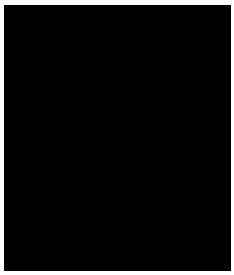
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH



MÔ HÌNH HÓA TOÁN HỌC (CO2011)

Đề bài tập lớn

"Đặc tả Smart Contract bằng Linear Logic"

GVHD: Nguyễn An Khương
Huỳnh Tường Nguyên
Trần Văn Hoài
Lê Hồng Trang
Trần Tuấn Anh

Trợ giảng: Nguyễn Trung Việt

SV thực hiện: Nguyễn Văn A – 22102134
Trần Văn B – 88471475
Lê Thị C – 36811334
Phạm Ngọc D – 97501334
Kiều Thị E – 12341334

Tp. Hồ Chí Minh, Tháng 04/2018

Mục lục

1 Giới thiệu đề tài

1.1 Giới thiệu về smart contract

1.1.1 Smart contract là gì?

Một hợp đồng thông minh (smart contract) [Gom+17],[CD16], là một giao thức quản lí hợp đồng dựa trên công nghệ blockchain. Smart contract là một bộ giao thức đặc biệt với mục tiêu là để xác nhận hay tiến hành quá trình đàm phán và thực hiện hợp đồng. Các smart contracts cho phép chúng ta triển khai giao dịch mà không cần thông qua một bên thứ ba trung gian [CM18]. Những giao dịch này dễ dàng truy vết và không thể bị can thiệp từ bên ngoài hoặc sửa đổi lại. Các smart contracts chứa trong mình toàn bộ những thông tin chi tiết về các điều khoản và thực hiện chúng một cách tự động.

Một ví dụ thực tế, nếu smart contract được ứng dụng trong ngành bảo hiểm thì những công ty bảo hiểm sẽ đưa ra những điều kiện hưởng bảo hiểm vào trong smart contract. Đầu vào sẽ là hóa đơn bệnh viện. Nếu một người mua bảo hiểm bị chấn thương và thảo các yêu cầu bảo hiểm thì smart contract sẽ thực hiện tự động và số tiền sẽ được chuyển vào tài khoản của người đó, không cần mất công làm hồ sơ yêu cầu được bồi thường.

1.1.2 Hoạt động của một smart contract

Smart contract chỉ có thể thực hiện những mã lệnh đã lập trình sẵn từ các lập trình viên. Đầu tiên, tài sản và điều khoản hợp đồng đều được mã hóa và chuyển vào một block thuộc hệ thống blockchain. Smart contract này tiếp đó sẽ được phân phối và sao chép lại bởi các node hoạt động trên nền tảng đó. Sau khi có nhận lệnh triển khai thì hợp đồng sẽ được triển khai theo đúng như điều khoản định sẵn. Đồng thời, smart contract cũng sẽ tự động kiểm tra quá trình thực hiện những cam kết nêu trong hợp đồng.

1.1.3 Cách sử dụng smart contract trong thực tế

Trong thực tế khi thực hiện và triển khai một smart contract chúng ta thường có hai giai đoạn chính:

- Giai đoạn phát triển và lập trình: Viết mã code cho smart contract bằng Solidity (ngôn ngữ chuyên dụng dành riêng cho việc viết smart contract).
- Giai đoạn triển khai: Đây là giai đoạn chúng ta sẽ triển khai smart contract lên mạng Ethereum. Có những cách sau:
  - Sử dụng trực tiếp Mist browser (là trình duyệt phục vụ riêng cho mục đích quản lý account của Ethereum)
  - Sử dụng Truffle (command truffle migrate)

Việc deploy các smart contracts rất nhanh và đơn giản. Mỗi một smart contract lúc triển khai sẽ có một địa chỉ để client có thể tương tác với nó.

1.1.4 Ứng dụng của Smart Contract

Smart contract có thể được ứng dụng trong rất nhiều lĩnh vực. Trong phần giới thiệu này chúng ta chỉ xem một số lĩnh vực đã và đang rất nhiều nghiên cứu sử dụng smart contract.

- Ứng dụng trong bảo hiểm: Các yêu cầu bảo hiểm thường rất nhiều và mất thời gian để xác thực bồi thường khách hàng. Với smart contract ta có thể thiết lập giao kèo giữa các bên tham gia để khi có sự kiện xảy ra thì việc bồi thường bảo hiểm sẽ tự động xảy ra chứ ko phải chờ làm thủ tục và duyệt bảo hiểm nữa.

