TRƯỜNG ĐẠI HỌC BÁCH KHOA TP HCM
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

- - - - - - - - - - - -

MÔ HÌNH HÓA TOÁN HỌC (CO2011)

Báo cáo 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: Nguyễn Minh Thám - 1613166
Nguyễn An Khang - 1611508
Huỳnh Công Thức - 1613494
Nguyễn Huỳnh Thoại - 1613379

Tp. Hồ Chí Minh, Tháng 05/2018

Mục lục

1 Giới thiệu về smart contract

1.1 Lịch sử smart contract

Năm 1994 cụm từ "Hợp đồng thông minh" được giới thiệu bởi nhà khoa học máy tính Nick Szabo. Mãi đến năm 1996 ông cùng nhà mật mã người Mỹ mới công bố khái niệm cho cụm từ trên. Theo ông, các điều khoản được qui định dưới dạng kỹ thuật số và hợp đồng thông minh là một loại giao thức giao dịch trên máy tính thực hiện các điều khoản trong hợp đồng.
Công nghệ này đi trước thời đại gần 1 thập kỉ. Sau ngần ấy năm người ta mới xem xét lại và thấy được tiềm năng của loại hợp đồng này bởi lẽ lúc bấy giờ không có đủ công nghệ và môi trường để sử dụng như hiện nay. Ví dụ Blockchain là môi trường hoàn hảo để áp dụng các hợp đồng thông minh. Mãi sau này tiền ảo ra đời thì người ta mới bắt đầu nghĩ đến việc cần một loại hợp đồng để quản lý trong một môi trường minh bạch như blockchain. Sự trỗi dậy mạnh mẽ bắt đầu khi Ethereum xuất hiện và phổ biến hợp đồng thông minh đến mọi người như một phương thức mới để thiết lập hợp đồng.

1.2 Ứng dụng smart contract

Nick Szabo khẳng định hợp đồng của ông - "thông minh" bởi vì chúng có nhiều chức năng và ưu điểm hơn hợp đồng giấy về bảo mật, tính toàn vẹn và minh bạch. Hợp đồng thông minh có nhiều ứng dụng trong đời sống hiện đại.

- Chẳng hạn có thể kể đến việc tự động ký giấy tờ, một người khi mất có nhiều giấy tờ như di chúc, giấy ủy quyền cần phải ký xác minh. Lúc này hợp đồng thông minh tỏ ra rất hữu dụng, nó sẽ thay người đó ký các giấy tờ đó và đảm bảo các điều khoản trong giấy tờ nguyên bản không sửa chữa đảm bảo quyền lợi cho các cá nhân liên quan.

- **Ứng dụng trong hợp đồng:** Một ứng dụng khác như hợp động thuê mướn đảm báo chắc chắn 2 bên sẽ được hưởng những điều khoản đã cảm kết trước đó.Như hợp đồng thuê nhà, người thuê nhà phải mã code từng có giá trị sử dụng nhất định để vào được nhà code này được cung cấp bởi "Hợp đồng thông minh" khi người đó đã thanh toán đủ số tiền theo điều khoản mà bên cho thuê đã đề ra. Tránh tính trạng bên thuê không trả tiền nhà hoặc bên cho thuê hủy hợp đồng. Như vậy kết hợp với công nghệ blockchain có thể ứng dụng trong nhiều lĩnh vực tài chính, bảo hiểm, ngân hàng, bầu cử... Tương tự với mỗi lĩnh vực thì sẽ có các loại hợp đồng khác nhau như hợp đồng cho thuê, hợp đồng vay mượn, hợp đồng trả lương mà nhóm đang tiến hành.

- **Ứng dụng trong bầu cử:** Trong việc bầu cử để đảm bảo tính trung thực cho số phiếu bầu cử của các chủ thể có liên quan trong hợp đồng. Đảm bảo các số liệu nhập từ các nguồn khi hoàn thành thì không thể sửa đổi. Tránh sự nhúng tay của tác nhân con người làm thay đổi kết quả.

- **Ứng dụng trong lĩnh vực tiền ảo:** Một ứng dụng nữa mà nhóm muốn nói thêm trong lĩnh vực tiền mã hóa hay tiền ảo đó là phát hành ICO (Initial Coin Offering). Giai đoạn ICO có nghĩa nhà đầu tư sẽ bỏ tiền đầu tư vào một dự án mà họ cho rằng có tiềm năng, dự án này sẽ đưa lại lượng coin (token tương ứng tỉ lệ quy đổi lúc đó). ở đây vai trò của hợp đồng thông minh đảm bảo tỉ lệ chuyển đổi là tương đương. Ngoài ra có thể bổ sung nhiều điều kiện để đảm bảo chuyển đổi thu lại được lượng coin nhiều vào thời điểm tùy ý mà nhà đầu tư dự đoán là tỉ lệ chuyển đổi là thấp nhất.

