phai la tong
                  if ((u1.getKey() == 0 || u1.getKey() == -37)
                        && (u2.getKey() == 0 || u2.getKey() == -37)) {
                        // 1. Ca hai la constant(integer or phan so)
                        AnyNode r = simplifyConstant(new AnyNode(-1, u1,
u2));
                        if (r.getValue() == 0 && r.getKey() == 0)
                              result = null;
                        else
                              result.add(r);
                  } else if (u1.getValue() == 0 && u1.getKey() == 0) {
                        // 2.1 u1 = 0
                        result.add(u2);
                  } else if (u2.getValue() == 0 && u2.getKey() == 0) {
                        // 2.2 u2=1
                        result.add(u1);
                  } else if (u1.term().equal(u2.term())) {
                        // hai toan hang co so hang bang nhau
                     AnyNode constant = simplifyConstant(new AnyNode(-1,
                                 u1.constant(), u2.constant()));
                        if (constant.getKey() == 0 && constant.getValue()
== 0)
                              result = null;
                        else    if    (constant.getKey()    ==    0    &&
constant.getValue() == 1) {
                              result = u1.term().leaf;
                        } else {
                              AnyNode t = new AnyNode(-3);
                              t.leaf.add(constant);
                              AnyNode term = u1.term();
                              if (term.getKey() == -3) {
                               for (int i = 1; i <= term.nop(); i++) {
                                          t.leaf.add(term.operand(i));
                                    }
                              } else
                                    t.leaf.add(term);
                              result.add(t);
                        }
                  } else if (!u1.compare(u2)) {
                        // neu u2 < u1
                        result.add(u2);
                        result.add(u1);
                  } else {
                        // neu 4 luat tren khong phu hop
                        result = list;
                  }
                  return result;
            } else { // it nhat 1 trong 2 cai la tong
                  if (u1.getKey() == -1 && u2.getKey() == -1) {
                        // ca hai cung la tong
                        result = mergeSums(u1.leaf, u2.leaf);
                  } else if (u1.getKey() == -1 && u2.getKey() != -1) {
                        // u1 la tong con u2 khong la tong
```

```
                        ArrayList<AnyNode> temp = new
ArrayList<AnyNode>();
                        temp.add(u2);
                        result = mergeSums(u1.leaf, temp);
                  } else if (u1.getKey() != -1 && u2.getKey() == -1) {
                        // u2 la tong con u1 khong la tong
                        ArrayList<AnyNode> temp = new
ArrayList<AnyNode>();
                        temp.add(u1);
                        result = mergeSums(temp, u2.leaf);
                  }
                  return result;
            }
      } else {
            // number of operators > 2
            AnyNode a = list.get(0);
            ArrayList<AnyNode> temp = simplifySumRec(rest(list));
            if (a.getKey() == -1) {
                  result = mergeSums(a.leaf, temp);
            } else {
                  ArrayList<AnyNode> t = new ArrayList<AnyNode>();
                  t.add(a);
                  result = mergeSums(t, temp);
            }
            return result;
      }
}
```

Phụ lục 13: Phương thức *mergeSums*

```
private static ArrayList<AnyNode> mergeSums(ArrayList<AnyNode> p,
            ArrayList<AnyNode> q) {
      if (p == null || p.size() == 0) {
            return q;
      } else if (q == null || q.size() == 0) {
            return p;
      } else {
            AnyNode p1 = p.get(0);
            AnyNode q1 = q.get(0);
            ArrayList<AnyNode> temp = new ArrayList<AnyNode>();
            temp.add(p1);
            temp.add(q1);
            ArrayList<AnyNode> h = simplifySumRec(temp);
            if (h == null) {
                  return mergeSums(rest(p), rest(q));
            } else if (h.size() == 2
                  &&  ((h.get(0).equal(q1)  &&  h.get(1).equal(p1))  ||
(h.get(1).equal(q1) && h.get(0).equal(p1)))) {
                  ArrayList<AnyNode> t = new ArrayList<AnyNode>();
                  if (h.get(0).equal(q1) && h.get(1).equal(p1)) {
                        t.add(q1);
                        return adjoin(t, mergeSums(p, rest(q)));
                  } else {
                        t.add(p1);
                        return adjoin(t, mergeSums(rest(p), q));
                  }
            } else {
                  return adjoin(h, mergeSums(rest(p), rest(q)));
```

```
            }
        }
}
```

Phụ lục 14: Phương thức *adjoin* sử dụng để nối hai danh sách nút

```
private static ArrayList<AnyNode> adjoin(ArrayList<AnyNode> a,
                    ArrayList<AnyNode> b) {
        ArrayList<AnyNode> r = new ArrayList<AnyNode>();
        r.addAll(a);
        r.addAll(b);
        return r;
}
```

Phụ lục 15: Phương thức *rest* sử dụng để loại bỏ phần tử đầu tiên của danh sách