- Ứng dụng trong bản quyền: người sở hữu bản quyền sẽ công khai thông tin bản quyền của người đó lên mạng blockchain, chính sách sử dụng bản quyền tác phẩm sẽ được thiết lập trong smart contract và mỗi khi tác phẩm của họ được sử dụng thì chính sách sử dụng sẽ được tuân thủ và thực hiện tức thời.

- Ứng dụng trong logistics: chuỗi cung ứng mà một hệ thống phức tạp gồm nhiều mối liên kết kế tiếp khác nhau. Mỗi liên kết cần phải nhận được xác nhận bởi cái ở trước để đủ điều kiện thực hiện phần việc của mình theo như hợp đồng. Ứng dụng 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 (Internet of Things).

1.2 Linear logic

1.2.1 Giới thiệu linear logic

Linear logic (LL) đã được giới thiệu bởi J.-Y.Girard năm 1987, dựa trên cơ sở các nghiên cứu về ngữ nghĩa của các phép toán [Gir87]. LL có thể được coi là sự tái phát triển của logic cổ điển ở chỗ nó đưa ra thêm khái niệm về tài nguyên, nguồn lực (resources) hoặc các công thức được xem như tài nguyên. Điều này đã khiến LL thu hút nhiều sự chú ý từ các nhà khoa học máy tính, vì nó là một cách hợp lý để xử lý và kiểm soát các tài nguyên, nguồn lực. LL gần với thực tế hơn và nó không giống như các mệnh đề mà ta thường sử dụng và thường được biểu diễn hành vi thực tế thông qua "consuming A yields B" ("chi tiêu A để được B"; và thường dùng ký hiệu $\multimap$, gọi là "Lollipop", với hàm ý là tuyến tính) [BF05]. Để nhận thấy sự khác biệt của của LL với các hệ thống logic khác, chúng ta có thể xem ví dụ sau (tất cả các ví dụ bên dưới được tham khảo từ [BF05], [Gir87], [Gir95]).

Ví dụ 1.1. Giả sử A là mệnh đề "*Tôi có $10*" và B là mệnh đề "*Tôi có một chiếc bánh pizza.*" Khi đó phát biểu "*nếu tôi có $10 thì tôi sẽ mua được một chiếc pizza*" có thể được viết như sau trong logic cổ điển $A \to B$. Và hiển nhiên nếu ta có cả "A *và* $A \to B$" thì trong logic cổ điển kết quả là ta sẽ có cả A và B, tức là có B (một chiếc bánh pizza) trong khi vẫn giữ được A ($10).

Điều này là hoàn hảo trong toán học, nhưng không phù hợp trong cuộc sống thực, vì trong thực tế phép kéo theo $\to$ ngụ ý quan hệ nhân quả (causal implication). Trong thực tế, một quan hệ nhân quả không thể được tái lập vì các tiền đề (premises) đã bị sửa đổi sau khi sử dụng vì A sẽ bị mất đi để nhận về B. Điều này không thể diễn tả được trong logic cổ điển. Tuy nhiên nếu ta diễn đạt theo linear logic thì: "$A \multimap B$" nói rằng tôi có thể chi tiêu $10 cho một chiếc bánh pizza, nhưng một khi tôi làm vậy tôi sẽ không còn có $10 nữa. Vì vậy LL thường biểu diễn cho một hành động thực tế hơn là một mệnh đề hình thức đơn thuần như trong logic cổ điển.

1.2.2 Các nguyên lý cơ bản của linear logic

1.2.2.a Hàm tuyến tính

LL thường được sử dụng để minh họa cho một hành vi thực tế và thường dùng ký hiệu $\multimap$ ("linear implication" - "hàm tuyến tính"). Một hành vi trao đổi trong thực tế sẽ được biểu diễn thành

biểu thức $A \multimap B$ có thể được gọi là "chi tiêu A để được B (mất đi A để có được B)". Ví dụ bên dưới sẽ minh họa điều này.