2 Linear logic

2.1 Lịch sử linear logic

Linear logic được giới thiệu bởi Jean-Yves Girard vào năm 1987. Và đã thu hút rất nhiều sự chú ý từ các nhà khoa học máy tính, vì linear logic là một cách hợp lý để xử lý tài nguyên và kiểm soát nguồn lực. Ý tưởng cơ bản của Linear logic là xem các công thức như một tài nguyên, có thể tiêu thụ hoặc sản xuất. Thay vì thay thế logic cổ điển, thì linear logic thể hiện sự chọn lọc từ logic cổ điển.

2.2 Ứng dụng linear logic

Một số ứng dụng của linear logic trong khoa học máy tính như:

- Ứng dụng trong việc kiểm chứng (verification) và đặc tả (specification). Linear logic cung cấp một số cách giải quyết vấn đề một cách trực quan hơn số với nhiều ngôn ngữ logic khác.
- Miễn là các phép "of course" ! và "why not" ? không có liên quan, chúng ta có thể kiểm soát sự chuẩn hóa (normalization) một cách rõ ràng hơn. Do đó, hiệu suất của chương trình sẽ được cải thiện hơn.
- Trong cơ sở dữ liệu, việc sử dụng logic cổ điển để biểu diễn các lý luận thành biểu thức logic thì không hợp lý so với thực tế. Trong khi linear logic biểu diễn được vấn đề này hợp lý hơn. Một trong những ví dụ cho thấy điều này là:

  Linear logic có thể biểu diễn được việc chúng ta mất chi phí nào đó để thực hiện được vấn đề nào đó. Còn logic cổ điển thì không thể hiện được vấn đề này.

2.3 Các nguyên lý cơ bản của linear logic

2.3.1 Hàm tuyến tính

Trong logic cổ điển, nếu chúng ta có A và cố A -> B thì thì chúng ta sẽ có được cả A và B(không thích hợp trong thực tế).
Trong Linear Logic, một hành vi trao đổi trong thực tế sẽ được biểu diễn bởi hàm tuyến tính: $A \multimap B$(mất A sẽ có được B).
Ví dụ 2.3.1: Nếu ông A có $1 thì sẽ mua được một cây kem được diễn đạt bằng:

$$\$1 \multimap cream$$

(Nếu ông A có $1 thì ông A sẽ có được một cây kem và không còn $1 nữa)

2.3.2 Các phép hội tuyến tính

Trong linear logic, có hai phép hội tuyến tính đó là $\otimes$(times) và $\&$(with). Cả hai ứng với hai cách sử dụng khác nhau cho cùng quan hệ và(And). Phép $\otimes$ ứng với phép hội($\wedge$)trong logic cổ điển. Nếu chúng ta viết $A \otimes B$ thì có nghĩa là có cả A và B. Phép $\&$ được xem như phép chọn , chỉ được chọn 1 trong 2(không được chọn cả 2). Nếu chúng ta viết là $A \,\&\, B$ thì có nghĩa là có thể chọn A hoặc có thể chọn B.
Ví dụ 2.3.2a: Nếu ông A có $1 thì sẽ mua được một cây kem và một viên kẹo được diễn đạt bằng:

$$\$1 \multimap cream \otimes candy$$

Ví dụ 2.3.2b: Nếu ông A có $1 thì sẽ mua được một cây kem. Nếu ông A có $1 thì sẽ mua được một viên kẹo được diễn đạt bằng:

$$\$1 \multimap cream \,\&\, candy$$

2.3.3 Các phép tuyển tuyến tính

Trong linear logic, có hai phép tuyển tuyến tính đó là $\oplus$(plus) và $\parr$(par). Phép $\oplus$ là phép đối ngẫu của phép $\&$. Nếu chúng ta viết $A \oplus B$ có nghĩa là hoặc A hoặc B nhưng quyền quyết định không phải do ta. Phép $\parr$ là phép đối ngẫu của phép $\otimes$. Nếu chúng ta viết $A \parr B$ có nghĩa là nếu không có A thì có B(buộc lựa chọn A hoặc B nhưng không thể không chọn hoặc chọn cả hai). Ta có phép phủ định của A trong Linear Logic là $A^{\perp}$. Do đó phép tuyển $A \parr B$ có thể được viết lại là $A^{\perp} \multimap B$ hoặc là $B^{\perp} \multimap A$.
Ví dụ 2.3.3a: Nếu ông A có $1 thì sẽ mua được một cây kem kèm theo một viên kẹo vào mùa xuân hoặc một cái bánh vào mùa thu được diễn đạt bằng:

$$\$1 \multimap cream \otimes (candy \oplus cake)$$

Ví dụ 2.3.3b: Nếu ông A có $1 thì sẽ mua được một cây kem được diễn đạt bằng:

$$\$1^{\perp} \parr cream$$

2.3.4 Phép phủ định và khẳng định tuyến tính

Phép phủ định tuyến tính A là đối ngẫu của A(kí hiệu $A^{\perp}$), ta có thể hiểu theo một cách trực quan là: "Một thứ gì đó tiêu thụ A".
Phép phủ định tuyến tính rất quan trọng, vì nó gắn liền với tất cả các hằng, các phép toán, lượng từ trong linear logic.
Tính chất:

$$(A^{\perp})^{\perp} = A$$

Ví dụ 2.3.4a: Ta có công thức sau:

$$A \otimes A^{\perp} \multimap 1$$

(Nếu chúng ta có tài nguyên A và ta có cái gì đó tiêu thụ A thì ta sẽ không còn tài nguyên nào, hằng 1 biểu thị sự vắng mặt tài nguyên)