```
public static ArrayList<AnyNode> rest(ArrayList<AnyNode> a) {
        a.remove(0);
        return a;
}
```

Phụ lục 16: Phương thức *simplifyFactorial* tính giai thừa của một là số nguyên

```
private static AnyNode simplifyFactorial(AnyNode node) {
        if (node.operand(1).getKey() == 0) {
                if (node.operand(1).getValue() < 0)
                        return new AnyNode(-38);
                else {
                        int value = factorial(node.operand(1).getValue());
                        return new AnyNode(0, value);
                }
        } else {
                return node;
        }
}
```

Phụ lục 17: Phương thức *factorial* tính giai thừa của một số nguyên

```
public static int factorial(int n) {
                if (n == 0)
                        return 1;
                else
                        return n * factorial(n - 1);
        }
```

Phụ lục 18: Phương thức *simplifyRationalNumber* tính giá trị của biểu thức hữu tỉ với các toán hạng là hằng số (số nguyên hoặc phân số).

```
private static AnyNode simplifyRationalNumber(AnyNode u) {
        if (u.getKey() != -37)
                // u khong la phan so
                return u;
```

```
	AnyNode result = new AnyNode();
	if (u.operand(1).getKey() == -37 || u.operand(2).getKey() == -37) {
		// tu hoac mau cua u la phan so
		AnyNode rNumerator = new AnyNode();
		AnyNode rDenominator = new AnyNode();
		if (u.operand(1).getKey() == -37)
			rNumerator = simplifyRationalNumber(u.operand(1));
		else
			rNumerator = u.operand(1);
		if (u.operand(2).getKey() == -37)
			rDenominator = simplifyRationalNumber(u.operand(2));
		else
			rDenominator = u.operand(2);
		int t = rNumerator.numerator() * rDenominator.denominator();
		int m = rNumerator.denominator() * rDenominator.numerator();
		return simplifyRationalNumber(new AnyNode(-37, t, m));
	} else {
		int numerator = (int) u.operand(1).getValue();
		int denominator = (int) u.operand(2).getValue();
		if (numerator == 0) {
			// tu so bang 0
			result.setKey(0);
			result.setValue(0);
		} else if (denominator != 0) {
			int g = gcd(numerator, denominator);
			if (g != 1) {
				if (denominator > 0) {
					numerator /= g;
					denominator /= g;
				} else {
					numerator /= -g;
					denominator /= -g;
				}
				if (denominator == 1)
					result = new AnyNode(0, numerator);
				else
					result   =   new   AnyNode(-37,   numerator,
enominator);
			} else {
				if (denominator == 1)
					result = new AnyNode(0, numerator);
				else
					result = u;
			}
		} else if (denominator == 0) {
			result.setKey(-38); // undefined
		}
		return result;
	}
}
```

Phụ lục 19: Phương thức *gcd* tính ước chung lớn nhất của hai số nguyên.

```
private static int gcd(int a, int b) {
```

```
        if (b != 0) {
                return gcd(b, a % b);
        } else
                return Math.abs(a);
}
```

Phụ lục 20: Phương thức *degreeMonomialSV*

```
private AnyNode degreeMonomialSV(AnyNode u, AnyNode x) {
        AnyNode result = new AnyNode();
        if (u.getKey() == 0 && u.getValue() == 0) {
                // neu u=0 thi bac cua don thuc la khong xac dinh
                result = new AnyNode(-38);
        } else if (u.getKey() == 0 || u.getKey() == -37) {
                // neu u la so nguyen hoac phan so
                result = new AnyNode(0, 0);
        } else if (u.equal(x)) {
                // u=x
                result = new AnyNode(0, 1);
        } else if (u.getKey() == -5) {
                // u = x^n
                AnyNode base = u.base();
                AnyNode exponent = u.exponent();
                if (base.equal(x) && exponent.getKey() == 0
                                && exponent.getValue() > 1) {
                        result = exponent;
                } else {
                        result = new AnyNode(-38);
                }
        } else if (u.getKey() == -3) {
                // u la tich cua hai toan hang
                if (u.nop() == 2) {
                        AnyNode a = degreeMonomialSV(u.operand(1), x);
                        AnyNode b = degreeMonomialSV(u.operand(2), x);
                        if (a.getKey() != -38 && b.getKey() != -38)
                                result = b;
                } else {
                        result = new AnyNode(-38);
                }
        } else {
                result = new AnyNode(-38);
        }
        return result;
}
```

Phụ lục 21: Phương thức *coefficientMonomialSV*

```
private ArrayList<AnyNode> coefficientMonomialSV(AnyNode u, AnyNode x) {
        ArrayList<AnyNode> result = new ArrayList<AnyNode>();
        AnyNode m = degreeMonomialSV(u, x);
        if (monomialSV(u, x) && m.getKey() != -38) {
                AnyNode c = new AnyNode();
                if (u.getKey() == 0 || u.getKey() == -37) {
```

