软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(6):1429−1443 [doi: 10.3724/SP.J.1001.2012.04047]

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

基于事件图的离散事件仿真模型并行检验方法*

夏 薇[1,2+], 姚益平[1], 慕晓冬[2], 柳 林[3]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](第二炮兵工程大学,陕西 西安 710025)

[3](海军装备研究院,北京 100161)

Parallel Model Checking for Discrete Event Simulation Models Based on Event Graphs

XIA Wei[1,2+], YAO Yi-Ping[1], MU Xiao-Dong[2], LIU Lin[3]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](Xi'an Hi-Tech Institute, Xi'an 710025, China)

[3](Naval Academy of Armament, Beijing 100161, China)

+ Corresponding author: E-mail: weiwei32329@163.com

Xia W, Yao YP, Mu XD, LIU L. Parallel model checking for discrete event simulation models based on event graphs. *Journal of Software*, 2012,23(6):1429−1443. http://www.jos.org.cn/1000-9825/4047.htm

Abstract: Informal verification methods for simulation models are vulnerable to subjective ingredients. The traditional model checking method has difficulty dealing with large-scale simulation models because of the state space explosion. The parallel model checking (PMC) method has been accepted and successfully implemented in industrial tools because of the completeness and high efficiency. Unfortunately, it is hard to use as it involves several difficulties, such as formal specifications, logics, and parallel computing. To solve the above problems, a parallel model checking method for simulation models based on event graphs is proposed in this paper. This method extends event graphs in synchronization and defines the syntax and semantics of the extended event graphs. It transforms the extended event graphs to distributed and parallel verification environment (DVE) model, and PMC method is successfully applied to simulation model verification filed. With this method, simulation participators can verify simulation models with PMC method without learning new formal modeling languages. The efficiency and completeness of simulation model verification are improved. The experimental results show the validity of this method, and the method can improve the application of PMC method in simulation field.

Key words: discrete event simulation model verification; parallel model checking; event graph; DVE modeling language; model transformation

摘 要: 非形式化仿真模型验证方法易受主观因素的影响且具有不完备性,而传统的形式化模型检验方法由于受到状态空间爆炸问题的影响,很难处理大规模的仿真模型.并行模型检验方法以其完备性、高效性已经在工业界中得到了成功的应用,但是由于涉及到形式化规约、逻辑学以及并行计算等多项技术,应用难度较大.针对上述问题,提出了基于事件图的离散事件仿真模型并行检验方法.该方法首先对事件图在模型同步方面进行了扩展,给出了扩

* 基金项目: 国家自然科学基金(61170048); 国家教育部博士点基金(200899980004)
收稿时间: 2010-11-29; 修改时间: 2011-02-17; 定稿时间: 2011-04-28

展事件图的形式化定义、语法及语义;然后将扩展事件图模型转换到分布并行验证环境的 DVE 模型,成功地将并行模型检验方法应用于仿真模型验证领域.该方法使得仿真人员无须学习新的形式化验证语言就能采用并行模型检验方法对仿真模型进行形式化验证,可降低模型并行验证的难度,从而有效提高模型验证的效率和完备性.实验结果表明了该方法的有效性,有利于扩展并行模型检验方法在仿真领域中的应用.

关键词: 离散事件仿真模型验证;并行模型检验;事件图;DVE 建模语言;模型转换

中图法分类号: TP391 **文献标识码**: A

仿真模型的正确性和可信度对仿真应用的发展起着决定性作用,能够快速、完备地验证大规模仿真模型的正确性和可信性是现实的迫切需要.仿真模型的校验问题近年来受到了仿真界的高度重视,逐渐发展为一套完整的校验、验证和确认(validation, verification & accreditation,简称 VV&A)技术理论体系.

VV&A 方法可以分为非正式方法、静态方法、动态方法以及形式化方法 4 大类[1,2]:非正式方法依赖于人工推理和主观判断,没有严格的数学描述和分析推理;静态方法用于评估静态模型设计和源代码,它不需要模型的机器执行,应用于检验模型的静态方面;动态方法则需要模型运行,大多数动态方法需要加入模型探测器,以收集模型运行中的行为信息;而形式化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,是对模型进行 VV&A 的最有效的方法,但是由于数学证明技术的难度及局限性,这种方法在仿真领域尚未得到广泛的应用[2].表 1 对这 4 类 VV&A 方法进行了比较[1,2].具有代表性的 VV&A 方法研究工作主要有:Hermann 根据直觉等方法验证仿真模型和真实对象之间的一致性[3];Aigner 将因素分析法、回归分析法应用于模型验证,Balci 和 Sargent 提出在评估工作中“费用-风险”的权衡分析[3];Balci 综述了模型可信度评估的各种方法,如静态分析、动态测试、约束分析和理论证明等[1],Sargent 从 1989 年开始在每年的冬季仿真会议的特邀报告中综述了目前在仿真领域使用较多的 VV&A 方法,如比较法、极端条件测试、历史数据确认、灵敏度分析等[4].国内一些研究者也对 VV&A 方法进行了长期的研究,提出了一些新的方法:哈尔滨工业大学的王子才院士在 VV&A 方法方面进行了长期的研究,提出了复杂仿真系统评估的理论框架,设计实现了 VV&A 辅助工具,主要采用了数据预处理、图灵测试法、统计学方法、特征匹配法、频谱分析法、专家评定法等[3].可以看出,目前仿真模型 VV&A 方法研究主要集中在非正式方法、静态方法以及动态方法,形式化方法的研究很少[5].但是模型中的错误大多不是发生在语法层面,不可能在编译阶段完全排除:依靠人工的方式进行错误排除易受主观因素的影响,而静态及动态分析技术是针对源程序或其语言子集的通用错误(例如内存泄漏错误、空指针引用、数组访问越界等)进行的[6],没有涉及到模型逻辑层面上的正确性(如模型有无死锁、活性、公平性等).可以看出,要想利用前 3 类方法对模型进行完备地验证是不可行的,它们只能证明错误存在,而不能证明错误不存在.因此,为了提高仿真模型的正确性以及仿真结果的可靠性,对仿真模型的形式化验证方法进行研究十分必要.

