计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(7):70−81 [doi: 10.15888/j.cnki.csa.007470]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于规则推理的实时信息物理监控系统①

彭　程[1,2], 乔　颖[1], 王宏安[1]

[1](中国科学院 软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100049)
通讯作者: 乔　颖, E-mail: qiaoying@iscas.ac.cn

摘　要: 基于规则的 CPS 监控方法在降低监控复杂度和提升监控灵活性等方面具有显著优势. 目前基于规则的 CPS 监控方法未考虑 CPS 监控场景的时间约束, 仅仅利用各种优化技术来缩短监控的响应时间. 为此, 本文基于实时规则引擎建立了一个 CPS 的实时监控系统 RTCPMS. 该系统采用 Rete 网络表示监控规则, 其核心是一个新的实时推理算法 Rete-TC. Rete-TC 算法引入了规则截止期, 通过基于优先级的 Beta 节点调度方法, 使得 CPS 监控的时间约束尽可能地被满足. 模拟实验与智慧建筑应用案例验证了 RTCPMS 系统的有效性, 且实验结果表明其核心算法 Rete-TC 的调度成功率优于传统的规则推理算法 Rete.

关键词: 监控系统; 规则推理; Rete; 时间约束; 信息物理系统

引用格式: 彭程,乔颖,王宏安.基于规则推理的实时信息物理监控系统.计算机系统应用,2020,29(7):70–81. http://www.c-s-a.org.cn/1003-3254/7470.html

Real-Time Cyber-Physical Monitoring and Control System Based on Rule Inference

PENG Cheng[1,2], QIAO Ying[1], WANG Hong-An[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Rule-based CPS monitoring methods have significant advantages in reducing monitoring complexity and improving monitoring flexibility. Currently, rule-based CPS monitoring methods do not consider the timing constraints of CPS monitoring scenarios, and only use various optimization techniques to shorten the response time. Based on the real-time rule engine, a real-time monitoring system of CPS, RTCPMS, is established. The system uses Rete network to represent the monitoring rules, and its core is a new real-time inference algorithm Rete-TC. The Rete-TC algorithm introduces the rule deadline, and the timing constraints of CPS monitoring is satisfied as much as possible by the priority-based Beta node scheduling method. The simulation experiment and smart building application case verify the effectiveness of the RTCPMS, and the experimental results show that the core algorithm Rete-TC has a better scheduling success ratio than the traditional rule inference algorithm Rete.

Key words: monitoring systems; rule inference; Rete; timing constraints; CPS

信息物理系统 (Cyber-Physical Systems, CPS) 是将计算过程和物理过程集成的系统, 利用嵌入式计算机和网络对物理过程进行监测和控制, 并通过反馈环实现计算和物理过程的相互影响[1]. CPS 概念自提出以来, 迅速在众多重要环节中承担关键任务, 广泛应用在智慧建筑[2]、生产制造[3]、交通运输[4]、医疗健康[5,6]

① 基金项目: 国家重点研发计划 (2018YFB0904505); 北京市科委项目 (KJY20190971)
Foundation item: National Key Research and Development Program of China (2018YFB0904505); Program of Science and Technology Commission of Beijing Municipality (KJY20190971)
收稿时间: 2019-11-25; 修改时间: 2019-12-19; 采用时间: 2020-01-02; csa 在线出版时间: 2020-07-03

和城市建设[7]等领域. CPS 本质就是构建一套赛博 (Cyber) 空间与物理 (Physical) 空间之间基于数据自动流动的状态感知、实时分析、科学决策、精准执行的闭环赋能体系, 解决生产制造、应用服务过程中的复杂性和不确定性问题, 提高资源配置效率, 实现资源优化[8]. CPS 监控是从物理设备产生的事件中感知关注场景并作出响应的过程, 是 CPS 的核心功能之一.

实时性是 CPS 的特点之一[9], 为 CPS 监控带来了时间约束问题, 即从与监控关注场景相关的所有原子事件到达系统开始到触发关注场景关联的动作之间的时间间隔 (也称为响应时间) 不应超过其上界 (这个上界称为截止期), 否则将造成严重的后果. 例如, 在智慧建筑 CPS 中, 存在诸多监控场景, 例如, 节能场景、环境舒适度调节场景和火灾监控场景等. 其中火灾监控场景具有时间约束要求, 国家标准 GB4717-2005 规定查询和处理数据等火灾报警操作不超过 10 s. 这个 10 s 就是一个从事件发生到触发火灾报警动作的截止期, 错失截止期可能会造成火灾事故, 从而造成生命和财产损失. 这样的 CPS 监控场景还有很多, 比如高速列车故障控制系统、电网故障诊断系统等.

在这样的 CPS 监控中, 事件与动作之间的因果关系往往使用规则来描述. 针对监测到的事件, 判断某条规则所描述的事件模式是否满足, 从而得出动作的过程, 称为针对此条规则的推理过程, 上述对监控事件作出响应的过程, 可视为一系列规则推理过程的集合[10]. 采用规则系统相较于过程式逻辑具有若干优势, 如规则灵活性更强 (当业务需求变化时, 更新规则库中规则即可, 应用无需重新编译部署)、更直观易理解 (例如智慧建筑中监控服务“当房间温度低于 18 度时, 则打开空调制热”)、更低的复杂度 (规则具有一致性表示) 等.

为此, 学者们对基于规则的 CPS 监控进行了探索. Sun 等人[11–13]使用基于 Rete[14]算法的规则引擎对大规模智能建筑产生的事件进行高效处理. Klein 等人[15]提出了一种结合 ECA (Event-Condition-Action)[16]和 CEP (Complex Event Processing)[17]的基于规则的方法来对 CPS 中产生的事件进行监控. 然而, 上述基于规则的 CPS 监控研究工作未考虑 CPS 监控的时间约束. 关于时间约束问题, 学者们也提出了一些方法, 如: 迭代推理[18](如 Anytime 算法)、多重方法推理[19](如 Design-to-Time 算法) 和渐进式推理[20,21](如 GREAT 算法和 PRIMES 算法). 这些方法为整个推理过程定义了截止期约束, 通过对推理运行时间与推理结果质量进行折中来满足这个截止期约束. 李想等[22]介绍了一种面向物联网的实时复杂事件处理引擎, 该引擎采用一种启发式复杂事件处理算法来优先处理能够产生结果的任务, 缩短了复杂事件处理的响应时间, 同时介绍了一种复杂事件处理程序的最坏响应事件估算算法, 但忽略了各监控任务不同的紧迫度. Liu 等人[23]使用复杂事件处理技术来对智能电网中产生的事件进行处理. 然而, 上述工作主要是缩短了 CPS 监控的响应时间, 但不能使 CPS 监控尽可能地满足其时间约束.

