计算机科学
COMPUTER SCIENCE

基于Event-B的可靠智能合约自动生成方法

朱健, 胡凯, 王军, 李洁, 叶亚飞, 时希言

引用本文

朱健, 胡凯, 王军, 李洁, 叶亚飞, 时希言. 基于Event-B的可靠智能合约自动生成方法[J]. 计算机科学, 2023, 50(10): 343-349.

ZHU Jian, HU Kai, WANG Jun, LI Jie, YE Yafei, SHI Xiyan. Reliable Smart Contract Automatic Generation Based on Event-B [J]. Computer Science, 2023, 50(10): 343-349.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220800134

基于 Event-B 的可靠智能合约自动生成方法

朱 健[1] 胡 凯[2,3] 王 军[1] 李 洁[2,3,4] 叶亚飞[3] 时希言[5]
1 中国电子科技集团第十四研究所 南京 210039
2 北京航空航天大学计算机学院 北京 100191
3 北京航空航天大学云南研究院 昆明 650233
4 北京物资学院信息学院 北京 101149
5 北京化工大学经济管理学院 北京 100029
(zhujian@buaa.edu.cn)

摘 要 智能合约是一种以代码的方式执行合同条款的可计算交易协议,其应用场景与规模日益增长,承载着多达数十亿美元的各类资产。由于其代码缺陷可能会造成严重的经济损失,因此智能合约的可信开发成为技术关键。为此,提出了一种基于集合论语言 Event-B 的智能合约可信验证与自动生成方法。Event-B 方法是一种基于精化的形式化方法,可用于规约、设计和验证软件系统。通过对智能合约的模型验证和可执行代码的自动生成技术,研发了自动转换工具 EB2S,打通了形式化模型和智能合约编程语言的语义鸿沟和技术壁垒。最后,选取典型的在线支付智能合约场景,应用基于 Event-B 的智能合约模型自动生成合约代码,验证了 EB2S 转换工具的有效性。

关键词:智能合约;Event-B 方法;自动代码生成;Solidity 合约;定理证明
中图法分类号 TP311.5

Reliable Smart Contract Automatic Generation Based on Event-B

ZHU Jian[1],HU Kai[2,3],WANG Jun[1],LI Jie[2,3,4],YE Yafei[3] and SHI Xiyan[5]
1 The 14th Research Institute of CETC,Nanjing 210039,China
2 School of Computer Science and Engineering,Beihang University,Beijing 100191,China
3 Yunnan Innovation Institute of Beihang University,Kunming 650233,China
4 School of Information,Beijing Wuzi University,Beijing 101149,China
5 School of Economics and Management,Beijing University of Chemical Technology,Beijing 100029,China

Abstract Smart contract is a new computable transaction agreement that executes contract terms in code. Its application scenarios and scale are growing with each passing day,carrying up to billions of dollars of various assets. However,smart contracts may cause serious economics losses due to code defects. Therefore,the trusted development of smart contract is particularly critical. This paper proposes a method of trusted verification and automatic generation of smart contracts based on the set theory language Event-B,which is a formal method based on refinement and can be used for specification,design and verification of software systems. Through the model verification of smart contracts and the automatic generation technology of executable codes,an automatic conversion tool EB2S is developed,which bridges the semantic gap and technical barriers between formal models and smart contract programming languages. Finally,this paper selects a typical online payment smart contract scenario,and the smart contract design and verification method based on Event-B is applied to automatically generate the smart contract code,which verifies the effectiveness of the conversion tool.

Keywords Smart contract,Event-B method,Automatic code generation,Solidity contract,Theorem proving

到稿日期:2022-08-15 返修日期:2023-01-12
基金项目:北京市自然科学基金(M22040);云南省重大科技专项(202103AN080001-001,202102AD080006)
This work was supported by the Natural Science Foundation of Beijing,China(M22040) and Science and Technology Major Project of Yunnan Province(202103AN080001-001,202102AD080006).
通信作者:叶亚飞(yeyf@bhynii.com)

1 引言

智能合约作为区块链上最核心的技术之一，其结合了纸质合同的法律强制执行特点和作为链上代码的自动执行属性，已经引起了业界的重视，在数字化的发展进程中起到越来越重要的作用。

智能合约本质上可以看作是区块链上部署的一段可信交易执行程序，它蕴含着契约关系和数字资产交易信息，承载了巨大的价值和交易属性，比一般的软件具有更加复杂的逻辑性和更高的安全性要求。目前，由智能合约引发的区块链安全问题也十分严峻。据研究人员统计，有超过 34 000 个智能合约都有可被利用的安全隐患[1]。这使得提高智能合约的安全性成为一个亟待解决的问题。

