

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



MÔ HÌNH HÓA TOÁN HỌC (CO2011)

Đề bài tập lớn

## Đặc tả Smart Contract bằng Linear Logic

---

**Giáo viên hướng dẫn:** Nguyễn An Khương  
Huỳnh Tường Nguyên  
Trần Văn Hoài  
Lê Hồng Trang  
Trần Tuấn Anh

**Trợ giảng:** Nguyễn Trung Việt

**Sinh viên thực hiện:** 1613575 - Trần Ngọc Tín  
1610852 - Huỳnh Sâm Hà  
1613535 - Nguyễn Văn Tiến  
161 - Thái Hoàng Nguyên  
161 - Huỳnh Song Anh Quân

---

Thành phố Hồ Chí Minh, Ngày 10 tháng 6 năm 2018

# Mục lục

Danh sách hình vẽ	1
Danh sách bảng	2
<b>1 Giới thiệu đề tài</b>	<b>3</b>
<b>2 Bài toán 1: Kiến thức ôn tập</b>	<b>3</b>
2.1 Lịch sử và ứng dụng của Linear Logic . . . . .	3
2.1.1 Lịch sử phát triển của Linear Logic . . . . .	3
2.1.2 Những ứng dụng của Linear Logic . . . . .	3
2.2 Lịch sử và ứng dụng của Smart Contract . . . . .	4
2.2.1 Lịch sử phát triển và cái nhìn tổng quan về Smart Contract . . . . .	4
2.2.2 Những ứng dụng dựa trên Smart Contract . . . . .	4
2.3 Làm rõ mấy cái phép toán trong ass . . . . .	6
2.4 Các ký hiệu $\top$ và $\perp$ . . . . .	6
2.5 Exponentials connectives . . . . .	6
2.5.1 Connective ! . . . . .	6
2.5.2 Connective ? . . . . .	7
2.5.3 Liên hệ của ! và ? trong linear logic . . . . .	8
2.5.4 Một số biểu thức liên quan . . . . .	8
<b>3 Bài toán 2: Ngữ cảnh smart contract</b>	<b>9</b>
3.1 Mô tả ngữ cảnh cho smart contract . . . . .	9
3.1.1 Động cơ và mục tiêu . . . . .	9
3.1.2 Tình huống cụ thể . . . . .	9
3.2 Mô tả các điều khoản cho ngữ cảnh smart contract . . . . .	10
<b>4 Bài toán 3: Đặc tả ngữ cảnh smart contract bằng linear logic</b>	<b>12</b>
<b>5 Bài toán 4: Dùng mã giả xây dựng smart contract</b>	<b>13</b>
<b>6 Bài toán 5: Dùng Solidity xây dựng smart contract</b>	<b>13</b>
6.1 Hiện thực code bằng Solidity . . . . .	13
6.2 Ảnh minh họa cho việc chạy trên môi trường remix . . . . .	22
<b>7 Nhận xét và kết luận</b>	<b>28</b>
<b>Tài liệu tham khảo</b>	<b>29</b>
<b>Phụ lục</b>	<b>30</b>

## Danh sách hình vẽ

1	Compile smart contract . . . . .	22
2	Compile smart contract . . . . .	22
3	Môi trường chạy và các tài khoản tự sinh . . . . .	23
4	Deploy smart contract DateTime . . . . .	23
5	Deploy smart contract Transport . . . . .	23
6	Thực thi các hàm bên A . . . . .	24
7	A gửi ether vào smart contract . . . . .	24
8	Thực thi các hàm bên B . . . . .	24
9	B gửi ether vào smart contract . . . . .	25
10	Các thông tin đầu vào smart contract (1) . . . . .	26
11	Các thông tin đầu vào smart contract (2) . . . . .	27
12	Các thông tin đầu vào smart contract (3) . . . . .	27
13	Thực thi khi B đến nhận hàng . . . . .	27
14	Kiểm tra tài khoản của B . . . . .	28



## Danh sách bảng

# 1 Giới thiệu đề tài

## 2 Bài toán 1: Kiến thức ôn tập

### 2.1 Lịch sử và ứng dụng của Linear Logic

#### 2.1.1 Lịch sử phát triển của Linear Logic

Linear Logic (Logic tuyến tính) là một dạng Substructural Logic (Logic thiếu mất 1 trong những quy luật cấu trúc) được đề xuất bởi Jean-Yves Girard vào năm 1987 như là một sự cải tiến của Classical Logic (Logic cổ điển) và Institutional Logic (Logic mang tính trực giác), kết hợp giữa luật đối tính của Classical Logic với các quy luật hình thành của Institutional Logic.

Mục tiêu của Linear Logic là cầu kết nối giữa logic và khoa học máy tính vì nó cho phép thể hiện và điều khiển những sự kiện của thế giới thực một cách tự nhiên. Một ví dụ điển hình là Law of Excluded Middle (LEM). LEM nói rằng "có A hoặc không có A", vốn là một điều hoàn toàn hợp lý trong cuộc sống của chúng ta. Đối với Classical Logic, LEM là một luật được chấp nhận, tuy nhiên điều đó lại ngược lại đối với Institutional Logic.

Linear Logic đã định nghĩa LEM theo 2 cách là  $A \oplus \neg A$  và  $A \wp A$ .

Cách thứ nhất tương đương với công thức của phép tuyển trong Institutional Logic và cách thứ hai tương đương với "A suy ra A" là điều luôn đúng. Cả 2 cách này đều được chấp nhận trong Institutional Logic. Từ đó, ta có thể thấy Linear Logic là một sự kết hợp hoàn hảo giữa Classical và Institutional.

#### 2.1.2 Những ứng dụng của Linear Logic

Linear Logic có rất nhiều công dụng trên nhiều lĩnh vực khác nhau. Ví dụ:

- Điện toán lượng tử Khác với máy tính thông thường sử dụng bit để lưu trữ các trạng thái, máy tính lượng tử sử dụng qubit. Ở máy tính thông thường, chi phí tính toán một phép toán một lần hay nhiều lần là như nhau. Vì vậy mà ta có thể sử dụng logic cổ điển để thể hiện các phép toán trên máy tính thông thường. Tuy nhiên, đối với máy tính lượng tử, việc thực hiện một phép toán nhiều lần sẽ có chi phí tính toán cao hơn là thực hiện chỉ 1 lần. Sử dụng Linear Logic sẽ thích hợp hơn trong trường hợp này vì việc quản lý tài nguyên là một vấn đề cần được xem xét.

- Ngôn ngữ học Linear Logic có thể được dùng để kiểm tra và chứng minh ngữ pháp và ngữ nghĩa của một ngôn ngữ. Như việc kiểm tra xem ngữ pháp của một ngôn ngữ có cấu trúc nhất định hay ngữ nghĩa của một câu nói có thể hiện được một điều đúng đắn trong logic.

- Lập trình Vì Linear Logic cho phép quản lý tài nguyên một cách hiệu quả hơn, chúng ta có thể áp dụng nó vào việc lập trình để quản lý bộ nhớ hay việc thu gom rác một cách hiệu quả hơn. Ngoài ra, Linear Logic giúp thể hiện những hiện tượng trong thế giới thực một cách tự nhiên hơn, giúp cho việc lập trình trở nên đa dạng và dễ dàng hơn.

## 2.2 Lịch sử và ứng dụng của Smart Contract

### 2.2.1 Lịch sử phát triển và cái nhìn tổng quan về Smart Contract

Theo dòng lịch sử chúng ta sẽ thấy công nghệ đã thay đổi rất nhiều thứ, kể cả nhận thức của mỗi chúng ta. Các công nghệ mới được sinh ra và dần dần thay thế các công nghệ đã trở nên lạc hậu. Bắt đầu từ nguyên của internet, niềm tin luôn là một thứ gì đó khá xa xỉ khi các tổ chức, cá nhân bắt tay, giao tiếp với nhau trên mạng internet. Đứng trước bài toán đó, vào những năm 1990, nhà mật mã học Nick Szabo đã đưa ra 1 khái niệm hoàn toàn mới là Smart Contract. Ông định nghĩa một smart contract là một giao thức máy tính tạo ra để số hóa, kiểm chứng và thực thi các thỏa thuận và nghĩa vụ được quy định trong một hợp đồng. Smart contract cho phép thực thi các giao dịch một cách đáng tin cậy mà không cần một bên thứ ba làm chứng. Các giao dịch đó có thể theo dấu và không thể đảo ngược được.