此外, 学者们还通过改进典型的规则匹配 Rete 算法来缩短匹配时间以及进行一些功能扩展如模糊推理、事件处理和并行化等方面, 但未考虑 Rete 算法在推理过程中的时间约束问题. William Van Woensel 等人[24]提出了一种改进的 Rete 算法, 支持在内存受限环境下进行规则推理. 文献[25,26]对 Rete 的语法结构进行扩展, 即根据规则的 and、or 等逻辑操作将规则构建成一个抽象语法树, 并在语法树的基础上建立 Rete 网络, 逻辑操作符本身作为一个 Rete 节点参与事实匹配. 这种做法减少了规则拆分开销, 给规则匹配带来了便利. 文献[27]对规则中的条件进行了重排序. 规则中条件顺序是指规则条件中的各个约束的排列顺序, 它决定了条件的执行顺序, 是决定规则匹配效率的关键因素. Xiao 等人[28]将 Beta 内存分成若干单元, 每个单元分配一个 id, 对事实用哈希函数计算索引, 索引号即为某个单元的位置, 通过索引快速找到相应单元进行匹配, 如果匹配不成功, 则将该对象组成一个新的单元加入相关内存. 孙新等人[29]提出了一种基于共享度模型的改进 Rete 算法, 可以提升 Rete 网络的推理速度. Ju 等人[30]研究将 Rete 算法在 Apache Spark 上进行并行化. 汪成亮等人[31]提出了智能环境下分布式 Rete 算法, 将推理网络中的计算任务分布式地部署到最靠近参数采集的传感器节点中, 让这些传感器充分参与到推理计算工作中.

为此, 本文基于实时规则引擎建立了一个 CPS 的实时监控系统 RTCPMS (Real-Time Cyber-Physical Monitoring System), 该系统采用 Rete 网络表示监控规则. RTCPMS 将实时推理技术和 CPS 监控结合, 其核心是一个新的实时推理算法 Rete-TC. Rete-TC 算法引入了规则截止期, 通过基于优先级的 Beta 节点调度方

法, 使得 CPS 监控的时间约束尽可能地被满足. 模拟实验与智慧建筑应用案例验证了 RTCPMS 系统的有效性, 且实验结果表明其核心算法 Rete-TC 的调度成功率优于传统的规则推理算法 Rete.

本文接下来的组织结构如下: 第 1 节介绍了研究基础; 第 2 节介绍了 Rete-TC 实时规则推理算法; 第 3 节介绍了系统框架; 第 4 节进行了模拟实验评估; 第 5 节进行应用案例研究; 第 6 节进行总结了及下一步工作.

1 研究基础

1.1 术语和定义

在 CPS 中, 原子事件是不可再分的, 只存在发生和不发生两种状态, 其来源包括传感器或其他感知设备. 关注场景是按照定义在规则中的事件条件从原子事件中检测得到.

定义 1. 原子事件 (atomic event)

原子事件由属性类型名和属性组成, 可表示为 (EventTypeName Attributes), 其中 EventTypeName 表示事件的类型, Attributes = ($(Name_1\ Value_1)$, ⋯, $(Name_n\ Value_n)$) 表示事件的属性, 属性是有序的. 每个属性由属性名和属性值构成. 例如, 一个事件类型为“ClassA”的原子事件可以表示为“(ClassA (val_1 365) (val_2 928) (val_3 153) (val_4 497))”.

定义 2. 约束 (constraint)

约束可以表示为 ($Param_1$ op $Param_2$). 它由 $Param_1$, op 和 $Param_2$ 组成. 这里 $Param_1$ 是事件属性名, 为变量; op 为常见的逻辑操作符, 例如>, <, ==, >=, <=; $Param_2$ 可以是一个常量或一个变量 (另一个事件的属性名). 当约束为“变量-op-常量”时, 它用于确定一个事件的属性和常量之间的关系, 称为常量约束, 例如, (Person.age > 15). 当约束为“变量-op-变量”时, 它用于确定事件属性关系, 称为变量约束, 例如, (C1.attr > C2. attr).

定义 3. 事件条件 (event condition)

事件条件可表示为:

($ EventTypeBindName : EventTypeName ($Constraint_1$ OP $Constraint_2$ OP ⋯$Constraint_n$))

其中 $ 为变量绑定符，EventTypeBindName 为事件对象绑定的变量名，用于支持将多个同类型事件对象绑定到不同的 $ EventTypeBindName 上 (因为同一事件对象在一条规则中可能出现多次). EventTypeName 含义参见定义 1. Constraint 参见定义 2. OP 指“and” (也表示为“&&”)，“or”(也表示为“||”) 连接操作符. 事件条件有时也称为事件模式.

定义 4. 实时规则 (real-time rule)

实时规则由用户定义, 可表示监控场景. 规则由规则名、规则截止期、事件条件集和动作集组成. 其中事件条件集也称作左手部分 (Left-Hand Side, LHS), 动作集也被称作右手部分 (Right-Hand Side, RHS). 基于 Drools[32] 规则语言, 我们定义的实时规则格式如下:

rule “name”
deadline *n*
when
事件条件 1∪事件条件 2∪⋯∪事件条件 *N*
then
动作 1∪动作 2∪⋯∪ 动作 *N*
end

“deadline *n*”表示规则的截止期为 *n* 秒, 截止期大小一般由应用确定. 如果 *n* 取 0, 则表示规则无时间约束.

1.2 Rete 推理算法

1.2.1 Rete 基本概念

Rete 是一个经典且高效的规则匹配算法, 已被广泛应用到许多主流的规则推理引擎中. Rete 算法作为一种前向链推理算法, 其核心思想是采用增量匹配的概念, 根据内容动态构造匹配树, 以达到显著降低计算量的效果. 目前主流规则推理引擎 (例如 Drools) 使用 Rete 算法. Rete 算法将规则转换成 Rete 网络, 表示各种规则条件间的数据依赖. 当事实或事件对象进入系统后, 它将在 Rete 网络上匹配传播, 直到抵达规则的终止节点, 即完成一条规则的匹配, 进而触发该规则预定义动作. Rete 网络包括两部分: Alpha 网络和 Beta 网络. Alpha 网络包括 4 类节点: 根节点 (Alpha root node)、类型节点 (Object type node)、条件节点 (Alpha node) 和 Alpha 内存节点 (Alpha memory node). 根节点是所有事实或事件对象进入 Rete 网络的入口, 是一个虚节点, 无实际意义. 类型节点是根节点的直接子节点, 保存了规则中事实或事件对象对应的类型. 如果某对象类型的模式有条件约束, 则该约束会进入 Alpha 网络形成条件节点, 完成事实或事件中属性的常量约束测试 (例如, 检查属性 Person.age > 15), 这样可以过滤掉大部分无意义的事实或事件对象. Alpha 内存 (Alpha memory) 节点是 Alpha 网络的最后一层节点, Alpha 内存存储所有通过常量约束测试的事实对象, 也称为

Alpha 匹配 (Alpha Match, AM). Beta 网络包含 3 类节点: 连接节点 (Beta node) 和终止节点 (Terminal node) 和 Beta 网络根节点 (Beta root node). 连接节点的输入来自上一级的 Beta 节点和 Alpha 内存节点, 负责执行变量约束测试 (例如, C1.attr > C2.attr), 即判断两个事实或事件对象在某一属性值上的关系, 通过该测试的事实或事件数据被称为部分匹配 (Partial Match, PM), 即成功匹配了规则的一部分约束但还没完全匹配所有约束的部分事实或事件数据, 每个 Beta 节点都保存着通过上一级 Beta 节点变量约束测试的 PM 数据, 并使用这些数据完成本节点的变量约束测试, 之后将匹配结果传递给下一级 Beta 节点, 直至抵达规则终止节点; 规则终止节点是出度为 0 的节点, 是 Rete 网络的叶子节点, 也是一条规则的最后一个节点, 当事实或事件数据到达该节点时, 表明该条规则被触发. Beta 网络根节点为一个虚节点, 不具有实际意义.

