文章编号:1001-2486(2011)06-0066-06

一种支持并行离散事件仿真建模和并行模型检验的建模语言*

夏 薇[1,2],姚益平[1],慕晓冬[2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073; 2. 第二炮兵工程大学 计算机系,陕西 西安 710025)

摘 要:并行离散事件仿真(Parallel Discrete Event Simulation, PDES)模型的正确性和可信度对PDES应用的发展起着决定性作用。然而,现有的并行离散事件仿真开发环境都没有提供仿真模型检验功能。并行模型检验(Parallel Model Checking, PMC)方法以其完备性、高效性已经在工业界中得到了成功的应用,但是PDES和PMC是基于不同的建模语言实现的,现阶段要对PDES模型进行验证需要专门设计基于PMC建模语言的模型,不仅耗费资源、时间,而且容易出错。为了实现PDES和PMC的有机结合,提出了一种支持PDES和PMC的统一的建模语言——扩展事件图(Extended Event Graph, EEG),它对事件图在模型同步方面进行了扩展,然后通过转换机制,该建模语言使得用户只需建立一个模型,就能够既进行PDES又利用并行模型检验方法对PDES模型进行形式化验证。最后通过实验验证了基于EEG建立的模型既可以进行PDES又能够进行PMC。

关键词:并行离散事件仿真;模型检验;事件图;逻辑进程范型;DVE建模语言;模型转换
中图分类号:TP391 **文献标识码**:A

Modeling Language for the Integration of Parallel Discrete Event Simulation Modeling and Parallel Model Checking

*XIA Wei*1, [1,2], *YAO Yi-ping*[1], *MU Xiao-dong*[2]
(1. College of Computer, National Univ. of Defense Technology, Changsha 410073, China;
2. Department of Computer Science, Xi'an Hi-Tech Institute, Xi'an 710025, China)

Abstract: The correctness and reliability of parallel discrete event simulation models play an important role in the development of PDES. Most of the existing PDES developing environments do not support model verification at present. The completeness and high effectiveness of parallel model checking helps it successful in making its way into industrial tools. While PDES and PMC are implemented in different modeling languages, in the current practice, to check whether or not a PDES model contains any errors, a prototype that is solely written for model checking purposes has to be built. This process is an onerous, time-consuming and error-prone task. The current research presented a modeling language, namely Extended Event Graph (EEG) to combine PDES and PMC, and it extended the classical event graph in aspect of synchronization. This modeling language makes it possible for users to achieve PDES and PMC with only one model via transformation mechanisms, thus saving both the time and effort of developers. The experimental results confirm the validity of this language and it can support both PDES and PMC.

Key words: parallel discrete event simulation; model checking; event graph; logical process paradigm; Distributed and Parallel Verification Environment modeling language; model transformation

并行离散事件仿真(Parallel Discrete Event Simulation, PDES)在执行效率方面具有得天独厚的优势,近年来得到了各国的高度重视,成为仿真界的主要研究领域之一,是研究复杂系统的重要手段[1]。在PDES应用开发早期检验出模型存在的错误,不仅能够提高仿真系统的正确性,而且能够提高仿真应用的开发效率。然而,目前大多数PDES开发环境都不具备模型验证功能。

形式化验证方法是基于严格的数学描述与推理的方法,能够对所研究的问题域提供100%的覆盖率,对模型的验证更可信、更具有说服力,是对模型进行验证最有效的方法之一[2]。模型检验(model checking)是一种自动验证有限并发系统的方法,是形式化验证方法中应用最广泛的一

* **收稿日期**:2011-06-16
基金项目:国家自然科学基金资助项目(61170047,61170048);国家教育部博士点基金资助项目(200899980004)
作者简介:夏薇(1983—),女,博士生。