Table 1 Comparison of VV&A methods

表 1 VV&A 方法比较

方法	特点						
	标准化程度	复杂性	人力资源	计算机资源	效果	附加软件	重要性
非正式方法	较低	低	最高	低	有限	否	高
静态方法	适中	适中	偏低	适中	较好	否	高
动态方法	低	偏高	偏高	高	较好	是	高
形式化方法	高	高	高	高	最好	是	最高

模型检验(model checking)是一种自动验证有限并发系统的方法,它是应用广泛的一种形式化验证方法[7].经过 28 年的发展,模型检验方法在采用某些智能算法的情况下可以检验的系统状态空间达 10^{476}[8],在 VLSI 电路、通信协议、实时嵌入式系统和安全算法等验证方面得到了成功的应用[8].但是传统的模型检验技术具有很高的内存要求,需要非常密集的计算,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,许多研究人员开始通过使用功能强大的共享内存多处理器(如多核

机)或在多台机器上分布共享内存(如工作站集群)的方法,利用更多的计算资源成倍地提高模型检验的效率[9].在并行处理器以及工作站集群上的测试结果显示,无论是在问题规模的扩大,还是计算时间的缩短,以及处理器数量的可扩展性方面,并行模型检验方法较之串行方法都有明显的优势,该方法已经在工业界中得到了成功的应用[9].许多早期的模型检验工具都有并行版本,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[10].因此,并行模型检验方法为大规模离散事件仿真应用的模型验证提供了可行性.

目前,大多数离散事件仿真系统模型是基于高级编程语言(如 C,C++,Java)或 DEVS 范型实现的[11],而模型检验工具的输入模型一般都是基于各种自动机、Petri 网以及进程代数等数学模型[8],缺乏支持离散事件仿真和模型检验的统一的建模语言.因此在现阶段,对仿真模型的形式化模型检验方面的研究主要可以分为两个方向:一是根据模型检验的原理,研究适合于不同仿真模型的模型检验算法,但是对基于高级编程语言实现的模型使用模型检验方法较为复杂,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[7];另一种则是研究模型转换算法,应用比较成熟的模型检验工具对仿真模型进行验证,其研究思路是[12]:

(1) 选择合适的建模语言建立系统仿真模型;

(2) 选择一种合适的模型检验工具,掌握该工具所允许的输入模型样式以及性质规约语言;

(3) 如果系统仿真模型与模型检验工具的输入模型不一致,则进行模型转换工作;

(4) 最后,输入模型进行验证.

事件图(event graph,简称 EG)是一种图形化的离散事件仿真建模方法[13],它精简的表达方式以及强大的建模能力使其成为离散事件建模与仿真的理想工具,近年来在离散事件仿真领域得到了广泛的应用.自动机、Petri 网以及进程代数等可作为已有模型检验工具输入的建模语言在大型、复杂系统的仿真模型中应用较为困难,主要是因为他们描述能力的增强会在某种程度上增加对这些语言进行分析的难度,增加对系统模型性质的判断和计算的困难[13].尽管 DEVS 范型也是应用较为广泛的一种离散事件建模语言,但它是类自动机方法,范式规则非常复杂.

DiVinE(distributed and parallel verification environment)是马萨里克大学(Masaryk University)开发的一个开源、可扩展的模型检验工具集,它的不同版本可以支持基于不同平台的验证[14,15].例如,DiVinE Cluster 适用于分布式内存的工作站集群平台,DiVinE Multi-cores 是运行于共享内存多处理器平台上的并行模型检验工具,而 DiVinE CUDA 是使用 GPU 加速的验证工具等 [14,15].这些并行模型检验方法已经在工业界中得到了成功的应用[9].但是并行模型检验方法涉及到形式化规约、形式化验证、逻辑学以及并行计算等多学科,而仿真参与者大多数是面向领域的专家,因而该方法直接在仿真领域应用比较困难.

现阶段,没有直接支持仿真模型的形式化模型检验的方法或工具,仿真人员如果要对仿真模型进行形式化验证,需要专门建立适用于模型检验器的模型,这不仅浪费时间而且容易出错.鉴于事件图强大的建模能力以及 DiVinE 并行模型检验的能力,本文选择仿真模型形式化模型检验的第 2 个研究方向,利用成熟的并行模型检验工具对仿真模型进行关键性质的验证.由于事件图模型与 DiVinE 工具的输入模型 DVE 模型不一致,因而需要进行事件图模型到 DVE 模型的转换,模型转换算法是研究的重点内容.

针对目前主流的非形式化仿真模型验证方法存在的不完备问题,以及传统串行形式化验证方法效率低的问题,本文将并行模型检验方法应用于基于事件图的离散事件仿真模型验证中,以提高仿真模型验证的完备性及效率.本文的主要贡献在于:

(1) 事件图是一种用于离散事件建模与仿真的图形化建模语言,DiVinE 是一种基于时间自动机的并行模型检验工具,本文首次尝试将并行形式化验证方法用于离散事件仿真模型的验证;

(2) 本文通过分析、比较事件图模型与 DVE 模型间的映射关系,对事件图在模型同步方面进行了扩展,并对扩展事件图的语法及语义进行了形式化定义;

(3) 在此基础上,提出了将扩展事件图模型转换为 DVE 模型的方法.这种通过转换机制实现的仿真模型形式化验证方法,使得仿真人员不需要学习新的形式化验证语言,只需要建立一个事件图模型就能够既进行离散事件仿真又利用并行模型检验方法对离散事件仿真模型进行形式化验证.该方法不仅

充分利用了事件图的简单易用性,同时还能够发挥并行模型检验方法在完备性以及高效性方面的优势,从而有效地提高仿真模型验证的完备性及效率.更重要的是,这种通过转换机制的方法,对仿真应用开发者隐藏了模型检验的细节,使他们只需要关注于仿真模型的构造,不用关注模型检验算法的细节,降低了基于并行模型检验的仿真模型验证的难度,简化了模型检验的过程,有利于扩大形式化验证技术在仿真领域的应用范围.