1.2.2 Rete 网络构建流程

假设有规则集 Ri($1 \le i \le N$), Rete 网络构建流程如下:

(1) 建立 Rete 网络的根节点 Alpha root node 和 Beta 网络根节点 Beta root node;

(2) 处理规则 Ri;

(3) 处理 Ri 中的事件条件 Cj($1 \le j \le n$), 假如 Cj 不存在于 Alpha 网络中, 则将 Cj 作为 Alpha(j) 节点插入 Alpha 网络;

(4) 对 Cj 中的变量约束, 新建 Beta 节点, 该 Beta 节点的左输入是上一级的 Beta 节点 (如果是第一个 Beta 节点, 左输入为 Beta root node 节点), 右输入是当前 Cj 的事件对象的 Alpha(j);

(5) 重复 (3) ~ (4) 处理完所有事件条件;

(6) 重复 (2) ~ (5) 处理完所有规则.

由于本文所提 Rete-TC (Timing Constraints) 算法重点改进 Rete 算法的 Beta 网络部分, 下面将举例对 Rete 中的 Beta 网络部分进行重点介绍, Alpha 网络不再详细赘述.

假设有如下规则:

```
rule “R1”
when
    $1 : C1 (attr1 == 120 && attr2 == 220)
    $2 : C2 (attr1 == 55 && attr2 == $1.attr2)
    $3 : C3 (attr1 == $2.attr1 && attr2 == $2.attr2)
then
    Actions
end
rule “R2”
when
    $1 : C1 (attr1 == 120 && attr2 == 220)
    $2 : C2 (attr1 == 55 && attr2 == $1.attr2)
    $4 : C4 (attr1 == $1.attr1)
    $5 : C5 (attr2 == $2.attr2)
then
    Actions
end
rule “R3”
when
    $1 : C1 (attr1 == 120 && attr2 == 220)
    $2 : C2 (attr1 == 55 && attr2 == $1.attr2)
    $4 : C4 (attr1 == $1.attr2)
    $3 : C3 (attr1 == $2.attr1 and attr2 == $2.attr2)
then
    Actions
end
```

根据上述规则建立的 Rete 网络如图 1 所示. 图 1 中左半部分 β1~β6 为 Beta 网络中的 Beta 节点, 也称作连接节点. 连接节点的输入来自上一级的 Beta 节点 (β1 节点为 Beta 网络中第一个节点, 其左输入为 Beta 根节点) 和 Alpha 内存节点, 负责执行变量约束测试. 例如, β2 节点的输入来自上一级的 Beta 节点 β1 和 Alpha 内存节点 AM2, 负责执行变量约束测试“C2 : attr2 == $1.attr2”. β3 节点的左输入来自于 β2 和 Alpha 内存节点 AM3, 负责执行变量约束测试“C3 : attr1 == $2.attr1”和“C3 : attr2 == $2.attr2”. 其中 β1 和 β2 节点为 R1、R2 和 R3 规则共享, β4 节点为规则 R2 和 R3 共享.

定义 5. 规则路径

从 Beta 网络根节点到某个规则终止节点之间相连的 Beta 节点构成的路径.

定义 6. 规则路径节点集 RP

规则路径节点集为某条规则的规则路径上所有 Beta 节点的集合. 例如, 图 1 中的规则 R1 的路径节点集 $RP_{R1} = \{\beta1, \beta2, \beta3\}$.

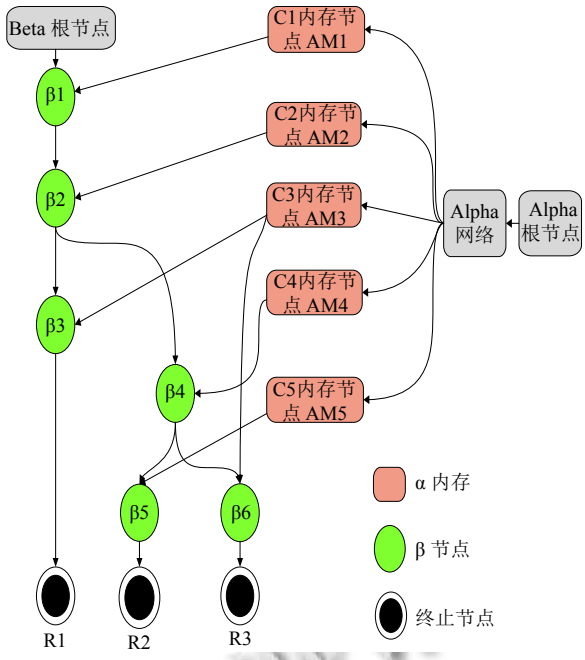

图 1 Rete 网络示例

1.2.3 Rete 算法的匹配过程

Rete 网建立后, 事件对象从 Alpha 根节点进入网络, 如果相关的 Alpha 常量约束测试通过, 则激活后继相关 Beta 节点. Beta 网络中的激活有两种方式, 如果 Beta 节点接收到左输入 Beta 节点传递来的部分匹配 (Partial Match, PM), 称为左激活. 左激活发生时, Beta 节点会遍历其相关的 Alpha 内存中的 Alpha 匹配 (AM) 和新到来的 PM 进行变量约束测试; 如果 Beta 节点接受右输入 Alpha 网络传递过来的 AM, 称为右激活. 此时, Beta 节点会遍历该节点的 PM 和输入的 AM 进行变量约束测试. 左激活和右激活中的变量约束测试如果通过则产生新的部分匹配 (PM) 向后继 Beta 节点传递并激活相关 Beta 节点.

激活的 Beta 节点存放在 Beta 节点激活队列中, 如图 2 所示. 激活的 Beta 节点按激活时间依次存入 Beta 激活队列的队尾, 后续依次从 Beta 激活队列的队首取出 Beta 节点进行处理, 即按照 FCFS (First Come First Service) 的方式处理.