种[3]。但是,模型检验需要非常密集的计算,因而串行模型检验方法具有很高的内存要求,即便是采用了状态空间爆炸缓解技术后,许多大规模模型的验证花费数天也不能完成。随着高性能计算机的不断发展,许多研究人员开始研究并行模型检验(Parallel Model Checking, PMC)方法,通过利用更多的计算资源来成倍地提高模型检验的效率,该方法已经在工业界中得到了成功的应用[4]。许多模型检验工具都具有并行版本,例如 Murφ 用 1 个计算节点检验协议 SCI 的时间是运用 32 个计算节点的 26.6 倍[5]。因此,PMC 方法为 PDES 应用的模型验证提供了可行性。

PDES 通常采用逻辑进程(Logical Process, LP)范型[6],目前 PDES 开发环境中基于 LP 范型构建的模型大都是基于高级编程语言(如 C, C ++ ,Java)或并行仿真语言的,缺乏严格的语义;而模型检验工具的输入模型大多是基于各种自动机或 Petri 网的数学模型。由于缺乏支持并行离散事件仿真和模型检验的统一的建模语言,现阶段要采用模型检验对并行离散事件仿真模型进行验证,要么需要仿真模型满足检验工具的要求,从而加大了仿真模型开发的难度;要么需要形式化定义程序语言的语义,处理复杂的数据结构和动态特性,从而增加了模型检验的难度。因此,需要一种统一的建模语言作为 PDES 和 PMC 之间的桥梁,使得用户只需要建立一个模型就能同时实现 PDES 和 PMC,降低 PMC 的应用难度,提高 PDES 模型的逻辑正确性。

事件图(Event Graph, EG)是 Schruben 于 1983 年首先提出的一种图形化的离散事件仿真建模方法[7],它通过事件以及事件之间的逻辑、时序关系来刻画离散事件系统的动态特性。作者已经提出并实现了基于事件图的并行离散事件仿真建模方法[8]。因此,针对目前 PDES 开发环境大都不具备仿真模型验证能力的问题,提出了一种支持 PDES 和 PMC 的统一的建模语言——扩展事件图,该建模语言作为桥梁,可以将并行模型检验工具 DiVinE (Distributed and Parallel Verification Environment)[9]集成到 PDES 开发环境中,从而使 PDES 开发环境具备仿真模型验证能力。

1　事件图与逻辑进程之间的映射关系

事件图是一种基于事件调度策略的图形化离散事件仿真建模方法[7],它将事件作为仿真模型的基本模型单元,按照事件发生的先后顺序不断执行相应的事件。而 LP 范型的建模思想是在事件调度机制"之上"实现的,它明确地使用了事件调度法所定义的事件表和时间推进机制,整个仿真的生命周期由事件计算的序列组成,它可被看成是一种微型的面向事件仿真[6]。一个完整的事件图模型可被看作是由多个子事件图组成的,每个子图对应一个 LP。关键的差别在于事件图模型是共享状态变量的,而 LP 模型与之相反,为了能够并行处理 LP, LP 之间是不能共享变量的,LP 之间只能通过发送消息来通知相关 LP 进行事件调度、修改局部状态变量,而且需要提供额外的机制实现同步控制。

然而,事件图本身并没有直接提供交互机制,在目前支持事件图建模的 Sigma™[10]、Simkit[10]和 Ptolemy II[11]三个仿真平台中,Simkit 和 Ptolemy II 在实现不同的事件图交互时分别采取了不同机制,而 Sigma™没有提供交互机制,不支持层次化建模方式。例如在 Simkit 中,用户可以使用 Listener Event Graph Objects (LEGOs)框架提供的事件监听模式(SimEvent listener pattern)、性质变化监听模式(property change listener pattern)、适配器模式(adapter pattern)三种不同的模式来实现不同子事件图之间的同步[12]。Ptolemy II 对事件图进行了扩展,它根据面向执行者的语义,通过调控器(director)来层次化组合基于执行者语义的模型,如 DE (actor-oriented discrete-event)模型和有限状态机模型等[11]。但是,它们目前还不支持基于 LP 范型的并行仿真。在 LP 范型中,不同 LP 之间的通信是通过发送带有事件发送者、接收者以及时间戳信息的事件来实现的。因此,为了能够直观、简洁地对基于 LP 范型的并行离散事件仿真建模,并且便于将该模型转换为可执行的代码,屏蔽并行离散事件仿真的技术细节,本文需要对基本事件图进行扩展,增加其对不同子图间交互机制的表达能力。