针对智能合约，通常采用软件测试或人工审计的方式排查漏洞，提高智能合约代码的可靠性和健壮性。但这些方法不能保证智能合约的正确性，因为软件测试或人工审计只能用于检测合约漏洞，即使检测不到漏洞也无法严格地证明合约没有漏洞。另外，测试方法是从系统的局部入手，理论上无法覆盖系统的全部执行情况，因此无法考虑到系统的整体性。当合约存在一些超出测试范围的漏洞时，将导致不可控的损失。相比软件测试，人工审计更需要投入巨大的人力成本和时间成本，且需要很专业的人员才能做到，而且谁来审计、如何审计、怎么审计都是问题。因此，在处理大量存在潜在漏洞的智能合约时，需要一种更加高效、严谨和准确的验证手段。

形式化方法[2]是一种基于严格数学模型的软硬件设计方法，是在安全关键软件系统中被广泛采用且高效的验证方法，同时也是国际软件验证的最高级别标准。它可以为系统和软硬件的规约、设计与验证等相关活动提供支持。在形式规约和验证的基础上，形式化建模主要是构造、证明形式规约之间的等价转换和精化关系，以系统的形式模型为指导，逐步精化，开发出满足需要的系统，也称为构造即正确(Correct by Construction)的开发[3]。Event-B[4]是由 B 方法[5]演化而来的，其灵感来源于 Back 等提出的行为系统[6-7]。它是一种支持精化的形式化建模语言[8]，可以将程序或系统抽象成基于类型集合论的离散系统，并生成证明义务(Proof Obligations)，然后验证系统的属性。

应用形式化方法验证智能合约已经成为热门研究领域，编写出可靠、安全的智能合约代码是应用形式化方法的主要目标。直接将形式化方法和智能合约结合会面临诸多限制，如开发者需要掌握深厚的数学知识，学会使用复杂的符号语言和相关形式化工具。此外，当前针对智能合约的开发主要是从需求到代码，即根据单个需求开发对应的智能合约代码，在面对新的需求时，直接修改代码容易产生新的漏洞，因此合约代码很难复用，导致合约开发效率低下。智能合约语言所支持的类型检查也不能完备地验证安全属性和期望属性，因此基于当前模式开发出来的智能合约代码的安全性较低。

为解决上述问题，本文提出从 Event-B 模型到 Solidity 智能合约语法的转换方法。首先基于形式模型来验证智能合约的安全属性和期望属性，然后通过自顶向下的方式提出具体的转换规则，同时应用 Java 语言实现了转换工具 EB2S。该工具可帮助精通 Event-B 方法的专家在建立严格的形式化模型并验证所有的证明义务后，将 Event-B 模型自动转换到 Solidity 智能合约中，即生成安全、可靠的可执行智能合约代码。

本文的主要贡献与创新如下：

(1)提出了基于 Event-B 方法的可信智能合约的生成方法，并分析了 Event-B 模型与 Solidity 合约的语义一致性问题，保证高效、自动地生成可信的智能合约代码。

(2)定义了转换规则运算子，自顶向下提出从 Event-B 模型到 Solidity 合约的转换规则，并基于转换规则研发了自动转换工具 EB2S。

(3)选取典型在线支付场景，应用 Event-B 方法实现了合约模型的设计、仿真模拟和属性验证，并应用 EB2S 工具生成可执行合约代码，验证了工具的有效性。

2 相关工作

与大多数软件一样，智能合约也面临着隐私泄露、安全漏洞和其他各种问题。2016 年 4 月，The DAO 智能合约因存在重入漏洞被黑客发现并发起攻击，损失超过 5 000 万美元[9]；2017 年 7 月，Pairty 公共合约库爆出一个代码注入漏洞，损失了超过 3 000 万美元的以太币，官方在修复漏洞中又产生了新的漏洞，导致了约 2.3 亿美元的以太币被永久冻结[10]；2018 年 4 月，BEC 与 SMT 两个符合 ERC20 标准的众筹智能合约被发现整数溢出漏洞，被黑客利用，凭空造出 2 256 量级的代币，导致市值跌破归零[11]。频频爆出的智能合约安全事件，使得越来越多的研究人员考虑使用形式化方法来提高智能合约的安全性。

我们广泛搜集了基于形式化方法验证智能合约领域的相关研究[1]，将应用在智能合约上的形式化验证方法分为两类。

第一类是应用形式化语言或工具验证现有的智能合约程序，例如，2016 年 Bhargavan 等[12]通过定义从 Solidity 智能合约到 F^* 函数式编程语言的转换规则，从而使用 F^* 来规约 Solidity 智能合约程序；2019 年，Nielsen 等[13]通过 Coq 证明助手定义了智能合约的规约，该规约支持合约之间的调用功能；2020 年，Zhu 等[14]使用 Event-B 语言对 Solidity 智能合约代码进行建模与验证，并实现了自动转换工具。这类方法可以有效地验证已有智能合约程序的安全属性，但程序建模的自动化程度较低，且越复杂的合约程序对形式化语言或工具的表达能力要求更高。当合约程序发生变动时，需要重新建模验证，复杂度与成本也较高。