```
                    // neu u la so nguyen hoac phan so
                    c = u;
              } else if (u.equal(this.var)) {
                    // u=x
                    c = new AnyNode(0, 1);
              } else if (u.getKey() == -5) {
                    // u = x^n
                    AnyNode base = u.base();
                    AnyNode exponent = u.exponent();
                    if (base.equal(x) && exponent.getKey() == 0
                                && exponent.getValue() > 1) {
                          c = new AnyNode(0, 1);
                    } else {
                          c = new AnyNode(-38);
                    }
              } else if (u.getKey() == -3) {
                    // u la tich cua cua 2
                    if (u.operand(1).getKey() == 0 || u.operand(1).getKey()
== -37) {
                          // toan hang dau tien cua u la so nguyen hoac phan
so
                          c = u.operand(1);
                    } else {
                          c = new AnyNode(0, 1);
                    }
              }
              result.add(c);
              result.add(m);
        } else {
              result.add(new AnyNode(-38));
              result.add(new AnyNode(-38));
        }
        return result;
}
```

Phụ lục 22: Phương thức *coefficientSV*

```
public AnyNode coefficientSV(int j){
        if(this.isPoly){
              for(int i = 1; i <= this.poly.nop(); i++){
                    MonomialSV temp = new MonomialSV(this.poly.operand(i),
this.var);
                    if(temp.getDeg().getValue() == j){
                          return temp.getCoeffi();
                    }
              }
              return new AnyNode(0, 0);
        }
        return new AnyNode(-38);
}
```

Phụ lục 23: Phương thức *degreeSV*

```
private AnyNode degreeSV(AnyNode u, AnyNode x){
	MonomialSV d = new MonomialSV(u, x);
	if(d.getDeg().getKey() != -38){
		return d.getDeg();
	}else if(AnyNode.kind(u).equals("+")){
		int temp = 0;
		for(int i = 1; i <= u.nop(); i++){
			AnyNode f = new MonomialSV(u.operand(i), x).getDeg();
			if(f.getKey() == -38){
				return new AnyNode(-38);
			}else{
				temp = f.getValue() < temp ? temp : f.getValue();
			}
		}
		return new AnyNode(0, temp);
	}else{
		return new AnyNode(-38);
	}
}
```

Phụ lục 24: Phương thức *coefficientGme*

```
public ArrayList<AnyNode> coefficientGme(AnyNode x) {
	AnyNode u = this.getMono();
	ArrayList<AnyNode> s = new ArrayList<AnyNode>();
	s.add(x);
	ArrayList<AnyNode> result = new ArrayList<AnyNode>();
	if (monomialGpe(u, s)) {
		if (u.equal(x)) {
			result.add(new AnyNode(0, 1));
			result.add(new AnyNode(0, 1));
			return result;
		} else if (u.getKey() == -5) {
			AnyNode base = u.base();
			AnyNode exponent = u.exponent();
			if (base.equal(x) && exponent.getKey() == 0
					&& exponent.getValue() > 1) {
				result.add(new AnyNode(0, 1));
				result.add(exponent);
				return result;
			}
		} else if (u.getKey() == -3) {
			AnyNode c = u;
			AnyNode m = new AnyNode(0, 0);
			ArrayList<AnyNode> f = new ArrayList<AnyNode>();
			for (int i = 1; i <= u.nop(); i++) {
				GeneralMonomial temp = new
generalMonomial(u.operand(i), s);
				f = temp.coefficientGme(x);
				if (f.get(0).getKey() == -38) {
					result.add(new AnyNode(-38));
					return result;
				} else if (f.get(1).getKey() == 0
						&& f.get(1).getValue() != 0) {
```

```
                                m = f.get(1);
                                AnyNode t = new AnyNode(-5, new AnyNode(-5,
x, m),
                                                new AnyNode(0, -1));
                                c = Simplify.simplify(new AnyNode(-3, c,
t));
                          }
                    }
                    result.add(c);
                    result.add(m);
                    return result;
              }
              if (Operation.free(u, x)) {
                    result.add(u);
                    result.add(new AnyNode(0, 0));
                    return result;
              } else {
                    result.add(new AnyNode(-38));
                    return result;
              }
        } else {
              result.add(new AnyNode(-38));
              return result;
        }
}
```

Phụ lục 25: Phương thức *degreeGme*

```
private AnyNode degreeGme(AnyNode u, ArrayList<AnyNode> s) {
        if (isMember(u, s))
              // u thuoc S
              return new AnyNode(0, 1);
        else if (u.getKey() == -5)
              // u = x^n
              return u.exponent();
        else if (u.getKey() == -3) {
              // u = u1*u2*...*un
              int degree = 0;
              for (int i = 1; i <= u.nop(); i++) {
                    if (u.operand(i).getKey() != -5) {
                          if (isMember(u.operand(i), s))
                                degree                                    +=
u.operand(i).exponent().getValue();
                    } else {
                          if (isMember(u.operand(i).operand(1), s))
                              degree += u.operand(i).exponent().getValue();
                    }
              }
              return new AnyNode(0, degree);
        } else {
              // u khong thuoc
              if (u.getKey() == 0 && u.getValue() == 0)
                    // u = 0
                    return new AnyNode(-38);
```

