软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(9):2608−2619 [doi: 10.13328/j.cnki.jos.005773]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向合同的智能合约的形式化定义及参考实现*

王璞巍[1,2], 杨航天[1], 孟 佶[1], 陈晋川[1,2], 杜小勇[1,2]

[1](中国人民大学 信息学院,北京 100872)

[2](数据工程与知识工程教育部重点实验室(中国人民大学),北京 100872)

通讯作者: 陈晋川, E-mail:jcchen@ruc.edu.cn

摘 要: 智能合约是区块链系统的核心组件,在现实中广泛应用.然而,目前没有关于智能合约的统一定义,在不同的区块链平台上,智能合约的实现也相差甚远.这样将影响公众对智能合约的认知,也对产业的发展造成障碍.回顾了智能合约的发展历史,梳理其概念的变化过程.归纳智能合约的本质,对现有智能合约的实现进行了分析和对比.给出了面向合同的智能合约的形式化定义,为智能合约的标准化奠定基础.提出了独立于区块链平台的、通用的智能合约实现方法.在目前广泛应用的联盟链区块链平台 Hyperledger Fabric 上面进行了具体实现.最后对未来工作进行了展望.

关键词: 区块链;智能合约;以太坊;超级账本

中图法分类号: TP311

中文引用格式: 王璞巍,杨航天,孟佶,陈晋川,杜小勇.面向合同的智能合约的形式化定义及参考实现.软件学报,2019,30(9):2608−2619. http://www.jos.org.cn/1000-9825/5773.htm

英文引用格式: Wang PW, Yang HT, Meng J, Chen JC, Du XY. Formal definition for classical smart contracts and a reference implementation. Ruan Jian Xue Bao/Journal of Software, 2019,30(9):2608−2619 (in Chinese). http://www.jos.org.cn/1000-9825/5773.htm

Formal Definition for Classical Smart Contracts and Reference Implementation

WANG Pu-Wei[1,2], YANG Hang-Tian[1], MENG Ji[1], CHEN Jin-Chuan[1,2], DU Xiao-Yong[1,2]

[1](School of Information, Renmin University of China, Beijing 100872, China)

[2](Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education (Renmin University of China), Beijing 100872, China)

Abstract: Smart contract is one of the key components of blockchain systems, and has been widely applied in practice. However, there are no uniform definitions for smart contract. Moreover, the implementations of smart contracts in different platforms have quite large differences. This situation will affect the public perception of smart contracts and will cause obstacles to the development of the blockchain industry. This study recalls the history of the development of smart contracts, combing out the changes of the concepts, summarizes the essence of smart contracts, and analyzes and compares existing smart contract implementations. The formal definition of classical smart contracts is proposed, which may lay the foundation for the standardization of smart contracts. A common implementation

* 基金项目: 国家重点研发计划(2016YFB1000702); 贵州财经大学与商务部国际贸易经济合作研究院联合基金(2017SWBZD08)

Foundation item: National Key Research and Development Plan Task of China (2016YFB1000702); Guizhou University of Finance and Economics and the Ministry of Commerce International Trade and Economic Cooperation Research Institute Joint Fund (2017SWBDZ08)

本文由“区块链数据管理”专题特约编辑于戈教授、牛保宁教授、金澈清教授推荐.

收稿时间: 2018-06-09; 修改时间: 2018-08-28; 采用时间: 2018-12-14; jos 在线出版时间: 2019-04-10

CNKI 网络优先出版: 2019-04-09 17:32:20, http://kns.cnki.net/kcms/detail/11.2560.TP.20190409.1732.003.html

method independent of the blockchain platforms is also given. And a reference implementation based on Hyperledger Fabric is carried out as well. Finally, the conclusion is presented and the future work is listed.

Key words: blockchain; smart contract; Ethereum; hyperledger

智能合约(smart contract)的概念由跨领域的学者 Szabo 于 1994 年提出[1].Szabo 将智能合约定义为实现合同条款的计算机程序,且在不需要可信第三方情况下,能够确保合同的正确履行.在区块链技术出来之前,智能合约并未在实际中应用,因为很难在没有可信第三方情况下保证合同的正确执行.比特币系统是第一个区块链的应用系统,也第一次实现了智能合约的理念.区块链网络所具备的去中心化、去信任以及不可篡改的特质,使其成为智能合约天然的执行平台.伴随着区块链技术向多个传统行业的渗透,智能合约也得到了广泛应用,其理念和实现技术也有了迅速的发展.与此同时,越来越多的学者认为,智能合约就是运行在区块链上的程序,不再与合同绑定.本文讨论的智能合约是指经典的定义,即,用程序定义的合同条款.

在实际应用中,可以通过智能合约防范履约风险,让合同履行实现自治,无需法院等权威机构来督促合约的执行.随着区块链技术的不断发展,智能合约被应用到不同领域.以农业保险为例,农业保险品种小、覆盖范围低,经常会出现骗保事件.将智能合约与农业保险结合之后,一旦检测到农业灾害,就会自动启动赔付流程,这样赔付效率更高[2].此外,在医疗数据共享领域,通过智能合约来规范数据的产生者(病人)、数据的保管者(医院)以及数据的使用者(数据分析公司)三方的权责利,促进医疗大数据的共享[3].

虽然智能合约已经得到了广泛应用,但一直缺乏严格的定义.目前看到,对智能合约的描述都是采用自然语言,表达上不够精确,且存在分歧.有些文献认为,智能合约应该是数字化的商务合同.商务合同是指有关各方在进行商务合作的时候需要共同遵守的协议条款,里面的协议常常涉及各种商业逻辑.例如,我们摘取了航班延误保险合同的一部分协议:“被保险人持本人购买的有效机票以乘客身份乘坐航班时,遭遇搭乘航班较预定起飞时间延误,且延误时间达到保险单所载明的时间,保险人按保险单所载明金额给付保险金”.这部分协议包含的商业逻辑是“只有当被保险人购买了机票,以乘客身份乘坐航班,且航班延误时间超过预定时间,保险人才会支付保险金”.但是,数字化商务合同的一个关键问题在于:如何形式化描述这些商业逻辑.