2　事件图与 DVE 模型之间的映射关系

系统建模语言可分为基于状态的建模语言和基于事件(行为)的建模语言两种。事件被定义为系统中一种不具持续性的动作,它在某个特定的时间点上发生,可以使系统从一个状态迁移到另一个状态;状态则是对系统在某一个时间段内的描述,在一个持续时间内保持不变。事件图是事件驱动的建模范型,通过事件或行为的执行使得系统从一个状态变化到其他的状态;而 DVE 语言则是基于状态的建模语言[9],通过状态以及状

态的改变来描述系统。但是事件和状态之间有着紧密的联系:系统的每一个状态可以由一对事件来定义,一个发生事件表示进入该状态,以及一个结束事件表示离开该状态。图 1 表示了事件与状态之间的关系,其中(a)图为事件驱动的建模范型,(b)图为基于状态的建模范型,事件和状态在这两种语言中是互补的。

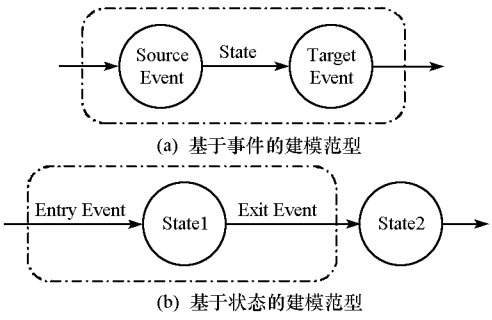

图 1 事件与状态之间的关系

Fig. 1 The relationship between events and states

根据事件和状态之间的关系,事件图到 DVE 语言的映射过程简单地说就是:边转换为顶点,顶点转换为边。一个事件图模型可以看作是由多个子事件图组成的,DVE 中的基本建模单元是系统(system),它由多个进程(process)组成[13],因而每个子事件图对应于 DVE 中的一个进程 process。DVE 中的状态迁移是进程的一个状态到另一个状态的转换,包括卫式(guard)、同步(synchronization)以及结果(effect)[13]。事件图中的事件发生所引起的系统状态的变化对应于 DVE 中的状态迁移 trans,事件发生的条件对应于 DVE 中状态迁移的卫式 guard,事件发生所引起的状态迁移函数($s = f_A(s)$)对应于 DVE 语言的 effect。

事件图中的调度(取消)边能够在事件节点之间传递参数[7],这种能力使得建模者能够通过参数表示同类子系统的不同实例,而不需要将这些实例一一表示出来,这极大地简化了建模过程。例如,在流水线系统的事件图模型中,到达的顾客由 n 个工作站串行地处理其服务需求,每个工作站相当于一个多服务员(或单服务员)排队系统[10]。当顾客在一个工作站的服务完成之后,该顾客将进入到下一个工作站接受服务。当顾客在所有工作站的服务都完成之后,他将离开系统。事件图模型可以通过事件所带的参数表示该事件发生的工作站标号,该参数值由调度边上的表达式来确定。由于 DVE 模型不具备这种通过参数表示多个工作站的能力,在建模时需要将每个工作站的状态变化都一一表达出来,因此事件图模型中的每个工作站都对应于 DVE 模型中的一个进程(即有 n 个工作站,就有相应的 n 个 DVE 进程)。由此也可以看出,采用事件图建模语言能够简化 DVE 的建模过程。