Một cách dễ hiểu để mô tả smart contract (dịch ra ngôn ngữ Việt là *hợp đồng thông minh*) là hình dung công nghệ này như một máy bán nước tự động. Thông thường, người làm hợp đồng sẽ phải tìm đến luật sư hay đem đi công chứng, trả tiền cho họ và chờ đợi để lấy giấy tờ tài liệu. Bằng cách sử dụng hợp đồng thông minh, ta chỉ cần trả một bitcoin (giả sử pháp luật Việt Nam cho phép ta sử dụng đồng tiền kỹ thuật số Bitcoin trong giao dịch mua bán như trong ví dụ này) vào máy bán hàng tự động, đã được phát triển trên nền tảng Blockchain và những gì bạn yêu cầu sẽ được trả lại trực tiếp vào tài khoản của bạn. Hơn nữa, hợp đồng thông minh không chỉ xác minh những quy định, quyền lợi và nghĩa vụ giống như hợp đồng truyền thống mà nó còn tự động thực thi những điều trên, không thông qua một bên thứ 3 trong môi trường thiếu niềm tin.

Công nghệ nguyên thủy của hợp đồng thông minh đã từng là một bài toán tư duy "*ngủ yên*" trong hơn một thập kỷ. Thế nhưng mọi thứ đã thay đổi với sự ra đời và phát triển của công nghệ Blockchain. Tuy Bitcoin đã đặt ra những nền tảng cơ bản cho việc thiết lập hợp đồng trên nền tảng Blockchain, nó vẫn còn chưa thể thỏa mãn mọi nhu cầu trong đời sống xã hội hiện nay. Mãi cho đến khi Ethereum ra đời thì ý tưởng hợp đồng thông minh mới chính thức phổ biến, cung cấp phương thức mới để thiết lập hợp đồng. Ngày nay, số lượng các tổ chức, công ty nghiên cứu về các lĩnh vực trong công nghệ Blockchain và ứng dụng của smart contract đã và đang tăng lên đáng kể, cho thấy sự phát triển đầy tiềm năng của công nghệ này.

### 2.2.2 Những ứng dụng dựa trên Smart Contract

Smart contract kết hợp với blockchain có thể ứng dụng trong rất nhiều lĩnh vực, từ tài chính, bảo hiểm, ngân hàng cho đến các nhu cầu như giải trí, bầu cử (voiting), bình chọn,... Với mỗi lĩnh vực ta lại sử dụng một loại hợp đồng thông minh khác nhau, ví dụ hợp đồng vay tiền, hợp đồng đóng tiền định kỳ, hợp đồng mua bán, hợp đồng chuyển nhượng,...

Khi một lĩnh vực được áp dụng hợp đồng thông minh hay còn gọi là smart contract,

mọi quyết định hoặc giao dịch thuận theo hợp đồng đều được xử lý tự động khi thỏa điều kiện đã đưa ra. Ví dụ:

- Hợp đồng vay tiền có lãi suất:

Giả sử rằng bạn vay một số tiền có lãi suất và ký hợp đồng này, khi đến một ngày nhất định, hợp đồng thông minh sẽ tự trừ tiền trong tài khoản của bạn và gửi (cộng thêm) cho người cho vay theo tỉ lệ đúng như đã ký kết.

- Hợp đồng chuyển nhượng:

Giả sử nếu bạn trả 50% số tiền bạn sẽ được giữ cọc món hàng, nếu bạn trả đủ 100% bạn sẽ được nhận món hàng đó, nếu bạn trả 50% nhưng không thanh toán đủ trong thời gian quy định thì bạn sẽ mất cọc, tất cả đều được tự động xử lý với hợp đồng thông minh.

- Bầu cử:

Giả sử ta đang xét trong một quá trình bầu cử cần sự minh bạch, rõ ràng và không xảy ra gian lận giữa người tổ chức và các ứng cử viên. Nếu ta thực hiện theo cách truyền thống, ví dụ như bỏ phiếu hoặc bình chọn bằng tin nhắn, có thể thấy còn thiếu sự minh bạch, và có thể xảy ra gian lận. Cụ thể người chơi (ứng cử viên) không thể biết được những ai đã bầu cho các người chơi khác, và những người bầu cử cũng không biết được thực sự những ai đã bầu cho những người nào. Do đó thiếu đi sự minh bạch trong phương thức truyền thống. Nếu ta sử dụng nền tảng công nghệ Blockchain với smart contract, mọi thứ sẽ được giải quyết ổn thỏa. Kết quả bỏ phiếu sẽ được chuyển vào Blockchain, do smart contract quản lý và phân phối về các node trong mạng lưới. Toàn bộ dữ liệu sẽ được mã hóa và hoàn toàn ẩn danh, mọi người cũng có thể biết chính xác có bao nhiêu phiếu đã bầu cử cho những người nào.

- Logistics

Chúng ta đều biết rằng chuỗi cung ứng là một hệ thống kéo dài và gồm nhiều liên kết khác nhau. Mỗi liên kết cần phải nhận được xác nhận bởi một mắt xích (xem như một node) ở trước đó để đủ điều kiện thực hiện phần việc của mình theo như hợp đồng.

Đây là một quá trình kéo dài, lãng phí và kém năng suất, nhưng với smart contract thì mỗi bộ phận tham gia đều có thể theo dõi tiến trình công việc để từ đó hoàn thành nhiệm vụ đúng hạn. Smart contract bảo đảm tính minh bạch trong điều khoản hợp đồng, chống gian lận.

Nó còn có thể cung cấp cho ta khả năng giám sát quá trình cung ứng nếu như được tích hợp chung với mạng lưới vạn vật kết nối Internet (hay còn gọi là Internet of Things). Có thể nói hai công nghệ này kết hợp với nhau tạo ra một nền công nghệ 4.0 hoàn thiện.

- ICO

Một ứng dụng rộng rãi của hợp đồng thông minh trong lĩnh vực tiền mã hóa đó là phát hành ICO. Một hợp đồng thông minh được viết ra để khi nhà đầu tư gửi

vào địa chỉ của hợp đồng một số tiền, họ sẽ nhận lại được một số token tương ứng với số tiền họ đã bỏ ra. Hợp đồng thông minh đó có thể bổ sung một số điều kiện như đóng bằng số tiền nhận được trong một khoảng thời gian quy định hoặc hủy các token không bán được.

Việc ứng dụng hợp đồng thông minh và blockchain trong ICO giúp các nhà đầu tư có thể theo dõi dự án đã gọi được bao nhiêu tiền, có đạt được mục tiêu hay không, và nhiều thông tin khác nữa.

Ngoài ra, smart contract còn có rất nhiều những ứng dụng khác trong đời sống xã hội cũng như kinh tế chính trị, lĩnh vực tài chính,...

## 2.3 Làm rõ mấy cái phép toán trong ass

## 2.4 Các ký hiệu $\top$ và $\perp$

Các ký hiệu  $\top$  và  $\perp$  được gọi là đơn vị lần lượt của các phép logic Additive Conjunction  $\&$  và Multiplicative Disjunction  $\wp$ . Nếu ta thêm ký hiệu  $\top$  vào một phép Additive Conjunction hoặc  $\perp$  vào Multiplicative Disjunction thì ý nghĩa của phép logic không bị thay đổi.

$$\begin{aligned} A &\multimap (A \& \top) \\ B &\multimap (B \wp \perp) \end{aligned}$$

## 2.5 Exponentials connectives

Trong linear logic, ngoài các connectives (dịch là phép tuyến) *multiplicatives* và *additives* như trên đã trình bày, còn một loại connective khác là *exponentials* (dịch ra là số mũ, ý chỉ cấp số mũ, tăng nhanh chóng hay vô hạn), trong đó bao gồm 2 loại connectives là  $!$  và  $?$ . Chi tiết của hai loại connectives này sẽ được trình bày trong section này.

### 2.5.1 Connective $!$

Đây là một connective trong linear logic, được đọc là "*of course*" (dịch ra là *tất nhiên*) hay "*bang*". Connective này diễn tả một tài nguyên có tiềm năng vô tận, ý nghĩa là dùng để chỉ sự sản sinh, hay có được một số lượng không giới hạn của một yếu tố nào đó. Cụ thể:

**!A:** có nghĩa là tạo ra một lượng không giới hạn yếu tố A.

#### Ví dụ 1

Trong thực đơn menu của một nhà hàng nêu rõ một phần ăn giá \$5 dành cho một người gồm các phần ăn như sau:

- + Hamburger
- + Fries or Wedges
- + Unlimited Pepsi



+ Ice-cream or Sorbet

Ta có thể chuyển bài toán trên thành linear logic với các connectives đã học. Cụ thể kí hiệu Hamburger (H), Fries (F), Wedges (W), Pepsi (P), Ice-cream (I) và Sorbet (S), khi đó ta có:

$$\$1 \otimes \$1 \otimes \$1 \otimes \$1 \otimes \$1 \multimap H \otimes (F \& W) \otimes !P \otimes (I \oplus S)$$

Ta thấy trong ví dụ này, ta sử dụng connective ! cho trường hợp ta tạo ra một lượng không giới hạn pepsis.