在实际应用中,几乎所有的区块链平台都宣称支持智能合约,但实现的方式差别较大.比特币的智能合约类似 Forth 语言[4],是基于堆栈的脚本,不支持循环,不是图灵完备.加上大小的限制,使得比特币的智能合约只能支持有限的逻辑,通常只作为账户拥有者的身份识别.以太坊的智能合约采用图灵完备的 solidity 语言来编写,合约具有状态.超级账本旗下的 Fabric(Hyperledger Fabric)平台采用 Go 语言编写智能合约,功能强大,但本身并不支持状态.对智能合约模糊的、不一致的描述,以及不同的实现,妨碍了公众对智能合约的认知,影响统一标准的制定,并阻碍行业的健康发展.

在很多应用场景中,智能合约担负商务合同的作用.此时,智能合约将面临两个问题.

1) 如何准确地描述商务合同?

2) 如何让程序员之外的人员理解?

商务合同和智能合约的鸿沟在于:前者采用自然语言,后者是计算机程序.未受过专业训练的人员,比如企业的法律顾问,无法理解程序,只能在计算机专家的帮助下对比智能合约和商务合同,这里面的难度和风险显而易见.

本文将提出面向合同的智能合约的形式化定义.该定义将基于合同的基本元素(承诺),用统一的形式化模型来描述商务合同和智能合约.这样做的优点在于:首先,给智能合约严格的、统一的定义;其次,领域专家和程序员将可以在双方都能理解的统一模型下交流,就像数据库中的概念模型.智能合约的形式化定义可以看成是面向智能合约的建模语言,方便非计算机专业的一般用户来描述他们的承诺,再从这些承诺自动地(或半自动地)转换到智能合约具体的实现语言(例如 solidity 和 go).

本文第 1 节将概述目前智能合约在学术界和工业界的进展,对比分析不同的智能合约实现.第 2 节将给出智能合约的形式化定义.第 3 节提出了通用的实现算法.第 4 节介绍我们在区块链平台 Hyperledger Fabric 中的参考实现.最后,第 5 节总结全文并展望未来工作.

1 智能合约概览

1.1 智能合约的定义

在最早的定义中,智能合约被描述为执行合同条款的计算机程序[1],或者是数字化的一组承诺[5],要求智能合约的执行无须可信的第三方.需注意,承诺是合同的基本要素.在 Wikipedia 的定义中,智能合约是一种旨在以信息化方式传播,验证或执行合同的计算机协议,允许在没有第三方的情况下进行可信交易,且这些交易可追踪且不可逆转[6].不难看出,上述对智能合约的定义,均要求合约能够实现现实中商务合同,且要求合约的执行在一个无须可信第三方的环境.我们将其称为"狭义的"或"经典的"智能合约,这也是本文讨论重点.

随着区块链的问世,人们终于找到了无中介的可信计算环境.只要网络中多数节点是正常的,那么整个网络最终的运行结果(共识)也就值得信赖.在这之后,智能合约得到了迅速发展,其概念也在发生变化.很多学者认为,智能合约就是运行在区块链上的程序.在文献[7]中,智能合约被定义为运行在区块链上的脚本,实现了多步骤过程的自动化.在文献[8]中,区块链就是可以被一个网络正确执行的计算机程序,该网络由互不信任的节点构成.Luu 等人[9]则认为:智能合约就是运行在区块链上的程序,由共识协议保证其正确执行.类似的观点还出现在文献[10−12]中.我们将这类智能合约称为"广义的"智能合约.

此外,文献[13,14]强调了智能合约是一个状态机.特别是以太坊中,智能合约的每次执行都可以看做是从一个状态转移到另一个状态[14].值得注意的是,商务合同的执行过程也可以被看做是一个状态机.一份合同的执行,正是按照预先规定的逻辑在不同状态之间转移的过程.

通过上面对智能合约的分析,我们认为智能合约是具备以下几个性质的一段程序.

1) 支持复杂的商业逻辑;
2) 支持状态机;
3) 可以自动执行;
4) 执行的结果能被参与各方认可;
5) 无须可信第三方.

1.2 智能合约的实现

比特币系统是第 1 个区块链的应用,在比特币系统中,区块链的作用仅仅是记录比特币在各个账户之间的转移.每一笔交易的输出(output)中,都需要写入一段脚本,作为该笔输出在未来被使用的条件.大多数情况下,脚本被用来检验 output 账户拥有者的身份.脚本也可以用来实现简单的逻辑,比如指定某个输出只有在区块高度达到某个阈值之后才能使用.但总的来说,比特币脚本所使用的语言较为简单,难以实现复杂的商业逻辑.此外,每个输出只能被使用一次,也不能表达状态机.

相对于比特币,以太坊平台可以支持各类智能合约.以太坊采用的语言是图灵完备的,可以支持复杂的商业逻辑.智能合约以字节码形式存储于区块中,可以被各个节点执行.合约在发布之后可以被多次调用,每次调用均需从之前的区块中读取合约的状态和代码,而调用执行的结果也会存入新的区块中.因此,智能合约在以太坊上的执行就是一个状态机,只不过这个状态机是运行在区块链这个去中心的分布式计算平台上.