Ví dụ 1.2. Tại của hàng pizza với \$10 chúng ta có thể mua một bánh Pizza (P). Có thể hiểu là khi thực hiện giao dịch trao đổi xong, chúng ta có một bánh Pizza nhưng đồng thời chúng ta không còn \$10.

$$\$10 \multimap P.$$

1.2.2.b Các phép hội tuyến tính

Trong LL chúng ta có hai phép hội hoàn toàn khác nhau, cùng diễn tả cho quan hệ *và (and)* trong ngôn ngữ tự nhiên.

Phép hội tuyến tính - nhân (linear conjunction - times) được biểu diễn bằng ký hiệu $\otimes$ ("time" - "phép nhân") tương đương với phép hội thông thường $\wedge$ trong logic cổ điển. Do đó, biểu thức $A \otimes B$ được gọi là "cả A và B" như Ví dụ 1.3 bên dưới.

Ví dụ 1.3. Mô hình hóa thực tế là đối với \$10, chúng ta có thể mua Pizza (P) và Soda (S):

$$\$10 \multimap P \otimes S.$$

Mặc khác chúng ta cũng có một phép hội tuyến tính khác được ký hiệu là $\&$ ("with") được xem như *phép chọn* và được gọi là "chọn từ A hoặc B (nhưng không thể có cả A và B)".

Ví dụ 1.4. Với ví dụ trên là đối với \$10, chúng ta có thể mua Pizza (P) và chọn một trong hai Soda (S) hoặc Milk (M):

$$\$10 \multimap P \otimes (S \& M)$$

1.2.2.c Các phép tuyển tuyến tính

Đối ngẫu với hai phép hội tuyến tính là $\&$ và $\otimes$ thì trong LL cũng có hai phép tuyển tương ứng lần lượt là $\oplus$ và $\parr$.

Đối ngẫu với $\&$ là *phép tuyển tuyến tính - cộng* (linear disjunction - plus) biểu diễn bằng ký hiệu $\oplus$ ("plus" - "phép cộng") tương đương với phép tuyển thông thường $\vee$ trong logic cổ điển. Biểu thức $A \oplus B$ được gọi là "hoặc A hoặc B (nhưng không biết A hay B sẽ được chọn)", tức là với $\oplus$ thì chúng ta không thể chủ động lựa chọn mà "quyền chọn lựa thuộc về một người khác hoặc yếu tố khác". Do chúng ta không thể chủ động lựa chọn nên kết quả có thể dẫn đến thiếu hoặc thừa nguồn lực.

Ví dụ 1.5. Chúng ta có \$10 để mua bánh pizaa có kèm theo salad. Tuy nhiên salad đi kèm lại phụ thuộc vào mùa như: fresh garden (G) vào mùa hè và coleslaw (C) vào mùa đông. Lưu ý rằng chúng ta không thể chủ động lựa chọn được loại salad! Điều này có thể được biểu diễn như sau.

$$\$10 \multimap P \otimes (G \oplus C).$$

Phép tuyển thứ hai $\parr$ ("par") là đối ngẫu của $\otimes$ được xem như là $A \parr B$ được gọi là "nếu không A thì có B (buộc lựa chọn A hoặc B nhưng không thể không chọn hoặc chọn cả hai)". Điều này sẽ gây ra sự khó hiểu và không dễ dàng trong việc sử dụng nên thay vì vậy chúng ta có thể có thể phát biểu theo một cách khác. Đầu tiên chúng ta có định nghĩa phủ định của A được biểu diễn trong LL là $(A^{\perp})^{\perp} = A$. Khi đó với hàm tuyến tính $A \multimap B$, chúng ta có thể viết lại là $A^{\perp} \parr B$.

Ví dụ 1.6. Quay lại ví dụ về mua bánh pizza được phát biểu rằng dùng \$10 chúng ta có thể đổi được một bánh pizza (P). Dùng $\otimes$ chúng ta có thể viết lại như sau.

$$\$10^{\perp} \otimes P.$$

Ngoài ra LL còn có các phép toán khác là ! ("of course") và ? ("why not"). Sinh viên có thể tìm hiểu thêm trong [Gir87], [BF05] phần này là yêu cầu Bài tập 3.