### Ví dụ 2

Xét một ví dụ khác liên quan đến *Constraint Handling Rules (CHR)*. Xét bài toán tung đồng xu ta luôn có một trong hai kết quả là mặt ngửa và mặt sấp mỗi khi tung đồng xu. Cụ thể trong logic cổ điển ta có thể diễn tả như sau

$$(throw(Coin) \Leftrightarrow Coin = head) \wedge (throw(Coin) \Leftrightarrow Coin = tail)$$

Có nghĩa khi tung đồng xu, không phải mặt ngửa thì là mặt sấp. Còn trong linear logic, chúng ta có thể dùng biểu thức sau để nhấn mạnh tính đúng đắn của định luật trên bằng connective ! (dịch ra là *tất nhiên*) như sau:

$$!(throw(Coin) \multimap ((Coin = head) \& (Coin = tail)))$$

Biểu thức trên sử dụng connective ! để chỉ ra việc ta có một tiềm năng vô hạn về việc khi chi tiêu A (tung đồng xu), ta sẽ có được B (không ngửa thì là sấp). Cũng có thể đọc là "*Tất nhiên khi tung đồng xu, không phải mặt ngửa thì là mặt sấp*".

### 2.5.2 Connective ?

Tiếp theo, ta sẽ nói về connective còn lại trong Exponentials connective trong linear logic, đó là connective ?, được đọc là "*why not*".

Ngược lại với connective !, connective ? diễn tả một thực tế về tài nguyên hiện tại có tiềm năng vô tận, tức mang ý nghĩa chi tiêu (tiêu thụ) một lượng không giới hạn một yếu tố, cụ thể:

**?A:** có nghĩa chi tiêu một lượng không giới hạn yếu tố A.

### Ví dụ

Trong thực tế, sẽ không có ví dụ nào minh họa được tính có sẵn nguồn lực vô hạn mà cụ thể, ta đều có thể ước tính được lượng cần sử dụng cần thiết để tạo ra một thứ gì đó. Lấy ví dụ như trong việc sử dụng các nguồn năng lượng tự nhiên như năng lượng gió, năng lượng mặt trời, năng lượng nước, bằng các thực nghiệm và tính toán, ta đều có thể ước tính được lượng ta cần sử dụng để tạo ra một khối lượng sản phẩm đầu ra sau cùng.

Cụ thể, trong ngành năng lượng thủy điện, sử dụng nước để tạo ra điện, lấy ví dụ người ta cần dùng  $500m^3$  nước để sản sinh ra  $1kJ$  điện năng. Tuy nhiên ta có thể giả sử việc sử dụng lượng nước bao nhiêu là không tính trước được, hoặc ta cần một lượng vô hạn điện năng, ta có thể sử dụng linear logic để biểu diễn bài toán như sau:

$$?(hydro) \multimap 1kJ \text{ hay } ?(hydro) \multimap !(electric)$$

Lấy một ví dụ khác trong ngành điện gió, ta cần một lượng động năng sinh ra từ gió không giới hạn để tạo ra một lượng điện năng cho toàn bộ thành phố (cũng chưa biết):

$$?(wind) \multimap !(electric)$$

### 2.5.3 Liên hệ của ! và ? trong linear logic

Trong linear logic, ta có mối quan hệ sau

$$(?A)^\perp = !(A^\perp) \text{ và } (!A)^\perp = ?(A^\perp)$$

Trong đó hai connectives này là dual của nhau.

### 2.5.4 Một số biểu thức liên quan

#### Biểu thức 1. $!(A \& B) \equiv !A \otimes !B$

Biểu thức này nói rằng có sự tương đương giữa hai yếu tố  $!(A \& B)$  và  $!A \otimes !B$ . Ta có thể giải thích như sau:

+  $!(A \& B)$ : có nghĩa là ta có được một lượng vô hạn các sự lựa chọn (mang tính chủ động, phụ thuộc cách chọn của ta) các yếu tố A, hoặc B. Do đó ở mỗi lần lựa chọn trong số vô hạn lần tạo ra, ta có thể tùy ý chọn A hoặc chọn B.

+  $!A \otimes !B$ : còn công thức này có nghĩa là ta tạo ra được cả A lẫn B, trong đó không giới hạn số lượng A, hay B tạo ra. Có nghĩa ta có được vô hạn lần A cũng như vô hạn lần B.

Qua đó ta thấy có sự tương đương giữa 2 công thức, đều muốn đề cập đến việc có được vô hạn các yếu tố A và B.

#### Biểu thức 2. $A \otimes !(A \multimap B) \vdash B \otimes !(A \multimap B)$

Đây là một sequent chỉ ra rằng khi có A và "luôn có" tính chất "Chỉ tiêu A sẽ được B", thì ta sẽ có B, cũng như tính chất trên không thay đổi. Điều này hiển nhiên là đúng và ở đây ta thấy được việc sử dụng connective ! để chỉ một tính chất luôn đúng mà ta có được (ở đây là chỉ tiêu A sẽ được B).

## 3 Bài toán 2: Ngữ cảnh smart contract

### 3.1 Mô tả ngữ cảnh cho smart contract

#### 3.1.1 Động cơ và mục tiêu

Trong các hoạt động kinh doanh, vận chuyển hàng hóa hiện nay, hợp đồng thông minh (smart contract) giúp giải quyết rất nhiều những khó khăn, rào cản mà việc xử lý thủ công mắc phải. Trước hết, bởi vì các smart contract sẽ thực hiện tự động những hoạt động mà trước đây thường phải thực hiện thủ công (như việc xác nhận đơn hàng) nên chúng có khả năng thúc đẩy tốc độ của quy trình kinh doanh. Ngoài ra, smart contract cũng đem lại sự đảm bảo về độ chính xác của giao dịch cao hơn, ít lỗi hơn, từ đó giảm thiểu đáng kể rủi ro khi thực hiện hợp đồng. Bên cạnh đó, một ưu điểm cần phải kể đến của smart contract đó là việc tối thiểu hóa sự tham gia của các bên thứ ba hay bên trung gian vào quá trình thực hiện hợp đồng. Sự tinh giản này giúp tiết kiệm đáng kể chi phí kinh doanh. Đặt trong bối cảnh một doanh nghiệp phải thực hiện hàng ngày một lượng khổng lồ các thủ tục xác nhận đơn hàng logistics, lợi ích mà smart contract đem lại sẽ không chỉ kể hết trong một hai trang giấy. Như vậy, với lý tưởng đảm bảo sự công bằng, minh bạch và tiết kiệm tối đa chi phí cho các bên tham gia giao kết hợp đồng, smart contract thực sự đã cho thấy những tiềm năng ứng dụng của nó, không chỉ đơn thuần áp dụng vào các hoạt động thương mại, mà còn được kỳ vọng sẽ hoạt động hiệu quả trong việc bầu cử, tiếp cận dữ liệu về sức khỏe và dân số, hỗ trợ quy trình bồi thường bảo hiểm hoặc quản lý chính phủ.

#### 3.1.2 Tình huống cụ thể

Với tình huống cụ thể là giao dịch liên quan đến vận tải đa phương thức (logistics) giữa công ty A và công ty B. Giả sử bên A (công ty A) có nhu cầu vận chuyển một lô hàng thủ công mỹ nghệ, và sử dụng dịch vụ vận tải của công ty B (gọi là bên B). Tuy nhiên bên A và bên B chưa có được niềm tin tưởng lẫn nhau. Cụ thể bên A băn khoăn rằng liệu bên B có bảo quản tốt hàng hóa, đảm bảo về chất lượng lẫn số lượng, cũng như thời gian chuyển hàng tới địa điểm chỉ định đúng hẹn hay không? Còn bên B thì đặt ra câu hỏi là liệu bên A có đảm bảo uy tín, không phải là công ty lừa đảo, hay hàng hóa của bên A là hợp pháp hay bất hợp pháp, cũng như việc thù lao khi bên B giao hàng đến nơi có được chuyển đúng hẹn? Để giải quyết những khúc mắc này, 2 bên phải giao kết một hợp đồng thỏa thuận về việc vận chuyển hàng hóa, dẫn đến những phát sinh về dịch vụ tư vấn luật như soạn thảo hợp đồng, phòng ngừa các rủi ro pháp lý,... Vậy cách nào để đơn giản hóa quy trình thỏa thuận giữa 2 bên A và B, khi mà không có niềm tin trong môi trường doanh nghiệp như trong tình huống này?