```
            else
                  return new AnyNode(0, 0);
      }
}
```

Phụ lục 26: Phương thức *multi* đơn thức

```
public GeneralMonomial multi(GeneralMonomial u) {
      GeneralMonomial result = new GeneralMonomial();
      if (!this.isMono || !u.isMono) {
            // this hoac u khong la don thuc
            setUndefined(result);
      } else {
            result.s = union(this.s, u.s);
            AnyNode coeffi = Simplify.simplify(new AnyNode(-3,
this.coeffi,
                        u.coeffi));
            AnyNode var = Simplify.simplify(new AnyNode(-3, this.var,
u.var));
            AnyNode mono = Simplify.simplify(new AnyNode(-3, coeffi,
var));
            result.coeffi = coeffi;
            result.var = var;
            result.mono = mono;
            result.isMono = true;
            result.deg = degreeGme(result.mono, result.s);
      }
      return result;
}
```

Phụ lục 27: Phương thức *div* đơn thức

```
public GeneralMonomial div(GeneralMonomial u) {
      AnyNode coeffi = Simplify.simplify(new AnyNode(-5, u.coeffi,
                  new AnyNode(0, -1)));
      AnyNode var = Simplify.simplify(new AnyNode(-5, u.var, new AnyNode(0,
                  -1)));
      AnyNode mono = Simplify.simplify(new AnyNode(-3, coeffi, var));
      u.coeffi = coeffi;
      u.mono = mono;
      u.var = var;
      return this.multi(u);
}
```

Phụ lục 28: Phương thức *sub* đơn thức

```
public GeneralMonomial sub(GeneralMonomial u) {
      AnyNode coeffi = Simplify.simplify(new AnyNode(-3, u.coeffi,
                  new AnyNode(0, -1)));
      AnyNode mono = Simplify.simplify(new AnyNode(-3, coeffi, u.var));
      u.coeffi = coeffi;
      u.mono = mono;
      return this.add(u);
```

```
}
```

Phụ lục 29: Phương thức *add* đơn thức

```
public GeneralMonomial add(GeneralMonomial u) {
	GeneralMonomial result = new GeneralMonomial();
	if (!this.getVar().equal(u.getVar()) || !this.isMono || !u.isMono) {
		setUndefined(result);
	} else {
		// neu 2 don thuc dong dang
		AnyNode coeffi = Simplify.simplify(new AnyNode(-1,
this.coeffi, u.coeffi));
		AnyNode mono = Simplify.simplify(new AnyNode(-3, coeffi,
this.var));
		result.coeffi = coeffi;
		result.mono = mono;
		result.isMono = true;
		result.var = this.var;
		result.deg = this.deg;
		result.s = intersection(this.s, u.s);
	}
	return result;
}
```

Phụ lục 30: Các phương thức *polynomialExpansion*, *add*, *sub*, *multi*, *div*, *degreeGpe* ,*leadingCoefficientGPE*, *coefficientGPE*, *degreeGpe*(*AnyNode*),

- Phương thức *coefficientGPE*

```
public AnyNode coefficientGPE(AnyNode x, int i) {
	AnyNode u = this.getPoly();
	if (u.getKey() == -1) {
		if (u.equal(x)) {
			if (i == 1)
				return new AnyNode(0, 1);
			else
				return new AnyNode(0, 0);
		}
		AnyNode result = new AnyNode(-1);
		result.leaf.add(new AnyNode(0, 0));
		for (int j = 1; j <= u.nop(); j++) {
			GeneralMonomial mono = new GeneralMonomial(u.operand(j),
this.s);
			ArrayList<AnyNode> list = mono.coefficientGme(x);
			if (list.get(0).getKey() == -38)
				return new AnyNode(-38);
			else if (list.get(1).getValue() == i)
				result.leaf.add(list.get(0));
		}
		return Simplify.simplify(result);
	} else {
		GeneralMonomial mono = new GeneralMonomial(u, this.s);
		ArrayList<AnyNode> list = mono.coefficientGme(x);
		if (list.get(0).getKey() == -38)
```

```
                return new AnyNode(-38);
            else if (list.get(1).getValue() == i)
                return list.get(0);
            else
                return new AnyNode(0, 0);
        }
}
```

- Phương thức *degreeGpe*

```
public AnyNode degreeGpe() {
        if (this.isPoly) {
            if (this.poly.getKey() == 0 && this.poly.getValue() == 0)
                // da thuc 0
                return new AnyNode(-38);
            else {
                int deg = 0;
                for (int i = 0; i < this.listMono.size(); i++) {
                    int degree = listMono.get(i).getDeg().getValue();
                    deg = (deg < degree) ? degree : deg;
                }
                return new AnyNode(0, deg);
            }
        } else
            return new AnyNode(-38);
}
```

- Phương thức *leadingCoefficientGpe*