第二类是基于形式模型生成可信智能合约代码。2018 年，Mavridou 等[15]提出了 FSolidM 框架，该框架允许开发人员将智能合约定义为有限状态机，并提供了一种在图形界面上创建有限状态机并自动生成以太坊智能合约的方法。但是有限状态机表示的合约状态有限，且只能接受正则语言，表达能力受限。

除了应用形式模型指导生成可执行合约代码外，还有一些基于领域特定语言向目标合约语言转换的工作。例如，2018 年，Choudhury 等[16]通过本体和语义规则对领域特定知识进行编码，利用抽象语法树合并约束，支持自动生成领域特定智能合约代码模板，领域专家对本体和语义规则提炼的

质量对最终结果影响较大；2020 年，Gao 等[17]通过数据抓取以及聚类分析，自动生成交易类智能合约模板，但其主要针对交易类合约验证，通用性不足；2021 年，Zhu 等[18]提出了从智能法律合约语言 SPESC 到以太坊智能合约的转换方法，该方法支持半自动化地编写智能合约程序，但缺乏对法律合约语言或智能合约的形式化验证，安全性保证没有充分说明。该类方法可以更好地满足特定交易场景的需求，并提高了生成智能合约程序的效率，但无法严格保证程序的正确性。

为了建立从需求分析到形式模型，再高效地生成安全可信智能合约代码的设计链条，本文提出了一种基于 Event-B 方法的智能合约形式化验证与自动生成方法，帮助开发者快速开发与需求一致的智能合约。其中智能合约语言选择 Solidity 语言作为研究对象，完整的 Event-B 模型包括两部分：机器(Machine)和文本(Context)。其中机器定义了一系列行为，称为事件(Event)，其作用类似于命令式语言中的函数。事件执行需要满足的条件称为守卫条件(Guard)；上下文主要用于定义全局变量和定理。选择该方法的主要原因是：(1)Solidity 语言图灵完备且应用广泛，有很高的验证价值；(2)Event-B 支持从抽象到具体的精化方式来建模，可以保证模型和合约需求的一致性；(3)Event-B 是基于集合论的语言，具有强大的表达力，并且有成熟的 Rodin 开发平台[19]作为支撑，可以开发插件，并支持模型检测和交互式定理证明，功能非常强大。

这种设计模式弥补了其他智能合约设计与生成方法中没有对转换的源语言或目标语言进行严格的逻辑、漏洞等的形式化验证的缺陷，在需求验证无误的情况下，本文设计的模式保证了生成代码与需求模型的一致性。

3　可靠智能合约自动生成方法

3.1　可验证智能合约的生成架构机制

本节研究基于模型驱动的正向智能合约建模验证与代码生成方法，其设计流程如图 1 所示。

(1)针对用户的需求(智能合约的业务流程、安全属性和期望属性等)构建形式化模型，并逐步精化，形成一个或多个精化模型；

(2)基于形式化模型进行逐步求精、逻辑推理、验证和仿真，验证证明义务；

(3)如果验证未通过，则反馈用户的修改需求，并迭代优化模型，直到全部证明义务通过；

(4)自动代码生成，通过自动代码转换工具 EB2S，将形式化模型转换为可信合约代码。

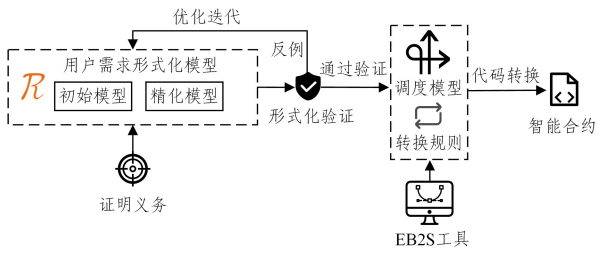

图 1　基于 Event-B 模型的智能合约生成方法

Fig. 1　Smart contract generation method based on Event-B model

通常形式化领域专家精通建立形式模型和推理证明，但对智能合约语言并不是很熟悉；反过来，智能合约的开发者也很难建立形式化模型并进行形式化验证。因此，打通形式化模型和智能合约代码的技术壁垒是非常有意义的。本文提出并实现了 EB2S 自动转换工具，它将 Event-B 形式化模型作为输入，输出 Solidity 智能合约代码。该工具作为 Rodin 插件安装使用，Rodin 是基于 Eclipse 平台开发的 IDE，操作方便。

3.2　从 Event-B 到 Solidity 语言转换规则

本文开发 EB2S 工具以完成 Event-B 到 Solidity 的自动转换，本节介绍基于 EB2S 运算子的转换规则。由于定义的所有转换规则都是确定的，当给定相同的输入时，不会出现不同的输出结果。转换规则均以推理规则的形式给出，为了方便介绍语法转换的理论，首先给出以下形式化定义。

定义 1(横线运算符)　对于有限次的逻辑推导过程 $p\rightarrow q$，其中 p 为前件，q 为后件，则横线运算符上方描述前件，横线下方描述后件。

定义 2(转换规则运算子)