Lúc này, smart contract có thể giải quyết câu hỏi này. Khi hai bên tham gia vào mô hình này, hai bên cung cấp những thông tin về thời gian, địa điểm, hàng hóa, các quy định bồi thường vào smart contract, thông tin này sau đó được gửi cho cả 2 phía. Bên A cần chuyển vào số tiền thù lao phải chi trả cho bên B nếu bên B hoàn tất công việc. Còn bên B cần chuyển vào số tiền mà bên B có thể sẽ phải bồi thường nếu xảy ra sự cố trong quá trình vận chuyển, số tiền này tối thiểu phải là lượng tiền bồi thường lớn nhất mà bên

B gánh phải trong suốt quá trình hợp đồng xảy ra. Khi 2 bên thỏa thuận, smart contract chính thức có hiệu lực. Trong suốt quá trình nhận hàng, kiểm tra hàng, vận chuyển, tiếp nhận, kiểm tra lại hàng hóa của 2 phía, smart contract sẽ luôn thực thi khi một điều kiện nào đó kích hoạt nó, bao gồm việc bồi thường của 2 bên, hoặc hợp đồng bị hủy, hoặc hoàn tất.

Chi tiết của các điều kiện trong bản hợp đồng sẽ kích hoạt smart contract như sau: diễn ra ở các giai đoạn B đến nhận hàng từ A, B kiểm tra hàng hóa của A, A nhận hàng khi B vận chuyển và cuối cùng là A kiểm tra lại hàng hóa của mình.

- Khi B đến nhận hàng từ A  
Gồm các quy định về thời gian nhận hàng, B cần phải đến sớm hoặc đúng hơn thời gian nhận hàng, nếu đến trễ hoặc không đến, hợp đồng sẽ bị hủy. Bên A có trách nhiệm phải chuẩn bị hàng hóa đầy đủ để giao cho bên B. Nếu bên A không có hàng hóa, bên A sẽ phải bồi thường khoản tiền chi phí di chuyển cho bên B và sau đó hợp đồng kết thúc.
- Khi B kiểm tra hàng hóa của A  
Gồm các quy định về thông tin hàng hóa, bên A phải đảm bảo hàng hóa của mình đúng như thông tin hàng hóa quy định trong smart contract. Bên A sẽ phải chịu bồi thường và hủy hợp đồng.
- Khi bên A nhận hàng hóa từ bên B  
Gồm các quy định về thời gian, bên B phải giao hàng cho bên A đúng địa điểm thời gian trong smart contract. Nếu đến trễ hoặc không đến, bên B sẽ phải chịu bồi thường.
- Khi bên A kiểm tra lại hàng hóa  
Gồm các quy định về hàng hóa, bên B phải đảm bảo hàng hóa còn nguyên vẹn. Nếu vi phạm, bên B sẽ phải chịu bồi thường. Nếu không có gì xảy ra, smart contract xem như hoàn tất.

### 3.2 Mô tả các điều khoản cho ngữ cảnh smart contract

Trong ngữ cảnh của bài toán trên, ta có 2 đối tượng cần xem xét là bên thuê A và bên vận chuyển B. Giữa 2 đối tượng có những quan hệ và ràng buộc, cần được chuyển thành những điều khoản cụ thể được quy định trong smart contract. Smart contract đóng vai trò là bên thứ 3 quy định các điều khoản này, tạo nên niềm tin giữa 2 bên khi 2 bên không có niềm tin tưởng cho nhau. Cụ thể ta có thể cụ thể thành các điều khoản sau cho smart contract:

#### Article 1.

Bên A đưa ra thông tin về hàng hóa cần bên B vận chuyển (gồm thông tin khách hàng, tính chất, số lượng, giá trị hàng hóa, thời gian dự kiến, địa điểm giao/nhận hàng). Tất cả các thông tin được smart contract ghi lại và public cho 2 phía.

#### **Article 2.**

Bên B sau khi nhận thông tin hàng hóa được gửi từ bên A, bên B sẽ tiến hành xác nhận đơn hàng sẽ gửi lại thông tin đơn hàng kèm theo các thông tin bổ sung về thời gian chi tiết, chi phí. Tất cả thông tin được smart contract ghi nhận lại và public qua cho 2 phía.

#### **Article 3.**

Bên A gửi toàn bộ số tiền vào smart contract mà bên B sẽ được nhận nếu bên B hoàn thành xong việc đúng như quy định trong hợp đồng.

#### **Article 4.**

Bên B gửi vào smart contract số tiền đặt cọc tối thiểu phải bằng số tiền bồi thường tối đa mà bên B phải chịu nếu có phát sinh những vi phạm các điều khoản trong hợp đồng.

#### **Article 5.**

Nếu đúng thời gian gửi hàng mà đúng 1 bên vẫn chưa gửi số tiền quy định tại điều 3 và 4 thì smart contract sẽ gửi tiền trả về cho bên kia. Hoặc nếu chưa có bên nào gửi tiền thì hợp đồng kết thúc.

#### **Article 6.**

Nếu bên B đến nơi nhận hàng trễ hơn thời gian quy định thì smart contract tự động gửi số tiền ban đầu về cho cả hai bên và kết thúc hợp đồng.

#### **Article 7.**

Nếu bên B đến địa điểm gửi hàng đúng giờ mà bên A vẫn chưa chuẩn bị hàng đầy đủ hoặc hàng không đúng như mô tả thì smart contract sẽ tự động chuyển 1 số tiền bồi thường chi phí di chuyển cho bên B. Sau đó gửi số tiền còn lại cho mỗi bên rồi kết thúc hợp đồng.

#### **Article 8.**

Nếu bên B giao hàng trễ hơn so với thời gian quy định nhưng không quá thời gian cho phép, hoặc không có hàng để giao hoặc làm thất thoát số hàng thì bên B phải bồi thường thiệt hại cho bên A với khoản tiền tương ứng. Trường hợp nếu bên B không giao hàng, thì phần chi phí vận chuyển sẽ được trả về cho bên A.

#### **Article 9.**

Sau khi bên B hoàn thành xong quá trình giao hàng thì smart contract sẽ gửi toàn bộ số tiền qua cho bên B. (gồm tiền chuyển hàng của A và tiền đặt cọc ban đầu của B trừ đi các khoản bồi thường nếu có). Bên A không nhận thêm bất cứ khoản phí nào. Đồng thời smart contract cũng không còn tiền trong tài khoản.

## 4 Bài toán 3: Đặc tả ngữ cảnh smart contract bằng linear logic

Dựa vào các điều khoản đã được nêu bên trên, dưới đây là phần đặc tả smart contract sử dụng linear logic.

- Nếu đến thời gửi hàng mà chưa có đủ tiền từ cả hai bên, smart contract sẽ trả tiền về bên đã gửi trước đó:

$$tg\_gui \otimes (moneyA \oplus moneyB) \multimap (walletA + moneyA) \wp (walletB + moneyB)$$

- Nếu bên B đến địa điểm nhận hàng nhưng đã quá thời gian qui định, smart contract sẽ trả tiền lại cho cả hai bên:

$$tg\_gui \otimes B^\perp \multimap (walletA + moneyA) \otimes (walletB + moneyB)$$

- Nếu bên B đến địa điểm nhận hàng đúng hẹn, nhưng bên A vẫn chưa chuẩn bị đủ số lượng hàng hoặc thông tin hàng hóa khác so với thông tin mô tả ban đầu thì smart contract sẽ chuyển một khoản bồi thường cho bên B, đồng thời phần tiền còn lại cũng sẽ được trả về cho hai bên:

$$tg\_gui \otimes B \otimes (goods \oplus (goods\_info \neq goods\_info2)) \multimap$$

$$(walletA + moneyA - moneyA\_boithuong) \otimes (walletB + moneyB + moneyA\_boithuong)$$

- Nếu bên B vận chuyển hàng đến nơi trễ, nhưng không vượt quá thời gian cho phép thì smart contract sẽ gửi một khoảng bồi thường cho bên A:

$$tg\_chophep \otimes B2 \multimap walletA + moneyB\_boithuong$$

- Nếu hết khoảng thời gian cho phép nhưng hàng vẫn chưa được vận chuyển đến, xem như bên B không giao hàng, lúc này smart contract sẽ chuyển toàn bộ số tiền ứng với giá trị đơn hàng cho bên A và chi phí vận chuyển (bên A không trả cho bên B):

$$het\_tg\_chophep \otimes goods^\perp \multimap walletA + moneyB + moneyA$$

- Nếu hàng được giao nhưng không đảm bảo số lượng, chất lượng,... tùy theo mức độ thiệt hại, smart contract gửi tiền bồi thường cho bên A:

$$goods2^\perp \multimap walletA + moneyB\_boithuong2$$

- Sau khi quá trình vận chuyển hàng kết thúc, smart contract sẽ chuyển tiền cho bên B, bao gồm phí vận chuyển hàng, và khoản tiền ban đầu bên B gửi vào smart contract trừ đi các khoản bồi thường (nếu có), tại thời điểm hết khoảng thời gian cho phép. Ngược lại, smart contract vẫn giữ số tiền này và chờ đến đúng thời gian sẽ thực thi:

$$goods \otimes het\_tg\_chophep \multimap walletB + moneyB + moneyA$$

$$goods \otimes het\_tg\_chophep^\perp \multimap (walletB + moneyB + moneyA)^\perp$$

## 5 Bài toán 4: Dùng mã giả xây dựng smart contract

## 6 Bài toán 5: Dùng Solidity xây dựng smart contract

### 6.1 Hiện thực code bằng Solidity

Dưới đây là phần hiện thực smart contract trên bằng Solidity, có thể chạy trên môi trường remix của ethereum. Mã nguồn gồm 2 file tương ứng 2 smart contract được hiện thực. Smart contract đầu tiên là Transport hiện thực nội dung của bản hợp đồng trên. Trong smart contract này, có sử dụng một smart contract khác là DateTime, đây là một open source trên mạng lưới ethereum, hỗ trợ chuyển đổi thời gian trên hệ thống ethereum.