```
public AnyNode leadingCoefficientGPE(AnyNode x) {
        AnyNode degree = this. degreeGpe (x);
        if (degree.getKey() == 0) {
            AnyNode coefficient = coefficientGPE(x, degree.getValue());
            return coefficient;
        }
        return new AnyNode(-38);
}
public AnyNode degreeGpe(AnyNode x) {
        ArrayList<AnyNode> v = new ArrayList<AnyNode>();
        v.add(x);
        AnyNode u = this.getPoly();
        GeneralPolynomial temp = new GeneralPolynomial(u, v);
        if (temp.isPoly)
            return temp.degreeGpe();
        else
            return new AnyNode(-38);
}
```

- Toán tử cộng hai đa thức *add*

```
public GeneralPolynomial add(GeneralPolynomial u) {
        AnyNode poly = new AnyNode(-1, expand(this.poly), expand(u.poly));
        ArrayList<AnyNode> s = union(this.s, u.s);
```

```
        GeneralPolynomial result = new GeneralPolynomial(poly, s);
        return result.collectTerms();
}
```

- Toán tử trừ hai đa thức *sub*

```
public GeneralPolynomial sub(GeneralPolynomial u) {
        AnyNode polyMinuend = expand(new AnyNode(-3, new AnyNode(0, -1),
u.poly));
        GeneralPolynomial minuend = new GeneralPolynomial(polyMinuend, u.s);
        return this.add(minuend);
sss}
```

- Toán tử nhân hai đa thức *multi*

```
public GeneralPolynomial multi(GeneralPolynomial u) {
        ArrayList<AnyNode> s = union(this.s, u.s);
        AnyNode poly = Simplify.simplify(expand(new AnyNode(-3, this.poly,
                        u.poly)));
        GeneralPolynomial result = new GeneralPolynomial(poly, s);
        return result;
}
```

- Toán tử chia hai đa thức *div*

```
public ArrayList<GeneralPolynomial> div(GeneralPolynomial v, AnyNode x) {
        if (v.getPoly().getKey() == 0 && v.getPoly().getValue() == 0)
                return null;
        else {
                ArrayList<GeneralPolynomial>        result        =        new
ArrayList<GeneralPolynomial>();
                AnyNode q = new AnyNode(0, 0);
                GeneralPolynomial r = new GeneralPolynomial();
                r = this;
                AnyNode m = r.degreeGpe(x);
                AnyNode n = v.degreeGpe(x);
                AnyNode lcv = v.leadingCoefficientGPE(x);
                while (m.getValue() >= n.getValue()) {
                        AnyNode lcr = r.leadingCoefficientGPE(x);
                        AnyNode  s  =  Simplify.simplify(new  AnyNode(-37,  lcr,
lcv));
                        AnyNode temp = new AnyNode(-3, s, new AnyNode(-5, x,
                                        new AnyNode(-1, m, new AnyNode(-3, n,
                                                        new AnyNode(0, -1)))));
                        q = Simplify.simplify(new AnyNode(-1, q, temp));
                        AnyNode   temp1   =   new   AnyNode(-3,   new   AnyNode(-1,
v.getPoly(),
                                        new  AnyNode(-3,  new  AnyNode(0,  -1),  new
AnyNode(-3,
                                                        lcv, new AnyNode(-5, x, n)))),
temp);
                        AnyNode   temp2   =   new   AnyNode(-1,   r.getPoly(),   new
AnyNode(-3,
```

```
                                    new AnyNode(0, -1), new AnyNode(-3, lcr, new
AnyNode(-5, x, m))));
                        AnyNode t = new AnyNode(-1, temp2, new AnyNode(-3, new
AnyNode(0, -1), temp1));
                        AnyNode polyR = expand(t);
                        r = new GeneralPolynomial(polyR, this.s);
                        m = r.degreeGpe(x);
                  }
                  result.add(new GeneralPolynomial(q, this.s));
                  result.add(r);
                  return result;
            }
}
```

- Phương thức *degreeGpe*(*AnyNode*)

```
public AnyNode degreeGpe(AnyNode x) {
      ArrayList<AnyNode> v = new ArrayList<AnyNode>();
      v.add(x);
      AnyNode u = this.getPoly();
      GeneralPolynomial temp = new GeneralPolynomial(u, v);
      if (temp.isPoly)
            return temp.degreeGpe();
      else
            return new AnyNode(-38);
}
```

- Phương thức *polynomialExpansion*

```
public GeneralPolynomial polynomialExpansion(GeneralPolynomial v,
            AnyNode x, AnyNode t) {
      if (this.getPoly().getKey() == 0 && this.getPoly().getValue() == 0)
{
            return this;
      } else {
            ArrayList<GeneralPolynomial> d = this.div(v, x);
            GeneralPolynomial q = d.get(0);
            GeneralPolynomial r = d.get(1);
            GeneralPolynomial temp = q.polynomialExpansion(v, x, t)
                  .multi(new GeneralPolynomial(t, s)).add(r);
            AnyNode t1 = expand(temp.getPoly());
            GeneralPolynomial   result   =   new   GeneralPolynomial(t1,
temp.getS());
            return result;
      }
}
```

- Phương thức *collectTerms*