超级账本是目前影响最为广泛的联盟链项目,其中最重要的系统是 Fabric.与以太坊和比特币不同的是:在 Fabric 中,智能合约(即 Fabric 中的链码,ChainCode)并不存储于区块,而是一直运行在联盟节点之上.链码采用 Go 语言,图灵完备,可以实现各种复杂的商业逻辑.但是链码没有状态.在比特币或以太坊上运行的智能合约可以看做是某个合约模板的多个实例,但是在 Fabric 中没有合约的实例,所有对区块链上数据的读写都必须经过链码来执行,类似于 MVC 模型中的 M(model).

EOS(enterprise operation system)是一款为商用分布式应用设计的区块链操作系统,旨在实现分布式应用的性能扩展.其与比特币、以太坊不同的是:EOS 采用了 DPOS(delegated proof of stake,委托权益证明)的共识机制,同时对智能合约系统进行优化,并支持多种语言编写合约,为使用者提供一个没有手续费的智能合约使用平台.

BCOS(BlockChain OpenSource)是聚焦于企业级应用服务的区块链技术开源平台.BCOS 由微众银行、万向

区块链和矩阵元三方开发,为分布式商业提供区块链技术基础设施及服务,并于 2017 年 7 月 31 日完全开源.其智能合约主要使用 solidity 语言开发,智能合约设计理念与以太坊类似.

在表 1 中,对比分析了上述的智能合约系统.可以看出以太坊,EOS,BCOS 对智能合约的实现较为完善.而比特币和 Fabric 则存在不足.在第 4 节中,我们将基于 Fabric 系统实现一个带有状态的智能合约.

Table 1 Comparison of major smart contracts

表 1 目前主流的智能合约对比分析

	支持复杂的商业逻辑	支持状态机	可以自动执行	结果被各方认可	无须权威第三方
比特币	×	×	√	√	√
以太坊	√	√	√	√	√
Fabric	√	×	√	√	√
EOS	√	√	√	√	√
BCOS	√	√	√	√	√

1.3 智能合约的安全性

智能合约的安全性一直是人们关心的问题,而由智能合约的漏洞暴露出来的安全性事件层出不穷,例如 2017 年 6 月的 TheDAO 事件[15].TheDAO 是搭建在以太坊网络上的智能合约,黑客利用递归函数的漏洞将面向公众筹集的 350 万个以太坊代币转向自己的地址,造成了巨大的经济损失.2018 年 4 月,黑客再次利用 BEC 代币的智能合约漏洞向 BEC 代币发起攻击[16],通过计算溢出,凭空产生 5.7896×10^{58} 个代币,同时,黑客将这些代币转移到交易平台进行抛售,导致 BEC 代币市值几乎归零.

目前有多个公司从事与智能合约的安全性验证工作,如链安、慢雾等.传统的安全公司也在开始进军智能合约安全领域,比如 360 公司最近发现了 EOS 平台的重大漏洞,在文献[17]中,提出了一种智能合约的形式化验证方法.

2 智能合约的形式化定义

基于以上认识,我们以承诺(commitment)作为基本元素来构建智能合约.承诺是商务合同的基本单位[18],表示一方对另一方的义务.一份合同通常由一组相互关联的承诺组成.在人工智能领域,有不少关于形式化描述商业承诺的工作[19,20].本文将承诺用于描述智能合约,根据智能合约的特点对承诺进行了重新定义,并在此基础上完成了智能合约的形式化定义.

2.1 承 诺

定义 1(承诺).承诺是一个五元组 $C(x,y,p,r,tc)$,含义是承诺人 x(promisor)向被承诺人 y (promisee)做出承诺,如果前提 p(premise)达成,就产生结果 r(result).其中,

- 前提 p 和结果 r 的值是布尔值.值为 true 表示前提已达成(或结果已完成),值为 false 表示前提未达成(或结果未完成);
- tc(time-constraints)表示该承诺的有效期,tc 为 true,承诺才会有效.

一个承诺的生命周期内可能有 5 种不同的状态.

1) 激活(act):前提 p 和结果 r 都为 false,时间未超出承诺的有效期.表示承诺有效,在等待前提 p 的达成和结果 r 的完成;
2) 就绪(bas):前提 p 为 true,结果 r 为 false,时间未超出承诺的有效期.表示承诺已生效且前提 p 已经达成,在等待结果 r 的完成;
3) 满足(sat):前提 p 和结果 r 都为 true.表示前提 p 已达成,结果 r 也已完成,承诺已被履行;
4) 过期(exp):前提 p 和结果 r 都为 false,时间已超出承诺的有效期.表示在承诺失效时,前提 p 未能达成而结果 r 也未能完成;
5) 违约(vio):前提 p 为 true,但结果 r 为 false,时间已超出承诺的有效期.表示当承诺失效时,尽管 b 已达成

了前提 p,但 a 仍然未履行其承诺完成结果 r,已经违约.

如图 1 所示:如果在创建承诺时,前提 p 和结果 r 都为 false,承诺进入激活状态,我们将这样的承诺成为有条件承诺,也就是承诺人履行该承诺存在前提条件;如果前提 p 为 true,结果 r 为 false,承诺进入就绪状态,我们将这样的承诺成为无条件承诺,也就是承诺人将无条件履行该承诺;在激活状态下,如果时间超出了承诺的有效期,前提 p 和结果 r 都未能达成(完成),承诺进入了过期状态,此时承诺已经失效;在激活状态下,如果前提 p 达成(也就是使得 p=true),该承诺变成无条件承诺,进入就绪状态;在就绪状态下,如果时间超出了承诺的有效期,结果 r 依然未能完成,承诺进入违约状态,表示承诺人已违约.在就绪状态下,在有效期内,承诺人完成了结果 r(也就是使得 r=true),进入满足状态,表示承诺人履行了其承诺.