本文第 1 节、第 2 节对事件图以及 DVE 建模语言进行简单介绍.第 3 节根据事件图以及 DVE 建模语言之间的映射关系,给出扩展事件图的形式化定义以及其语法结构和语义.第 4 节对基于 ANTLR 语法分析器的扩展事件图到 DVE 模型的转换器进行介绍,并通过实验证明该转换方法的有效性.最后总结全文并对下一步的工作进行展望.

1 事件图

事件图是 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[13],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.事件图是一个有向图,主要由以下元素组成[13]:

1) 状态变量:表示系统的状态;
2) 事件:有向图中的每个顶点表示事件(或状态转移);
3) 事件之间的逻辑、时序关系:有向图中每条有向边表示一对事件之间的逻辑、时序关系,即事件之间的调度关系:当源事件 A 发生时,系统状态 s 将改变为 $s=f_A(s)$,如果该有向边上的条件(i)成立,那么目标事件 B 将在一定的时间延迟 t 后被调度,如图 1(a)所示.布尔条件和时间延迟是有向边的辅助元素,有的调度关系不需要条件或者时延.

为了丰富事件图的表达能力,Schruben 对事件图进行了扩展,增加了取消边以及通过有向边在事件之间传递参数的能力[13].取消边[13]相反于调度边的操作,采用虚线表示,如图 1(b)所示:当事件 A 发生,那么如果条件(i)为真,那么第一个要发生的事件 B 将从未来事件列表中移除;如果没有事件 B 被调度,那么什么也不会发生.

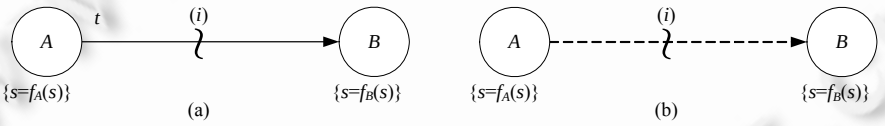


Fig.1 Event graph

图 1 事件图

事件图的第 2 个扩展是通过有向边在事件之间传递参数[13],如图 2 所示.其中图 2(a)表示当事件 A 发生时,如果条件(i)为真,事件 B 将在时间延迟 t 后发生;当事件 B 发生时,其参数 k 必须与表达式 j 匹配.对于取消边而言(图 2(b)),需要取消的事件 B,其参数 k 将由表达式 j 给出.

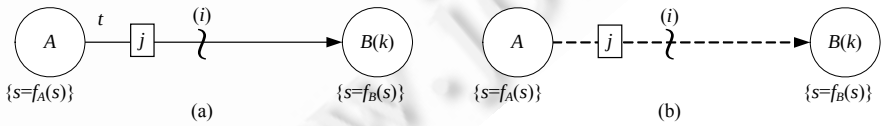


Fig.2 Parameters on schedule (cancelling) edges

图 2 调度(取消)边在事件之间传递参数

事件图模型被定义为八元组[13]:$M=(V,E,S,F,C,T,A,B)$,其中:

- V 是顶点集合,对应于事件图中的事件;
- E 是一组有向边,表示事件之间的调度/取消关系;

- S 是状态变量的集合;
- $F=\{f_v:S\to S,\forall v\in V\}$,每个顶点的状态变迁函数;
- $C=\{c_e:S\to\{0,1\},\forall e\in E\}$,每条边对应的条件函数($c_e$==1 表示条件满足,$c_e$==0 表示条件不满足);
- $T=\{t_e\in\Re_0^+,\forall e\in E\}$,每条边的时间延迟;
- $A=\{A_e,e\in E\}$,每条边的属性值列表;
- $B=\{B_v,v\in V\}$,每个顶点的参数列表.

事件图与 Petri 网、时间自动机等可视化建模语言相比,Petri 网、时间自动机都是基于状态的建模语言,它们的顶点表示系统的状态;而事件图中的顶点代表系统事件,表示系统状态的变迁[13],更符合离散事件建模与仿真的思想.目前有两款软件直接支持基于事件图的离散事件仿真——Sigma™ 和 Simkit[16],但是这两款软件都不具备模型验证功能.

2 DVE 建模语言

DiVinE 是基于自动机的 LTL(linear temporal logic)模型检验工具,其模型描述语言 DVE 是一种扩展的有限自动机,部分源自于 UPPAAL 建模语言[17].

DVE 中的基本建模单元是系统(system),它由多个进程(process)组成.进程由唯一的名称来标识,进程之间通过一组通道(channel)以及共享变量进行通信.一个完整的进程定义由局部变量声明、进程状态列表、可接受进程状态列表、约束进程状态列表、初始状态声明以及状态迁移列表组成.约束状态优先于其他状态,它表示从一个进程的约束状态的迁移不能被其他进程的动作中断[17].

Process P_1 {
  variables
  stares
  transitions}

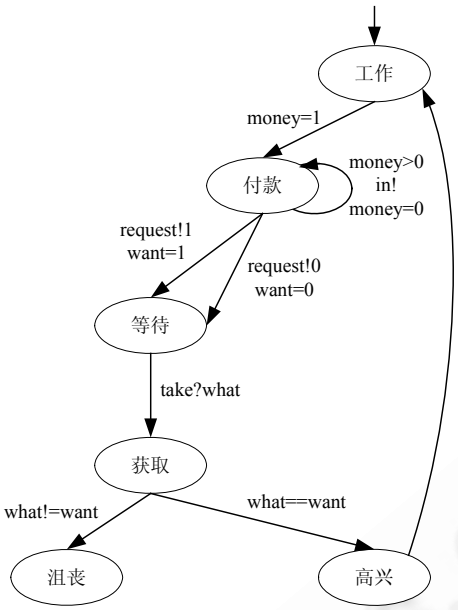
其中,状态迁移是进程的一个状态到另一个状态的转换,包括卫式(guard)、同步(synchronization)以及结果(effect)[17].

trans
$q_0\to q_1$ {guard(x==1)&&(y==2);
  sync toK!(S*2+y);
  effect x=x+1, y=y−1;};

其中:

- 卫式是布尔表达式,它表示变迁发生所必须要满足的条件,决定一个状态迁移是否能够被激活;
- 同步标识为 Expr!或者 Expr?(Expr 表示表达式),也可以为空.同步通过通道 channel 来实现,例如 make!0 与另外一个行为 make?product 同步;此外,同步也可以传送数据;
- 结果由赋值表达式组成,用于表示变迁所引起的系统状态变量的改变,可包含多个表达式.