Trong logic cổ điển ta đã làm quen với phép toán $\top$: "tautology", là hằng đúng, biểu diễn sự khẳng định giả thuyết đúng đắn nào đó, ví dụ $\top = A \vee \neg A$.
Trong logic tuyến tính, hằng số $\top$ biểu thị mục tiêu sử dụng hết mọi tài nguyên, hay là một cái " thùng chứa những tài nguyên không để mất tới" .
Ví dụ 2.3.4b: Nếu ông A có $1 thì sẽ mua được một cây kem và một viên kẹo được diễn đạt bằng:

$$\$1 \multimap cream \otimes candy$$

(Nếu có $1 thì Cream và Candy được mua cùng lúc)

Giả sử nếu ta có $1 và ta chỉ để ý đến Cream mà không cần Candy thì ta phải biểu diễn như thế nào?

$$\$1 \otimes (\$1 \multimap cream \otimes candy) \nvdash Cream$$

Để biểu diễn ngữ nghĩa cho việc mua Cream không cần quan tâm đến Candy thì ta viết như sau:

$$Cream \otimes \top$$

Như vậy ta có được biểu thức đầy đủ nhất là:

$$\$1 \otimes (\$1 \multimap cream \otimes candy) \vdash Cream \otimes \top$$

2.3.5 Các phép toán "of course" ! , "why not" ?

Ở những ví dụ trước về hàm tuyến tính, nếu ta có $1 thì ta có thể mua một cây kem, và quá trình ấy chỉ diễn ra đúng một lần vì $1 đã tiêu hao đi. Đặt lại vấn đề nếu họ có rất nhiều $1 ấy đến mức vô tận, hiển nhiên thì sẽ mua được kem nhiều lần. Phép ! dùng để diễn đạt sự dồi dào và lặp lại của nguồn lực nào đó một cách vô tận, hay nói cách khác là "sản xuất vô tận tài nguyên A(kể cả 0)".

Ta có biểu thức sau:

$$A \to B = (!A) \multimap B$$

(Theo logic cổ điển nếu chúng ta có A $\to$ B và có A thì sẽ có được B và A vẫn còn, điều này tương đương logic tuyến tính đó là nếu ta có A lặp lại vô tận thì sẽ có được B và vẫn còn A)

$$!(A \,\&\, B) = !A \otimes !B$$

(Nếu có sự lựa chọn A hoặc B được lặp lại vô tận cũng đồng nghĩa ta sẽ có được A vô tận và B vô tận)

Ví dụ 2.3.5a : Anh ta có nhiều hơn $1 và sau khi mua xong kem rồi thì anh ta vẫn còn nhiều $1 đô khác trong ví, anh ta có thể mua tiếp nếu thích:

$$!\$1 \multimap cream.$$

Phép toán ? là đối ngẫu của phép toán !, hay nói cách khác là $?A$ là tiêu thụ vô hạn tài nguyên A(bao nhiêu cũng được, kể cả 0).

Ví dụ 2.3.5b : Ta có biểu thức sau:

$$?Cream = (!Cream)^{\perp}$$

(nghĩa là nếu có một cái gì đó tiêu thụ Cream được sản xuất vô hạn, thì cũng đồng nghĩa với việc tiêu thụ vô hạn)

3 Đặc tả smart contract bằng linear logic

3.1 Mô tả ngữ cảnh cho smart contract

Công việc bán thời gian là loại hình công việc việc đang rất phổ biến trong xã hội hiện nay. Tuy nhiên, có một vấn đề nan giải hiện nay đó là việc chủ thuê không thanh toán cho nhân viên bán thời gian khi họ đã hoàn thành công việc. Thực trạng này đã gây ra nhiều bức xúc trong cộng đồng. Do đó, để giải quyết vấn đề trên, nhóm em đã đưa ra ý tưởng để giải vấn đề trên như sau: người chủ muốn nhân viên làm việc thì phải chuyển tiền lương trước cho nhân viên thông qua một smart contract . Khi tiền đã được chuyển đủ thì nhân viên bắt đầu thực hiện công việc. Nếu

nhân viên hoàn thành công việc thì smart contract sẽ chuyển số tiền lương đang giữ cho nhân viên đó. Ngược lại, nhân viên không hoàn thành thì số tiền sẽ được trả về lại tài khoản của người chủ.