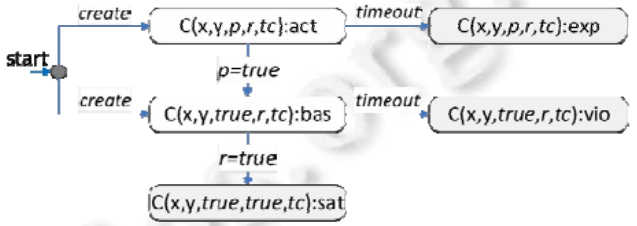

Fig.1 The lifecycle of a commitment

图 1 承诺的生命周期图

从承诺生命周期图中我们可以看到,承诺的有效期与激活状态和就绪状态有关.本文中,我们将承诺有效期(time constraints)定义如下.

定义 2(承诺有效期). 承诺有效期是一个二元组 tc:=($pdact$,$pdbas$),其中:$pdact$ 表示承诺进入激活(act)状态之后,对前提 p 的完成时间限制;$pdbas$ 表示承诺进入就绪(bas)状态之后,对结果 r 的完成时间限制.若这两个限制满足,tc 为 true;否则,tc 为 false.

例如,tc=(24,24)表示前提 p 需要在承诺进入激活状态 24h(默认单位为 h)之内达成(即 $pdact$=24),否则承诺将进入过期状态;结果 r 需要在承诺进入到就绪状态之后 24h 之内完成(即 $pdbas$=24),否则承诺进入违约状态.

在承诺中,前提或结果可能是预期的动作.例如,商家 a 向客户 b 做出一个承诺,如果 b 在系统上预存 100 元货物钱,a 就给 b 寄出相应货物.这里,前提是预存钱的动作(b 在系统上预存 100 元钱),而结果是寄货物的动作(a 给 b 寄出购买的货物).本文中,我们将动作(action)定义如下.

定义 3(动作). 一个动作表示为 $action$:=$actname$($executor$,$object$,$input$,$output$),其中:

- $actname$ 是动作的名称,$executor$ 是动作的执行者,$object$ 是动作的作用对象,$input$ 是输入参数,$output$ 是输出参数(act,$exectuor$ 是必需的(required),而 $object$,$input$ 和 $output$ 都是可选的(optional));
- 动作的值是一个布尔值,$action$=fasle 表示没有完成该动作,$action$=true 表示该动作已完成.动作的默认值为 false.

例如,$transfer$(a,b,10,$receipt$)表示 a 向 b 转账 10 元的动作,其中,$transfer$ 是动作名,a 是动作的执行者,b 是动作的作用对象,10 是输入参数,$receipt$(收据)是输出参数.$transfer$(a,b,10,$receipt$)的默认值是 false,表示 b 还没有收到 a 转过来的 10 元钱,也就是动作还未完成.如果 $transfer$(a,b,10,$receipt$)=true,则表示该动作已经完成(a 收到 b 转账过来的 10 元钱).

此时,前面提到的那个承诺就可以写为

$$C_1=C(a,b,deposit(b,s,100),send(a,b,goods),(24,24)).$$

这是一个有条件承诺,其含义是:如果客户 b 在承诺激活的 24h 内达成前提 $deposit$(b,s,100)=true,那么商家 a 就会在承诺就绪后的 24h 内完成结果 $send$(a,b,$goods$)=true.但是假设有这样一种情况:如果客户 b 在承诺 C_1 进入过期状态之后才在系统 s 上预存 100 元钱,这时候承诺 C_1 已失效(可能是商家 a 的优惠购买活动已经结束),系统 s 如何处理这一笔预存款呢?此时,系统 s 往往需要做出一个关于“承诺 C_1 过期之后如何处理客户预存款”

的承诺.可以看到,承诺的前提可能与另一个承诺的状态有关系.系统 s 可以做出如下承诺:

$$C_2=C(s,b,C_1\_exp,deposit(b,s,x),refund(s,b,x),(\infty,24)).$$

其中,

- C_1_exp 是一个布尔变量:如果承诺 C_1 处于过期(exp)状态,值为 true;否则,值为 false;
- 有效期 $tc=(\infty,24)$表示承诺在激活状态不会过期,但如果在就绪状态下超过 24h 还未完成结果,系统 s 将会违约.这个承诺的具体含义是:系统 s 向客户 b 承诺,如果 b 在承诺 C_1 过期之后还在系统上预存钱,系统将会在 24h 内退钱给 b.

这里,我们通过一个机票延误险的案例来说明我们定义的承诺.在机票延误险场景中,乘机人购买机票之后在系统上预存 10 元保费;保险公司再在系统上预存相应的赔偿金 1000 元;如果保险公司没有按时预存赔偿金,系统就直接将保费退还给用户;如果保险公司预存了赔偿金,若航班没有延误,或者延误时间少于 4h,系统就将乘机人预存的保费转账给保险公司,同时退还保险公司预存的赔偿金.如果航班延误超过 4h,系统也会将保费转账给保险公司,但是会将保险公司预存的赔偿金赔偿给乘机人.假设 a 表示保险公司,b 表示乘机人,c 表示航空公司,s 表示系统,MAX-T 表示保险最大有效期,他们的承诺如下.

