RTCS:一种具有精确语义的实时构件描述机制

徐丽萍　贾红卫　卢炎生

(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　大规模和复杂的实时系统可以显著获益于基于构件的软件开发方法,即通过已有的经过验证的可复用构件来构造实时系统,如能将这一集成过程自动化,将会显著提高实时系统的开发效率。通过对实时任务特性的分析,在Timed CSP等形式化工具的基础上,提出了一种具有精确语义的实时构件描述机制—RTCS,并探讨了在实时CORBA架构内利用RTCS实现构件自动生成的方法。

关键词　实时构件,构件描述,构件集成

RTCS: A Method of Real-Time Component Specification with Precise Semantics

XU Li-Ping　JIA Hong-Wei　LU Yan-Sheng

(Department of Computer Science and Technology, Huazhong University of Sci. &Tec., Wuhan 430074)

Abstract　Large and complex real-time systems can benefit significantly from a component-based development approach where new systems are constructed by composing reusable and previously tested components, if this process of component composition can be automated, the efficiency of system development would be largely promoted. This paper presents an approach with precise semantics, based on formal tools such as Timed CSP, to the specification and specification matching of real-time components(RTCS). In the context of Real-Time CORBA, this paper also discusses the method of using RTCS to automate the composition of real-time components.

Keywords　Real-time component, Component specification, Component composition

1　引言

传统的实时系统大多采用低级的程序设计语言编写,并在专用的硬件和操作系统之上运行,运行效率、资源利用率以及与硬件的集成是考虑的主要方面,模块化和复用性是在第二位的。但随着开发要求的提高(时间、成本),实时系统的构件化越来越受到重视。

实时系统的构件化远比普通软件复杂,因为实时任务具有时间和同步的约束。实时任务可以看做是对多个功能构件的并发执行施加约束而完成的,也就是进行调度,因此,要实现实时构件复用,必须消除功能和约束间的耦合,将功能构件作为复用的基础,对不同的实时任务,设计不同的调度构件,协调功能构件的行为以完成实时任务[1]。

另一方面,对于特定的实时应用领域而言,功能构件可以事先定制,变化的通常是任务,即调度构件。而且,客户端和服务端的交互主要是启动实时任务并对其进行监控,调度构件的接口可以是恒定的,这使得当任务要求发生变化时,生成相应的服务端构件并替换客户端的对象引用即可。如能根据任务要求自动生成调度构件,将会大大提高系统的可扩展性和节约开发成本。

基于以上分析,将功能构件和调度构件加以区分有利于提高实时系统的构件化程度,遵循这一原则,本文提出一种具有精确语义的构件描述机制—RTCS(Real-Time Component Specication),包括功能构件和调度构件的描述及匹配(见图1)。精确的构件描述对于构件查找、构件验证和构件集成的自动化具有重要意义。在实时CORBA架构内,本文还探讨了通过分析构件描述自动生成调度构件的方法。

本文第2节定义功能构件描述及匹配,第3节定义调度构件描述及匹配,第4节探讨构件描述在实时CORBA框架内的应用。

2　功能构件描述

功能构件是实时系统中完成特定功能的一个独立实体,是调度构件调度的对象。构件描述的主要目的之一是构件匹配,即在已有的构件中查找满足要求的构件,这要求构件描述必须具有精确的语义,目前软件构件的描述语言有许多种[2~5],但都不涉及构件的性能特征,这对于实时构件是不够的。例如,一个实时服务,每20毫秒从传感器读取一次数据,处理后发送到监视器,显然此服务本身的计算时间不能超过20毫秒,如果构件描述不包括性能特征,则无法表达对构件的性能要求,也无法确定一个构件实现是否满足要求。

所以在实时应用环境中,功能构件描述既要刻画构件具有什么样的功能,还要刻画构件对QoS(Quality of Service)有什么样的保证。后者是必须的,因为任何实时任务的执行都必须具有可预测性(predicability),这也是实时构件和非实时构件的本质区别所在。在本文中,QoS是指构件的非功能方面的特性,如性能、可靠性和安全性等[6]。

构件的功能特征取决于构件对外提供什么样的服务,这些服务对应于构件内的方法,所以构件的描述和匹配以方法描述和匹配为基础。文[4]定义了不同严格程度的方法匹配,下面先给出QoS描述及匹配的定义,然后对文[4]的方法描述加以改进,定义包含QoS特征的功能构件描述和匹配。

2.1　QoS描述

在实时系统中,功能构件每个方法的执行都必须是可预测的,即必须具有明确的QoS规格。从全系统的角度看,对于每一个操作,并非QoS越高越好,否则可能造成系统资源的浪费,因为较高的QoS通常以占用较多的资源(内存、带宽等)为代价。

徐丽萍　副教授,研究方向为软件工程。贾红卫　硕士,研究方向为软件工程。卢炎生　教授,博士生导师,主要研究方向为软件工程、特种数据库。

为了提高系统的吞吐量，对于同一服务，可以提供多个具有不同 QoS 的实现，以满足不同的任务要求。从下一节可以看到，这些以 QoS 区分的服务，既可在不同构件中，也可在同一构件中。

定义 1（QoS 描述） QoS 描述是一个集合 $\{q_1, \cdots, q_n\}$，集合的每一个元素是一个二元组 $\langle name, requirement\rangle$，*name* 是 QoS 要素的名称，*requirement* 是关于 *name* 的一个断言。

例如：采样时间 STime 小于 10ms，处理时间 PTime 小于 6ms，QoS 描述为{〈STime，Stime<10ms〉，〈PTime，PTime<6ms〉}。