在 DVE 建模语言中,不同进程之间通过 channel 实现交互以及数据传递功能[13],因此,综合考虑事件图与 LP 范型及 DVE 模型之间的映射关系,本文通过添加通道来表示不同子事件图之间的交互以及数据流关系,从而便于将基于事件图的模型自动转换为可执行的并行仿真代码和模型检验工具识别的模型,屏蔽并行离散事件仿真和模型检验的技术细节。

3 扩展事件图的定义

扩展后的事件图的形式化定义如下。

定义 1 扩展事件图(Extended Event Graph, EEG)是由事件和事件之间的关系组成的有向图 $EEG = (E, S)$。E 表示事件的集合,$E = <M, F, P, SC>$,由图中的顶点表示,其中:

$M = \{M_e : \forall e \in E\}$,有限的状态集合;

$F = \{f_e : M \rightarrow M, \forall e \in E\}$,每个事件的状态变迁函数;

$P = \{P_e, \forall e \in E\}$ 每个顶点的参数列表;

$SC = \{SC_e \in \{?, !\}, \forall e \in E\}$ 表示同步通道的集合,? 表示接收,! 表示发送;

S 表示调度边或取消边的集合,$S = <L, From, To, C, T, PR, A>$,由有向边来表示,其中:

$L = \{Schedule, Cancel\}$ 表示边的类型;

$From = \{From_e : \forall e \in E\}$ 表示调度事件;

$To = \{To_e : \forall e \in E\}$ 表示被调度事件;

$C = \{c_s \in \{True, False\}, \forall s \in S\}$,每条边对应的条件函数;

$T = \{t_s \in \mathscr{R}_0^+ \quad \forall s \in S\}$,每条边的时间延迟;

$PR = \{PR_s \in \{Lowest, Lower, Low, Default, High, Higher, Highest\}, \forall s \in S\}$ 表示每条边的优先级;

$A = \{A_s, \forall s \in S\}$ 表示每条边的属性值列表。

若用 SG 表示子图,即 $SG \subseteq EEG$,那么一个 $EEG = \{SG_1, SG_2, \cdots, SG_n\}$,$n \in N$,每个子图 SG 对应于一个 LP 或者是 DVE 中的进程 Process。

定义 2 系统状态(σ)是一个函数(e)映射,其中 σ 表示状态,e 表示事件。所有表达式的语义定义为:

$[\![false]\!](e) = 0$

$[\![true]\!](e) = 1$

$[\![number]\!](e) = number$

$[\![id[expr]]\!](e) = e(id)([\![expr]\!](e))$

$[\![(expr)]\!](e) = [\![expr]\!](e)$

$[\![id]\!](e) = e(id)$

$[\![expr_1\ binary\text{-}op\ expr_2]\!](e) = [\![expr_1]\!](e)\ binary\_op\ [\![expr_2]\!](e) \cdots binary\_op \in \{<, \leqslant, ==, !=, +, -, *, /, \%, \&\&, ||\}$,

Boolean operators(!=, ==) return 0 or 1.

$$[\![id_1.id_2]\!](e) = \begin{cases} 1 & e(id_1) = id_2 \text{ the event } id_2 \text{ in its sub-graph } id_1 \\ 0 & \text{otherwise} \end{cases}$$

定义 3　系统变迁定义为:

$transition \equiv id_1.id_2\{condition\ sync\ function\}$ 且 $function \equiv assignment_1, \cdots, assignment_n$,其中 id_1 表示子图的名称,id_2 表示引起变迁的事件,$id_1.id_2$ 表示子图 id_1 中 id_2 事件发生。

变迁通过以下 3 种方式来改变系统状态:

1) 改变子图的状态;

2) 改变通道缓冲区的内容;

3) 改变变量的值。

因此,变迁的语义可以定义为:

1) $[\![id_1.id_2]\!](e) = e[parent(transition)/id_2]$

2) $[\![sync]\!](e)$

$$= \begin{cases} e & \text{if } sync \equiv \varepsilon \\ e[id_{ch}/id_{ch} \cdot e(syncvalue)] & \text{if } sync \equiv sync\ id_{ch}!\ syncvalue \\ e[id_{ch}/tail(id_{ch}), syncvalue/head(id_{ch})] & \text{if } sync \equiv sync\ id_{ch}?\ syncvalue \end{cases}$$

3) $[\![assignment]\!](e)$

$$= \begin{cases} [\![id_{var}/[\![expr_{var\ lue}]\!](e)]\!](e) & \text{if } assignment \equiv id_{var} = expr_{value} \\ [\![id_{var}([\![expr_{index}]\!](e))/[\![expr_{value}]\!](e)]\!](e) & \text{if } assign\ ment \equiv id_{var}[expr_{index}] = \mathrm{expr}_{value} \end{cases}$$

4　实验分析

4.1　基于 EEG 的 PDES 和 PMC 过程

ANTLR (Another Tool for Language Recognition)[14] 是根据一种可以嵌入如 Java, C++等辅助代码段的文法来自动构筑出相对该文法的语法分析器的语言工具框架。ANTLR 可以根据需要生成其中任何一种语言的语法分析器,程序员通常使用它来为领域语言创建语法分析器。本文使用 ANTLR 根据 EEG 的语法结构自动生成 EEG 建模语言的语法分析器,使得用户根据 EEG 建模规则建立的模型可分别直接转换为基于 LP 范型的 PDES 模型和 DVE 模型。根据 EEG 的定义,本文设计的 EEG 的 ANTLR 语法表示如图 2 所示。

```
//Extened event graph grammar
EEG: = Declaration EventPartition EdgeDefList
Declaration: = ε | Declaration StateVariableDecl
               | Declaration Parameters | Declaration
               AttributeDecl
StateVariableDecl: = ε | ( < TypeName > < DeclId >
(InitDecl)) +
ParametersDecl: = ε | ( < TypeName > < DeclId >
(InitDecl)) +
AttributeDecl: = ε | ( < TypeName > < DeclId >
(InitDecl)) +
TypeName: = int | string | double | integer | Boolean | Random
Event: = < EventName > < % CausedState >
(StateTransition) (Parameter)
StateTransition: = Expresstion
SubEG: = ε | EventName(",EventName) *
EdgeDefList: = ε | Edge(",#Edge) *
Edge: = < EdgeType > < SourceEvent > < TargetEvent >
        < · Conditions >
        < @ Priority > ($ TimeDelay) (Attribute)
EdgeType: = Schedule | Cancel
Conditions: = ε | (Expression Compare Expression) +
Compare: = > | > = | < | < = | = = | ! =
Priority: = default | lowest | lower | low | high | higher | highest
Channel: = ε | send | receive
Send: = (Condition)? < ChannelExpr! >"("EventList")"
Receive: = < ChannelExpr? >"("EventList")"
EventList: = (EventName"," TimeStamp (";" EventName"," 
TimeStamp) *)?
EEG: = SubEG(",SubEG) *
```

图 2　ANTLR 的 EEG 语法

Fig. 2　The EEG grammar in ANTLR

本文在定义 EEG 语法时使用了一些特殊的算子,如表 1 所示。

基于 EEG 模型的 PDES 和 PMC 过程如图 3 所示。1)根据系统描述建立 EEG 模型。2)EEG 模型输入到转换器,首先由 ANTLR 自动生成的 EEGlexer 和 EEGparser 判断该模型语法的正确性;若该模型有效,则分别由根据 LP 范型和 DVE 模型原语建立的 EEgClassBuilder 类被解析为并行离散事件仿真器和 DiVinE 验证工具所识别的可执行模型文件。3)生成的可执行文件分别在仿真器和验证器上运行,输出结果。通过这种转换器进行 PDES 和 PMC 的方法使得仿真领域的专家或仿真应用的开发人员使用一种建模语言就可以对仿真系统进行性能分析和形式化验证。