Đây là smart contract Transport:

```
1 pragma solidity ^0.4.18;
2
3 import './DateTime.sol';
4
5 contract Transport {
6
7     DateTime private dt;
8
9     address public addr_A;
10    address public addr_B;
11
12    uint8 public p_info; // product infomation
13
14    uint public m_A;
15    uint public m_B;
16
17    uint public t_B_nhan; // timestamp B received product to transfer
18    uint public t_A_nhan; // timestamp A received product after B
19    transfer
20    uint32 public dt_B_tre_nhan; // delta time
21    uint32 public dt_A_tre_gui;
22    uint32 public dt_B_tre_gui;
23
24    string public a_A_gui; // address A send product to B
```

```
24     string public a_A_nhan; // address A receive product after B
    transfer
25
26     uint32 public m_B_tre;
27     uint32 public m_B_huy;
28     uint32 public m_B_tre_gui;
29     uint32 public m_B_khong_hang;
30     uint32 public m_B_hang_hong;
31
32     uint public m_A_tre_gui;
33     uint public m_A_khong_dung_hen;
34     uint public m_A_khong_dung_hang;
35
36     // flags
37     bool private provided_info_A = false;
38     bool private provided_time_A = false;
39     bool private provided_money_A = false;
40     bool private provided_money_B = false;
41     bool private sent_deposits_A = false;
42     bool private sent_deposits_B = false;
43
44     bool private passed_B_nhan_hang = false;
45
46
47     constructor(address address_A, address address_B, address
    addrDateTime) public {
48         addr_A = address_A;
49         addr_B = address_B;
50         dt = DateTime(addrDateTime);
51     }
52
53     modifier onlyA() {
54         require(msg.sender == addr_A);
55         _;
56     }
57
58     modifier onlyB() {
59         require(msg.sender == addr_B);
60         _;
61     }
62
63     modifier only_A_or_B() {
64         require(msg.sender == addr_A || msg.sender == addr_B);
65         _;
66     }
67
68     modifier agreementCompleted() {
69         require(provided_time_A && provided_info_A
70             && provided_money_A && provided_money_B
71             && sent_deposits_A && sent_deposits_B);
72         _;
73     }
74
75     modifier passed_B_nhanHang() {
76         require(passed_B_nhan_hang);
77         _;
78     }
79
```



```
80     function provideTime_A(  
81         uint32 _dt_A_tre_gui,  
82         uint32 _dt_B_tre_nhan,  
83         uint32 _dt_B_tre_gui,  
84         uint16 year_B, uint8 month_B, uint8 day_B, uint8 hour_B, uint8  
            minute_B,  
85         uint16 year_A, uint8 month_A, uint8 day_A, uint8 hour_A, uint8  
            minute_A  
86     )  
87     onlyA public {  
88         if (!provided_time_A) {  
89             dt_A_tre_gui = _dt_A_tre_gui;  
90             dt_B_tre_nhan = _dt_B_tre_nhan;  
91             dt_B_tre_gui = _dt_B_tre_gui;  
92             t_B_nhan = dt.toTimestamp(year_B, month_B, day_B, hour_B,  
                minute_B);  
93             t_A_nhan = dt.toTimestamp(year_A, month_A, day_A, hour_A,  
                minute_A);  
94         }  
95         provided_time_A = true;  
96     }  
97  
98     function provideInfo_A(  
99         uint8 _p_info,  
100         string _a_A_gui,  
101         string _a_A_nhan  
102     )  
103     onlyA public {  
104         if (!provided_info_A) {  
105             p_info = _p_info;  
106             a_A_gui = _a_A_gui;  
107             a_A_nhan = _a_A_nhan;  
108         }  
109         provided_info_A = true;  
110     }  
111  
112     function provideMoney_A(  
113         uint32 _m_B_tre,  
114         uint32 _m_B_tre_gui,  
115         uint32 _m_B_khong_hang,  
116         uint32 _m_B_hang_hong  
117     )  
118     onlyA public {  
119         if (!provided_money_A) {  
120             m_B_tre = _m_B_tre;  
121             m_B_tre_gui = _m_B_tre_gui;  
122             m_B_khong_hang = _m_B_khong_hang;  
123             m_B_hang_hong = _m_B_hang_hong;  
124         }  
125         provided_money_A = true;  
126     }  
127  
128     function sendDeposits_A() onlyA payable public {  
129         if (!sent_deposits_A) {  
130             m_A = msg.value;  
131         }  
132         sent_deposits_A = true;  
133     }
```

```
134
135     function provideMoney_B(
136         uint32 _m_A_tre_gui,
137         uint32 _m_A_khong_dung_hen,
138         uint32 _m_A_khong_dung_hang
139     )
140     onlyB public {
141         if (!provided_money_B) {
142             m_A_tre_gui = _m_A_tre_gui;
143             m_A_khong_dung_hang = _m_A_khong_dung_hang;
144             m_A_khong_dung_hen = _m_A_khong_dung_hen;
145         }
146         provided_money_B = true;
147     }
148
149     function sendDeposits_B() onlyB payable public {
150         if (!sent_deposits_B) {
151             m_B = msg.value;
152         }
153         sent_deposits_B = true;
154     }
155
156     function checkCurrentBalance() only_A_or_B public view returns (uint
157     ) {
158         return address(this).balance;
159     }
160
161     function trigger_B_nhan_hang_de_gui(
162         uint16 year_B, uint8 month_B, uint8 day_B, uint8 hour_B, uint8
163         minute_B,
164         uint16 year_A, uint8 month_A, uint8 day_A, uint8 hour_A, uint8
165         minute_A,
166         uint8 p_nhan
167     )
168     only_A_or_B agreementCompleted public {
169         uint t_B_den_nhan = dt.toTimestamp(year_B, month_B, day_B,
170         hour_B, minute_B);
171         uint t_A_gui_hang = dt.toTimestamp(year_A, month_A, day_A,
172         hour_A, minute_A);
173
174         if (t_B_den_nhan > t_B_nhan) {
175             // B den nhan hang tre
176             if (t_B_den_nhan - t_B_nhan <= dt_B_tre_nhan) {
177                 // chua qua thoi han, chi can boi thuong
178                 addr_A.transfer(m_B_tre);
179                 m_B -= m_B_tre;
180             } else {
181                 // qua han, huy hop dong
182                 addr_A.transfer(m_A + m_B_huy);
183                 selfdestruct(addr_B);
184             }
185         }
186
187         if (t_A_gui_hang > t_B_nhan) {
188             // A gui hang tre
189             if (t_A_gui_hang - t_B_nhan <= dt_A_tre_gui) {
190                 // chua qua thoi han, chi can boi thuong
191                 addr_B.transfer(m_A_tre_gui);
192             }
193         }
194     }
195 }
```

```
187         m_A -= m_A_tre_gui;
188     } else {
189         // qua han, huy hop dong
190         addr_A.transfer(m_A - m_A_khong_dung_hen);
191         selfdestruct(addr_B);
192     }
193 }
194
195 if (p_nhan != p_info) {
196     // A gui khong dung hang, huy hop dong
197     addr_A.transfer(m_A - m_A_khong_dung_hang);
198     selfdestruct(addr_B);
199 }
200
201 passed_B_nhan_hang = true;
202 }
203
204 function trigger_A_nhan_hang_B_giao(
205     uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute,
206     bool hang_on
207 )
208 only_A_or_B passed_B_nhanHang public {
209     uint t_B_gui_den = dt.toTimestamp(year, month, day, hour, minute
210 );
211
212     if (t_B_gui_den > t_A_nhan) {
213         // B gui hang den tre
214         if (t_B_gui_den - t_A_nhan <= dt_B_tre_gui) {
215             // chua qua thoi han, chi can boi thuong
216             addr_A.transfer(m_B_tre_gui);
217             m_B -= m_B_tre;
218         } else {
219             // qua han, khong co hang, huy hop dong
220             addr_A.transfer(m_A + m_B_khong_hang);
221             selfdestruct(addr_B);
222         }
223     }
224
225     if (!hang_on) {
226         // B gui hang khong on, boi thuong
227         addr_A.transfer(m_B_hang_hong);
228         m_B -= m_B_hang_hong;
229     }
230
231     // complete
232     selfdestruct(addr_B);
233 }
```