下面以一个简单的自动售货机实例直观地说明 DVE 的语法结构[17].该模型由两个有限状态机组成:消费者和自动售货机.消费者通过工作赚钱在自动售货机上消费,获取想要的物品.如果得到的物品是他所期望的,那么消费者就会很高兴并继续工作;反之,他会变得沮丧.自动售货机根据消费者的选择为消费者提供不同种类的物品,两个状态机之间通过通道(例如 in!,request!0,take?what 等)进行同步.以消费者为例,一个消费者包括工作(初始状态)、付款、等待、获得产品、高兴以及沮丧这 6 个状态,其状态图及相应的 DVE 模型如图 3 所示.



状态描述:
工作状态(挣钱)—— 初始状态
付款状态——将钱放入自动售货机
等待状态 —— 选择饮料品种,等待产品
获取状态 ——拿到饮料
高兴状态——如果拿到的饮料是其所选则的品种
沮丧 ——如果拿到的饮料不是其所选的饮料

```
process man {
  byte what, want, money;
  state working, payment, waiting, got, happy, sad;
  init working;
  trans
    working → payment {effect money=1;},
    payment → payment {guard money>0;
                       sync in!;
                       effect money=0;},
    payment → waiting {sync request!0;
                       effect want=0;},
    payment → waiting {sync request!1;
                       effect want=1;},
    waiting → got {sync take?what;},
    got → happy {guard what==want;},
    got → sad {guard what!=want;},
    happy → working{ };
}
```

Fig.3 Finite state machines of a man and the corresponding DVE model

图 3 消费者的有限状态机模型及其相应的 DVE 模型

3 扩展事件图

3.1 事件和状态

系统建模语言可分为基于状态和基于事件(行为)两种.前者是通过状态来描述系统,而后者则是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.状态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;而事件被定义为系统中一种不具持续性的动作,它具有瞬时性,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.事件图是事件驱动的建模范型,而 DVE 语言则是基于状态的建模语言.在 DVE 语言中,状态是显式表示的,而事件是隐式的,事件图建模范型与之相反.但是,事件和状态之间存在紧密的联系:系统的每一个状态可以由一对事件来定义,一个开始事件(entry event)表示进入该状态,以及一个结束事件(exit event)表示离开该状态.图 4 表示了事件与状态之间的关系,其中,图 4(a)为事件驱动的建模范型,图 4(b)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.

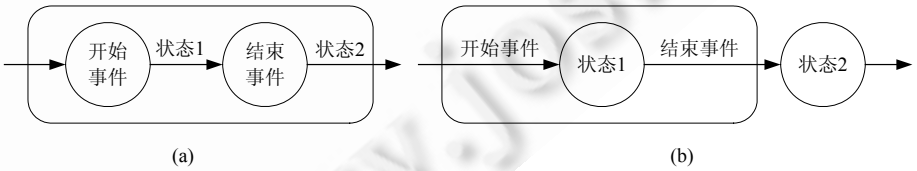


Fig.4 Relationship between events and states

图 4 事件与状态之间的关系

3.2 事件图与DVE之间的映射关系

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.下面将详细介绍事件图到 DVE 语言的转换关系.

一个系统的事件图模型可以看作是由多个子事件图组成的,每个子事件图对应于 DVE 模型中的一个进程 process.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(i)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 的 effect,如图 5 所示.

由于事件图建模范型没有直接提供同步机制,Sigma™ 和 Simkit 在实现不同子事件图互连时分别采取了不同的机制.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)来实现不同子事件图间的同步及通信[18].在 DVE 建模语言中,不同进程间通过 channel 实现同步以及数据传递功能.因此,为了能够简明地表示这类对同步以及数据传递有需求的系统,本文对事件图进行扩展,通过添加通道表示不同子事件图间的同步以及数据流关系,在图 5 中用 sync 表示,与 DVE 语言的同步表达式 sync 对应.

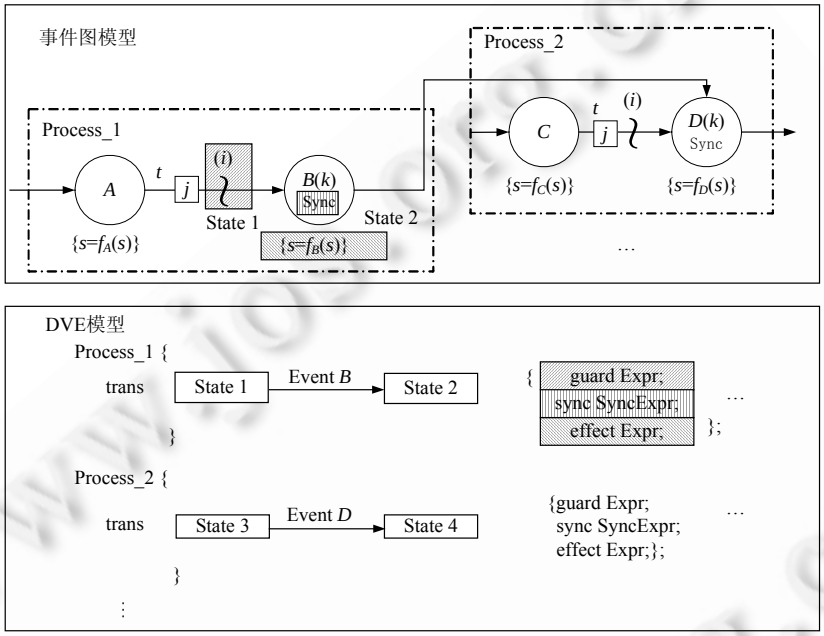


Fig.5 Mapping relationship between EEG and DVE

图 5 EEG 与 DVE 的对应关系

事件图中的调度(取消)边能够在事件节点之间传递参数.事件图的这种表达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,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(如图 6 所示)[16],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.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.图 6 中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,该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.

DVE 建模语言不支持对时间特性的表达[17],因此事件图模型中的时间延迟在 DVE 中没有对应关系.事件图和 DVE 之间的对应关系如图 5 所示,图中用不同方式表示出了事件图和 DVE 元素之间的对应关系.