表 1 EEG 中的特殊算子

Tab. 1 Some special operators in EEG

Operator	Meaning
%	The state caused by an event occurrence
@	The priority of schedule or cancel
·	The condition of schedule or cancel
$	Time delay
#	The combine of several event that scheduled by one event
!	Send channel
?	Receive channel

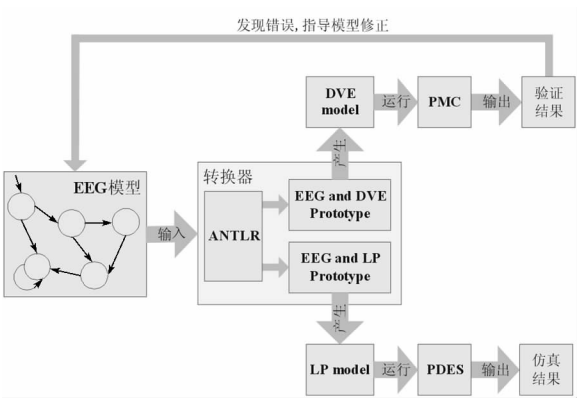

图 3 基于 EEG 的 PDES 和 PMC 过程

Fig. 3 The EEG based PDES and PMC

4.2 实验环境及结果

本文实验基于离散事件仿真和模型检验领域中经典的哲学家就餐问题,实验主要关注两个转换问题:即 EEG 是否能够自动转换为基于 LP 范型的并行仿真可执行代码和 DVE 模型。

实验 1 EEG 到 LP 模型的转换实验。本文首先根据 EEG 语法建立了会进入"死锁"状态的 5 个哲学家的模型(如图 4 所示),通过本文提出的转换机制,该模型自动转换为能够由课题组开发的银河速跑(YH-SUPE)[15]仿真实验平台进行 PDES 的可执行代码。仿真平台的运行环境是主频 2.72GHZ,内存 2G 的双核 PC 机。本文基于串行算法和时间弯曲并行算法对转换后的 LP 模型进行了测试,通过改变模型规模和处理单元(Processing Element,PE)数量对两种算法进行了比较,对比结果如图 5 所示。实验结果表明,在本文提出的转换机制下,EEG 模型能够成功地转换为 LP 模型,并且在并行仿真算法下能够有效提高仿真的效率。但是在实验过程中,并不是每次的仿真运行结果都会显示出模型具有死锁状态,从而也可以看出,虽然仿真模型没有语法层面上的错误,能够通过编译后执行,但并不能代表其在逻辑上是正确的,因此,仿真模型的逻辑正确性验证十分必要。

```
//[deadlock]
model Dining Philosopher
process Dining Philosopher[Fork(i):Integer - > Fork(i):1]
initial Dining Philosopher LP{Dining Philosopher[pil(n):Thinking,think,0,3]}
//n is the number of philosopher, Thinking is the trigger event,
//the number 0,3 denotes the philosopher numbered from 0 and total number is 3
TakeLeftFork(i)% one (Fork(i) =0):=
{(Schedule TakeRightFork(i)) @ default. if{Fork((i +1)%n) ==1}$1.0}
TakeRightFork(i)% eat(Fork((i+1)%n) =0):=
{(Schedule DropLeftFork(i))@default. $3.6}
DropLeftFork(i)% finish(Fork((i+1)%n) =1):=
{(Schedule DroprightFork(i))@default. $0.0}
DroprightFork(i)% think(Fork(i) =1):=
{(Schedule TakeLeftFork(i))@default. if{Fork(i) ==1}
$0.0}
```