1.3 Ví dụ mẫu về đặc tả các smart contracts bằng linear logic

1.3.1 Mô tả một ngữ cảnh thuê nhà cho smart contract

Mỗi một giao dịch thuê nhà giữa hai khác hàng cần rất nhiều thao tác thủ công. Những thao tác này nảy sinh xuất phát từ sự chưa tin tưởng nhau giữa khách hàng thuê nhà và người cho thuê. Thường thì người cho thuê sau khi nhận được tiền mới giao chìa khóa hoặc mã khóa (code) vào nhà cho khách hàng. Ngược lại khách hàng cảm thấy không yên tâm khi phải giao tiền trước khi chưa được vào nhà. Điều này sẽ được giải quyết dễ dàng khi giữa hai người có sự thiết lập thông qua một hợp đồng thông minh. Tình huống cụ thể như sau.

> "Một người cho thuê một căn hộ thông minh. Căn hộ thông minh này có tính năng khóa bằng mã khóa (code.) Điều này thuận tiện khi khách thuê nhà chỉ cần cấp một mã khóa để có thể vào ở căn hộ này và hợp đồng cho thuê thành công. Một quá trình thuê sẽ bắt đầu từ người thuê nhà sẽ lựa chọn và thuê một căn hộ thông minh với giá là 1000 \$ một tháng bắt đầu từ 8h00 AM ngày 28/04/2018. Khách thuê sẽ chuyển tiền \$ 3000 (Khách hàng này thuê 3 tháng) cho người cho thuê thông qua một smart contract. Smart contract này sẽ giữ lại số tiền của khách hàng và chờ người cho thuê chuyển lại mã khóa vào nhà. Khi người cho thuê chuyển mã khóa xong thì số tiền sẽ được chuyển vào tài khoản người cho thuê. Tuy nhiên smart contract sẽ có những ràng buộc nhất định như nếu người cho thuê gửi mã khóa trước 8h00 AM ngày 28/04/2018 thì smart contract vẫn giữ đó và chờ đến đúng 8h00 AM ngày 24/04 mới giao cho khách hàng cũng như chuyển tiền vào tài khoản của người cho thuê. Trong trường hợp bên cho thuê đến ngày hợp đồng một trong hai bên vẫn trưa hoàn tất thủ tục thông tin hoặc tiền sẽ gửi trả về bên còn lại."

1.3.2 Mô tả một ngữ cảnh thuê nhà dưới dạng các điều khoản (articles)

- Article 1. Bên A gửi tiền thuê nhà 3 tháng đến smart contract trước 8h00 AM ngày 28/04/3018. Số tiền sẽ bị smart contract giữ lại.
- Article 2. Bên B gửi mã code cho smart contract trước 8h00 ngày 28/04/2018 và bị giữ lại.
- Article 3. Đến 8h00 ngày 28/04/2018 smart contract chuyển mã khóa cho người thuê nhà và tiền đến chủ căn hộ.
- Article 4. 8h00 AM ngày 28/04/3018, Bên A gửi tiền rồi nhưng bên B vẫn không gửi mã khóa thì Bên A sẽ nhận lại được toàn bộ tiền.
- Article 5. 8h00 AM ngày 28/04/3018, Bên A chưa gửi tiền trong khi bên B đã gửi mã khóa thì thì mã khóa sẽ bị hủy và không gửi mã khóa về bên A.

1.3.3 Đặc tả ngữ cảnh thuê nhà bằng linear logic

Trong phần đặc tả này chúng ta sẽ có hai cách đặc tả, cách bên dưới là đặc tả chi tiết cho thành từng phần.

- Khách hàng phải chi \$1000 cho mỗi tháng thuê nhà (month):

$$\$1000 \multimap month.$$

- Smart contract sẽ gửi mã khóa (code) đến khách hàng thuê A trong trường hợp hợp đồng thành công:

$$A \otimes code \multimap send(code, A).$$

- Trong trường hợp hợp đồng không thành công, smart contract sẽ trả mã khóa về chủ thuê:

$$B \otimes code \multimap return(code, B).$$

- Khách thuê nhà 3 tháng với số tiền \$3000 chúng ta có thể viết lại dưới dạng sau.

$$\$3000 := \$1000 \otimes \$1000 \otimes \$1000.$$

- Khi hợp đồng thành công, smart contract sẽ tự động chuyển 3 tháng tiền nhà (tương đương \$3000) vào trong tài khoản người cho B:

$$(walletA + \$3000) \parr (walletB + \$3000).$$

- 8h00 AM, ngày 28/04/2018 (times) nếu có đầy đủ mã code và tiền thực hiện chuyển tiền cho chủ căn hộ và chuyển mã code cho khách thuê:

$$times \otimes code \otimes \$3000 \multimap (walletA - \$3000) \otimes (walletB + \$3000) \otimes send(code, A) \otimes (month \otimes month \otimes month).$$