Tình huống cụ thể như sau:

Một người chủ quán coffee ở xa muốn thuê nhân viên làm việc bán thời gian cho quán coffee của mình. Quán coffee này có hệ thống giám sát nhân viên bằng việc bằng việc điểm danh qua vân tay giữa đầu buổi và cuối buổi, và thông qua hệ thống camera cài đặt. Bất cứ ai thỏa yêu cầu về độ tuổi, ngoại hình, và muốn làm việc có thể liên lạc với ông chủ và sau khi thỏa thuận, giữa hai người sẽ thanh toán tiền lương thông qua hệ thống smart contract mà chúng em xây dựng. Quá trình này sẽ diễn ra cụ thể như sau:

- Người A muốn làm việc cho ông chủ B trong thời gian 1 tuần (từ ngày 30-04-2018 đến ngày 06-05-2018) và phải đặt cọc 100 000 VNĐ cho ông chủ B thông qua smart contract. Lịch đăng ký thời gian làm việc như sau:
  - Ngày 30-04: làm việc từ 7h tới 11h.
  - Ngày 01-05: làm việc từ 15h tới 21h.
  - Ngày 02-05: làm việc từ 7h tới 11h.
  - Ngày 03-05: làm việc từ 18h tới 22h.
  - Ngày 04-05: làm việc từ 13h tới 17h.
- Lương của nhân viên sẽ được tính theo giờ, với 15.000 VNĐ / 1 giờ. Hơn thế nữa, nếu nhân viên A làm việc vào những ngày nghỉ lễ thì lương sẽ tăng lên là 20.000 VNĐ / 1 giờ. Do đó, lương của nhân viên A là $15\,000 \times 12 + 20\,000 \times 10 = 380\,000$ VNĐ. Trước 9h ngày 29-04-2018, ông chủ B phải chuyển tiền lương trước cho nhân viên A thông qua smart contract. Số tiền chuyển không thể nhỏ hơn tiền lương được tính theo giờ trong tuần của nhân viên A. Vào đúng lúc 9h ngày 29-04-2018, nếu ông chủ B chưa chuyển đủ tiền lương cả tuần cho nhân viên cho hệ thống smart constract thì hợp đồng bị hủy, nhân viên không cần đi làm.
- Mỗi buổi làm việc, nhân viên phải điểm danh vào đầu buổi và cuối buổi để hệ thống smart constract tính giờ làm việc của nhân viên
- Vào đúng 9h ngày 07-05-2018:
  - Nếu nhân viên A đi làm đầy đủ và đúng thời gian đã thỏa thuận (tức là nhân viên A đi làm 100% số thời gian đã đăng ký lúc đầu), thì smart contract sẽ chuyển tiền lương do B gửi và tiền đặt cọc của A vào tài khoản của nhân viên A.
  - Nếu nhân viên A đi làm từ 80% đến nhỏ hơn 100% số thời gian đã đăng ký lúc đầu, thì smart contract sẽ chuyển tiền lương do B gửi vào tài khoản của nhân viên A và tiền đặt cọc của A vào tài khoản của ông chủ B.
  - Nếu ông chủ B tự ý chấm dứt hợp đồng (tức là cho nhân viên A nghỉ việc), thì smart contract sẽ chuyển tiền lương do B gửi và tiền đặt cọc của A vào tài khoản của nhân viên A.
  - Nếu nhân viên A tự ý chấm dứt hợp đồng (tức là nhân viên A đi làm dưới 80% thời gian đã đăng ký lúc đầu), thì smart contract sẽ chuyển tiền đặt cọc của A và tiền đặt cọc của B vào tài khoản của ông chủ B.

3.2 Mô tả ngữ cảnh dưới dạng các điều khoản

- **Article 1:** Trước 9h này 29-04-2018, ông chủ B phải chuyển đủ tiền lương trong tuần (là 380 000 VNĐ) đến smart contract. Số tiền sẽ bị smart contract giữ lại.
- **Article 2:** Trước 9h ngày 29-04-2018, nhân viên A phải chuyển đủ tiền đặt cọc (100 000 VNĐ) đến smart contract để được làm việc. Số tiền sẽ bị smart contract giữ lại.
- **Article 3:** Tại thời điểm 9h ngày 29-04-2018, nếu smart contract không nhận đủ tiền lương từ bên B (là 380 000 VNĐ) hay tiền đặt cọc từ bên A (là 100 000 VNĐ) thì hợp đồng sẽ bị hủy. Khi đó, smart contract sẽ chuyển số tiền đã nhận được của bên A vào tài khoản bên A và số tiền đã nhận được của bên B vào tài khoản bên B.
- **Article 4:** Vào 9h ngày 07-05-2018, nếu thời gian làm việc của nhân viên A bằng 100% thời gian làm việc đã đăng ký, thì smart contract sẽ chuyển tiền lương do B gửi và tiền đặt cọc của A vào tài khoản của nhân viên A.
- **Article 5:** Vào 9h ngày 07-05-2018, nếu nhân viên A đi làm từ 80% đến nhỏ hơn 100% số thời gian đã đăng ký lúc đầu, thì smart contract sẽ chuyển tiền lương do B gửi vào tài khoản của nhân viên A và tiền đặt cọc của A vào tài khoản của ông chủ B.
- **Article 6:** Vào 9h ngày 07-05-2018, nếu nhân vân A tự ý chấm dứt hợp đồng (tức là nhân viên A đi làm dưới 80% thời gian đã đăng ký lúc đầu), thì smart contract sẽ chuyển tiền đặt cọc của A và tiền đặt cọc của B vào tài khoản của ông chủ B.
- **Article 7:** Vào 9h ngày 07-05-2018, nếu ông chủ B đã tự ý hủy hợp đồng thì smart contract sẽ chuyển tiền lương do B gửi và tiền đặt cọc của A vào tài khoản của nhân viên A.