1.2.4 Rete 算法的局限性

Rete 网络中 Beta 节点的处理是按 Beta 节点激活时间顺序处理, 即 FCFS (First Come First Service) 的方式, 这会造成时间约束强的监控场景中的 Beta 节点无法得到及时处理, 从而无法尽可能满足监控场景的时间约束. 例如, 假设规则 1 的规则路径节点集 RP1 = {β1, β2, β3, β6}, 规则 2 的规则路径节点集 RP2 = {β1, β2, β4, β7}. 规则 1 代表的是建筑节能监控场景 (例如, 当房间无人时, 关闭灯光和空调). 规则 2 代表的是建筑火灾监控场景 (例如, 当监控到房间发生火灾时, 立刻进行报警等). 显然规则 2 比规则 1 具有更强的时间约束. 假设 β4 代表火灾监控场景中的某个条件, 此时发生的事件使得 β4 被激活, 但 β4 会被放置在激活队列的队尾, 而 Rete 算法按照激活队列的中 Beta 节点激活顺序依次处理. 这可能会造成时间约束强的监控场景 (例如, 火灾监控场景) 得不到及时处理.

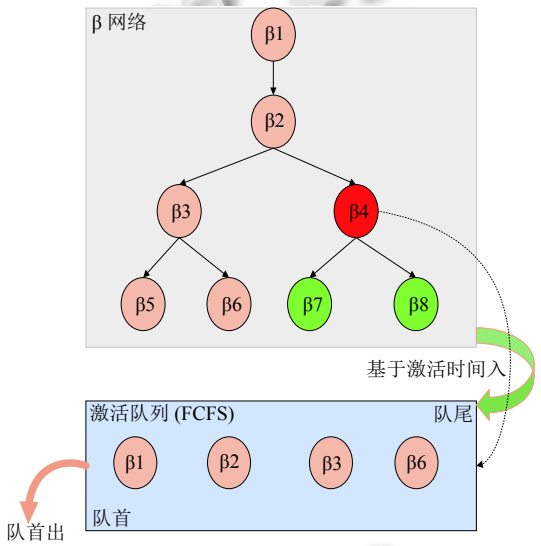

图 2 Rete 网中 Beta 节点处理顺序示例

1.3 基于规则的监控场景表示

假设当前存在这样一个建筑火灾监控场景: 当房间 4 个角落的烟雾传感器和温度传感器值同时超过阈值时, 则表示房间发生严重的火灾. 该火灾监控场景必须在 3 s 内监测到并触发火灾报警相关的行动, 否则将造成严重的损失. 上述建筑火灾监控场景可用规则表示如下:

```
rule “Room fire alarm”
deadline 3
when
   $j : J (temp>55 && smoke>100)
   $g : G (temp>55 && smoke>100)
   $e : E (temp>55 && smoke>100)
   $b : B (temp>55 && smoke>100)
then
   Raise fire alarm
end
```

2 Rete-TC 实时规则推理算法

当前可用规则表示监控场景,在规则推理中常使用 Rete 算法, 但 Rete 算法未考虑规则的截止期, 执行具有更晚截止期的规则可能会延迟具有较早截止期的规则, 这样可能会造成时间约束强的规则错失截止期, 从而无法尽可能满足监控场景的时间约束.

为了克服上述 Rete 算法的缺点, 基于 Rete, 我们提出了一种适用于实时监控的实时推理算法 Rete-TC (Timing Constraints). Rete-TC 算法引入了规则截止期, 通过基于优先级的 Beta 节点调度方法, 使得 CPS 监控尽可能满足时间约束.

2.1 Beta 节点调度

规则的截止期代表了该条规则的紧迫度. 所以, 具有更早截止期的规则应优先处理.

建立了 Beta 节点的优先级计算方法:

$$P(\beta) = \min_{r \in R_\beta} \{deadline(r)\} \quad (1)$$

其中, r 为一条规则, $r \in R_\beta$, R_β 是规则的路径节点集中包含 β 的规则集合. 例如, 图 1 中的规则 R1 路径节点集 $RP_{R1} = \{\beta1, \beta2, \beta3\}$. $deadline(r)$ 表示规则 r 的截止期; $P(\beta)$ 越小, β 节点的优先级越大. Beta 节点处理按照 Beta 节点的优先级从大到小的顺序进行, 从而使得具有更早截止期的规则将被优先触发. 另外, 当 $P(\beta)$ 为 0 时, β 节点按照激活时间顺序放入激活队列的末尾, 表示该 Beta 节点对应的规则无时间约束要求.

2.2 Rete-TC 算法描述

Rete-TC 算法有 3 部分组成, 分别是算法 1 Beta 节点优先级计算、算法 2 Rete-TC 网络构造算法、算法 3 Rete-TC 匹配, 各算法伪码如下. 算法 1 按照公式 (1) 计算每条规则产生的 Beta 节点的优先级, 优先级作为每个 β 节点的属性. 算法 1 在算法 2 Rete-TC 网络构造算法中调用.

按照式 (1)Beta 节点的优先级计算方法, 图 3 中的 β4 节点将依据优先级进入到激活队列中的相应位置 (具体位置可由 β4 节点的优先级与激活队列中已有的 Beta 节点的优先级比较决定, 最终激活队列中 Beta 节点优先级从队首到队尾优先级递减), 从而 β4 代表的具有强时间约束要求的火灾监测场景可以获得优先处理.

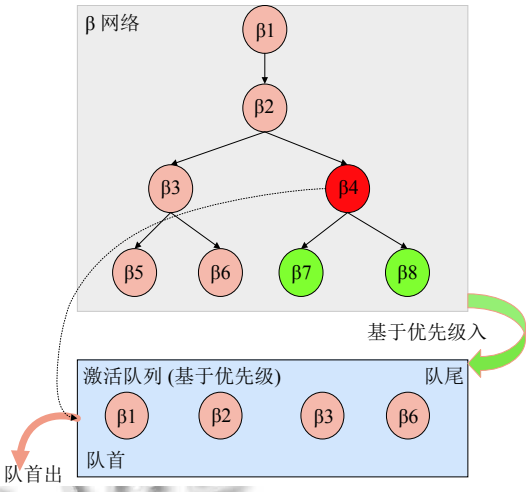

图 3 Rete-TC 网中 Beta 节点处理顺序示例

算法 2. Rete-TC 网络构造算法完成 Rete-TC 网络构造, 与 Rete 网络构造区别是增加 Beta 节点优先级计算 (在 Rete 网络构造的结尾处即第 17 行通过调用算法 1 实现). 由此可见, β 节点优先级在 Rete-TC 网络构造完成时已确定. 算法 2 中第 4~6 行完成 Alpha 网络的构造. 第 7~14 行完成 Beta 网络的构造. 算法 3 Rete-TC 匹配完成规则推理, 在 Rete 匹配算法的基础上主要增加了第 5、12 和 15 行 Beta 节点激活队列更新维护模块 UpdateAtiveNodesByPriority, 其余保持不变.UpdateAtiveNodesByPriority 模块内部主要采用红黑树数据结构, 支持在 O(log$N$) 时间复杂度内对优先级队列中元素进行排序 (N 为优先级队列中 Beta 节点个数), 相较于 Rete 算法中对激活的 Beta 节点管理采用的 FIFO 数据结构 (时间复杂度为 O(1)), 匹配时间虽略有增加, 但影响不大, 我们在实验评估环节进行了验证分析. 算法 3 中第 2~8 行完成 Alpha 节点处理, 第 9~17 行完成 Beta 节点处理.