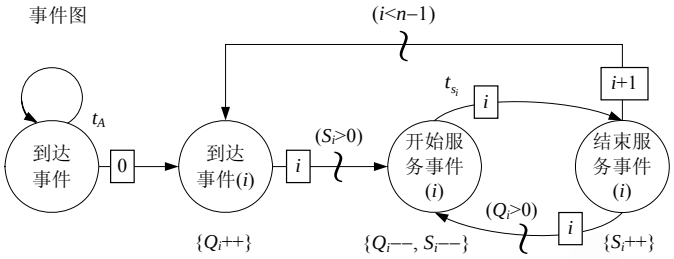


Fig.6 Transfer line event graph

图 6 流水线系统的事件图模型

3.3 扩展事件图的定义

定义 1. 扩展事件图(extended event graph,简称 EEG)是由事件和事件之间的关系组成的有向图,

$$EEG=\langle E,S,SC\rangle.$$

- E 表示事件的集合,$E=\langle M,F,P\rangle$,其中:
  $M=\{M_e,\forall e\in E\}$,状态变量集合;
  $F=\{f_e:M\to M,\forall e\in E\}$,每个事件的状态变迁函数;
  $P=\{P_e,\forall e\in E\}$,每个事件对应的参数列表;
- S 表示调度边或取消边的集合,$S=\langle L,From,To,C,T,PR,A\rangle$,其中:
  $L=\{L_s\in\{Schedule,Cancel\},\forall s\in S\}$,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;
  $C=\{C_s\in\{\text{True,False}\},\forall s\in S\}$,表示每条边对应的条件函数;
  $T=\{t_s\in\mathcal{R}_0^+,\forall s\in S\}$,表示每条边的时间延迟;
  $PR=\{PR_s\in\{Lowest,Lower,Low,Default,High,Higher,Highest\},\forall s\in S\}$,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.
- $SC=\{SC_e\in\{?,!\},\forall e\in E\}$,表示同步通道的集合,“?”表示接收,“!”表示发送.

若用 SG_i 表示子事件图,即 $SG_i\in EEG$,那么一个扩展事件图可表示为 $EEG=\{SG_i,SC\}$,$1\leqslant i\leqslant n$, $n\in N$.

3.4 扩展事件图的语法及语义

3.4.1 扩展事件图的语法

一个完整的 EEG 模型由声明、子图定义、事件定义、调度(取消)边定义以及系统类型定义组成.

EEG::=*Declaration SGDefList EventDefList EdgeDefList System*.

声明包括状态变量声明、参数声明、属性声明以及通道声明.

Declaration::=*ε*|*StateVariableDecl*|*ParametersDecl*|*AttibuteDecl*|*ChannelsDecl*|.

声明常量时,需要标明关键字 Const.变量既可以是标量也可以是矢量.

Const::=*ε*|Const;

VarDecl::=*Const TypeName DeclIdList*;

TypeName::=*int*|*string*|*double*|*integer*|*Boolean*|*Random*

DeclIdList::=*DeclId*|*DeclIdList*,*DeclId*

DeclId::=*id VectorDecl VarInit*

VectorDecl::=*ε*|[*number*]

矢量初始化记为$\{value_1,value_2,\ldots,value_n\}$.

VarInit::=*ε*|*Initializer*

Initializer::=*Expr*|{*VecInitList*}

VecInitList::=*VecInit*|*VecInitList*,*VecInit*

VecInit::=*Expr*

通道包括有类型和无类型两种,有类型通道具有缓冲区,能够缓存数据,数据传送是异步的[17].

Channels::=channel *ChannelDeclList*|channel *TypedChannelDeclList*

ChannelDeclList::=*ChannelDecl*|*ChannelDeclList*,*ChannelDecl*

ChannelDecl::=*id*

TypedChannelDeclList::=*TypedChannelDecl*|*TypedChannelDeclList*,*TypedChannelDecl*

TypedChannelDecl::=*id*[*number*]

TypeList::=*TypeId*|*TypeList*,*TypeId*

子图由唯一的名称来标识,它与 DVE 模型中的进程一一对应,包括局部变量声明、初始化、事件列表、调度(取消)边列表.

SGDefList::=*SGDef SGDefList*

SGDef::=*SG id* {*SGBody*}

SGBody::=*SGLocalDeclList Events Edges*

事件由唯一的名称来标识.一个完整的事件定义由局部变量声明,参数声明,以及状态转移函数组成.

Event::=*EventDecl*|*EventDeclList*,*EventDecl*

EventDecl::=*EventName*,*%CausedState*,*LocalVarDecl*,*ParaDecl*,*Transitions*,*Sync*

Transitions::=*ε*|[*TransitionList*]

TransitionList::=*Transition*|*TransitionList*,*Transition*

Transition::=*Assignment*

Assignment::=*Expr*=*Expr*

Sync::=*ε*|*Sync SyncExpr*

SyncExpr::=*id*!*SyncValue*|*id*?*SyncValue*

SyncValue::=*ε*|*Expr*|{*ExprList*}

ExprList::=*Expr*|*ExprList*,*Expr*

调度(取消)边由一对事件来定义,它包括边的类型(调度或取消)、调度事件、被调度事件、事件调度的条件、优先级、时间延迟以及相关属性的声明.