- $C_3=C(b,s,\text{true},buy\_ticket(b,c,flightno),(\infty,24))$.含义是乘机人向系统承诺:将会向航空公司 c 购买航班号为 $flightno$ 的机票($buy\_ticket(b,c,5210)$).这个承诺是无条件承诺,没有前提条件,承诺初始为就绪状态.承诺有效期是(∞,24),即乘机人将在 24h 购买机票;
- $C_4=C(b,s,C_3\_sat,deposit(b,s,10),(24,0.5))$.含义是乘机人向系统承诺:如果承诺 C_3 进入满足状态(也就是按时购买了机票),他就在系统上预存 10 元保费($deposit(b,s,10)$).承诺有效期是(24,0.5),即乘机人需要在 24h 内购买机票,以及乘机人将在购买机票后 0.5h 内在系统上预存保费;
- $C_5=C(a,s,C_3\_sat,C_4\_sat,deposit(a,s,1000),(72,1))$.含义是保险公司向系统承诺:如果承诺 C_3 和 C_4 都进入了满足状态(也就是乘机人按时购买了机票并且在系统上按时预存了 10 元保费),它将在系统上预存 1000 元赔偿金($deposit(a,s,1000)$).承诺有效期是(72,1),即保险公司只在 72h 内接受乘机人预存保费,且将在乘机人购买机票和预存保费之后 1h 内在系统上预存赔偿金;
- $C_6=C(s,b,C_5\_vio,refund(s,b,10),(\text{MAX-}T,2))$.含义是系统向乘机人承诺:如果承诺 C_5 进入违约状态(也就是保险公司未能按时预存保险金),系统将向用户退还预存保费($refund(s,b,10)$).承诺有效期是(MAX-T,2),即,系统将在保险公司违约之后 2h 内退还保险金;
- $C_7=C(s,b,C_5\_satflight\_delay(c,flightno,4.0),transfer(s,a,10),compensate(s,b,1000),(\text{MAX-}T,24))$.含义是系统向乘机人承诺:如果承诺 C_5 进入满足状态(也就是保险公司预存了赔偿金),并且航空公司 c 航班号为 $flightno$ 的飞机延误 4h 以上($flight\_delay(c,flightno,4.0)$),系统将把 10 元保费转账给保险公司($transfer(s,a,10)$),同时向乘机人赔偿 1000 元($compensate(s,b,1000)$);承诺有效期是(MAX-$T$,24),即,系统将在完成投保且航班延误后 24h 内,将保费转账给保险公司和支付用户赔偿金;
- $C_8=C(s,a,C_5\_sat\neg flight\_delay(c,flightno,4.0),transfer(s,a,10),refund(s,a,1000),(\text{MAX-}T,24))$.含义是系统向保险公司承诺:如果承诺 C_5 进入满足状态(也就是保险公司预存了赔偿金),并且航空公司 c 航班号为 $flightno$ 的飞机没有延误,或延误在 4h 内($\neg flight\_delay(c,flightno,4.0)$),系统将把 10 元保费转账给保险公司($transfer(s,a,10)$),同时向保险公司退还 1000 预存赔偿金($refund(s,a,1000)$).

2.2 智能合约:有限自动机

定义 4(智能合约). 智能合约就是定义在一组承诺之上的有限自动机 $SC:=(CC,A,S,s_0,\delta,F)$,其中,

- $CC=\{C_1,C_2,\ldots,C_n\}$是一个有限的承诺集合;
- A 是这些承诺涉及到的动作的集合(包括超时动作,也就是时间超出了承诺的有效期);
- $S=\{s_0,s_1,s_2,\ldots,s_m\}$是一个有限的状态集合.状态 s_i 由 CC 中所有承诺的状态共同决定:

$$s_i=\{C_1:state,C_2:state,\ldots,C_n:state\}(state\in\{act,sat,bas,exp,vio\});$$

- s_0 是初始状态,其中,CC 中所有承诺或者处于激活状态(有条件承诺),或者就绪状态(无条件承诺);

- δ:$S\times A\to S$ 是状态变迁函数.A 中的动作会促使 CC 中承诺的状态发生变化,从而引起智能合约的状态发生变化;
- $F\in S$ 是一个有限的终止状态集合.

智能合约由一组承诺构成,例如第 2.1 节给出了机票延误险场景中的 6 个承诺 $CC=\{C_3,C_4,\ldots,C_8\}$,这些承诺里涉及的动作有:*buy_ticket*(*b*,*c*,5210)乘机人购买机票,*deposit*(*b*,*s*,10)乘机人预存延误险保费,*deposit*(*a*,*s*, 1000)保险公司预存赔偿金,*refund*(*s*,*b*,10)系统退还乘机人保费,*refund*(*s*,*a*,1000)系统退还保险公司预存赔偿金,*flight_delay*(*c*,*flightno*,4.0)航空公司航班延误超过 4h,*transfer*(*s*,*a*,10)系统将乘机人保费转账给保险公司,*compensate*(*s*,*b*,1000)系统向乘机人支付赔偿金,*timeout* 时间超过承诺有效期.

承诺是有状态的,对于智能合约的调用,会触发承诺的状态发生改变.因而,智能合约又是指定义在一组承诺之上的有限自动机.如图 2 所示,可以在这些承诺之上生成一个智能合约(有限自动机),其初始状态为 $s_0=\{C_3$:*bas*,C_4:*act*,C_5:*act*,C_6:*act*,C_7:*act*,C_8:*act*$\}$,也就是除了承诺 C_3 是就绪状态之外,其他的承诺都是激活状态.在初始状态 s_0 上,乘机人在承诺 C_3 的有效期内执行购买机票的动作 *buy_ticket*(*b*,*c*,*flightno*)之后,智能合约的状态就会变化为 $s_2=\{C_3$:*sat*,C_4:*bas*,C_5:*act*,C_6:*act*,C_7:*act*,C_8:*act*$\}$.也就是承诺 C_3 的状态从就绪(*bas*)变为满足(*sat*),承诺 C_4 的状态从激活(*act*)变为就绪(*bas*),其余承诺的状态没有变化.