Rete-TC 算法过程为先调用算法 2 Rete-TC 网络构造, 之后调用算法 3 进行 Rete-TC 匹配.

算法 1. Beta 节点优先级计算

输入: Rule Set
输出: The priority of each Beta node

```
01:  for Ri in Rule Set do
02:    for βi generated by Ri do
03:      The priority of βi← Equation(1)
04:    end for
05:  end for
```

算法 2. Rete-TC 网络构造

输入: Rule Set
输出: Rete-TC 网络

```
Create alpha root node and beta root node
for i = 1; i ≤ RuleNum; i++ do
    for j = 1; j < ConditionNum of Ri; j++ do
        if Condition_j 不在 Alpha 网络 then
            将 Condition_j 作为 Alpha(j) 节点插入 Alpha 网络
        end if
        if j ≥ 2 then
            if j == 2 then
                将 Beta(1) 节点插入到 Beta 网络, 左输入为 Beta root node 和右输入为 Alpha(1)
            end if
            if j > 2 then
                将 Beta(j) 节点插入 Beta 网络中, 其父节点为 Beta(j–1) 和 Alpha(j):
            end if
        end if
    end for
end for
调用算法 1 进行 Beta 节点优先级计算
```

算法 3. Rete-TC 匹配

输入: Events
输出: fired rules

```
while true do
    if root->alphaNodes[event->eventtype] then          //事件类型对应的 Alpha 类型节点存在
        while alpha condition node not empty do
            if Alpha node constant constraints test pass then //Alpha 节点常量测试通过
                UpdateAtiveNodesByPriority(child beta node) //依据优先级将 Alpha 节点关联的子 Beta 节点添加到 Beta 节点激活队列
            end if
        end while
    end if
    if head of Beta node activeated queue then //从 Beta 节点的激活队列取队首且不为空
        if Beta node's alpha match not empty then //该 Beta 节点 Alpha 匹配不为空
            doRightActivatedHandle(am, Beta node) //进行右激活处理
            UpdateAtiveNodesByPriority (child beta node) //依据优先级更新 Beta 节点激活队列
        elif Beta node's partial match not empty then //该 Beta 节点部分匹配不为空
            doLeftActivatedHandle(am, Beta node) //进行左激活处理
            UpdateAtiveNodesByPriority (child beta node) //依据优先级更新 Beta 节点激活队列
        end if
    end if
end while
```

3 系统框架

RTCPMS 系统框架如图 4 所示. RTCPMS 基于实时规则推理, 从 CPS 产生的大量事件中对关注的场景进行实时监控. RPCPMS 主要由事件抽取器、实时数据库系统、实时监控推理引擎和决策与反馈控制模块组成.事件抽取器持续采集 CPS 传感器产生的数据并抽取事件存入安捷 (Agilor)[33]实时数据库系统. 实时监控推理引擎通过 Agilor 提供的发布/订阅接口来访问实时数据库中的事件. 实时监控推理引擎由规则库、事件库、规则解析模块和基于 Rete-TC 的实时推理模块组成. 规则库存放领域专家制定的监控规则; 事件库存放由安捷实时数据库系统发布的原子事件. 规则解析模块将规则库中的规则翻译成 Rete-TC 网络; 基于 Rete-TC 的实时推理模块从事件库中取出事件在 Rete-TC 网络上传递, 进行规则匹配, 匹配成功的规则输出到决策和反馈模块. 决策和反馈模块执行规则中预设的动作, 例如触发一个报警或给用户反馈.

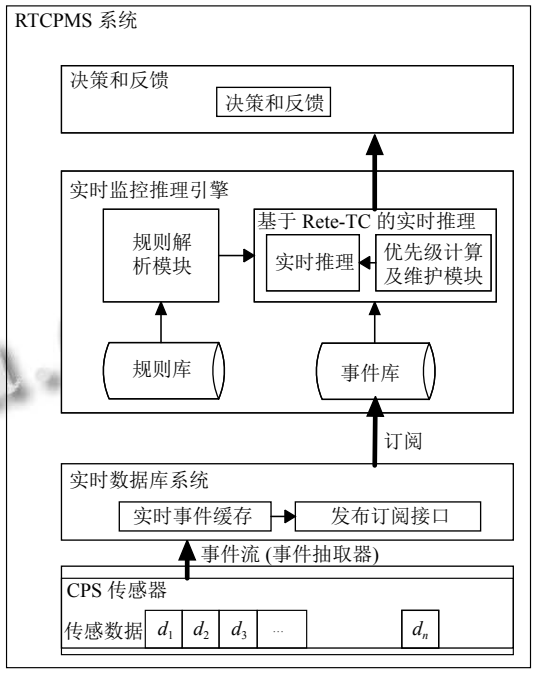

图 4 RTCPMS 的架构

4 实验评估

4.1 实验设置

CPS 的场景千变万化, “事件风暴”[34]是应用中存在的一类常见场景. 在这类场景中, 大量事件往往在一个较短的时间窗口内集中发生. 我们的模拟实验在此

场景下开展. 通过对 Rete-TC 和 Rete 算法进行对比, 来验证本文所提 Rete-TC 算法的有效性. 本次实验的规则集共包含 50 条规则. 每个规则的事件模式数为 1 至 8 个, 事件类型 120 种, 原子事件数 10 万到 30 万. 每个事件具有 2 到 4 个属性, 每个属性的属性值符合均匀分布 U[0, 1000], 各规则的截止期由应用需求决定, 一般设置后不再改变, 这里假设截止期服从均匀分布 U[1, 10], 所有的原子事件模拟产生.

规则集的参数情况见表 1.

表 1 规则集的参数

参数	值
事件模式数	1~8
事件类型数	120
事件属性数	2~4
规则数	50

我们使用成功率和匹配时间两个度量指标. 成功率的定义如下:

$$Success\ Ratio = N_{success}/N_{total} \quad (2)$$

其中, $N_{success}$ 表示在规则截止期前触发的规则数, N_{total} 表示所有触发的规则数.

匹配时间的定义如下:

$$Match\ Time = \text{Rete} - \text{TC} \text{ 或 Rete处理完指定输入规模下的所有事件所需时间} \quad (3)$$

实验环境使用 Intel(R) Core(TM) i7-8750H CPU @2.20 GHz, 16 GB 内存和 Windows 10.

4.2 成功率

我们对 Rete-TC 和 Rete 的成功率进行对比, 实验结果如图 5 所示. 图 5 展示了在输入的原子事件规模从 1.0×10^5 到 3.0×10^5 范围变化时, Rete-TC 和 Rete 的成功率变化情况.