Edge::=*TypeName*,*SourceEvent*,*TargetEvent*,.*Condition*,*@Priority*,*$Delay*,*AttrDecl*|*#Edge*

TypeName::=*Schedule*|*Cannel*

Condition::=*ε*|*Expr*

Priority::=*Default*|*Lowest*|*Lower*|*Low*|*High*|*Higher*|*Highest*

表达式用于表示变量初始化、状态变迁、事件发生的条件以及不同子事件图间的同步等.表达式的递归定义如下:

Expr::=False|True|*number*|*id*|[*Expr*]|(*Expr*)|*UnaryOp Expr*|$Expr_1$<$Expr_2$

|$Expr_1$≤$Expr_2$|$Expr_1$==$Expr_2$|$Expr_1$!=$Expr_2$|$Expr_1$>$Expr_2$|$Expr_1$≥$Expr_2$

|$Expr_1$+$Expr_2$|$Expr_1$–$Expr_2$|$Expr_1$*$Expr_2$|$Expr_1$/$Expr_2$|$Expr_1$%$Expr_2$|$Expr_1$&&$Expr_2$

|$Expr_1$||$Expr_2$|!*Expr*|?*Expr*

3.4.2 扩展事件图的语义

定义 2. 设 V_N 和 V_T 都是非空有穷集,$S\in V_N$ 且 $V_N\cap V_T=\varnothing$.记 $V=V_N\cup V_T$,并称 $G=\langle V_N,V_T,P,S\rangle$ 为文法.其中:

- V_N 称为 G 的变量集或非终极符字母表,并称 V_N 中元素为 G 的变量或非终极符;

- V_T称为 G 的终极符字母表,并称 V_T中元素为 G 的终极符;
- P 为有穷集,称为 G 的产生式集,并称 P 中元素为 G 的产生式.P 中元素都有如下形式:

$$\alpha\to\beta,\alpha\in V^*V_NV^*,\beta\in V^*.$$

- $S\in V_N$,称为 G 的句子符号.

定义 3. 设文法 $G=\langle V_N,V_T,P,S\rangle,\omega\in V_T^*,\alpha,\beta\in V^*$:

1) 若$\alpha\overset{*}{\Rightarrow}\beta$,则称$\alpha$可派生出$\beta$,或称$\beta$为$\alpha$的一个派生;

2) 若$S\overset{*}{\Rightarrow}\alpha$,则称$\alpha$为$G$的一个句型;

3) 若$S\overset{*}{\Rightarrow}\omega$,则称$\omega$为$G$产生的一个字;

4) 令$L(G)=\{\omega\in V_T^*\mid S\overset{*}{\Rightarrow}\omega\}$

并称 $L(G)$为 G 所产生的语言.例如,*event*_1,*event*_2 等表示由 L(*Event*)产生的语言.

定义 4. 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:

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

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

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

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

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

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

$[\![expr_1\ \ binary\_op\ \ expr_2]\!](e)=[\![expr_1]\!](e)\ \ binary\_op\ \ [\![expr_2]\!](e)$

$...binary\_op\in\{<,\le,==,!=,+,-,*,/,\%,\&\&,\|\},$

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

$$[\![id_1.id_2]\!](e)\ =\begin{cases}1 & e(id_1)=id_2\quad \text{表示子图 } id_1\text{中的事件 } id_2\\ 0 & \text{otherwise}\end{cases}$$

定义 5. 系统变迁定义为

transition≡$id_1.id_2${*condition sync function*} and *function*≡*assignment*$_1$,…,*assignment*$_n$,

其中,id_1 表示子事件图的名称,id_2 表示引起变迁的事件,$id_1.id_2$ 表示子事件图 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) 改变进程的状态;

(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) $$[[assignment]](e)=\begin{cases}[[id_{var}/[[expr_{varlue}]](e)]](e), & \text{if } assignment\equiv id_{var}=expr_{value}\\ [[id_{var}([[expr_{index}]](e))/[[expr_{value}]](e)]](e), & \text{if } assignment\equiv id_{var}[expr_{index}]=expr_{value}\end{cases}$$

本文在定义 EEG 时使用了一些特殊的算子,见表 2.

Table 2 Some special operators in EEG

表 2 EEG 中的特殊算子

算子	含义
%	事件发生所引起的状态
@	调度/取消边的优先级
.	事件调度/取消的条件
$	事件延迟
#	某事件所能调度的所有事件的连接符
!	发送通道
?	接收通道

4 实验分析

4.1 EEG到DVE的转换过程

ANTLR(another tool for language recognition)[19]是根据一种可以嵌入如 Java,C++或 C#等辅助代码段的文法来自动构建出相对该文法的识别器、编译器或翻译器(识别器、编译器或翻译器在下文中简称为语法分析器)的一种语言工具框架.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 的语法分析器,使用户根据 EEG 建模规则建立的模型直接转换为可在并行模型检验工具上运行的 DVE 模型.ANTLR 的基本单元是文法,文法是对某一种语言语法规则的描述,通常由一组规则组成,每一规则描述了该语言的若干短语,每一条规则有一个或者多个供选方案[19].ANTLR 生成的部分 EEG 语法结构图如图 7 所示.

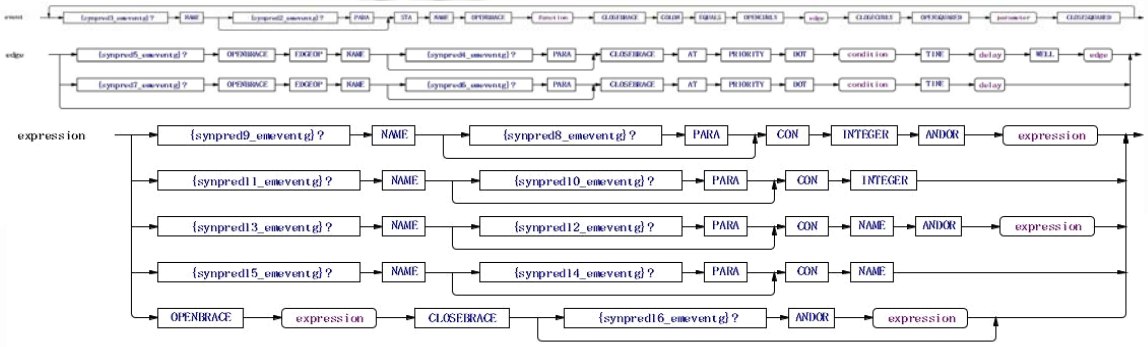


Fig.7 Parts of syntax diagram of EEG generated by ANTLR

图 7 ANTLR 生成的部分 EEG 语法结构图

仿真模型的并行检验过程为:

1) 建立系统的 EEG 模型;
2) 将 EEG 模型输入到转换器(如图 8 所示),首先由 ANTLR 自动生成的 Lexer 和 Parser 判断该模型语法的正确性,若该模型有效,则由根据 DVE 模型原语建立的 ClassBuilder 类被解析为 DiVinE 所识别的.dve 文件;
3) DiVinE 编译、运行生成的.dve 文件,输出检验结果.