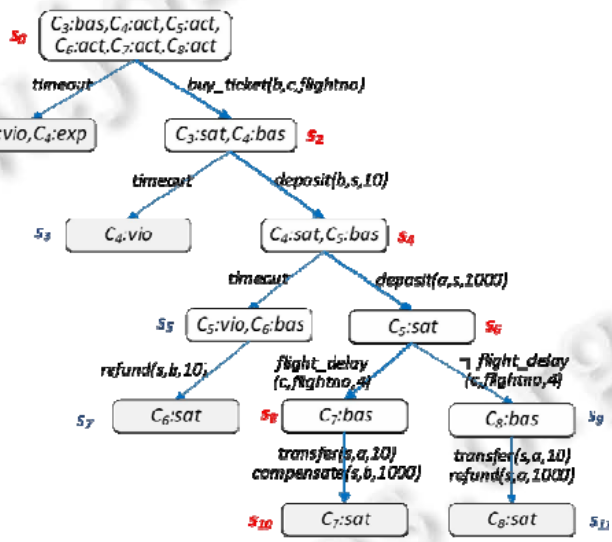

Fig.2 An example of smart contract

图 2 智能合约:有限自动机

为了简便,我们用发生了变化的那些承诺状态来表示智能合约的新状态.例如,图 2 就将 s_2 表示为 C_3:*sat*,C_4:*bas*,因为从 s_0 变化到 s_2,只有承诺 C_3 和 C_4 的状态发生变化.在 s_2 状态下,若乘机人执行预存保费的动作(*deposit*(*b*,*s*,10)),智能合约的状态就会变化为 $s_4=C_4$:*sat*,C_5:*bas*.紧接着,如果保险公司执行预存赔偿金的动作(*deposit*(*a*,*s*,1000)),智能合约的状态会变化为 $s_6=C_5$:*sat*.紧接着,如果航班真的发生了延误(*flight_delay*(*c*,*flightno*,4)),智能合约进入状态 $s_8=C_7$:*bas*.系统再将保费转账给保险公司,但将保险公司 1000 元的预存金赔偿给客户,智能合约的状态最终进入终止状态 $s_{10}=C_7$:*sat*.这就是从智能合约从初始状态到终止状态的一条路径:$s_0\to s_2\to s_4\to s_6\to s_8\to s_{10}$.根据初始状态 s_0 和这条路经,可以得出终止状态 s_{10} 的完整表示$\{C_3$:*sat*,C_4:*sat*,C_5:*sat*,C_6:*act*,C_7:*sat*,C_8:*act*$\}$,也就是承诺 C_3~C_5 和 C_7 最终进入满足状态.此外,从图 2 可以看出,这个智能合约还有多条执行路径.

3 智能合约的通用实现方法

本节讨论如何在区块链平台上实现智能合约.考虑到合约内容的多样性,要求区块链平台能够运行图灵完

备的程序.本节先介绍智能合约在区块链平台的执行过程,然后提出一种通用的算法.

3.1 智能合约在区块链平台的运行过程

如图 3 所示,合约发布到链上,有一个初始状态.每次调用(包括指令和参数)都需要读取合约的逻辑和上一个状态,执行之后将新的状态存入区块.智能合约就是运行在区块链上的状态机.

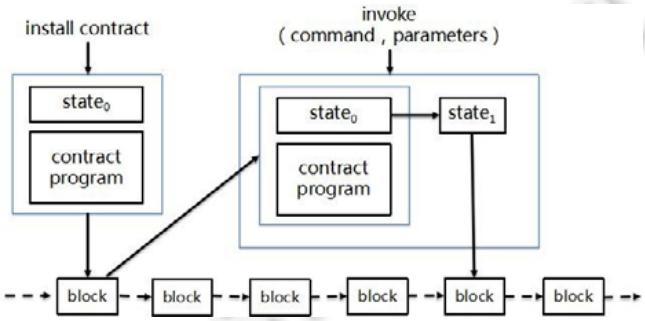

Fig.3 The process of executing smart contracts on blockchain platforms

图 3 智能合约在区块链平台上执行的通用过程描述

3.2 通用算法

智能合约是一个状态机,因此必然包含状态机和一系列接口函数.对智能合约的每次调用,都通过接口函数来完成,相当于对状态机施加的动作.状态机将实现合约所包含的一系列承诺,以及承诺之间的约束.算法 1 介绍了对智能合约的调用过程.

算法 1. invokeContract.

Input:

contractID; //合约的 ID;

invokerID; //调用者 ID,即公钥;

operation; //操作,包括参数;

sig; //调用者对操作的签名;

Output:

布尔值; //表示调用是否成功.

contractInfo←*loadContract*(*contractID*); //*contractInfo* 包含合约的状态以及各属性的值

If (!*verifySig*(*invokerID*,*operation*,*sig*)) //检验签名是否正确

Return false;

If (!*verifyOperation*(*contractInfo*,*invokerID*,*operation*))

//检查在合约当前的状况下,调用者是否可以执行这个操作

Return false;

contractInfo←*executeContract*(*contractInfo*,*invokerID*,*operation*,*sig*);

//执行合约,修改相应属性的值,并根据合约的预制规则得到新的合约状态

//本次操作记录及签名也会保存到 *contractInfo*

If (*saveContract*(*contractInfo*))

Return true;

Else

Return false;

注意输入参数 *sig*,对合约的每次调用操作均必须由调用者签名.

- 算法第 1 步,从区块链上读取指定的合约信息,存储到结构体 *contractInfo* 中;
- 第 2 步将检验签名的有效性,*operation* 相当于明文,*sig* 为对应密文,*verifySig* 函数将尝试使用调用者公钥(*invokerID*)去验证签名;
- 第 3 步是检查操作的有效性,根据合约的逻辑,判断当前状态下该调用者是否可以执行相应操作.这里,合约的逻辑实现了所有的承诺以及承诺之间的约束;
- 第 4 步将执行操作,根据状态转移矩阵修改合约的状态,并修改合约的属性值.注意,本次操作和签名也会保存作为合约的操作日志;
- 最后一步是将新的合约信息保存到区块链上.