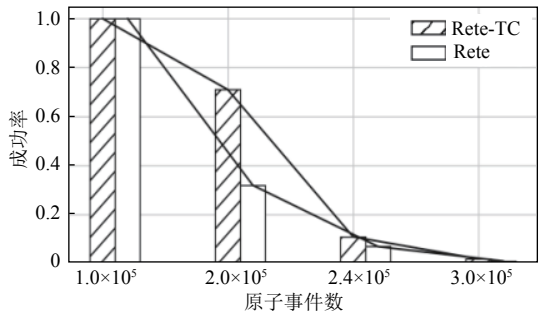

图 5 Rete-TC 和 Rete 的成功率

当原子事件数量小于 1.0×10^5, Rete-TC 和 Rete 下触发的所有规则都满足截止期.当输入的原子事件规模大于 1.0×10^5, 激活队列中的 Beta 节点的数量增多, Rete-TC 和 Rete 都会出现在规则截止期内规则对应的 Beta 节点未处理完情况, 从而导致 Rete-TC 和 Rete 触发的部分规则错失截止期. 当输入的原子事件规模超过 3.0×10^5 时, 大量规则触发, 激活队列中的 Beta 节点数激增, Rete-TC 和 Rete 在规则截止期内未能处理完相关 Beta 节点的情况激增, Rete-TC 和 Rete 的成功率趋于零. 总体上, 随着原子事件数的增加, Rete-TC 和 Rete 的成功率降低, 但是 Rete-TC 成功率均高于 Rete.

Rete-TC 依据式 (1) 对 Beta 节点的优先级进行计算, 充分考虑了各规则不同重要性, 其激活队列中的 Beta 节点按优先级排序, 这样时间约束强的规则得到及时处理, 从而使得各触发规则尽可能满足截止期约束. 相反, Rete 激活队列中的 Beta 节点按激活时间排序, 未考虑 Beta 节点所在规则的截止期, 忽略了各规则不同重要性, 执行具有更晚截止期的规则可能会延迟具有较早截止期的规则, 这样可能会造成时间约束强的规则错失截止期, 不利于各触发规则满足截止期约束.

4.3 匹配时间

我们对 Rete-TC 和 Rete 的匹配时间进行对比, 实验结果如图 6 所示. 从图 6 得知, 在相同的原子事件规模输入下, Rete-TC 相比 Rete 匹配时间略微增加, 也就是说, 本文在 Rete-TC 算法增加的 Beta 节点优先级调度特性对处理原子事件的效率的影响非常小.另外,由于 Rete-TC 算法本质上是改变了处理 Beta 节点的顺序, 不会对推理结果正确性产生影响. 每次实验也验证了 Rete-TC 和 Rete 的推理结果是完全一致的.

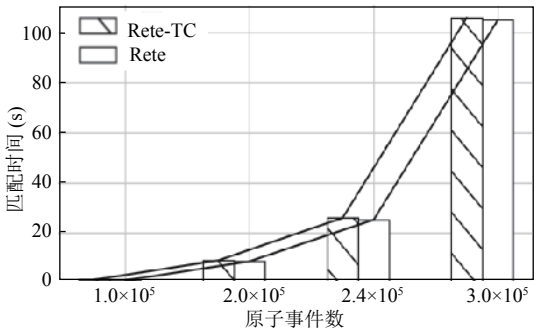

图 6 Rete-TC 和 Rete 的匹配时间

在相同的原子事件输入规模下, Rete-TC 相比 Rete 匹配时间略增的原因讨论如下:

Rete-TC 中激活的 Beta 节点优先级队列的数据结构采用红黑树, 支持在 O(log$N$) 时间复杂度内对优先级队列中元素进行排序, 其中 N 为优先级队列中 Beta 节点个数. 而 Rete 算法中对激活的 Beta 节点进行管理采用 FIFO 数据结构, 时间复杂度为 O(1). 在本文的实验环境下 Rete-TC 中 Beta 节点优先级激活队列中 Beta 节点最大数量级约为 10^4, Beta 节点优先级队列单次操作时间约为 10^{-6} s, Rete-TC 算法用于维护 Beta 节点优先级队列的时间数量级约为 10^{-2} s, 所以 Rete-TC 相比 Rete 匹配时间略增.

4.4 截止期讨论

规则的截止期通常由应用需求确定, 确定后一般不再改变. 但也可能存在某些特殊场景, 例如由于监控外部环境或者监控需求发生特殊变化, 规则截止期需动态改变, 下面对此问题给出一种解决思路. 解决思路为: (1) 在 Rete-TC 网络建立阶段采用哈希表建立每个 Beta 节点关联的规则集合, Rete-TC 匹配运行阶段直接使用即可. (2) 在 Beta 节点激活队列更新维护模块 UpdateAtiveNodesByPriority 中获取准备进入激活队列的 Beta 节点关联的规则集合中各规则最新截止期. (3) 在 Beta 节点激活队列更新维护模块 UpdateAtive-NodesByPriority 中对准备进入激活队列的 Beta 节点的优先级按照式 (1) 进行计算, 之后按 UpdateAtive-NodesByPriority 原有逻辑运行即可. 由于步骤 (2) 和 (3) 计算过程简单, 对 Beta 节点激活队列更新维护模块的运行时间开销影响小. 关于动态截止期的问题后续可进一步深入研究.

5 应用案例

随着 CPS 的快速发展, 建筑 CPS 逐渐兴起, 传统的建筑正在演变成集环境感知、实时分析、科学决策和精准执行功能于一体的智慧建筑. 以建筑 CPS 的监控为例, 如图 7 所示. 图 7 中建筑物理环境蕴含的隐性数据通过环境感知转换为显性数据, 进而能够在信息空间进行实时分析, 从而将显性数据转换为有价值的信息. 信息经过综合处理形成最优决策对物理空间实体进行精确调节. 图 7 中有 3 类场景需要监控, 分别是火灾场景、节能场景和环境舒适度调节场景. 其中, 火灾场景具有时间约束. 节能场景和环境舒适度调节场景只需尽可能快监控即可, 时间约束较为宽松, 错失截止期不会造成严重后果. 每个场景都由一系列原子事件所喻示. 例如, 房间内温度升高事件和烟雾产生事件可能喻示火灾发生. 房间内检测到无人事件且用电设备运行中事件喻示识别到潜在的节能场景, 需进行节能管理, 否则会造成能源浪费.

图 7 建筑 CPS 监控示意

为了验证 RTCPMS 系统的有效性, 我们以北京市建筑设计研究院有限公司某大楼作为案例研究对象. 该大楼总共有 13 层, 每层有 25 个房间, 每个房间大约 20 个传感器. 其中的某个房间的传感器布局如图 8 所示.