3.3 Đặc tả ngữ cảnh bằng linear logic

- Ông chủ B phải chi 15 000 VNĐ hoặc 20 000 VNĐ cho một giờ làm việc (hour) của nhân viên tùy theo ngày thường hay ngày lễ:

$$15000 \;\&\; 20000 \multimap hour$$

- Nhân viên A sẽ đặt cọc 100 000 VNĐ để được làm việc trong 1 tuần (week):

$$100000 \multimap week$$

- Ông chủ B phải chi 380 000 VNĐ cho một tuần làm việc của nhân viên:

$$380000 := 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 15000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes 20000 \otimes$$

- Khi hợp đồng thành công và nhân viên A làm đủ 100% số giờ đã đăng ký (100times_working), thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương và 100 000 VNĐ tiền đặt cọc vào trong tài khoản của nhân viên A(walletA) :

$$100times\_working \otimes (walletA + 380000 + 100000)$$

- Khi hợp đồng thành công và nhân viên A làm từ 80% đến nhỏ hơn 100% số giờ đã đăng ký (80times_working), thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương vào trong tài khoản của nhân viên A(walletA) và chuyển 100 000 VNĐ tiền đặt cọc vào tài khoản của ông chủ B(walletB):

$$80times\_working \otimes (walletA + 380000) \otimes (walletB + 100000)$$

- Khi ông chủ B hủy hợp đồng (cancel(contract,B)), thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương và 100 000 VNĐ tiền đặt cọc vào trong tài khoản của nhân viên A(walletA) :

$$cancel(contract, B) \otimes (walletA + 380000 + 100000)$$

- Khi nhân viên A hủy hợp đồng (cancel(contract,A)) tức là nhân viên A làm dưới 80% số giờ đã đăng ký, thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương và 100 000 VNĐ tiền đặt cọc vào trong tài khoản của ông chủ B(walletB) :

$$cancel(contract, A) \otimes (walletB + 380000 + 100000)$$

- Vào đúng lúc 9h ngày 07-05-2018 (time2), nếu nhân viên A làm từ 80% đến nhỏ hơn 100% số giờ đã đăng ký (80times_working) và smart contract nhận đầy đủ tiền đặt cọc của A và tiền lương do B gửi, thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương vào trong tài khoản của nhân viên A(walletA) và chuyển 100 000 VNĐ tiền đặt cọc vào tài khoản của ông chủ B(walletB):

$$time2 \otimes 100000 \otimes 380000 \otimes 80times\_working \multimap (walletA + 380000) \otimes (walletB + 100000)$$

- Vào đúng lúc 9h ngày 07-05-2018 (time2), nếu nhân viên A làm đủ 100% số giờ đã đăng ký (100times_working)và smart contract nhận đầy đủ tiền đặt cọc của A và tiền lương do B gửi, thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương và 100 000 VNĐ tiền đặt cọc vào trong tài khoản của nhân viên A(walletA) :

$$time2 \otimes 100000 \otimes 380000 \otimes 100times\_working \multimap walletA + 380000 + 100000$$

- Vào đúng lúc 9h ngày 07-05-2018 (time2), nếu nhân viên A hủy hợp đồng (cancel(contract,A)) và smart contract nhận đầy đủ tiền đặt cọc của A và tiền lương do B gửi, thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương và 100 000 VNĐ tiền đặt cọc vào trong tài khoản của ông chủ B(walletB) :

$$time2 \otimes 100000 \otimes 380000 \otimes cancel(contract, A) \multimap walletB + 380000 + 100000$$

- Vào đúng lúc 9h ngày 07-05-2018 (time2), nếu ông chủ B hủy hợp đồng (cancel(contract,B)) và smart contract nhận đầy đủ tiền đặt cọc của A và tiền lương do B gửi, thì smart contract sẽ tự động chuyển 380 000 VNĐ tiền lương và 100 000 VNĐ tiền đặt cọc vào trong tài khoản của nhân viên A(walletA) :

$$time2 \otimes 100000 \otimes 380000 \otimes cancel(contract, B) \multimap walletA + 380000 + 100000$$