下面,从两方面对算法 1 进行分析,我们将讨论其性能及其在执行过程中如何与外界可信地交互.

算法 1 只是一个框架,在具体执行时会调用不同的智能合约,因此整体执行性能取决于智能合约的复杂程度.因此,我们在这里只能讨论在区块链环境中执行一些基本操作的时间开销.算法 1 所涉及的基本操作主要是从区块链上读取合约的状态,以及向区块链上写入更新后的合约状态.通常,读取操作可以基于节点的本地状态数据库完成,时间开销很小.而写入操作需要和网络中其他节点保持一致,其时间开销主要取决于所采用的共识协议类型以及网络的规模.一般而言,公链的写入耗时较长,如比特币网络在 1h 左右(即 6 次确认),以太坊则在 10s 左右.联盟链的网络规模较小,单次写入耗时可以在亚秒级.本文在实验中采用的是 Hyperledger Fabric,在单机上虚拟多个节点,实验中平均每次写入耗时约数百毫秒;

算法 1 中对智能合约进行调用时,有时会涉及与外部的交互问题,比如确认航班是否延误.目前采用的办法是利用类似 oraclize 这样的预测机,保证各个节点拿到可信的相同的外部数据.Oraclize 依赖于 TLS 公证(TLSnotary)来提供诚实地从互联网页面安全获取信息的能力.

4 参考实现

本节将介绍基于 Hyperledger Fabric 系统对航空延误保险例子的实现.在第 4.1 节简述基于 Fabric 平台搭建的区块链网络;第 4.2 节介绍程序模块.我们的代码可以从 https://github.com/ RucBlockchain/SmartContract 链接下载.

4.1 平台介绍及系统部署

在 Hyperledger Fabric 1.0 中有两种类型节点.一类是 peer 节点,负责存储区块链.Chaincode(链码)部署在 peer 节点上,可以对区块链进行读写.多个 peer 节点构成一个组织(organization),每个组织有一个 CA 认证中心,负责分发公钥和私钥.多个组织构成一个通道(channel),只有通道内的 peer 节点才能参与交易(也就是一个联盟链);另一类是 orderer(排序)节点,不负责存储区块链,只负责对 peer 节点提交的交易进行排序,批量打包后生成新的区块,再发送到 peer 节点加入到区块链之中.

我们的实验构建了一个 Hyperledger Fabric 网络,其中有一个通道,包含了两个组织(org1 和 org2),每个组织里有两个 peer 节点,另外有 3 个 orderer 节点提供共识服务.Chaincode 是一段由 go 语言编写的代码,可以对区块链进行读写操作.如图 4 所示,Hyperledger Fabric 1.0 上,chaincode 的执行过程如下.

1) 用户向某个 peer 节点(称为提交节点)发出调用 chaincode 的请求;
2) 提交节点执行 chaincode 代码.当需要进行交易(就是往区块链写入交易)的时候,向 channel 中的几个 peer 节点(称之为背书节点)发出背书请求;
3) 背书节点模拟执行交易(只是模拟,此时并不会真正将交易写入区块链),得到模拟执行的结果;
4) 提交节点收集背书节点返回的模拟结果;
5) 提交节点向某一个排序节点提交交易请求(包括背书结果);
6) 排序节点是对收到的交易进行批量打包生成新的区块;
7) 排序节点将新生成的区块发送到提交节点所在 channel 中的每个 peer 节点;
8) 最后,peer 节点验证区块中每个交易的背书结果是否符合事先设定好的要求,只会将符合要求的交易

写入到区块链之中.

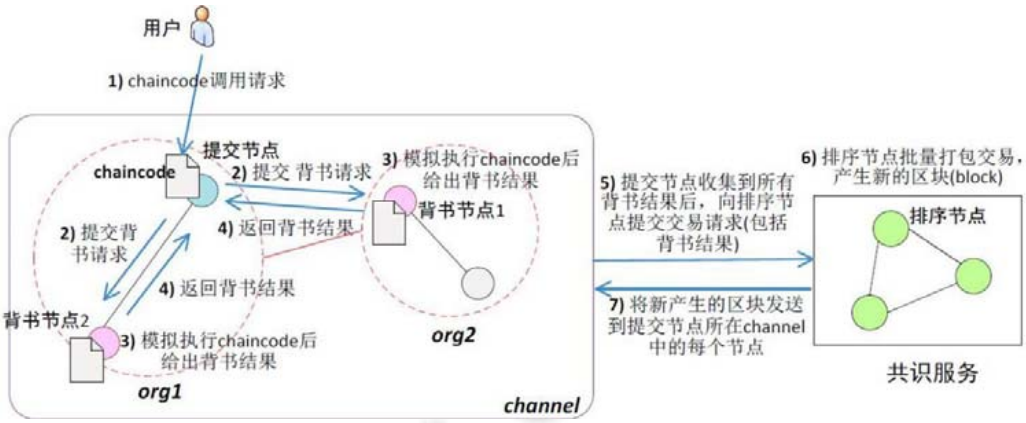

Fig.4 The process of executing chaincodes on hyperledger fabric 1.0

图 4 Hyperledger Fabric 1.0 上,chaincode 的执行过程

在这个执行过程中,我们发现:1) 通过 chaincode 写入区块链的交易都是被背书点验证的,且通过排序服务在每个节点上都保证了数据的一致性,不需要权威第三方再进行验证;但是,2) chaincode 实际上只提供了访问区块链的接口(API),不会记录 chaincode 的状态以及实现状态转移.在我们的实验中,在 chaincode 基础上实现了一种支持状态机的通用智能合约.

4.2 基于chaincode的智能合约实现思路

我们基于 chaincode 实现了一个状态机,可以在区块链存储智能合约的状态以及实现状态转移.如图 5 所示,外部调用通过入口函数进行转发.状态机包括初始化模块、事件响应模块、事件注册模块及具体的 *action* 函数.