图 4 带有死锁状态的哲学家就餐问题模型

Fig. 4 The dining philosopher model with deadlock

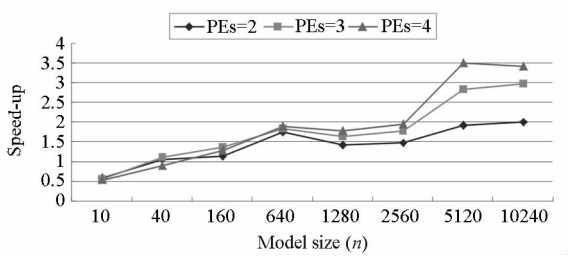

图 5 并行仿真的结果

Fig. 5 The PDES results

实验 2 EEG 模型到 DVE 模型的转换。本实验仍然使用图 4 所示的模型作为输入,通过转换机制,该模型自动转换为能够由 DiVinE 工具检验的 DVE 模型,验证结果如图 6 所示,验证结果显示该模型具有死锁状态。在此基础上,本文通过改变哲学家就餐问题的模型规模,将 DiVinE 与 UPPAAL 模型检验工具[16]以及 PRISM 概率模型检验工具[17]进行比较。DiVinE 的测试环境是双向 2.53GHz 的 4 核 Xeon 处理器 E5540,8GB 内存,内核版本 2.6.18 机群系统,UPPAAL 和 PRISM 的测试环境是主频 2.72GHZ,内存 2G 的双核 PC 机。DiVinE 的输入为基于扩展事件图的模型,UPPAAL 的输入为时间自动机模型,PRISM

检验器的输入是基于 PRISM 语言的模型。从图 7 的对比结果可以看出,随着模型规模的不断增大,并行模型检验方法显示出了明显的优势,证明并行模型检验方法能够有效地提高仿真模型验证的效率,此外每次验证结果都显示该模型会进入死锁状态。实验 2 证明了 EEG 模型能够成功转换为 DiVinE 工具有识别的建模语言,同时并行模型检验方法能够有效地提高仿真模型指定性质验证的完备性和效率。

```
[root@ib-cn1 divine-mc-1.4]# ./_build/tools/divine-mc verify \-\-report examples
/phils.1.dve
WARNING: deadlock state reached (at most one deadlock per thread is reported):
[fork:(1|1|1|1|1)]
phil_0:[one]
phil_1:[one]                    死锁状态，one表示每个哲学家都拿到一只餐叉
phil_2:[one]
phil_3:[one]
phil_4:[one]
seen total of: 242 states (0 accepting) and 806 transitions
encountered total of 0 errors and 1 deadlocks          模型的状态空间
Workers: 2
Initial-Storage-Size: 4097
Storage-Growth-Factor: 2
Handoff-Threshold: 50
Input-File: examples/phils.1.dve
Trail-File: examples/phils.1.trail
Algorithm: Reachability
Order: BFS
Partitioning: Static
Generator: DiVinE
Storage: Pooled-Partitioned
User-Time: 0.004999
System-Time: 0.001999
Wall-Time: 0.23981
Termination-Signal: 0
Memory-Used: 57316
Finished: Yes
```

图 6　DiVinE 对模型的验证结果

Fig. 6　The verification results in DiVinE

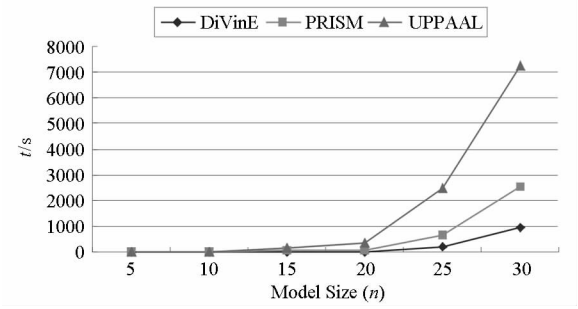

图 7　模型验证结果对比图

Fig. 7　The comparison of these verification results

本节通过两个实验证明了 EEG 模型能够成功地转换为用于 PDES 的 LP 模型和用于 PMC 的 DVE 模型,用户只需建立一个系统模型就能够同时实现 PDES 和 PMC,不仅能够减少仿真应用开发的工作量,而且能够帮助开发者在建模过程中尽可能早地发现模型中的逻辑错误,提高仿真应用的逻辑正确性。