- Vào lúc 9h ngày 29-04-2018 (time1), nếu chưa có tiền lương do B gửi (380 000 VNĐ) hoặc chưa có tiền đặt cọc của A (100 000 VNĐ), thì smart contract sẽ trả tiền lương (380 000 VNĐ) về ông chủ B hoặc tiền đặt cọc (100 000 VNĐ) về nhân viên A :

$$time1 \otimes (100000 \parr 380000) \multimap (walletA + 100000) \parr (walletB + 380000)$$

- Nhân viên A và ông chủ B đều đã gửi tiền trước 9h ngày 29-04-2018, thì smart contract sẽ giữ lại và chờ đúng ngày sẽ thực thi hợp đồng:

$$time1^{\perp} \otimes 100000 \otimes 380000 \multimap (walletA + 100000)^{\perp} \otimes (walletB + 380000)^{\perp}$$

Chú ý: walletA và walletB là tài khoản của nhân viên A và ông chủ B sau khi chuyên tiền cho smart contract(tiền đặt cọc và tiền lương)

3.4 Dùng mã giả để xây dựng một smart contract

```
input : owner, staff, balance, deposit, time_working_register, time_working_actual,
timestamp_contract_1, timestamp_contract_2, deposit_money := 100000,
balance_money := 380000, percent_accept := 0.8;
output:

if Converting system time to timestamp >= timestamp_contract_1 then
if deposit = deposit_money & balance = balance_money then
if Converting system time to timestamp = timestamp_contract_2 then

if time_working_actual = time_working_register then
Transfer deposit to "staff";
Transfer balance to "staff";
end
else if cancel(staff) and time_working_actual >= percent_accept*↩
   time_working_register then
Transfer deposit to "owner";
Transfer balance to "staff";
end
else if cancel(owner) and time_working_actual >= percent_accept*↩
   time_working_register then
Transfer deposit to "staff";
Transfer balance to "staff";
end
else if cancel(staff) and time_working_actual < percent_accept*↩
   time_working_register
Tranfer deposit to "owner";
Tranfer balance to "owner";
end
else if cancel(owner) and time_working_actual < percent_accept*↩
   time_working_register
Tranfer deposit to "staff";
Tranfer balance to "staff";
end
end
else if Converting system time to timestamp < timestamp_contract_2 then
Do not anything, waiting...;
end
end
else
if balance = balance_money then
Transfer balance to "owner";
end
else if deposit = deposit_money then
Transfer deposit to "staff";
```

```
end
else
Do nothing, finish contract...;
end
end
end
else
Do not anything, waiting...;
end
```

3.5 Dùng Solidity để xây dựng một smart contract

Để xây dựng smart contract cho ngữ cảnh 3.1, nhóm chúng em sử dụng Solidity hiện thực bằng 2 file cơ bản là: **DateTime.sol**, **Contract.sol** và sử dụng **https://remix.ethereum.org/** để biên dịch.

Listing 1: Contract.sol

```
pragma solidity ^0.4.17;

import "./DateTime.sol";
contract Owner_Staff{
address public owner;
address public staff;
uint public salary_money;
uint public deposit_money;
string strtime;
uint public timestamp1;
uint public timestamp2;
DateTime dt;
uint public percent_accept;
bool public cancel_owner;
bool public cancel_staff;
uint public time_working_actual;
uint public time_working_register;

bool public k;

constructor(address addDateTime) public{
dt = DateTime(addDateTime);
owner = msg.sender;
}

function set1(uint x) public{
percent_accept = x;
}

function set2(bool x) public{
cancel_owner = x;
}

function set3(bool x) public{
cancel_staff = x;
}

function set4(bool x) public{
```

```
cancel_staff = x;
}

function set5(uint x) public{
time_working_actual = x;
}

function set6(uint x) public{
time_working_register = x;
}

function sendSalary() public payable{
if(owner == msg.sender){
salary_money += msg.value;
}
}

function createStaff() public{
staff = msg.sender;
}

function sendDeposit() public payable{
if(staff == msg.sender){
deposit_money += msg.value;
}
}

function createTimestamp1(uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute) public{
timestamp1 = dt.toTimestamp(year,month,day,hour,minute);
}

function createTimestamp2(uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute) public{
timestamp2 = dt.toTimestamp(year,month,day,hour,minute);
}


function transfer1(uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute) public payable{
if(timestamp1 == dt.toTimestamp(year,month,day,hour,minute)){
if(salary_money>0 && deposit_money>0){
k=true;
}
}
else if ((timestamp1 < dt.toTimestamp(year,month,day,hour,minute)) && (deposit_money>0) && (salary_money==0)){
staff.transfer(deposit_money);
deposit_money -= deposit_money;
}
else if((timestamp1 < dt.toTimestamp(year,month,day,hour,minute)) && (deposit_money==0) && (salary_money>0)){
owner.transfer(salary_money);
salary_money -= salary_money;
}
}

function transfer2(uint16 year, uint8 month, uint8 day, uint8 hour, uint8 minute) public payable{
if((timestamp2 == dt.toTimestamp(year,month,day,hour,minute)) && (k==true)){
```

```
if(time_working_actual == time_working_register){
staff.transfer(deposit_money);
deposit_money -= deposit_money;
staff.transfer(salary_money);
salary_money -= salary_money;
}
else if((cancel_staff == true) && (time_working_actual >= (percent_accept/100)*time_working_register)){
staff.transfer(salary_money);
salary_money -= salary_money;
owner.transfer(deposit_money);
deposit_money -= deposit_money;
}
else if((cancel_owner == true) && (time_working_actual >= (percent_accept/100)*time_working_register)){
staff.transfer(deposit_money);
deposit_money -= deposit_money;
staff.transfer(salary_money);
salary_money -= salary_money;
}
else if((cancel_staff == true) && (time_working_actual < (percent_accept/100)*time_working_register)){
owner.transfer(deposit_money);
deposit_money -= deposit_money;
owner.transfer(salary_money);
salary_money -= salary_money;
}
else if((cancel_owner == true) && (time_working_actual < (percent_accept/100)*time_working_register)){
staff.transfer(deposit_money);
deposit_money -= deposit_money;
staff.transfer(salary_money);
salary_money -= salary_money;
}
}
}
}
```