- 8h00 AM, ngày 28/04/2018 nếu chưa có mã code hoặc tiền thì trả tiền về người thuê hoặc trả mã code cho chủ căn hộ:

$$times \otimes (code \parr \$3000) \multimap (walletA + \$3000) \parr return(code, B).$$

- Bên A và B đều gửi mã khóa trước ngày 28/04/2018 thì smart contract sẽ giữ lại và chờ đúng ngày sẽ thực thi hợp đồng:

$$times^{\perp} \otimes code \otimes \$3000 \multimap (walletB + \$3000)^{\perp} \otimes send(code, A)^{\perp}.$$

Chúng ta có thể có thể đặc tả theo một cách khác gọn hơn và thể hiện rõ được những điểm mạnh của LL như sau:

- Khách thuê nhà 3 tháng với số tiền \$3000 chúng ta có thể viết lại dưới dạng sau.

$$\$3000 := \$1000 \otimes \$1000 \otimes \$1000.$$

- Thời hạn giao nhà là: 8h00 AM, ngày 28/04/2018 ("times" - "là thời hạn thực hiện hợp đồng") nếu có đầy đủ mã code ("code" - "mã chìa khóa thông minh để vào nhà") và tiền $3000 thì smart contract thực hiện chuyển tiền cho chủ căn hộ B ("walletB" - "cộng thêm vào ví của chủ căn hộ số tiền tương ứng") và chuyển mã code cho khách thuê ("sendA" - "gửi tới khách hàng A mã code để vào nhà bằng email hoặc tin nhắn"):

$$times \otimes code \otimes \$3000 \multimap walletB \otimes sendA.$$

- Nếu đến 8h00 AM, ngày 28/04/2018 nếu chưa có mã code ("code") hoặc tiền thì smart contrac sẽ trả tiền về người thuê A ("walletA" - "cộng thêm vào ví của khách hàng số tiền tương ứng") hoặc trả mã code cho chủ căn hộ ("returnB" - gửi trả mã code về chủ căn hộ):

$$times \otimes (code \oplus \$3000) \multimap walletA \parr return(B).$$

- Bên A và B đều gửi mã code trước ngày 28/04/2018 thì smart contract vẫn giữ tiền và không gửi mã code:

$$times^{\perp} \otimes code \otimes \$3000 \multimap walletB^{\perp} \otimes sendA^{\perp}.$$

1.3.4 Dùng mã giả để xây dựng một smart contract cho ngữ cảnh thuê nhà

Trong trường hợp này các code gán và lấy dữ liệu trở nên đơn giản nên sẽ tập trung vào hàm chuyển đổi giao dịch. Trong hàm này hệ thống sẽ so sánh ngày hiện tại và ngày định trong hợp đồng nếu bằng nhau thì kiểm tra số tiền và mã code cho các bên. Nếu ngược lại còn một bên chưa gửi code hoặc tiền thì hệ thống sẽ chuyển tiền ngược về cho khách hàng và code về cho chủ thuê được minh hoạ bởi đoạn mã giả sau.

```
input : owner, customer, balance, balance, apartKeys, timestamp_contract
output:
if Converting system time to timestamp = timestamp_ contract then
    if exists(apartKeys) & balance>0 then
        Transfer money (balance) to the owner;
        Transfer code (apartKeys) to the customer;
    else
        Transfer money (balance) to the customer;
    end
end
else if Converting system time to timestamp > timestamp_ contract then
    if balance>0 then
        Transfer money (balance) to the customer;
    end
end
else
    Do not anything, waiting...;
end
```

Algorithm 1: Giải thuật giao dịch mã khoá căn hộ và tiền đến khách hàng và chủ thuê trong smart contract