(1)EB2S 运算子：表示 Event-B 代码(包括变量、单个语句或多个语句等)经过 EB2S 转换规则得到的运算结果。

(2)TypeOf 运算子：用于计算 Event-B 变量在 Solidity 语法中对应的类型。

(3)Pred 运算子：用于计算 Event-B 变量或单个语句转换得到的 Solidity 的变量或语句。

本文按照从整体到局部的转换方法介绍转换规则。首先定义规则 M，它将一个完整的 Event-B 机器和它引用的文本转换到 Solidity 合约。本文规定在转换前必须验证通过所有的证明义务，因为转换一个不能通过证明义务的机器是没有意义的；另一方面，转换通过证明义务的机器，不需要考虑将 Event-B 中的见证者(Witness)和变体(Variant)转换到 Solidity 合约中。见证者用于规约在抽象模型中的变量，但该变量在精化后的模型中不再使用。变体用于证明 Event-B 某个事件的收敛性，即系统的执行一定会终止。这两个元素用于辅助证明模型的精化属性和事件收敛性，而在 Solidity 合约中没有对应的功能和需求，因此不考虑将其转换到 Solidity 合约。

$$\frac{\begin{array}{cc} EB2S(sets\ s)=s' & EB2S(variables\ v)=v' \\ EB2S(constants\ c)=c' & EB2S(invariants\ I(s,c,v))=I' \\ EB2S(axioms\ X(s,c))=X' & EB2S(events\ e)=e' \\ \multicolumn{2}{c}{EB2S(theorems\ T(s,c))=T'} \end{array}}{EB2S\left(\begin{array}{ll} context\ ct\ x & machine\ M\ seesct\ x \\ sets\ s & variable\ v \\ constants\ c & invariants\ I(s,c,v) \\ axioms\ X(s,c) & events\ e \\ the\ orems\ T(s,c) & end \\ end & \end{array}\right)}(M)$$

$$=\begin{array}{l} \text{contract } M\{ \\ s'c'X'T'v'I'e' \\ \} \end{array} \tag{1}$$

由于 Solidity 语言没有关于抽象集的概念和相关的数据结构，因此需要基于 Solidity 语法规则构造一个语义一致的数据结构，选择数组作为和抽象集对应的数据结构，限制数组中不能有相同的元素，即数组中存储的元素值不能一样。通过 Set 规则将集合转换为数组，TypeOf 运算子用于计算集合 s 的类型。

$$\frac{TypeOf(s)=T1}{EB2S(sets\ s)=T1[]public\ s';}(\text{Set}) \tag{2}$$

Cons 规则和 Var 规则分别用于转换 Event-B 中的常量和变量到 Solidity 语言中对应的常量和变量。

$$\frac{TypeOf(c)=T1}{EB2S(constants\ c)=T1\ constant\ c'}(\text{Cons}) \tag{3}$$

$$\frac{TypeOf(v)=T2}{EB2S(variables\ v)=T2\ public\ v'}(\text{Var}) \tag{4}$$

将公理、定理和不变量都转换为 assert 断言。我们定义了 Axiom，Theorem 和 Inv 3 个转换规则，其中，Pred 运算子用于计算 Event-B 变量或语句转换得到的 Solidity 的变量或语句。

$$\frac{Pred(X(s,c))=X'}{EB2S(axiomsX(s,c))=assert(X')}(\text{Axiom}) \tag{5}$$

$$\frac{Pred(T(s,c))=T'}{EB2S(theoremsT(s,c))=assert(T')}(\text{Theorem}) \tag{6}$$

$$\frac{Pred(I(s,c,v))=I'}{EB2S(invariantsI(s,c,v))=assert(I')}(\text{Inv}) \tag{7}$$

初始化事件在所有事件中最先执行，并只执行一次，用于初始化机器中的变量，因此直接转换为 Solidity 合约中的构造函数。构造函数是使用 construct 关键字声明的特殊函数，用于初始化合约中的变量。式(8)定义了 Init 规则，将 EB2S 运算子递归地应用在初始化事件，可实现对整个 initialisation 事件的转换。

$$\frac{EB2S(A(s,c,v))=A'}{EB2S(eventsinitialisationthenA(s,c,v)end)=constructor()public\{A';\}}(\text{Init}) \tag{8}$$

普通事件可以转换为两个 Solidity 函数，一个守卫函数用于测试对应事件的守卫条件是否满足，一个执行函数用于执行对应事件的动作。在 Event-B 中，事件的所有守卫条件是同时评估的，但每次只能执行一个事件。根据 Event-B 语义，如果事件的守卫条件不满足，事件不会执行，因此不会改变系统状态。为了满足其语义，本文针对两种情况分别进行了对应的转换。