```
public GeneralPolynomial collectTerms() {
      // mang chua cac don thuc co phan bien khac nhau
      ArrayList<GeneralMonomial>          listMono          =          new
ArrayList<GeneralMonomial>();
      AnyNode u = this.getPoly();
      if (u.getKey() != -1) {
            // Da thuc chi co mot phan tu (Ex: 2x)
            return this;
```

```
	} else {
		// Da thuc la tong cac don thuc (Ex: 2xy+5x+7)
		if (isMember(u, this.getS()))
			// neu u thuoc S
			return this;
		int n = 0;
		for (int i = 1; i <= u.nop(); i++) {
			GeneralMonomial f = new GeneralMonomial(u.operand(i),
this.s);
			int j = 1;
			boolean combined = false;
			while ((!combined) && j <= n) {
				if (f.getVarPart().equal(listMono.get(j -
1).getVarPart())) {
					AnyNode sumCoeffi = new AnyNode(-1,
f.getCoeffi(),listMono.get(j - 1).getCoeffi());
					listMono.get(j - 1).coeffi = Simplify
							.simplify(sumCoeffi);
					combined = true;
					AnyNode mono = new AnyNode(-3,
listMono.get(j - 1).getCoeffi(), listMono.get(j - 1).getVar());
					listMono.get(j - 1).mono = mono;
				}
				j++;
			}
			if (!combined) {
				listMono.add(f);
				n++;
			}
		}
		this.setListMono(listMono);
		if (listMono.size() == 1)
			this.poly = listMono.get(0).mono;
		else {
			this.poly.leaf.clear();
			for (int i = 0; i < listMono.size(); i++) {
				this.poly.leaf.add(listMono.get(i).mono);
			}
		}
	}
	GeneralPolynomial result = new GeneralPolynomial(this.getPoly(),
			this.getS());
	return result;
}
```

Phụ lục 31: Thủ tục *ExpandProduct*

Procedure *ExpandProduct* (r, s);
Input
	r, s : biểu thức đại số đã ở dạng khai triển với số mũ của các lũy thừa là số nguyên
Output
	Dạng khai triển của r ∗ s;
Local Variables
	f;
Begin

```
        if Kind(r) = "+" then
                f := Operand(r, 1);
                Return(ExpandProduct(f, s) + ExpandProduct(r − f, s));
        Else if Kind(s) ="+" then
                Return(ExpandProduct(s, r));
        else
                Return(r ∗ s);
End
```

Phụ lục 32: Thủ tục $ExpandPower$

```
Procedure ExpandPower(u, n);
Input
        u : là một biểu thức đại số đã ở dạng khai triển với các lũy thừa có số mũ nguyên
        n : là một số nguyên không âm
Output
        Dạng khai triển của u^n
Local Variables
        f, r, k, s, c;
Begin
        if Kind(u) ="+" then
                f := Operand(u, 1);
                r := u − f;
                s := 0;
                for k := 0 to n do
                        c := n!/(k! (n − k)!);
                        s := s + ExpandProduct(c ∗ f^(n−k), ExpandPower(r, k));
                Return(s)
        else
                Return(u^n)
End
```

Phụ lục 33: Thủ tục $RationalizeSum$

- Thủ tục $RationalizeSum$

```
Procedure RationalizeSum(u, v);
Input
        u, v : các biểu thức đại số ở dạng hữu tỉ hóa;
Output
        Một biểu thức đại số ở dạng hữu tỉ hóa;
```

Local Variables m, n, r, s;
Begin
    m := $Numerator(u)$;
    r := $Denominator(u)$;
    n := $Numerator(v)$;
    s := $Denominator(v)$;
    if r = 1 and s = 1 then
        Return(u + v)
    else
        Return($RationalizeSum(m * s, n * r)/(r * s)$)
End

Phụ lục 34: Các phương thức *numerator*, *denominator*, *rationalGRE*, *rationalExpand*, *rationalzeExpression*, *rationalzeSum*.

- Phương thức *numerator*

```
public static AnyNode numerator(AnyNode u) {
      if (u.getKey() == -37)
            // u la so huu ti
            return u.operand(1);
      else if (u.getKey() == -5) {
            // u la luy thua
            if ((u.exponent().getKey() == 0 && u.exponent().getValue() <
0) || (u.exponent().getKey() == -37 && (u.exponent().numerator() < 0 ||
u.exponent().denominator() < 0))) {
                  // so mu cua u la so nguyen hoac phan so co gia tri < 0
                  return new AnyNode(0, 1);
            } else
                  // so mu cua u > 0
                  return u;
      } else if (u.getKey() == -3) {
            AnyNode v = u.operand(1);
            AnyNode t = u.divNode(1);
            return    Simplify.simplify(new    AnyNode(-3,    numerator(v),
numerator(t)));
      } else
            return u;
}
```

- Phương thức *denominator*