Đây là smart contract DateTime:

```
1 pragma solidity ^0.4.16;
2
3 contract DateTime {
4     /*
5      *   Date and Time utilities for ethereum contracts
6      */
7 }
```

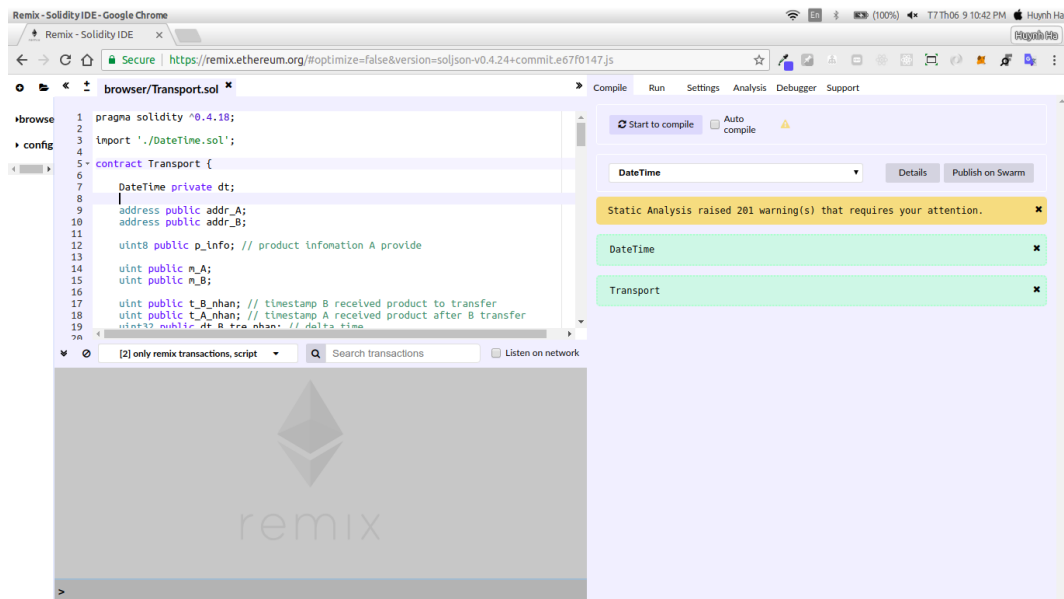
```
7      */
8      struct _DateTime {
9          uint16 year;
10         uint8 month;
11         uint8 day;
12         uint8 hour;
13         uint8 minute;
14         uint8 second;
15         uint8 weekday;
16     }
17
18     uint constant DAY_IN_SECONDS = 86400;
19     uint constant YEAR_IN_SECONDS = 31536000;
20     uint constant LEAP_YEAR_IN_SECONDS = 31622400;
21
22     uint constant HOUR_IN_SECONDS = 3600;
23     uint constant MINUTE_IN_SECONDS = 60;
24
25     uint16 constant ORIGIN_YEAR = 1970;
26
27     function isLeapYear(uint16 year) public pure returns (bool) {
28         if (year % 4 != 0) {
29             return false;
30         }
31         if (year % 100 != 0) {
32             return true;
33         }
34         if (year % 400 != 0) {
35             return false;
36         }
37         return true;
38     }
39
40     function leapYearsBefore(uint year) public pure returns (uint) {
41         year -= 1;
42         return year / 4 - year / 100 + year / 400;
43     }
44
45     function getDaysInMonth(uint8 month, uint16 year) public pure
46         returns (uint8) {
47         if (month == 1 || month == 3 || month == 5 || month == 7
48             || month == 8 || month == 10 || month == 12) {
49             return 31;
50         }
51         else if (month == 4 || month == 6 || month == 9 || month
52             == 11) {
53             return 30;
54         }
55         else if (isLeapYear(year)) {
56             return 29;
57         }
58         else {
59             return 28;
60         }
61     }
62
63     function parseTimestamp(uint timestamp) internal pure returns (
64         _DateTime dt) {
```

```
61         uint secondsAccountedFor = 0;
62         uint buf;
63         uint8 i;
64
65         // Year
66         dt.year = getYear(timestamp);
67         buf = leapYearsBefore(dt.year) - leapYearsBefore(
            ORIGIN_YEAR);
68
69         secondsAccountedFor += LEAP_YEAR_IN_SECONDS * buf;
70         secondsAccountedFor += YEAR_IN_SECONDS * (dt.year -
            ORIGIN_YEAR - buf);
71
72         // Month
73         uint secondsInMonth;
74         for (i = 1; i <= 12; i++) {
75             secondsInMonth = DAY_IN_SECONDS * getDaysInMonth
                (i, dt.year);
76             if (secondsInMonth + secondsAccountedFor >
                timestamp) {
77                 dt.month = i;
78                 break;
79             }
80             secondsAccountedFor += secondsInMonth;
81         }
82
83         // Day
84         for (i = 1; i <= getDaysInMonth(dt.month, dt.year); i++)
            {
85             if (DAY_IN_SECONDS + secondsAccountedFor >
                timestamp) {
86                 dt.day = i;
87                 break;
88             }
89             secondsAccountedFor += DAY_IN_SECONDS;
90         }
91
92         // Hour
93         dt.hour = getHour(timestamp);
94
95         // Minute
96         dt.minute = getMinute(timestamp);
97
98         // Second
99         dt.second = getSecond(timestamp);
100
101         // Day of week.
102         dt.weekday = getWeekday(timestamp);
103     }
104
105     function getYear(uint timestamp) public pure returns (uint16) {
106         uint secondsAccountedFor = 0;
107         uint16 year;
108         uint numLeapYears;
109
110         // Year
111         year = uint16(ORIGIN_YEAR + timestamp / YEAR_IN_SECONDS)
            ;
```