第一种情况是事件的守卫条件不满足，即不能执行 run_evt 方法，我们定义其后置条件为合约中变量的值不能发生改变。为了判断合约中变量是否被修改，本文定义了一个 *can_assign* 函数，以及两个全局数组变量 G 和 G'，由于合约的全局状态变量数目是确定的，因此可以定义一个数组 G 保存着合约中所有的全局变量。G' 表示经过了某个 run_evt 方法后的 G，如果该 run_evt 方法没有对变量修改，则 $G=G'$，反之，$G\neq G'$。*can_assign* 方法定义如图 2 所示，函数的输入为指定变量组成的数组，规定位于该数组中的元素可以进行赋值操作，允许其发生变化，而不在该数组中的元素则不能被修改。例如，*can_assign*(*null*)表示传入了空数组，即没有变量可以被修改；*can_assign*(S)表示传入了数组 S，即 S 中保存的任何合约变量都可以被修改，非 S 中的其他合约变量不能被修改。

第一种情况的断言如式(9)所示：

$$require(!guard\_evt(T1x))\&\&can\_assign(null)\&\&assert(true)) \tag{9}$$

第二种情况是事件的守卫条件满足，可以执行 run_evt 方法，本文规定其后置条件为合约状态的变化必须满足该事件的动作。其断言如式(10)所示。其中 $assert(A')$ 表示对函数中所有赋值语句进行断言，如赋值语句 $var=expr$，其断言为 $assert(var==expr)$。

$$assert((require(guard\_evt(T1x))\&\&can\_assign(D')\&\&assert(A')) \tag{10}$$

接下来介绍赋值操作转换规则。在 Solidity 语言中变量都是值类型，它们的比较操作使用==符号。考虑赋值后的表达式 E' 可能会发生变化，因为表达式中的变量可能会发生变化，而被赋值的变量应当等于赋值前的表达式 E'。为了转换这条安全属性，本文定义一个常量 EC，用于记录赋值前的表达式 E' 的值，如式(11)所示：

$$\frac{Pred(E(s,c,v))=E'}{EB2S(v;E(s,c,v))=constantEC=E';v=E';assert(v==EC)}(\text{Assign}) \tag{11}$$

```
function can_assign(T[] Array) public returns(bool)
{
        bool tmp = true;
        for (int i =0; i < G.length; i++){
                if((G[i] != G'[i])&&(!Array.has(G[i]))){
                        tmp = false;
                        return tmp;
        }
        return tmp;
}
```

图 2 *can_assign* 函数

Fig. 2 Function *can_assign*

3.3 基于 Event-B 的智能合约调度模型

Event-B 作为形式化语言，与智能合约的命令式语言在执行模式方面具有明显差别，无法直接进行语言转换。本节设计基于 Event-B 语言的智能合约调度模型来解决这一问题，该调度模型是 3.2 节其他基本元素转换规则的必要补充。

当至少有一个事件的守卫条件满足，机器就会不断执行，直到所有事件的守卫条件都不被满足。事件的执行满足原子性，执行过程不可中断，执行结果全部成功，若出错则全部失败。将 Event-B 模型转换到 Solidity 合约时，需要在 Solidity 合约中实现 Event-B 模型的执行机制。图 3 给出了一个调度模型，建立了一个名为 Controller 的智能合约，合约中所有方法都可以通过 this.method()来调用，从而模拟实现符合 Event-B 语义的事件执行机制。Event-B 中事件的执行顺序是随机的，Controller 中也通过随机数模拟事件的执行顺序，通过控制语句保证了事件执行的原子性。

```
contract Controller {
  ...
  int n = N ; /// N等于合约中除了初始化函数的所有函数数量
  while(this.guard_evt_1() || this.guard_evt_2() || ... || this.guard_evt_N()) {
    uint r = rand(N) ; /// 获得一个0-N之间的随机数
    if (r == 0 && this.guard_evt_1()) this.run_evt_1() ; break ;
    ...
    if (r == N-1 && this.guard_evt_N()) this.run_evt_N(); break ;
  }
}
```

图 3　基于 Event-B 的智能合约调度模型

Fig. 3　Smart contract scheduling model based on Event-B

4　EB2S 工具实现

EB2S 工具采用 Java 语言，基于 Rodin 插件形式开发，在 Rodin3.5 版本上进行测试和应用。图 4 给出了该工具的总体架构，EB2S 工具调用 Rodin 平台提供的 API 获取到 Event-B 机器中的元素，如集合、常量等。然后应用转换规则，便可得到对应的 Solidity 合约文件。

Rodin 作为基于 Eclipse 开发的平台，由很多插件组成，图 4 的虚线矩形表示插件，如编辑器、证明义务生成器、证明器、模型检测工具和定理证明器等。EB2S 是 Rodin 的新插件，输入一个 Event-B 模型，可以转换得到 Solidity 合约。