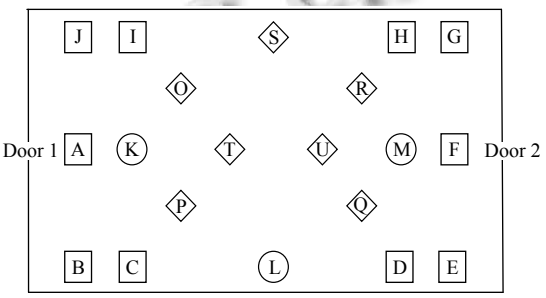

图 8 大楼某房间中的传感器布局

以该大楼的 3 个监控场景需求为例. (1) 关于火灾监控场景. 我们在位置 A–J 处部署一个温度传感器和一个烟雾传感器. 如果房间 4 个角落的传感器感知的数据同时超出阈值, 这表明该房间已发生严重火灾; 每个房间有两个防火门, 当火灾蔓延到防火门的时候, 防火门需要及时关闭. (2) 关于环境健康参数方面的智能控制场景. 在 K–M 位置处, 我们部署环境健康相关的监控传感器来收集 PM25、温度和湿度的数据. (3) 关

于节能管理场景. 在 O–U 的位置, 我们部署人员探测传感器, 当探测到无人时, RTCPMS 自动关闭灯光、空调和空气净化器. (1) 应最大化地满足时间约束, (2)(3) 需尽可能快的处理.

根据上述监控需求, 我们制定规则, 如表 2 所示.

表 2 智慧建筑监控规则

规则名称	规则内容
房间火灾报警	rule “Room fire alarm” deadline 3 when $j: J (temp>55 && smoke>100) $g: G (temp>55 $g: G (temp>55 && smoke>100) $e: E (temp>55 && smoke>100) $b: B (temp>55 && smoke>100) then Raise fire alarm end
关闭防火门	rule “Close the fire door” deadline 3 when $j: J (temp>55 && smoke>100) $d1:A(temp>55&&smoke>100&& $j.timestamp < timestamp) then Close the fire door1 end
启动空调制冷	Rule “Start the air conditioning cooling” deadline 60 when $a: ac (temp>30 && humidity>60%RH &&Infrared=1) then Start the air conditioner cooling
启动空气净化器	rule “Start the air purifier” deadline 60 when $a: ap ((pm25>75 || pm10>150) && Infrared=1) then Start the air purifier end
启动空调制热	rule “Start the air conditioning heating” deadline 60 when $a: ach (temp<13&&humidity< 30%RH && Infrared=1) then Start the air conditioning heating
能耗管理	rule “Energy management” deadline 60 when $a: personnel (infrared==0) then Turn off the lighting, air conditioning, and air purifier end

传感器产生的事件由采集程序进行采集并存储在实时数据库中. 大型建筑的火灾场景是一种典型的事件风暴场景, 当火灾发生时, 在短时间内, 用于火灾监控的各类传感器将上报大量事件, 如温度超标、烟雾浓度超标等.

应用案例的实验结果如图 9 所示. 图 9 中 RTCPMS 使用 Rete-TC 算法, RTCPMS-WITHOUT-TC 使用 Rete 算法. 当原子事件数在 1.0×10^5 至 2.5×10^5 范围内, RTCPMS 可以使得火灾报警规则在截止期内触发. RTCPMS-WITHOUT-TC 只能确保原子事件数在 1.0×10^5 至 2.0×10^5 范围内, 火灾报警规则在截止期内触发. 当输入的原子事件数超过 2.5×10^5 时, RTCPMS 和 RTCPMS-WITHOUT-TC 都会发生规则错失截止期情况. 但 RTCPMS 处理火灾报警类规则的成功率始终高于 RTCPMS-WITHOUT-TC (平均高 12%). 从这个智慧建筑的应用案例可以看出, 本文建立的 RTCPMS 系统可以更有效的满足 CPS 监控的时间约束.

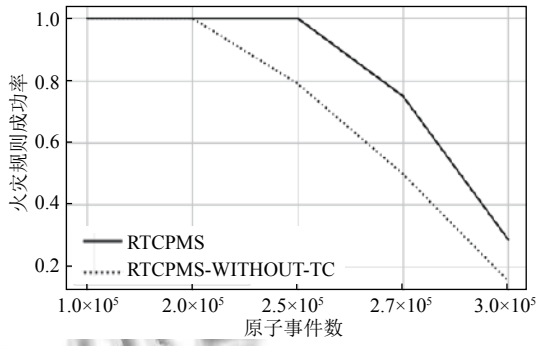

图 9 案例运行结果

6 总结

CPS 是将计算过程和物理过程集成的系统, 利用嵌入式计算机和网络对物理过程进行监测和控制, 并通过反馈环实现计算和物理过程的相互影响. CPS 监控是从物理设备产生的事件中感知关注场景并作出响应的过程, 是 CPS 的核心功能之一. 在 CPS 监控中, 事件与动作之间的因果关系往往使用规则来描述. 采用规则系统相较于过程式逻辑具有若干优势, 为此, 学者们提出了基于规则的 CPS 监控方法, 然而 CPS 的实时性为 CPS 监控带来了时间约束问题. 目前已有的基于规则的 CPS 监控方法未考虑 CPS 监控场景的时间约束, 仅仅利用各种优化技术来缩短监控的响应时间. 为此, 本文基于实时规则引擎建立了一个 CPS 的实时监

控系统 RTCPMS. 该系统采用 Rete 网络表示监控规则.

RTCPMS 将实时推理技术和 CPS 监控结合, 其核心是一个新的实时推理算法 Rete-TC. Rete-TC 算法引入了规则截止期, 通过基于优先级的 Beta 节点调度方法, 使得 CPS 监控的时间约束尽可能地被满足. 模拟实验与智慧建筑应用案例验证了 RTCPMS 系统的有效性, 且实验结果表明其核心算法 Rete-TC 的调度成功率优于传统的规则推理算法 Rete.

CPS 中, 由于传感器扰动, 可能会产生不精确的事件, 当前基于规则推理的 CPS 监控不适用于此场景. 下一步的研究中, 我们将基于 Rete-TC, 研究实时的模糊推理技术, 来支持对不精确的事件进行实时监控, 从而提升 CPS 实时监控的鲁棒性. 另外, 当前基于规则的 CPS 监控方法需要事先由领域专家制定规则, 不适应于复杂的动态场景的自适应监控. 随着机器学习的广泛使用, 将机器学习与规则进行结合也是一个有潜力的研究方向.

致谢. 在此, 我们向对本文的工作给予帮助的中国科学院软件研究所人机交互技术与智能信息处理实验室的姚乃明博士、冷昶博士和马翠霞老师以及评阅此文的各位专家表示感谢.