采用这种通过转换器进行并行模型检验的方法,使得仿真领域的专家或仿真应用的开发人员不需要学习新的形式化建模语言,就可以对仿真模型进行形式化验证.

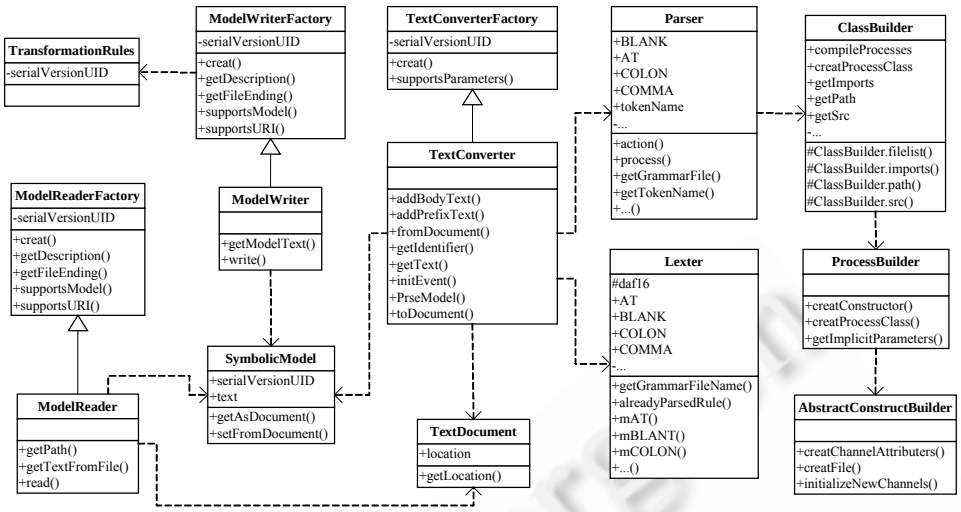


Fig.8 The class structure of the transformer

图 8 转换器的类结构

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.哲学家就餐问题表述为[8]:假设有 5 位哲学家围坐在一张圆形餐桌旁,做以下两件事情之一:吃饭或者思考.餐桌中间有一大碗意大利面,每两个哲学家之间有一只餐叉.假设哲学家必须用两只餐叉吃东西.他们只能使用自己左、右手边的那两只餐叉.本文主要关注两方面的问题:(1) 转换机制的有效性,即 EEG 模型是否能够转换为 DVE 模型;(2) 并行模型检验方法对仿真模型验证的效率如何.本文并行验证实验所采用的平台为:双向 2.53GHz 的四核 Xeon 处理器 E5540,8GB RAM,内核 2.6.18 的麒麟 3.1 操作系统.

实验 1. 转换机制的有效性实验.

要保证哲学家就餐模型的正确性就需要确保模型不会出现以下 3 种情况:(1) 死锁,每个哲学家都拿着一只餐叉,永远都在等另一支餐叉,即没有哲学家能够进入吃饭状态;(2) 活锁,假设规定当哲学家等待另一只餐叉超过一分钟后就放下自己手里的那一支餐叉,等一分钟后进行下一次尝试,但如果所有哲学家在相同时刻拿起左边的餐叉,那么这些哲学家就会等待一分钟后同时放下手中的餐叉,再等一分钟,又同时拿起这些餐叉,如此反复,虽然系统总会进入到下一个状态,但是哲学家们仍然不能够进入到吃饭状态;(3) 非公平性,即有某个或某几个哲学家想吃饭但是却始终吃不到饭.这 3 种情况对应于模型检验中的安全性(坏事情不会发生)、活性(好事情总会发生)以及公平性问题,只有保证了哲学家就餐模型满足了这些性质,它在逻辑上才是正确的.

本文首先根据 EEG 语法建立了最直观的哲学家就餐问题模型:当哲学家感到饥饿时,尝试随机地拿起他左边或右边的餐叉,如果拿到一支餐叉且另一支空闲,哲学家则拿起另一支餐叉进入吃饭状态;否则,等待另一支餐叉空闲.吃饭结束后,放下两支餐叉继续思考.很显然,当所有哲学家同时拿到左餐叉或右餐叉时,该模型会进入死锁状态.通过本文提出的转换机制,该模型自动转换为能够由 DiVinE 工具检验的 DVE 模型.DiVinE 的验证结果如图 9(a)所示,验证结果显示,验证过程遍历了模型存在的所有 242 个可能状态以及 806 个状态迁移,发现模型存在一个死锁状态——每个哲学家同时拿到左边的一支餐叉.假设规定当哲学家等待另一只餐叉超过一分钟后就放下自己手里的那一只餐叉,再等一分钟后进行下一次尝试,这个策略虽然消除了死锁,但仍然有可能发生活锁,图 9(b)显示了对该模型的验证结果.图 9(c)的验证结果表明,该模型不满足公平性.图 9(d)~图 9(f)分别显示了消除以上问题的正确模型的验证结果.

实验 1 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 DVE 模型,利用形式化模型检验方

法提高了仿真模型检验的完备性.

Fig.9 Verification result of dining philosopher models in DiVinE

图 9 DiVinE 对哲学家就餐模型的验证结果

实验 2. 仿真模型并行验证的高效性实验.

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[20]以及 PRISM 概率模型

检验工具[21]进行比较.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.对于后两个模型检验证器而言,首先需要根据其规约语言建立与 DiVinE 一样的具有死锁状态的哲学家就餐问题模型.实验结果如图 10 所示,从对比结果可以看出,当模型规模较小时,并行模型检验方法的优势并不明显.而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,能够有效地提高仿真模型验证的效率.实验 2 中验证的模型最大状态空间达到 7.503027757327091E19,此时,UPPAAL 串行验证的时间达到 351.875s;如果继续增大模型状态空间,其验证时间将让人们难以忍受,而 DiVinE 则用了 56.764s.对于更加复杂的大规模仿真模型而言,其验证方法及原理相同,采用这种并行验证方法都能够提供较高的模型验证效率.