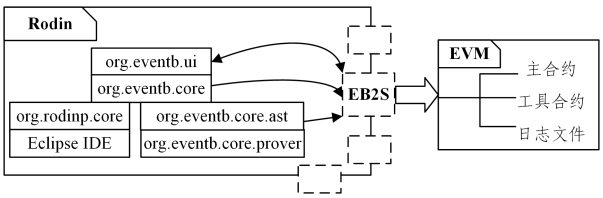

图 4　EB2S 工具内部结构

Fig. 4　Internal structure of EB2S tool

EB2S 工具是基于 Rodin 的插件开发模式来实现的，并且作为 Rodin 的插件集成在 Rodin 平台上。用户需要安装 Rodin 平台，并同时安装 EB2S 插件。完成安装工作后，点击 Rodin 主视图上的 EB2S 图标，打开该插件后，会弹出一个交互式工具界面，如图 5 所示，工具中间展示“请选择一个 Event-B 项目用于转换”字样，工具会自动识别所有位于 Event-B 资源管理器中的 Event-B 项目。用户可以通过下拉菜单选择需要转换的项目名称，然后选择“全部转换”或“部分转换”模式进行转换，前者会直接转换整个项目下的所有机器以及机器引用的文本，后者提供了下拉菜单，指定需要转换的机器。输入指定的项目文件后，工具会在项目所在目录下生成一个 Solidity 文件夹，里面保存着生成的日志文件与 Solidity 合约文件。“全部转换”模式会直接根据最后一次精化得到的机器来转换。

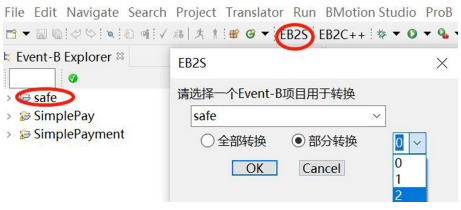

图 5　EB2S 工具界面

Fig. 5　Interface of EB2S tool

5　应用案例与测试

本部分基于一个典型的在线支付智能合约场景，应用模型驱动开发思想，采用 Event-B 方法建立合约模型并对属性进行规约，应用精化技术加入监管规则，使得合约满足合法性和公信性；接着基于 Rodin 平台对模型进行了仿真模拟、模型检测和定理证明；最后通过 EB2S 工具生成可执行代码，验证了工具的有效性。

5.1　智能合约 Event-B 模型

这里考虑一个通用的在线支付智能合约的案例[20]。假设商家 M 和顾客 C 完成了商品的交付环节，但是顾客 C 不能立即付全款，为了维持良好的客户关系同时希望用户尽快结清尾款，商家需要制定一种激励付款策略。由于交易是基于区块链平台进行的，商家 M 期望实现这样一个智能合约：为顾客设定一个付款期限 S1，如果顾客能在 S1 前完成付款，则给出 10% 的优惠折扣；如果用户超出期限 S1，但在 S2 之前完成付款，则给出 5% 的优惠折扣；当用户超过期限 S2 还没有付款，则没有任何折扣，需要全额付款。这个合约模型相对简单，且可以在购买房屋、订购大量商品等涉及巨额资金交易的场景中落地应用。同时，为了设计的智能合约满足合法性和公信性，在模型精化的过程中，我们加入监管规则用于保证每笔交易的合法性，有效避免了类似洗钱的非法行为。

根据需求为以上过程设置 6 个状态常量：work，completed，done，Other，delay2，done2，overdue2。其中 work 用于初始化模型状态，经过 Signal 事件，完成商品交易，等待顾客付款，之后系统状态变为 completed。Collect_1_Y 和 Collect_1_N 事件分别用于捕捉用户是否在期限内付款的信息，如果用户在付款期限 S1 之前完成付款，则系统状态变成 Done，表示完成付款，且顾客享受到优惠折扣；反之，系统状态变为 Delay2，增加两个状态转移事件 Collect_2_Y 和 Collect_2_N，用于捕捉顾客是否在第二个期限内完成付款的信息，分别对应 done2 和 overdue2 的状态。考虑到后续的合约状态可能会变多，因此这里定义了一个常量 Other，专门用于后续对 STATE 集合的精化。6 个状态变量共同组成了集合 STATE。抽象集合的好处在于不关注集合中元素的具体类型，在后续的精化中可以根据业务需求转换成整型、字符串等。

为了使设计的智能合约满足合法性和公信性，对模型进行精化，赋予每一笔交易一个唯一的合法 id，每一笔交易都需要申请合法的 id 号码才可以执行。

精化后的文本如图 6 所示。

```
context c2 extends c1
constants ini ID_old ID
axioms
 @axm2 ID⊆ℕ
 @axm3 ID\ID_old≠∅
 @axm4 ID_old⊂ID
 @axm5 ini∈ID_old
end
```

图 6　精化模型的文本

Fig. 6　Text of refinement model

首先定义 ID 常量作为合法交易 id 的集合，并定义 ID_old 常量作为已经执行过的合法交易 id 的集合，最后定义 ini 常量作为机器中的初始化 id 值。@axm4 表示 ID_old 是 ID 的子集，@axm5 表示 ini 属于 ID_old 的元素，表示交易的初始 id 是最后一次成功执行的合法 id 号，如果要执行新的交易，需要重新申请合法 id 号。