```
public AnyNode denominator(AnyNode u) {
      if (u.getKey() == -37)
            // u la so huu ti
            return u.operand(2);
      else if (u.getKey() == -5) {
            // u la luy thua
```

```
            if ((u.exponent().getKey() == 0 && u.exponent().getValue() < 0)||(u.exponent().getKey() == -37 && (u.exponent().numerator() < 0 || u.exponent().denominator() < 0)))
                  // so mu cua u la so nguyen hoac phan so co gia tri < 0
                  return Simplify.simplify(new AnyNode(-5, u, new AnyNode(0, -1)));
            else
                  // so mu cua u > 0
                  return new AnyNode(0, 1);
      } else if (u.getKey() == -3) {
            AnyNode v = u.operand(1);
            AnyNode t = u.divNode(1);
            return Simplify.simplify(new AnyNode(-3, denominator(v),
                              denominator(t)));
      } else if (u.getKey() == -38)
            return u;
      else
            return new AnyNode(0, 1);
}
```

- Phương thức *rationalGRE*

```
public boolean rationalGre(AnyNode u, ArrayList<AnyNode> s) {
      GeneralPolynomial gpNumerator = new GeneralPolynomial(numerator(u), s);
      GeneralPolynomial gpDenominator = new GeneralPolynomial(denominator(u),s);
      if (gpNumerator.isPoly && gpDenominator.isPoly)
            return true;
      else
            return false;
}
```

- Phương thức *rationalExpand*

```
public AnyNode rationalExpand(AnyNode u) {
      AnyNode numerator = numerator(u);
      AnyNode denominator = denominator(u);
      if (!isExpanded(numerator)) {
            numerator = expand(numerator);
      }
      if (!isExpanded(denominator)) {
            denominator = expand(denominator);
      }
      if (denominator.getKey() == 0 && denominator.getValue() == 0)
            return new AnyNode(-38);
      else {
            AnyNode result = new AnyNode(-3, numerator, new AnyNode(-5,
                              denominator, new AnyNode(0, -1)));
            return Simplify.simplify(result);
      }
}
```

- Phương thức *rationalzeExpression*

```
public AnyNode rationalizeExpression(AnyNode u) {
	if (u.getKey() == -5) {
		AnyNode f = u.operand(1);
		AnyNode result = new AnyNode(-5, rationalizeExpression(f),
					u.operand(2));
		return result;
	} else if (u.getKey() == -3) {
		AnyNode f = u.operand(1);
		AnyNode g = Simplify.simplify(new AnyNode(-3, u, new AnyNode(-
5, f,new AnyNode(0, -1))));
		AnyNode result = Simplify.simplify(new AnyNode(-3,
			rationalizeExpression(f), rationalizeExpression(g)));
		return result;
	} else if (u.getKey() == -1) {
		AnyNode f = u.operand(1);
		AnyNode g = rationalizeExpression(f);
		AnyNode   r   =   rationalizeExpression(Simplify.simplify(new
AnyNode(-1,u, new AnyNode(-3, new AnyNode(0, -1), f))));
		return rationalizeSum(g, r);
	} else
		return u;
}
```

- Phương thức *rationalzeSum*

```
private AnyNode rationalizeSum(AnyNode u, AnyNode v) {
	AnyNode m = numerator(u);
	AnyNode r = denominator(u);
	AnyNode n = numerator(v);
	AnyNode s = denominator(v);
	if (r.getValue() == 1 && s.getValue() == 1)
		// return m+n
		return (Simplify.simplify(new AnyNode(-1, m, n)));
	else {
		AnyNode t1 = rationalizeSum(
					Simplify.simplify(new AnyNode(-3, m, s)),
					Simplify.simplify(new AnyNode(-3, n, r)));
		AnyNode t2 = new AnyNode(-5,
					Simplify.simplify(new AnyNode(-3, r, s)),
					Simplify.simplify(new AnyNode(0, -1)));
		AnyNode result = new AnyNode(-3, t1, t2);
		return Simplify.simplify(result);
	}
}
```

Phụ lục 35: Các phương thức trong SMC

- Phương thức *taylorSeries*

```
public static AnyNode taylorSeries(AnyNode expression, AnyNode var,
		int n, int a) {
	AnyNode result = new AnyNode(-1);
	for (int k = 0; k <= n; k++) {
```

```
            AnyNode temp = new AnyNode();
            AnyNode derivative = higherDerivative(expression, var, k);
            derivative = derivative.substitute(new AnyNode(0, a), var);
            AnyNode factorial = new AnyNode(0, Simplify.factorial(k));
            AnyNode v = new AnyNode(-1, var, new AnyNode(0, -1 * a));
            AnyNode x = new AnyNode(-5, v, new AnyNode(0, k));
            temp.setKey(-3);
            ArrayList<AnyNode> t = new ArrayList<AnyNode>();
            t.add(derivative);
            t.add(x);
            t.add(new AnyNode(-5, factorial, new AnyNode(0, -1)));
            temp.leaf = t;
            result.leaf.add(temp);
      }
      return Simplify.simplify(result);
}
```

- Phương thức *derivative*