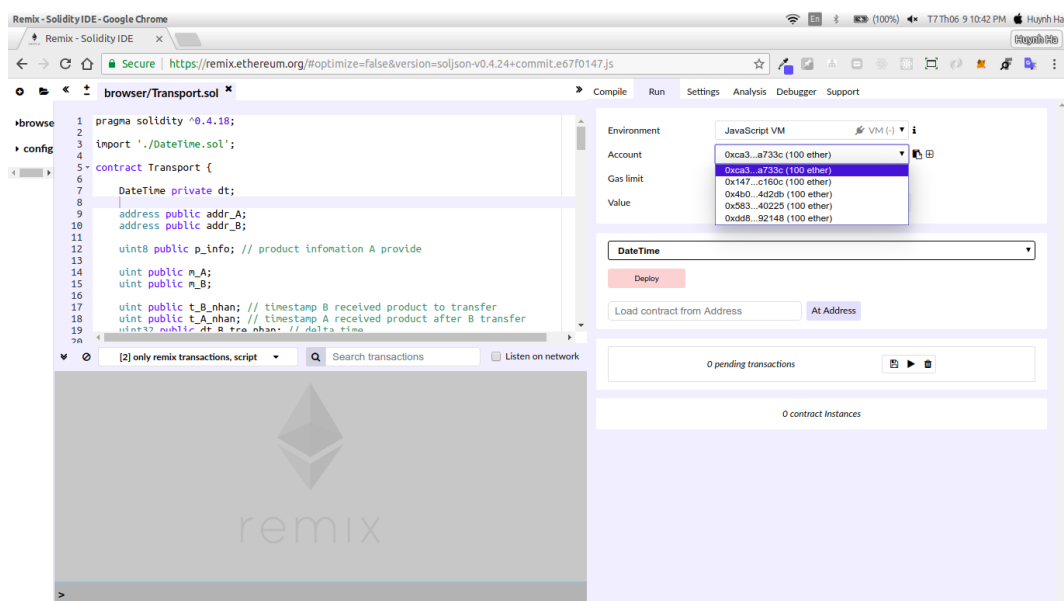
```
112         numLeapYears = leapYearsBefore(year) - leapYearsBefore(
113             ORIGIN_YEAR);
114         secondsAccountedFor += LEAP_YEAR_IN_SECONDS *
115             numLeapYears;
116         secondsAccountedFor += YEAR_IN_SECONDS * (year -
117             ORIGIN_YEAR - numLeapYears);
118         while (secondsAccountedFor > timestamp) {
119             if (isLeapYear(uint16(year - 1))) {
120                 secondsAccountedFor -=
121                     LEAP_YEAR_IN_SECONDS;
122             }
123             else {
124                 secondsAccountedFor -= YEAR_IN_SECONDS;
125             }
126             year -= 1;
127         }
128         return year;
129     }
130     function getMonth(uint timestamp) public pure returns (uint8) {
131         return parseTimestamp(timestamp).month;
132     }
133     function getDay(uint timestamp) public pure returns (uint8) {
134         return parseTimestamp(timestamp).day;
135     }
136     function getHour(uint timestamp) public pure returns (uint8) {
137         return uint8((timestamp / 60 / 60) % 24);
138     }
139     function getMinute(uint timestamp) public pure returns (uint8) {
140         return uint8((timestamp / 60) % 60);
141     }
142     function getSecond(uint timestamp) public pure returns (uint8) {
143         return uint8(timestamp % 60);
144     }
145     function getWeekday(uint timestamp) public pure returns (uint8)
146     {
147         return uint8((timestamp / DAY_IN_SECONDS + 4) % 7);
148     }
149     function toTimestamp(uint16 year, uint8 month, uint8 day) public
150     pure returns (uint timestamp) {
151         return toTimestamp(year, month, day, 0, 0, 0);
152     }
153     function toTimestamp(uint16 year, uint8 month, uint8 day, uint8
154     hour) public pure returns (uint timestamp) {
155         return toTimestamp(year, month, day, hour, 0, 0);
156     }
157     function toTimestamp(uint16 year, uint8 month, uint8 day, uint8
158     hour, uint8 minute) public pure returns (uint timestamp) {
```

```
162         return toTimestamp(year, month, day, hour, minute, 0);
163     }
164
165     function toTimestamp(uint16 year, uint8 month, uint8 day, uint8
        hour, uint8 minute, uint8 second) public pure returns (uint
        timestamp) {
166         uint16 i;
167
168         // Year
169         for (i = ORIGIN_YEAR; i < year; i++) {
170             if (isLeapYear(i)) {
171                 timestamp += LEAP_YEAR_IN_SECONDS;
172             }
173             else {
174                 timestamp += YEAR_IN_SECONDS;
175             }
176         }
177
178         // Month
179         uint8[12] memory monthDayCounts;
180         monthDayCounts[0] = 31;
181         if (isLeapYear(year)) {
182             monthDayCounts[1] = 29;
183         }
184         else {
185             monthDayCounts[1] = 28;
186         }
187         monthDayCounts[2] = 31;
188         monthDayCounts[3] = 30;
189         monthDayCounts[4] = 31;
190         monthDayCounts[5] = 30;
191         monthDayCounts[6] = 31;
192         monthDayCounts[7] = 31;
193         monthDayCounts[8] = 30;
194         monthDayCounts[9] = 31;
195         monthDayCounts[10] = 30;
196         monthDayCounts[11] = 31;
197
198         for (i = 1; i < month; i++) {
199             timestamp += DAY_IN_SECONDS * monthDayCounts[i -
                1];
200         }
201
202         // Day
203         timestamp += DAY_IN_SECONDS * (day - 1);
204
205         // Hour
206         timestamp += HOUR_IN_SECONDS * (hour);
207
208         // Minute
209         timestamp += MINUTE_IN_SECONDS * (minute);
210
211         // Second
212         timestamp += second;
213
214         return timestamp;
215     }
216 }
```