精化后的机器如图 7 所示，定义 *id_trans* 变量表示当前交易的 id 号。精化 m1 机器中的 Sigal 事件，在交易开始前，必须为当前交易变量 *id_trans* 赋予一个合法 id 号。@inv2 与@inv3 定义了两条符合监管规则的期望属性，前者表示当前交易 id 不合法时，模型的状态一直为 work，即交易无法执行，后者表示交易拥有合法 id 后，合约状态属于 STATE\{work}，即交易可以正常执行。

```
machine m2 refines m1 sees c2
variables state disc disc_f state_f id_trans
invariants
@inv1 id_trans∈ID
@inv2 id_trans∈ID_old⇒state_f=work
@inv3 id_trans∈ID\ID_old⇒state_f∈STATE\{work}
events
 event INITIALISATION
 then
 @act1 id_trans:=ini
 @act2 disc_f:=0
 @act3 state_f:=work
 @act4 disc:=0 ‥10
 @act5 state:=work end
 event Sigal extends Sigal
  any l
  where
   @grd3 l∈ID\ID_old
  then
   @act5 id_trans:=l end
 event Collect_1_Y extends Collect_1_Y end
 event collect_1_N extends collect_1_N end
 event Collect_2_Y extends Collect_2_Y end
 event Collect_2_N extends Collect_2_N end
end
```

图 7 精化模型的机器

Fig. 7 Machine of refinement model

该模型的证明义务全部通过，如图 8 所示。

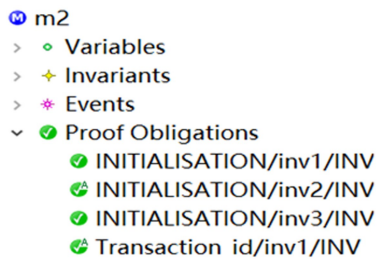

图 8 精化模型的证明义务

Fig. 8 Proof obligations of refinement model

5.2 EB2S 工具测试

使用 EB2S 插件将经过验证的 Event-B 模型转换得到的 Solidity 代码如图 9 所示。在 Event-B 机器中，ID 与 ID_old 是两个常量，表示交易 id 的无限抽象集合，而 Solidity 合约的数组类型是有限的，因此在转换到 Solidity 合约时，需要手动实例化。

```
pragma solidity >=0.7.0 <0.9.0;
contract m0{
…//此处省略默认生成的辅助函数，如unionSet等，因为后面不会用上
enum STATE {working,completed,done, done2,
overdue2,delay2}
STATE state_f;
uint disc_f;
uint id_trans;
uint constant ini;
uint[] constant ID;
uint[] constant ID_old;
constructor() public {
  state_f = working;
  disc_f = 0;
  id_trans = ini; }
function sigal(uint l) public returns (bool){
  require((ID\ID_old).has(l));
  id_trans = l;
      return true; }
function collect_1_Y() public returns (bool){
  require(state_f == completed);
  state_f = done;
  disc_f = 10;
  return true; }
function collect_1_N() public returns (bool){
  require(state_f == completed);
  state_f = delay2;
  disc_f = 5;
  return true; }
function collect_2_Y() public returns (bool){
  require(state_f ==delay2);
  state_f = done2;
  disc_f = 5;
  return true; }
function collect_2_N() public returns (bool){
  require( state_f==delay2);
  state_f = overdue2;
  disc_f = 0;
  return true; }
}
```

图 9 EB2S 生成的合约代码

Fig. 9 Contract code generated by EB2S

本文使用以太坊官网开源的 Remix[21] 作为集成开发环境测试生成的 Solidity 代码，测试步骤如下：

(1)编译直接通过，没有任何语法错误。

(2)运行智能合约，使用 JavaScript 虚拟机作为节点环境，将智能合约部署在给定地址位置上。

(3)按照 *signal*->*collect_1_N*->*collect_2_Y*->*collect_2_N* 的顺序执行函数，日志文件如图 10 所示，最后一个函数 *collect_2_N* 执行报错，是因为合约变量 *state_f* 已经是 done2(可以通过修改 returns 语句，将变量打印出来查看)，不满足函数的 require 断言。

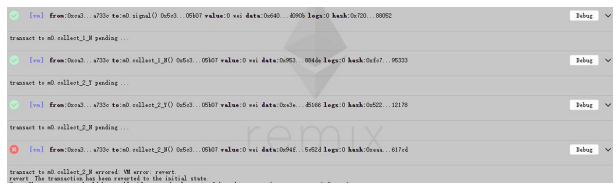

图 10 合约执行日志

Fig. 10 Contract execution log

(4)分析智能合约。基于 Remix 的分析工具可以对部署

的合约代码进行安全性分析，选择所有的安全性分析要素后，运行结果显示没有发现问题，说明从 Event-B 模型转换得到的合约代码具有很高的安全性。

结束语　智能合约应用场景丰富，需求迭代快，且上链不可更改，对安全性要求高。为了打通形式化方法和智能合约代码之间的壁垒，本文基于模型驱动代码的思想，分析了 Event-B 语言应用在 Solidity 智能合约建模的语义一致性，给出了从 Event-B 模型到 Solidity 智能合约的语法转换规则，针对 Event-B 的语法规则和语义，构造了多个 Solidity 数据结构和辅助函数。基于 Java 语言，实现了从 Event-B 到 Solidity 的自动转换工具 EB2S，提高了转换的效率。最后以一个经典的在线支付智能合约场景为例，展示了从设计模型、模型仿真和验证到自动代码生成的过程，通过分析生成的代码，验证了 EB2S 工具的有效性。