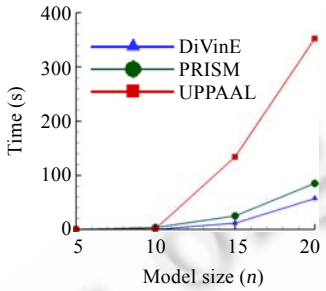


Fig.10 Model checking time for the dining philosopher model of different model size

图 10 不同规模的哲学家就餐问题验证时间

5 总 结

本文首先分析了现阶段常用的仿真模型验证方法,指出了对仿真模型进行形式化验证的必要性以及可行性.在分析、比较事件图和 DVE 建模语言的基础上,本文提出了一种对离散事件仿真模型进行并行形式化验证的方法,将并行模型检验方法用于对基于事件图的离散事件仿真模型进行验证.本文对事件图进行了必要的扩展,详述了扩展事件图的语法及语义,并根据扩展事件图和 DVE 模型之间的映射关系构建了模型转换器.基于该方法的离散事件仿真模型验证,使仿真人员无须学习新的形式化验证语言就能对仿真模型进行并行验证,不仅降低了仿真模型并行检验的难度,而且提高了验证的完备性以及验证效率.实验结果表明了该方法的有效性和实用性.本文下一阶段的主要工作是实现 EEG 模型的可视化并行验证,以及研究智能搜索算法进一步提高可验证的状态空间数.

致谢 在此,我们向对本文的工作给予支持和建议的国防科学技术大学计算机学院高性能仿真课题组全体成员表示感谢.

References:

[1] Whitner RB, Balci O. Guidelines for selecting and using simulation model verification techniques. In: MacNair EA, Musselman KJ, Heidelberger P, eds. Proc. of the '89 Winter Simulation Conf. New York: ACM Press, 1989. 559−568. [doi: 10.1145/76738.76811]

[2] Wang WP, ed. Modeling and Simulation of Discrete Event Systems. 2nd ed., Beijing: Science Press, 2007 (in Chinese).

[3] Wang ZC. Problem of evaluations of complex simulation systems. In: Proc. of the Conf. of the 20th Anniversary of System Simulation Institute Comes into Existence. 2009 (in Chinese with English abstract).

[4] Sargent RG. Verification and validation of simulation models. In: Rossetti MD, Hill RR, Johansson B, Dunkin A, Ingalls RG, eds. Proc. of the 2009 Winter Simulation Conf. Austin, 2009. 162−176. [doi: 10.1109/WSC.2009.5429327]

[5] Pace DK. Modeling and simulation verification and validation challenges. Johns Hopkins APL Technical Digest, 2004,25(2): 163−172.

[6] Liu WW. Extended temporal logic and symbolic model checking technology [Ph.D. Thesis]. Changsha: Nation University of Defense Technology, 2009 (in Chinese with English abstract).

[7] Clarke EM, Emerson EA, Sifakis J. Model checking: Algorithmic verification and debugging. Communications of the ACM, 2009, 52(11):75−84. [doi: 10.1145/1592761.1592781]

[8] Baier C, Katoen JP. Principles of Model Checking. England: The MIT Press, 2008.

[9] Brim L. Parallel model-checking. European Research Consortium for Informatics and Mathematics News (ERCIM News), 2004,58: 35−36.

[10] Kuang HB, Luo GM. Parallel software model checking. Computer Engineering, 2008,34(19):23−25,29 (in Chinese with English abstract).

[11] List of discrete event simulation software. http://dictionary.sensagent.com/list+of+discrete+event+simulation+software/en-en/

[12] Deng XN. Research on the validation methods of the C4ISR system requirements based on model checking and simulation [Ph.D. Thesis]. Changsha: Nation University of Defense Technology, 2008 (in Chinese with English abstract).

[13] Savage EL, Schruben LW, Yücesan E. On the generality of event-graph models. INFORMS Journal on Computing, 2005,17(1):3−9. [doi: 10.1287/ijoc.1030.0053]

[14] Barnat J, Brim L, Ročkai P. DiVinE 2.0: High-Performance model checking. In: Proc. of the 2009 Int'l Workshop on High Performance Computational Systems Biology. Trento: IEEE Computer Society, 2009. 31−32. [doi: 10.1109/HiBi.2009.10]

[15] DiVinE. http://divine.fi.muni.cz/

[16] Buss AH. Basic event graph modeling. Simulation News Europe, 2001,31:1−6.

[17] Šimeček P. The DVE modeling language [MS. Thesis]. 2006. 2−15. http://divine.fi.muni.cz/dve.pdf

[18] Buss AH, Sánchez PJ. Building complex models with LEGOS (listener event graph objects). In: Yücesan E, Chen C-H, Snowdon JL, Charnes JM, eds. Proc. of the 2002 Winter Simulation Conf. (WSC 2002). San Diego: IEEE Press, 2002. 732−737. [doi: 10.1109/WSC.2002.1172954]

[19] Parr T. The Definitive ANTLR Reference-Building Domain-Specific Languages. The Pragmatic Bookshelf, 2007.

[20] Behrmann G, David A, Larsen KG. A tutorial on uppaal. In: Proc. of the Formal Methods for the Design of Real-Time Systems (revised lectures). LNCS, Springer-Verlag, 2004. 200−236. [doi: 10.1.1.106.5799]

[21] PRISM. http://www.prismmodelchecker.org/

附中文参考文献:

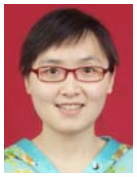
[2] 王维平等.离散事件系统建模与仿真.第 2 版,北京:科学出版社,2007.

[3] 王子才.关于复杂仿真系统评估问题.见:系统仿真学会成立 20 周年大会报告.2009.

[6] 刘万伟.扩展时序逻辑的推理及符号化模型检验技术[博士学位论文].长沙:国防科学技术大学,2009.

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

[12] 邓小妮.基于模型检验与仿真的 C4ISR 系统需求验证方法研究[博士学位论文].长沙:国防科学技术大学,2008.



夏薇(1983－),女,河南信阳人,博士生,主要研究领域为仿真模型验证.

慕晓冬(1965－),男,博士,教授,博士生导师,主要研究领域为计算机仿真技术.

姚益平(1963－),男,博士,教授,博士生导师,主要研究领域为并行与分布仿真,虚拟现实.

柳林(1977－),男,博士,主要研究领域为建模与仿真.