1.3.5 Dùng Solidity để xây dựng một smart contract cho ngữ cảnh thuê nhà

Để xây dựng một smart contract cho ngữ cảnh 1.3.1, chúng ta sử dụng Solidity hiện thực bằng 2 file cơ bản là: **RentalApartment.sol, DateTime.sol** và sử dụng https://remix.ethereum.org/ để chạy. Vì Solidity sử dụng thời gian là *timestamp* không dùng kiểu *datetime* nên chúng ta cần thêm một contract là **DateTime.sol**.

```
pragma solidity ^0.4.17;

import "./DateTime.sol";
contract RentalApartment{

    address public owner;
    address public customer;
    uint public balance;
    uint public timestamp;
    string strtime;
    DateTime dt;
    mapping (address => string) apartKeys;

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

    function sendApartmentKey(string new_apartKey) public {
        if(owner == msg.sender){
            apartKeys[owner] = new_apartKey;
        }
    }

    function createCustomer() public{
        customer = msg.sender;
    }

    function sendBalance() public payable{
        if(customer == msg.sender){
            balance += msg.value;
        }
    }

    function getApartKey() public constant returns (string){
        return apartKeys[msg.sender];
    }

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

    function transfer(uint16 year, uint8 month, uint8 day, uint8 hour, uint8
        minute) public payable{
        if(timestamp == dt.toTimestamp(year,month,day,hour,minute)){
            bytes memory stringTest = bytes(apartKeys[owner]);
            if(stringTest.length > 0 && balance>0){
                owner.transfer(balance);
                apartKeys[customer] = apartKeys[owner];
                balance -= balance;
            }else{
                customer.transfer(balance);
                balance -= balance;
            }
```

```
54          }else{
55              if(timestamp < dt.toTimestamp(year,month,day,hour,minute) && balance >
                    0){
56                  customer.transfer(balance);
57                  balance -= balance;
58              }
59          }
60      }
61
62  }
```