## 6.2 Ảnh minh họa cho việc chạy trên môi trường remix

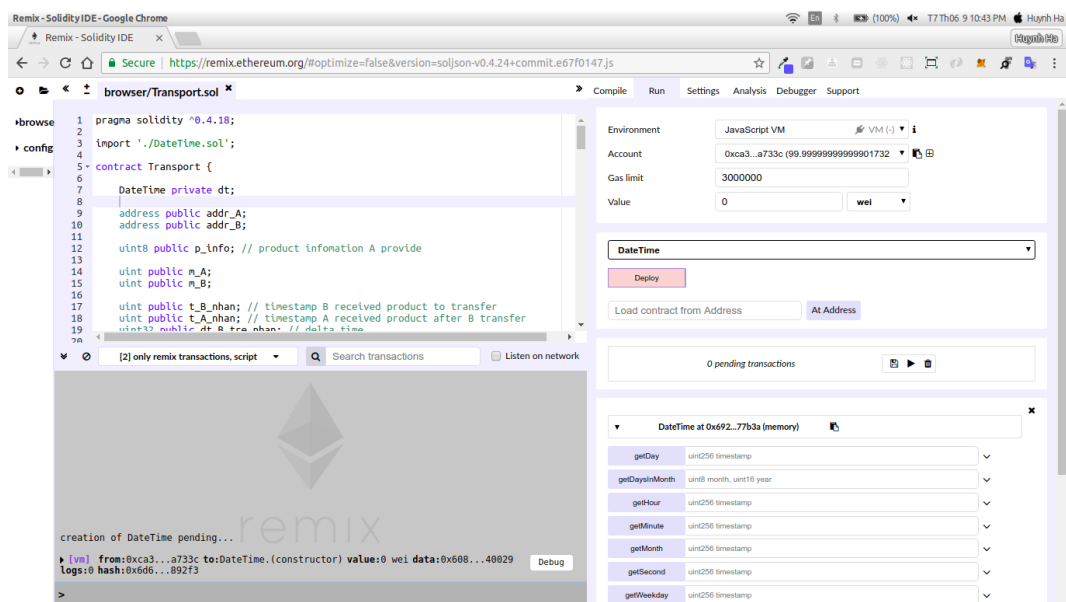


Hình 1: Compile smart contract

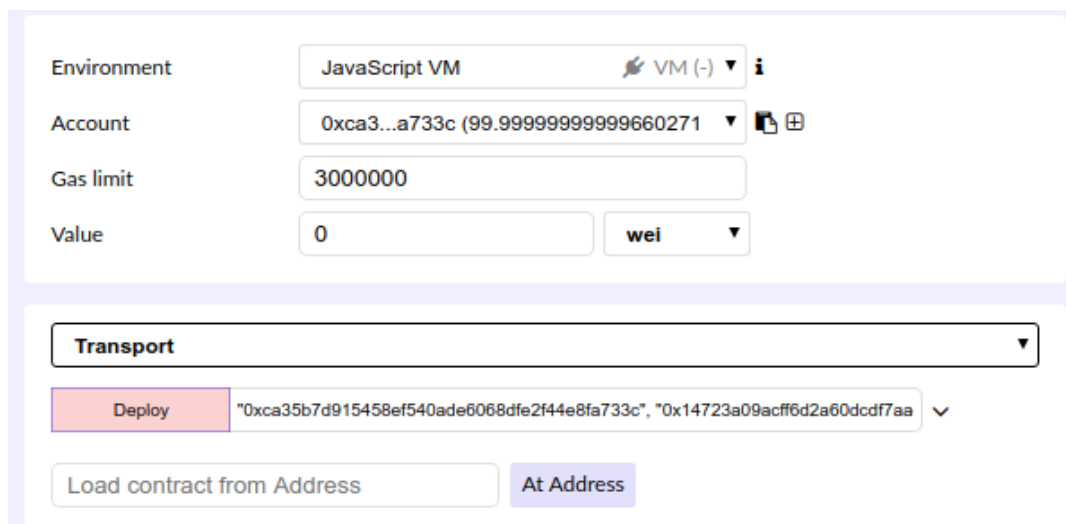


Hình 2: Compile smart contract

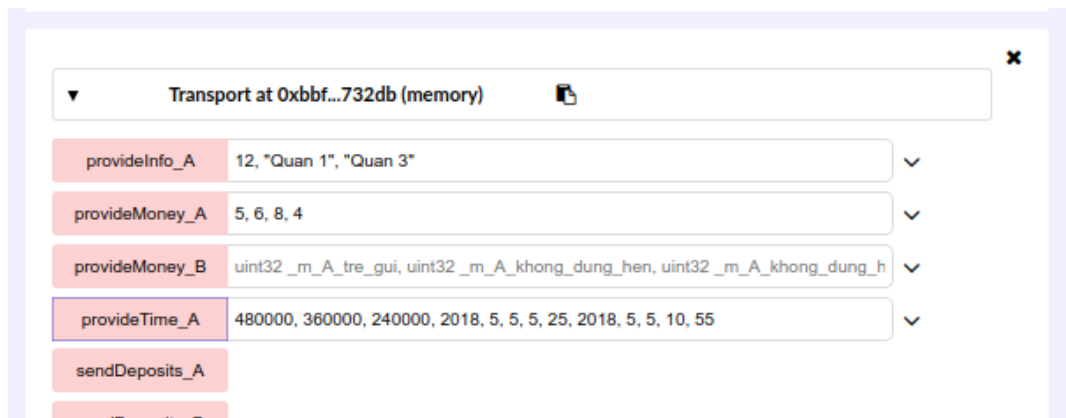




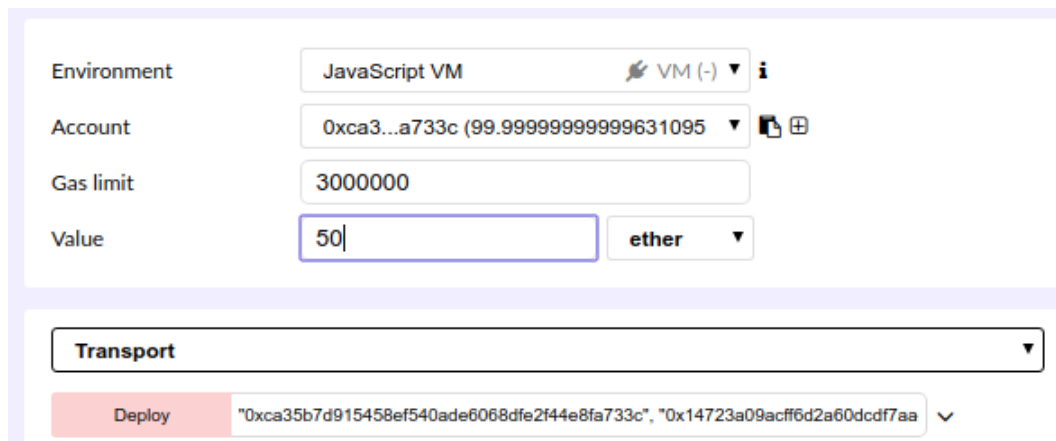
Hình 3: Môi trường chạy và các tài khoản tự sinh



Hình 4: Deploy smart contract DateTime



Hình 5: Deploy smart contract Transport



Environment: JavaScript VM VM (-) i

Account: 0xca3...a733c (99.99999999999631095) [icon] [icon]

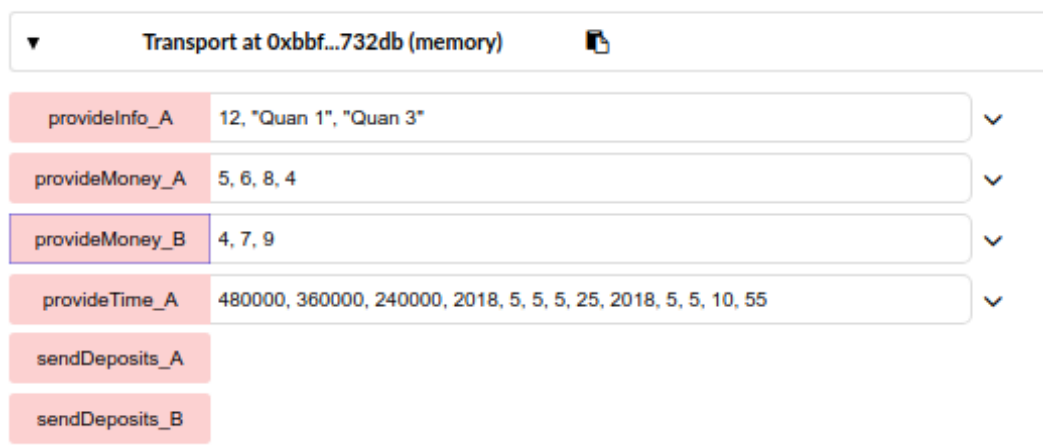
Gas limit: 3000000

Value: 50 ether

Transport

Deploy "0xca35b7d915458ef540ade6068dfe2f44e8fa733c", "0x14723a09acff6d2a60dcd77aa" ✓

Hình 6: Thực thi các hàm bên A



Transport at 0xbbf...732db (memory) [icon]

provideInfo\_A 12, "Quan 1", "Quan 3" ✓

provideMoney\_A 5, 6, 8, 4 ✓

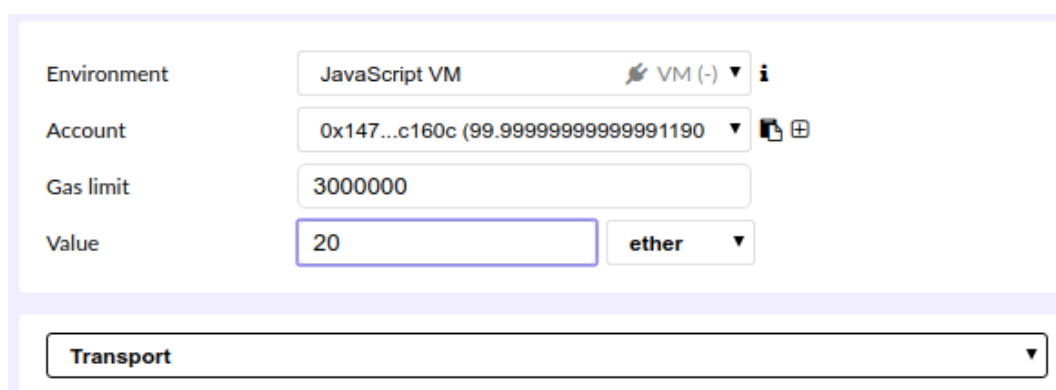
provideMoney\_B 4, 7, 9 ✓

provideTime\_A 480000, 360000, 240000, 2018, 5, 5, 5, 25, 2018, 5, 5, 10, 55 ✓

sendDeposits\_A

sendDeposits\_B

Hình 7: A gửi ether vào smart contract



Environment: JavaScript VM VM (-) i

Account: 0x147...c160c (99.9999999999991190) [icon] [icon]

Gas limit: 3000000

Value: 20 ether

Transport

Hình 8: Thực thi các hàm bên B

✕

▼ Transport at 0xbbf...732db (memory) 📄

provideInfo_A	12, "Quan 1", "Quan 3"	▼
provideMoney_A	5, 6, 8, 4	▼
provideMoney_B	4, 7, 9	▼
provideTime_A	480000, 360000, 240000, 2018, 5, 5, 5, 25, 2018, 5, 5, 10, 55	▼
sendDeposits_A		
sendDeposits_B		
trigger_A_nhan_hang_B_giao	uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute, bool hang_on	▼
trigger_B_nhan_hang_de_gui	uint16 year_B, uint8 month_B, uint8 day_B, uint8 hour_B, uint8 minute_B, uint16 ye	▼
a_A_gui		
O: string: Quan 1		
a_A_nhan		
O: string: Quan 3		
addr_A		
O: address: 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c		
addr_B		
O: address: 0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C		

Hình 9: B gửi ether vào smart contract

checkCurrentBalance	O: uint256: 7000000000000000000000
dt_A_tre_gui	O: uint32: 480000
dt_B_tre_gui	O: uint32: 240000
dt_B_tre_nhan	O: uint32: 360000
m_A	O: uint256: 5000000000000000000000
m_A_khong_dung_hang	O: uint256: 9
m_A_khong_dung_hen	O: uint256: 7
m_A_tre_gui	O: uint256: 4
m_B	O: uint256: 2000000000000000000000

Hình 10: Các thông tin đầu vào smart contract (1)

m_B	O: uint256: 20000000000000000000
m_B_hang_hong	O: uint32: 4
m_B_huy	O: uint32: 0
m_B_khong_hang	O: uint32: 8
m_B_tre	O: uint32: 5
m_B_tre_gui	O: uint32: 6
p_info	O: uint8: 12
t_A_nhan	O: uint256: 1525517700
t_B_nhan	O: uint256: 1525497900

Hình 11: Các thông tin đầu vào smart contract (2)

trigger_A_nhan_hang_B_giao	uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute, bool hang_on	▼
trigger_B_nhan_hang_de_gui	2018, 5, 5, 5, 30, 2018, 5, 5, 5, 0, 12	▼

Hình 12: Các thông tin đầu vào smart contract (3)

checkCurrentBalance	O: uint256: 69999999999999999995
---------------------	----------------------------------

Hình 13: Thực thi khi B đến nhận hàng

m\_B

0: uint256: 19999999999999999995

---

*Hình 14: Kiểm tra tài khoản của B*

## 7 Nhận xét và kết luận

## Tài liệu

- [LL14] A Very Rough Introduction to Linear Logic, John Wickerson, Imperial College London, January 7, 2014
- [LL13] What about Linear Logic in Computer Science?, Daniel Mihályi, Valerie Novitzká, Department of Computers and Informatics, Technical University of Košice Košice, Slovakia
- [BF05] A Linear-Logic Semantics for Constraint Handling Rules, Hariolf Betz and Thom Frühwirth, Faculty of Computer Science, University of Ulm, Germany
- [Gir95] LINEAR LOGIC : ITS SYNTAX AND SEMANTICS, Jean-Yves Girard Laboratoire de Mathématiques Discretes, UPR 9016 – CNRS
- [Gom+17] CREATING A TOKENIZED FUND IN THE ETHEREUM BLOCKCHAIN, Luis Gomez Quintana, November 2017 International Business



## Phụ lục