```
public static AnyNode derivative(AnyNode a, AnyNode b) {
            if (b.getKey() != CEILING) {
                  return new AnyNode(-38);
            } else {
                  if (a.equal(b)) {
                        return new AnyNode(0, 1);
                  } else if (a.getKey() == -5) {
                        return derivativePower(a, b);
                  } else if (a.getKey() == -1) {
                        return derivativeSum(a, b);
                  } else if (a.getKey() == -3) {
                        return derivativeProduct(a, b);
                  } else if (free(a, b)) {
                        return new AnyNode(0, 0);
                  } else if (a.getKey() <= -17 && a.getKey() >= -22) {
                        return derivativeTrigonometric(a, b);
                  } else if (a.getKey() == -28) {
                        // dao ham exp(u)
                        AnyNode u = a.operand(1);
                        AnyNode r = new AnyNode(-3, derivative(u, b), a);
                        return Simplify.simplify(r);
                  } else if (a.getKey() == -29) {
                        // dao ham ln(u)
                        AnyNode u = a.operand(1);
                        AnyNode r = new AnyNode(-3, derivative(u, b), new
AnyNode(-5,
                                    u, new AnyNode(0, -1)));
                        return Simplify.simplify(r);
                  }
                  return new AnyNode(-38);
            }
      }
```

- Phương thức *higherDerivative*

```
public static AnyNode higherDerivative(AnyNode expression,
            AnyNode var, int n) {
      try {
            while (n != 0) {
                  expression = derivative(expression, var);
                  n--;
            }
```

```
            return expression;
      } catch (Exception e) {
            System.out.println(e.toString());
            return new AnyNode(-38);
      }
}
```

- Phương thức *minF*

```
public static AnyNode minF(GeneralPolynomial a, GeneralPolynomial b,
AnyNode var){
      AnyNode da = a.degreeGpe(var);
      AnyNode db = b.degreeGpe(var);
      int max = 0;
      if(da.compare(db)){
            max = db.getValue();
      }else{
            max = da.getValue();
      }
      ArrayList<AnyNode> operators = new ArrayList<AnyNode>();
      for(int i = 0; i <= max; i++){
            AnyNode temp = new AnyNode(-3);
            AnyNode coff1 = a.coefficientGPE(var, i);
            AnyNode coff2 = b.coefficientGPE(var, i);
            if(coff1.compare(coff2)){     // coff1 < coff2
                  temp.leaf.add(coff1);
            }else{
                  temp.leaf.add(coff2);
            }
            temp.leaf.add(new AnyNode(-5, var, new AnyNode(0, i)));
            operators.add(temp);
      }
      AnyNode result = new AnyNode(-1, operators);
      return Simplify.simplify(result);
}
```

- Phương thức *maxF*

```
public static AnyNode maxF(GeneralPolynomial a, GeneralPolynomial b,
AnyNode var){
      AnyNode da = a.degreeGpe(var);
      AnyNode db = b.degreeGpe(var);
      int max = 0;
      if(da.compare(db)){
            max = db.getValue();
      }else{
            max = da.getValue();
      }
      ArrayList<AnyNode> operators = new ArrayList<AnyNode>();
      for(int i = 0; i <= max; i++){
            AnyNode temp = new AnyNode(-3);
            AnyNode coff1 = a.coefficientGPE(var, i);
            AnyNode coff2 = b.coefficientGPE(var, i);
            if(coff1.compare(coff2)){     // coff1 < coff2
                  temp.leaf.add(coff2);
            }else{
                  temp.leaf.add(coff1);
            }
            temp.leaf.add(new AnyNode(-5, var, new AnyNode(0, i)));
            operators.add(temp);
      }
```

```
        AnyNode result = new AnyNode(-1, operators);
        return Simplify.simplify(result);
}
```

- Phương thức *dedup*

```
public static AnyNode dedup(AnyNode expression, AnyNode var, int n, int a){
        AnyNode result = new AnyNode(-1);
        for (int k = 0; k <= n; k++) {
                AnyNode temp = new AnyNode();
                AnyNode derivative = higherDerivative(expression, var, k);
                derivative = derivative.substitute(new AnyNode(0, a), var);
                AnyNode factorial = new AnyNode(0, Simplify.factorial(k));
                AnyNode v = new AnyNode(-1, var, new AnyNode(0, -1 * a));
                AnyNode x = new AnyNode(-5, v, new AnyNode(0, k));
                temp.setKey(-3);
                ArrayList<AnyNode> t = new ArrayList<AnyNode>();
                t.add(derivative);
                t.add(new AnyNode(-5, factorial, new AnyNode(0, -1)));
                AnyNode ck = Simplify.simplify(new AnyNode(-3, t));
                if(ck.isNegative() == 1){
                        t.clear();
                }
                t.add(x);
                temp.leaf = t;
                result.leaf.add(temp);
        }
        return Simplify.simplify(result);
}
```