5　总 结

本文在对事件图与 LP 范型和 DVE 模型之间的映射关系分析的基础上,对事件图进行了相应的扩展,提出了一种支持 PDES 和 PMC 的统一的建模语言——扩展事件图,在此基础上实现了扩展事件图模型与并行仿真模型和 DVE 模型之间的转换。文章最后通过实验验证了 EEG 模型能够成功地转换为用于 PDES 的 LP 模型和用于 PMC 的 DVE 模型,用户只需建立一个系统模型就能够同时进行 PDES 和 PMC。下一阶段的工作是实现基于 EEG 的可视化建模环境。

参 考 文 献:

[1] 姚益平, 张颖星. 基于并行处理的分析仿真解决方案[J]. 系统仿真学报. 2008, 20(24): 6617 - 6621.

[2] 王维平,等. 离散事件系统建模与仿真 [M] (第二版). 北京: 科学出版社,2007.

[3] Clarke E M, Emerson E A, Sifakis J. Model Checking: Algorithmic Verification and Debugging [J]. Communications of the ACM, 2009, 52(11): 75 - 84.

[4] Brim L, Barnat J. Tutorial: Parallel Model Checking (Extended Abstract) [G]// LNCS 4595, 2007: 2 - 3.

[5] 邝宏斌,罗贵明. 并行软件模型检测[J]. 计算机工程, 2008, 34(19): 23 - 29.

[6] Fujimoto R M. Parallel and Distributed Simulation Systems[M]. A Wiley-Interscience publication. John Wiley & Sons, Inc, 2000, the United States of American. 2000.

[7] Schruben L. Simulation Modeling with Event Graphs [J]. Communications of the ACM, 1983, 26(11): 957 - 963.

[8] Xia W, Yao Y P, Mu X D, et al. Research on Technologies of Event Graph Based Parallel Discrete Event Simulation [C]// Proceedings of the 6th International Conference on Networked Computing and Advanced Information Management (NCM' 2010), Korea: 2010:69 - 74.

[9] Barnat J, Brim L, Rockai P. DiVinE 2.0: High-Performance Model Checking [C]//Proceedings of the 2009 International Workshop on High Performance Computational Systems Biology (HiBi 2009), Trento: IEEE Computer Society Press, 2009:31 - 32.

[10] Buss A H. Basic Event Graph Modeling [J]. Simulation News Europe. 2001, 31(1): 1 - 6.

[11] Feng T H, Lee E A, Schruben L W. Ptera: An Event-Oriented Model of Computation for Heterogeneous Systems [C]//Proceedings of the 10th ACM International Conference on Embedded Software (EMSOFT), Scottsdale, Arizona, USA, ACM Press, 2010: 219 - 228.

[12] Buss A H, Sánchez P J. Building Complex Models with LEGOS (Listener Event Graph Objects) [C]//Proceedings of the 2002 Winter Simulation Conference (WSC '02), San Diego: 2002: 732 - 737.

[13] Šimeček P. The DVE Modeling language (Chapter 2 of Pavel ś Master thesis) [D]. Brno: Masaryk University, 2006: 2 - 15[2011 - 02 - 16]. http://divine.fi.muni.cz/dve.pdf

[14] Parr T. The Definitive ANTLR Reference-Building Domain-Specific Languages [M]. Texas, USA: The Pragmatic Bookshelf, 2007.

[15] YH-SUPE2.0 用户手册. 国防科技大学计算机学院, 2010.

[16] Behrmann G, David A, Larsen K G. A Tutorial on Uppaal [G]// Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT'04), LNCS 3185, 2004: 200 - 236.

[17] Hinton A, Kwiatkowska M, Norman G, et al. PRISM: A Tool for Automatic Verification of Probabilistic Systems [G]//Proc. of 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS06), LNCS 3920, Heidelberg: Springer-Verlag, 2006: 441 - 444.