参考文献

1 Lee EA. CPS foundations. Design Automation Conference. Anaheim, CA, USA. 2010. 737–742. [doi: 10.1145/1837274.1837462]

2 Seshia SA, Hu SY, Li WC, *et al*. Design automation of cyber-physical systems: Challenges, advances, and opportunities. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017, 36(9): 1421–1434. [doi: 10.1109/TCAD.2016.2633961]

3 Kaihara T, Yao Y. A new approach on CPS-based scheduling and WIP control in process industries. Proceedings of the 2012 Winter Simulation Conference. Berlin, Gemany. 2012. 1–11. [doi: 10.1109/WSC.2012.6465319]

4 Gong Y, Li SJ. Fusion framework of urban traffic control and route guidance based on cyber-physical system theory. Journal of Highway and Transportation Research and Development (English Edition), 2013, 7(1): 82–89. [doi: 10.1061/JHTRCQ.0000029]

5 Shi JH, Wan JF, Yan HH, *et al*. A survey of cyber-physical systems. Proceedings of 2011 International Conference on Wireless Communications and Signal Processing. Nanjing, China. 2011. 1–6. [doi: 10.1109/WCSP.2011.6096958]

6 Li WJ, Meng WZ, Su CH, *et al*. Towards false alarm reduction using fuzzy if-then rules for medical cyber physical systems. IEEE Access, 2018, 6: 6530–6539. [doi: 10.1109/ACCESS.2018.2794685]

7 Hackmann G, Guo WJ, Yan GR, *et al*. Cyber-physical codesign of distributed structural health monitoring with wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(1): 63–72. [doi: 10.1109/TPDS.2013.30]

8 中国电子技术标准化研究院. 信息物理系统白皮书. 2017. 1–110.

9 耿少峰. 面向信息物理系统的主动式复杂事件处理技术研究[博士学位论文]. 长沙: 湖南大学, 2018.

10 李想. 基于主动规则的实时推理技术研究[博士学位论文]. 北京: 中国科学院大学, 2011.

11 Sun Y, Wu TY, Zhao GT, *et al*. Efficient rule engine for smart building systems. IEEE Transactions on Computers, 2015, 64(6): 1658–1669. [doi: 10.1109/TC.2014.2345385]

12 Sun Y, Wang XK, Luo H, *et al*. Conflict detection scheme based on formal rule model for smart building systems. IEEE Transactions on Human-Machine Systems, 2015, 45(2): 215–227. [doi: 10.1109/THMS.2014.2364613]

13 Sun Y, Wu TY, Li XM, *et al*. A rule verification system for smart buildings. IEEE Transactions on Emerging Topics in Computing, 2017, 5(3): 367–379. [doi: 10.1109/TETC.2016.2531288]

14 Forgy CL. Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 1982, 19(1): 17–37. [doi: 10.1016/0004-3702(82)90020-0]

15 Klein R, Xie JQ, Usov A. Complex events and actions to control cyber-physical systems. Proceedings of the 5th ACM International Conference on Distributed Event-Based System. New York, NY, USA. 2011. 29–38. [doi: 10.1145/2002259.2002265]

16 Chakravarthy S, Krishnaprasad V, Anwar E, *et al*. Composite events for active databases: Semantics, contexts and detection. Proceedings of the 20th International Conference on Very Large Data Bases (VLDB). San Francisco, CA, USA. 1994. 606–617.

17 Luckham D. The power of events: An introduction to complex event processing in distributed enterprise systems. In: Bassiliades N, Governatori G, Paschke A, eds. Rule Representation, Interchange and Reasoning on the Web. Berlin: Springer, 2008. [doi: 10.1007/978-3-540-88808-6_2]

18 Dean T, Boddy M. An analysis of time-dependent planning.

Proceedings of the 7th National Conference on Artificial Intelligence. St. Paul, UK. 1988. 49–54.

19 Garvey AJ, Lesser VR. Design-to-time real-time scheduling. IEEE Transactions on Systems, Man, and Cybernetics, 1993, 23(6): 1491–1502. [doi: 10.1109/21.257749]

20 Lesser VR, Pavlin J. Durfee E. Approximate processing in real-time problem solving. AI Magazine, 1988, 9(1): 49–61. [doi: 10.1609/aimag.v9i1.661]

21 Mouaddib AI, Charpillet F, Haton JP. GREAT: A model of progressive reasoning for real-time systems. Proceedings of the 6th International Conference on Tools with Artificial Intelligence. New Orleans, LA, USA. 1994. 521–527. [doi: 10.1109/TAI.1994.346447]

22 李想, 高红菊, 乔颖, 等. 面向物联网的实时复杂事件处理引擎. 小型微型计算机系统, 2015, 36(9): 2047–2053. [doi: 10.3969/j.issn.1000-1220.2015.09.025]

23 Liu GY, Zhu WD, Saunders C, *et al*. Real-time complex event processing and analytics for smart grid. Procedia Computer Science, 2015, 61: 113–119. [doi: 10.1016/j.procs.2015.09.169]

24 Van Woensel W, Abidi SSR. Optimizing semantic reasoning on memory-constrained platforms using the RETE algorithm. In: Gangemi A, Navigli R, Vidal ME, *et al*., eds. The Semantic Web. Cham: Springer, 2018. 682–696. [doi: 10.1007/978-3-319-93417-4_44]

25 Sottara D, Mello P, Proctor M. A configurable Rete-OO engine for reasoning with different types of imperfect information. IEEE Transactions on Knowledge and Data Engineering, 2010, 22(11): 1535–1548. [doi: 10.1109/tkde.2010.125]

26 Sottara D, Mello P, Proctor M. Adding uncertainty to a Rete-OO inference engine. In: Bassiliades N, Governatori G, Paschke A, eds. Rule Representation, Interchange and Reasoning on the Web. Berlin: Springer, 2008. 104–118. [doi: 10.1007/978-3-540-88808-6_13]

27 Özacar T, Öztürk Ö, Ünalir MO. Optimizing a rete-based inference engine using a hybrid heuristic and pyramid based indexes on ontological data. JCP, 2007, 2(4): 41–48. [doi: 10.4304/jcp.2.4.41-48]

28 Xiao D, Zhong XA. Improving rete algorithm to enhance performance of rule engine systems. Proceedings of 2010 International Conference on Computer Design and Applications. Qinhuangdao, China. 2010. [doi: 10.1109/iccda.2010.5541368]

29 孙新, 严西敏, 尚煜茗, 等. 一种基于共享度模型的改进 Rete 算法. 自动化学报, 2017, 43(9): 1571–1579. [doi: 10.16383/j.aas.2017.c160674]

30 Ju H, Oh S. Enabling RETE algorithm for RDFS reasoning on apache spark. Proceedings of 2018 IEEE 8th International Symposium on Cloud and Service Computing. Paris, France. 2018. 135–138. [doi: 10.1109/SC2.2018.00028]

31 汪成亮, 温鑫. 智能环境下分布式 Rete 算法. 计算机应用, 2016, 36(7): 1893–1898. [doi: 10.11772/j.issn.1001-9081.2016.07.1893]

32 规则推理引擎 Drools. http://www.drools.org/. [2019-06-13].

33 安捷实时数据库系统 Agilor. http://agilor.iscas.ac.cn/. [2019-06-13].

34 Albaghdadi M, Briley B, Evens M. Event storm detection and identification in communication systems. Reliability Engineering & System Safety, 2006, 91(5): 602–613. [doi: 10.1016/j.ress.2005.05.001]