Listing 1: RentalApartment.sol

```
1   pragma solidity ^0.4.16;
2
3   contract DateTime{
4
5           struct _DateTime {
6                   uint16 year;
7                   uint8 month;
8                   uint8 day;
9                   uint8 hour;
10                  uint8 minute;
11                  uint8 second;
12                  uint8 weekday;
13          }
14
15          uint constant DAY_IN_SECONDS = 86400;
16          uint constant YEAR_IN_SECONDS = 31536000;
17          uint constant LEAP_YEAR_IN_SECONDS = 31622400;
18
19          uint constant HOUR_IN_SECONDS = 3600;
20          uint constant MINUTE_IN_SECONDS = 60;
21
22          uint16 constant ORIGIN_YEAR = 1970;
23
24          function isLeapYear(uint16 year) public pure returns (bool) {
25                  if (year % 4 != 0) {
26                          return false;
27                  }
28                  if (year % 100 != 0) {
29                          return true;
30                  }
31                  if (year % 400 != 0) {
32                          return false;
33                  }
34                  return true;
35          }
36
37          function leapYearsBefore(uint year) public pure returns (uint) {
38                  year -= 1;
39                  return year / 4 - year / 100 + year / 400;
40          }
41
42          function getDaysInMonth(uint8 month, uint16 year) public pure returns (
                uint8) {
43                  if (month == 1 || month == 3 || month == 5 || month == 7 || month
                        == 8 || month == 10 || month == 12) {
44                          return 31;
45                  }
46                  else if (month == 4 || month == 6 || month == 9 || month == 11) {
47                          return 30;
```

```
48                  }
49                  else if (isLeapYear(year)) {
50                          return 29;
51                  }
52                  else {
53                          return 28;
54                  }
55          }
56
57          function parseTimestamp(uint timestamp) internal pure returns (_DateTime
                dt) {
58                  uint secondsAccountedFor = 0;
59                  uint buf;
60                  uint8 i;
61
62                  // Year
63                  dt.year = getYear(timestamp);
64                  buf = leapYearsBefore(dt.year) - leapYearsBefore(ORIGIN_YEAR);
65
66                  secondsAccountedFor += LEAP_YEAR_IN_SECONDS * buf;
67                  secondsAccountedFor += YEAR_IN_SECONDS * (dt.year - ORIGIN_YEAR -
                        buf);
68
69                  // Month
70                  uint secondsInMonth;
71                  for (i = 1; i <= 12; i++) {
72                          secondsInMonth = DAY_IN_SECONDS * getDaysInMonth(i, dt.
                                year);
73                          if (secondsInMonth + secondsAccountedFor > timestamp) {
74                                  dt.month = i;
75                                  break;
76                          }
77                          secondsAccountedFor += secondsInMonth;
78                  }
79
80                  // Day
81                  for (i = 1; i <= getDaysInMonth(dt.month, dt.year); i++) {
82                          if (DAY_IN_SECONDS + secondsAccountedFor > timestamp) {
83                                  dt.day = i;
84                                  break;
85                          }
86                          secondsAccountedFor += DAY_IN_SECONDS;
87                  }
88
89                  // Hour
90                  dt.hour = getHour(timestamp);
91
92                  // Minute
93                  dt.minute = getMinute(timestamp);
94
95                  // Second
96                  dt.second = getSecond(timestamp);
97
98                  // Day of week.
99                  dt.weekday = getWeekday(timestamp);
100         }
101
102         function getYear(uint timestamp) public pure returns (uint16) {
103                 uint secondsAccountedFor = 0;
104                 uint16 year;
105                 uint numLeapYears;
106
```

```
107                 // Year
108                 year = uint16(ORIGIN_YEAR + timestamp / YEAR_IN_SECONDS);
109                 numLeapYears = leapYearsBefore(year) - leapYearsBefore(ORIGIN_YEAR
                        );
110 
111                 secondsAccountedFor += LEAP_YEAR_IN_SECONDS * numLeapYears;
112                 secondsAccountedFor += YEAR_IN_SECONDS * (year - ORIGIN_YEAR -
                        numLeapYears);
113 
114                 while (secondsAccountedFor > timestamp) {
115                         if (isLeapYear(uint16(year - 1))) {
116                                 secondsAccountedFor -= LEAP_YEAR_IN_SECONDS;
117                         }
118                         else {
119                                 secondsAccountedFor -= YEAR_IN_SECONDS;
120                         }
121                         year -= 1;
122                 }
123                 return year;
124         }
125 
126         function getMonth(uint timestamp) public pure returns (uint8) {
127                 return parseTimestamp(timestamp).month;
128         }
129 
130         function getDay(uint timestamp) public pure returns (uint8) {
131                 return parseTimestamp(timestamp).day;
132         }
133 
134         function getHour(uint timestamp) public pure returns (uint8) {
135                 return uint8((timestamp / 60 / 60) % 24);
136         }
137 
138         function getMinute(uint timestamp) public pure returns (uint8) {
139                 return uint8((timestamp / 60) % 60);
140         }
141 
142         function getSecond(uint timestamp) public pure returns (uint8) {
143                 return uint8(timestamp % 60);
144         }
145 
146         function getWeekday(uint timestamp) public pure returns (uint8) {
147                 return uint8((timestamp / DAY_IN_SECONDS + 4) % 7);
148         }
149 
150         function toTimestamp(uint16 year, uint8 month, uint8 day) public pure
                returns (uint timestamp) {
151                 return toTimestamp(year, month, day, 0, 0, 0);
152         }
153 
154         function toTimestamp(uint16 year, uint8 month, uint8 day, uint8 hour)
                public pure returns (uint timestamp) {
155                 return toTimestamp(year, month, day, hour, 0, 0);
156         }
157 
158         function toTimestamp(uint16 year, uint8 month, uint8 day, uint8 hour,
                uint8 minute) public pure returns (uint timestamp) {
159                 return toTimestamp(year, month, day, hour, minute, 0);
160         }
161 
162         function toTimestamp(uint16 year, uint8 month, uint8 day, uint8 hour,
                uint8 minute, uint8 second) public pure returns (uint timestamp) {
```

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

Listing 2: DateTime.sol

2 Hướng dẫn và yêu cầu

2.1 Hướng dẫn

- Đọc kĩ và xử lý lại tất cả những thí dụ đã có trong file mẫu.

- Tìm hiểu kĩ cách soạn thảo văn bản bằng LaTeX.
- SV tự lập nhóm từ 4-5 SV mỗi nhóm.
- SV thảo luận và trao đổi về BTL này tại `https://ll4smartcontracts.slack.com`.
- Nhóm trưởng tạo các private channel trên Slack site ở trên và thêm các thành viên, instructors, và TA vào các channel.

2.2 Yêu cầu

- Thời gian làm bài: Từ 18/05/2018-11/06/2018. Đối với mỗi bài toán, yêu cầu sinh viên trình bày rõ ràng, sử dụng các phép toán.
- Viết báo cáo theo đúng **bố cục như trong file mẫu** bằng LaTeX.
- Mỗi nhóm khi nộp bài lên Sakai **cần phải nộp theo file log (nhật ký)** ghi rõ: tiến độ công việc, phân công nhiệm vụ, trao đổi của các thành viên,... GV sẽ đối chiếu với các thảo luận vào trao đổi trên Slack để kiểm tra tính trung thực của nội dung file nhật kí này.