Listing 2: DateTime.sol

```
pragma solidity ^0.4.16;

contract DateTime {
struct _DateTime {
uint16 year ;
uint8 month ;
uint8 day ;
uint8 hour ;
uint8 minute ;
uint8 second ;
uint8 weekday ;
}

uint constant DAY_IN_SECONDS = 86400;
uint constant YEAR_IN_SECONDS = 31536000;
uint constant LEAP_YEAR_IN_SECONDS = 31622400;

uint constant HOUR_IN_SECONDS = 3600;
```

```
uint constant MINUTE_IN_SECONDS = 60;

uint16 constant ORIGIN_YEAR = 1970;

function isLeapYear( uint16 year ) public pure returns ( bool ) {
if ( year % 4 != 0) {
return false ;
}
if ( year % 100 != 0) {
return true ;
}
if ( year % 400 != 0) {
return false ;
}
return true ;
}

function leapYearsBefore( uint year ) public pure returns ( uint ) {
year -= 1;
return year / 4 - year / 100 + year / 400;
}

function getDaysInMonth( uint8 month , uint16 year ) public pure returns (uint8) {
if ( month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month ==
    10 || month == 12) {
return 31;
}
else if ( month == 4 || month == 6 || month == 9 || month == 11) {
return 30;
}
else if ( isLeapYear ( year )) {
return 29;
}
else {
return 28;
}
}

function parseTimestamp( uint timestamp ) internal pure returns ( _DateTime dt) {
uint secondsAccountedFor = 0;
uint buf ;
uint8 i;

// Year
dt. year = getYear( timestamp );
buf = leapYearsBefore(dt. year ) - leapYearsBefore( ORIGIN_YEAR );

secondsAccountedFor += LEAP_YEAR_IN_SECONDS * buf ;
secondsAccountedFor += YEAR_IN_SECONDS * (dt. year - ORIGIN_YEAR - buf );

// Month
uint secondsInMonth ;
for (i = 1; i <= 12; i++) {
secondsInMonth = DAY_IN_SECONDS * getDaysInMonth(i, dt.year );
if ( secondsInMonth + secondsAccountedFor > timestamp ) {
dt. month = i;
break ;
}
secondsAccountedFor += secondsInMonth ;
}
```

```
78
79 // Day
80 for (i = 1; i <= getDaysInMonth(dt.month , dt. year ); i++) {
81 if ( DAY_IN_SECONDS + secondsAccountedFor > timestamp ) {
82 dt. day = i;
83 break ;
84 }
85 secondsAccountedFor += DAY_IN_SECONDS ;
86 }
87
88 // Hour
89 dt. hour = getHour( timestamp );
90
91 // Minute
92 dt. minute = getMinute( timestamp );
93
94 // Second
95 dt. second = getSecond( timestamp );
96
97 // Day of week .
98 dt. weekday = getWeekday( timestamp );
99 }
100
101 function getYear( uint timestamp ) public pure returns ( uint16 ) {
102 uint secondsAccountedFor = 0;
103 uint16 year ;
104 uint numLeapYears ;
105 // Year
106 year = uint16 ( ORIGIN_YEAR + timestamp / YEAR_IN_SECONDS );
107 numLeapYears = leapYearsBefore( year ) - leapYearsBefore( ORIGIN_YEAR);
108 secondsAccountedFor += LEAP_YEAR_IN_SECONDS * numLeapYears ;
109 secondsAccountedFor += YEAR_IN_SECONDS * ( year - ORIGIN_YEAR - numLeapYears );
110
111 while ( secondsAccountedFor > timestamp ) {
112 if ( isLeapYear(uint16( year - 1))) {
113 secondsAccountedFor -= LEAP_YEAR_IN_SECONDS ;
114 }
115 else {
116 secondsAccountedFor -= YEAR_IN_SECONDS ;
117 }
118 year -= 1;
119 }
120 return year ;
121 }
122
123 function getMonth( uint timestamp ) public pure returns ( uint8 ) {
124 return parseTimestamp(timestamp).month ;
125 }
126
127 function getDay( uint timestamp ) public pure returns ( uint8 ) {