- 初始化模块负责构造一个合约实例以及注册合约的参与各方;
- 事件注册模块将建立状态机的状态转移矩阵,即:接收到某个事件时,状态应如何转变;
- 事件响应模块将根据状态转移矩阵和输入的事件对状态机的状态进行修改;
- 最后,*action* 函数实现具体的动作(*action*),如 *buy_ticket*,*deposit* 等等.

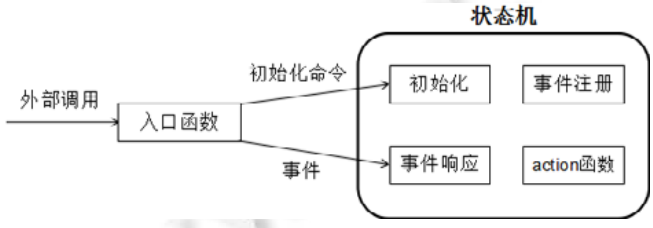

Fig.5 The architecture of smart contract program

图 5 智能合约程序架构

在航空延迟险的例子中,智能合约需根据航班的信息来决定是否赔付.在实际系统中,可以由多家保险公司或其他机构主动获取航班信息,并写入区块链.智能合约将根据这些数据来自动判定航班是否延误.多家公司数据相互印证,提高了结果的可信性.此外,由于写入区块链的数据将不可更改,客观上也会约束保险公司的行为.

5 总结和展望

本文总结了目前智能合约的发展状况,提出了对智能合约的形式化定义,以及通用的实现算法.在目前广泛

使用的联盟链平台 Hyperledger Fabric 上实现了上述算法,检验了形式化定义和算法的有效性.之后的工作将致力于开发一种可视化的智能合约的建模工具,使得用户可以轻松实现合同向智能合约的转换,并研究智能合约建模语言向程序语言(如 go,solidity,Java 等)的映射方法.

References:

[1] Szabo N. Smart contracts. 1994. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[2] https://www.sohu.com/a/245294932_750320

[3] https://www.sohu.com/a/225416045_490432

[4] https://www.forth.com/forth/

[5] Szabo N. Smart contracts: Building blocks for digital markets. 1996. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[6] Wikipedia: Smart contract. https://en.wikipedia.org/wiki/Smart_contract

[7] Christidis K, Devetsikiotis M. Blockchains and smart contracts for the Internet of things. IEEE Access, 2016,4:2292−2303. [doi: 10.1109/ACCESS.2016.2566339]

[8] Atzei N, Bartoletti M, Cimoli T. A survey of attacks on Ethereum smart contracts SoK. In: Maffei M, Ryan M, eds. Proc. of the 6th Int'l Conf. on Principles of Security and Trust. Vol.10204. New York: Springer-Verlag, 2017. 164−186. [doi: 10.1007/978-3-662-54455-6_8]

[9] Luu L, Chu DH, Olickel H, Saxena P, Hobor A. Making smart contracts smarter. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2016). New York: ACM Press, 2016. 254−269. [doi: 10.1145/2976749.2978309]

[10] Zhang F, Cecchetti E, Croman K, Juels A, Shi E. Town crier: An authenticated data feed for smart contracts. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2016). New York: ACM Press, 2016. 270−282. [doi: 10.1145/2976749.2978326]

[11] Fairfield J. Smart contracts, Bitcoin bots, and consumer protection. Washington and Lee Law Review Online, 2014,71(2):35−50.

[12] Seijas PL, Thompson S, McAdams D. Scripting smart contracts for distributed ledger technology. Report 2016/1156. Cryptology ePrint Archive, 2016.

[13] Mavridou A, Laszka A. Designing secure Ethereum smart contracts: A finite state machine based approach. arXiv.1711.09327. 2017.

[14] http://solidity.readthedocs.io/en/v0.4.21/common-patterns.html#state-machine

[15] https://www.jianshu.com/p/431cddaea961

[16] http://finance.ifeng.com/a/20180425/16197669_0.shtml

[17] Hu K, Bai XM, Gao LC, Dong AQ. Formal verification method of smart contract. Journal of Information Security Research, 2016,2(12):1080−1089 (in Chinese with English abstract). [doi: 10.3969/j.issn. 2096-1057.2016.12.003]

[18] Desai N, Chopra AK,Singh MP. Representing and reasoning about commitments in business processes. In: Proc. of the 22nd Conf. on Artificial Intelligence. AAAI, 2007. 1328−1333.

[19] Chesani F, Mello P, Montali M, Torroni P. Representing and monitoring social commitments using the event calculus. Autonomous Agents and Multi-Agent Systems, 2013,27(1):85−130. [doi: 10.1007/s10458-012-9202-0]

[20] Desai N, Chopra AK, Singh MP. Amoeba: A methodology for modeling and evolving cross-organizational business processes. ACM Trans. on Software Engineering Methodology, 2009,19(2): Article 6. [doi: 10.1145/1571629.1571632]

附中文参考文献:

[17] 胡凯,白晓敏,高灵超,董爱强.智能合约的形式化验证方法.信息安全研究,2016,2(12):1080−1089. [doi: 10.3969/j.issn. 2096-1057.2016.12.003]

王璞巍(1979－),男,贵州贵阳人,博士,讲师,主要研究领域为服务计算,区块链.

杨航天(1992－),男,硕士生,主要研究领域为区块链.

孟佶(1992－),女,硕士生,CCF 学生会员,主要研究领域为区块链.

陈晋川(1978－),男,博士,副教授,CCF 专业会员,主要研究领域为分布式数据管理,区块链.

杜小勇(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为高性能数据库实现技术,语义网技术,数字图书馆技术,智能信息检索技术.