2.3 Nộp bài

- SV chỉ nộp bài qua hệ thống Sakai đồng thời nộp bài qua Slack tại `https://ll4smartcontracts.slack.com`: nén tất cả các file cần thiết (file .tex, source codes, ...) thành một file tên theo mẫu là "*BTL-CO2011-TNMT-Ten nhom-(MSSV cac thanh vien).zip*" (thay "TNMT" bằng nhóm lớp tương ứng) và nộp trong mục Assignment.
- Lưu ý: mỗi nhóm **chỉ cần một thành viên là nhóm trưởng nộp bài.**

3 Đề bài

Sinh viên thực hiện một số yêu cầu sau:

Bài toán 1. Dựa vào tài liệu tham khảo, hãy trình bày chi tiết về lịch sử, ứng dụng của hợp đồng thông minh; trình bày chi tiết về lịch sử, ứng dụng của linear logic, đặc biệt là làm rõ hơn các phép toán đã nêu và trình bày chi tiết các phép toán $\bot$, $\top$, $!$, $?$ trong linear logic cùng với ví dụ minh họa cụ thể.

Bài toán 2. hãy xây dựng và mô tả một ngữ cảnh thực tế minh họa cho một smart contract có thể sử dụng linear logic để đặc tả được.

1. Mô tả một ngữ cảnh bằng lời như trong tiểu mục 1.3.1 đã trình bày.
2. Chuyển ngữ cảnh này thành các Articles (điều khoản) như trong Tiểu mục 1.3.2 đã trình bày.

Bài toán 3. Hãy dùng các kiến thức về linear logic để đặc tả ngữ cảnh trong Bài tập 2 một cách cụ thể như trong Tiểu mục 1.3.3 trình bày.

Bài toán 4. Sử dụng mã giả để lập trình smart contract minh họa cho ngữ cảnh được đưa ra trong Bài tập 2.

Bài toán 5. Sử dụng Solidity hoặc Go để lập trình smart contract minh họa cho ngữ cảnh trong Bài tập 2.

4 Cách đánh giá và xử lý gian lận

4.1 Đánh giá

Mỗi bài làm sẽ được đánh giá như sau.

Nội dung	Tỉ lệ điểm (%)
Trình bày về tổng quan về smart contract và linear logic được yêu cầu trong Bài toán 1 rõ ràng	10%
Xây dựng ngữ cảnh cụ thể tạo một smart contract	30%
Dùng linear logic để minh họa cho ngữ cảnh hợp lý	30%
Dùng mã giả để lập trình smart contract	5%
Dùng Solidity hoặc Go để hiện thực smart contract	10%
Trình bày kiến thức chuẩn bị rõ ràng, phù hợp	10%
Trình bày văn bản đẹp, đúng chuẩn	5%

4.2 Xử lý gian lận

Bài tập lớn phải được sinh viên (nhóm) TỰ LÀM. Sinh viên (nhóm) sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa các bài thu hoạch (nhất là phần kiến thức chuẩn bị). Trong trường hợp này, TẤT CẢ các bài nộp có sự giống nhau đều bị coi là gian lận. Do vậy sinh viên (nhóm) phải bảo vệ bài làm của mình.
- Sinh viên (nhóm) không hiểu bài làm do chính mình viết. Sinh viên (nhóm) có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những gì mình viết và trình bày tài liệu tham khảo cũng như trích dẫn đúng đắn.

Bài bị phát hiện gian lận thì sinh viên sẽ bị xử ý theo quy định của nhà trường.

Tài liệu

[BF05] Hariolf Betz and T Frühwirth. “A linear-logic semantics for CHR”. In: *11th Intl. Conf. Principles and Practice of Constraint Programming*. Citeseer. 2005.

[CM18] Ruzanna Chitchyan and Jordan Murkin. “Review of Blockchain Technology and its Expectations: Case of the Energy Sector”. In: *arXiv preprint arXiv:1803.03567* (2018).

[CD16] Konstantinos Christidis and Michael Devetsikiotis. “Blockchains and Smart Contracts for the Internet of Things”. In: *IEEE Access* 4 (2016), pp. 2292–2303.

[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.

[Gom+17] Luis Gomez Quintana et al. “Creating a tokenized fund in the Ethereum blockchain”. In: (2017).