尽管本文提出了转换规则的语法和语义描述，但仍未对 Event-B 模型与 Solidity 代码之间语义一致性进行严格的形式化验证，未来可建立 Event-B 到 Solidity 转换规则的一致性证明义务，并通过形式化方法验证该证明义务（使用 Coq 或 F* 等形式化语言），从而保证转换工具的一致性和正确性。

参 考 文 献

[1] ZHU J，HU K，ZHANG B J. Review on Formal Verification of SmartContract[J]. Acta Electronica Sinica，2021，49(4)：792-804.

[2] ZHANG G Q. Introduction to formalmethods[M]. Beijing：Tsinghua University Press，2015.

[3] RYZHYK L，BJØRNER N，CANINI M，et al. Correct by construction networks using stepwise refinement[C]// 14th USENIX Conference on Networked Systems Design and Implementation. USA：USENIX Association，2017：683-698.

[4] ABRIAL J. Modelling in Event-B：System and Software Engineering[M]. UK：Cambridge University Press，2009.

[5] SCHNEIDER S. The B-method：An introduction[M]. UK：University of Surrey，2001.

[6] BUTLER M，YADAV D. An incremental development of the mondex system in event-B[J]. Formal Aspects of Computing，2008，20(1)：61-77.

[7] BACK R J R，KURKI-SUONIO F. Distributed cooperation with action systems[J]. ACM Transactions on Programming Languages and Systems，1998，10(4)：513-554.

[8] BUTLER M. Decomposition structures for Event-B[C]// International Conference on Integrated Formal Methods. Germany：Springer，2009：20-38.

[9] BUTERIN V. Critical update re：Dao vulnerability [EB/OL]. https://blog. ethereum. org/2016/06/17critical-update-re-dao-vulnerability.

[10] The parity wallet hack explained [EB/OL]. https://blog. openzeppelin. com/on-the-parity-wallet-multisig-hack-405a8c12-e8f7.

[11] Batch overflow bug on Ethereum ERC20 token contracts and safeMath [EB/OL]. https://peckshield. medium. com/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536.

[12] BHARGAVAN K，DELIGNAT-LAVAUD A，FOURNET C，et al. Formal verification of smart contracts：short paper[C]// the 2016 ACM Workshop on Programming Languages and Analysis for Security. USA：ACM，2016：91-96.

[13] NIELSEN J B，SPITTERS B. Smart contract interactions in Coq [J]. arXiv：1911. 04732，2019.

[14] ZHU J，HU K，FILALI M，et al. Formal simulation and verification of Solidity contracts in Event-B[C]// 2021 IEEE 45th Annual Computers，Software，and Applications Conference(COMPSAC). USA：IEEE，2021：1309-1314

[15] MAVRIDOU A，LASZKA A. Designing secure ethereum smart contracts：A finite state machine-based approach[C]// International Conference on Financial Cryptography and Data Security. Germany：Springer，2018：523-540.

[16] CHOUDHURY O，RUDOLPH N，SYLLA I，et al. Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules[C]// 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications(GreenCom) and IEEE Cyber，Physical and Social Computing(CPSCom) and IEEE Smart Data(SmartData). USA：IEEE，2018：963-970.

[17] GAO Y C，ZHAO B，ZHANG Z. Research and implementation of a smart automatic contract generation method for Ethereum [J]. Journal of East China Normal University(Natural Science)，2020，2020(5)：21-32.

[18] ZHU Y，QIN B H，CHEN E，et al. An advanced smart contract conversion method and bidding contract design and implementation[J]. Chinese Journal of Computers，2021，44(3)：652-668.

[19] Rodin platform [EB/OL]. http://wiki. Event-B. org/index. php/Rodin_Platform.

[20] BANACH R. Verification-led smart contracts [C]// International Conference on Financial Cryptography and Data Security. Germany：Springer，2019：106-121.

[21] Remix [EB/OL]. http://remix. hubwiz. com/#optimize=false&version=soljson-v0. 4. 25-nightly. 2018. 7. 9+commit. c42583d2. js.

ZHU Jian，born in 1997，master，assistant engineer. His main research interests include formal methods and smart contract.

YE Yafei，born in 1982，master，engineer. Her main research interests include blockchain and smart contracts.

（责任编辑：何杨）