128 return parseTimestamp(timestamp).day ;
129 }
130
131 function getHour( uint timestamp ) public pure returns ( uint8 ) {
132 return uint8(( timestamp / 60 / 60) % 24) ;
133 }
134
135 function getMinute( uint timestamp ) public pure returns ( uint8 ) {
136 return uint8(( timestamp / 60) % 60) ;
137 }
```

```
138
139 function getSecond( uint timestamp ) public pure returns ( uint8 ) {
140 return uint8( timestamp % 60) ;
141 }
142
143 function getWeekday( uint timestamp ) public pure returns ( uint8 ) {
144 return uint8(( timestamp / DAY_IN_SECONDS + 4) % 7);
145 }
146
147 function toTimestamp( uint16 year , uint8 month , uint8 day ) public pure returns (↩
      uint timestamp ) {
148 return toTimestamp(year , month , day , 0, 0, 0);
149 }
150
151 function toTimestamp( uint16 year , uint8 month , uint8 day , uint8 hour ) public ↩
    pure returns ( uint timestamp ) {
152 return toTimestamp(year , month , day , hour , 0, 0);
153 }
154
155 function toTimestamp( uint16 year , uint8 month , uint8 day , uint8 hour , uint8 ↩
    minute ) public pure returns ( uint timestamp ) {
156 return toTimestamp(year , month , day , hour , minute , 0);
157 }
158
159 function toTimestamp( uint16 year , uint8 month , uint8 day , uint8 hour , uint8 ↩
    minute , uint8 second ) public pure returns ( uint timestamp ) {
160 uint16 i;
161 // Year
162 for (i = ORIGIN_YEAR ; i < year ; i ++) {
163 if ( isLeapYear (i)) {
164 timestamp += LEAP_YEAR_IN_SECONDS ;
165 }
166 else {
167 timestamp += YEAR_IN_SECONDS ;
168 }
169 }
170 // Month
171 uint8[12] memory monthDayCounts ;
172 monthDayCounts[0] = 31;
173 if ( isLeapYear(year)) {
174 monthDayCounts[1] = 29;
175 }
176 else {
177 monthDayCounts[1] = 28;
178 }
179 monthDayCounts[2] = 31;
180 monthDayCounts[3] = 30;
181 monthDayCounts[4] = 31;
182 monthDayCounts[5] = 30;
183 monthDayCounts[6] = 31;
184 monthDayCounts[7] = 31;
185 monthDayCounts[8] = 30;
186 monthDayCounts[9] = 31;
187 monthDayCounts[10] = 30;
188 monthDayCounts[11] = 31;
189
190 for (i = 1; i < month ; i ++) {
191 timestamp += DAY_IN_SECONDS * monthDayCounts[i - 1];
192 }
193 // Day
```

```
194 timestamp += DAY_IN_SECONDS * ( day - 1);
195 // Hour
196 timestamp += HOUR_IN_SECONDS * ( hour );
197 // Minute
198 timestamp += MINUTE_IN_SECONDS * ( minute );
199
200 // Second
201 timestamp += second ;
202
203 return timestamp ;
204 }
205 }
```

Tài liệu

[Gir87] Jean-Yves Girard. "Linear logic". In: *Theoretical Computer Science 50.1* (1987),pp. 1–101.

[Gir95] Jean-Yves Girard. "Linear logic: its syntax and semantics". In: *London Mathematical Society Lecture Note Series* (1995), pp. 1–42.

[BF05] Hariolf Betz and T Fruhwirth. ""A linear-logic semantics for CHR"". In: *1th Intl.Conf. Principles and Practice of Constraint Programming. Citeseer* (2005).

[Wiki1] Wikipedia. "*https://en.wikipedia.org/wiki/Smart_contract*", *Smart contract*, lần truy cập cuối: 11/06/2018.

[LTHDTM] Coin68. *https://coin68.com/bai-viet-noi-bat/li-giai-ve-hop-dong-thong-minh-smart-contract.html*", *Lý thuyết hợp đồng thông minh*, lần truy cập cuối: 11/06/2018.