有了 QoS 描述，还要明确一个构件满足给定 QoS 描述的含义：

定义 2（QoS 满足关系） 给定一个 QoS 描述 Q_{query}，一个方法的 QoS 描述 $Q_{liberary}$ 满足 Q_{query}，记为 $Q_{liberary}$ sat Q_{query}，当且仅当：

$\forall q_q \in \rho Q_{query} \cdot (\exists q_l \in Q_{liberary} \cdot (name(q_l) = name(q_q) \wedge requirement(q_l) \Rightarrow requirement(q_q)))$，其中 *name* 和 *requirement* 分别析取 QoS 要素名称和断言，ρ 是一系列的标识符重命名。

例如：{〈STime，Stime<8ms〉，〈PTime，PTime<6ms〉} sat {〈STime，Stime<10ms〉，〈PTime，Ptime<6ms〉}

2.2 功能构件描述

定义 3（功能构件描述） 功能构件描述是由多个方法描述构成的，定义了功能构件对外提供的服务及其 QoS 特征（以 QoS 描述表达）。

实时构件的执行是以方法为单位的，下面对文[4]中的方法描述及匹配进行改进，以包括 QoS 特征。

定义 4（方法描述） 方法描述刻画了一个方法的特征，包括名称、参数、返回值、执行前提（pre-condition）、执行结果（post-condition）和 QoS 特性。方法描述具有以下形式：

method method_name ((*Var*：*DomainSort*)*) → Var：RangeSort
requires pre-condition
ensures post-condition
peformance QoS-specification

方法描述 *M* 在逻辑上分为两部分：

I）方法的签名，记为 $signature(M)$，表示方法的参数及返回值。例如一个整形加法函数 IntAdd 的签名为：$signature(IntAdd) = int \times int \rightarrow int$。

II）方法的断言，记为 $predicate(M)$，刻画了方法的行为特征，$predicate(M) \overset{def}{=} pre\text{-}condition(M) \Rightarrow post\text{-}condition(M)$。

例如，设有方法 *member*，判断一个实数 *a* 是否是一维数组 *A*[1..100]的元素，判断在 10ms 内完成，则 *member* 的描述为：

method member(a:*real*，A:*real*[1..100])→β:*bool*
requires $\neg empty(A)$
ensures $\beta = a \in A$
peformance {〈ProcessTime，ProcessTime<10ms〉}
$signature(member) = real \times real[1..100] \rightarrow bool$，$predicate(member) = \neg empty(A) \Rightarrow \beta = a \in A$。

给定一种方法描述机制，可以定义不同严格程度的方法匹配，其中最严格的是精确匹配，但精确匹配可能会忽略比给定描述更“明确”的方法，即描述比方法实现更“一般”或是宽泛。例如一个描述 *S* 要求删除数组中任一元素，存在一个方法 *M* 删除给定数组的最后一个元素，虽然 *M* 和 *S* 不是精确匹配，但 *M* 满足 *S* 的功能要求。所以，在方法签名精确匹配的前提下，行为匹配应采用“一般匹配”（Generalized Match）。

下面定义包含 QoS 特征的方法匹配，其中签名匹配采用精确匹配，断言匹配采用“一般匹配”，相应的形式化推导见文[4,7]。

定义 5（签名匹配） 设 M_{query} 是待查询的方法描述，$M_{library}$ 是一个方法实现，$M_{library}$ 和 M_{query} 签名匹配，记为 $M_{library}$ $match_{sig}$ M_{query}，当且仅当：

$\exists \rho \cdot \rho signature(M_{library}) = signature(M_{query})$，其中 ρ 是一系列的变量重命名。

定义 6（断言匹配） 设 M_{query} 是待查询的方法描述，$M_{library}$ 是一个方法实现，$M_{library}$ 和 M_{query} 断言匹配，记为 $M_{library}$ $match_{pred}$ M_{query}，当且仅当：

$predicate(M_{library}) \Rightarrow predicate(M_{query})$。

定义 7（方法匹配） 设 M_{query} 是待查询的方法描述，$M_{library}$ 是一个方法实现，$M_{library}$ 和 M_{query} 匹配，记为 $M_{library}$ $match_{mthd}$ M_{query}，当且仅当：

$(M_{library}\ match_{sig}\ M_{query} \wedge M_{library}\ match_{pred}\ M_{query}) \wedge (performance(M_{library})\ sat\ performance(M_{query}))$，其中 *performance* 析取方法的 QoS 描述。

功能构件描述是由一个或多个方法描述构成的，因此功能构件匹配也通过方法匹配定义。

定义 8（功能构件匹配） 设 C_{query} 是待查询的功能构件描述，$C_{liberary}$ 是一个功能构件实现，$C_{liberary}$ 和 C_{query} 匹配，记为 $C_{liberary}$ match C_{query}，当且仅当：

$\forall\ m_{query} \in C_{query} \cdot \exists\ m_{liberary} \in C_{liberary} \cdot (m_{liberary}\ match_{mthd}\ m_{query})$

功能构件是完成实时任务的基础，也是实时系统中构件复用的主要内容。在定义功能构件描述及匹配的基础上，下一节给出调度构件描述的定义。

3 调度构件描述

调度构件的作用是调度多个功能构件完成特定的实时任务。下面以 Timed CSP[8,9]为工具，逐步给出调度构件描述的定义。Timed CSP 是一种形式化的描述语言，用于建模和推导带有明确时间约束的并发、通信系统。在实际应用中，可以提供图形化工具辅助生成调度构件描述。

定义 9（调度构件描述） 一个调度构件描述由任务描述、构件字典和行为描述三部分组成，刻画了实时任务的目标以及如何调度功能构件去实现这一目标。调度构件描述是一个自包含的体系，根据描述本身提供的信息，可以自动生成满足任务描述的构件实现。

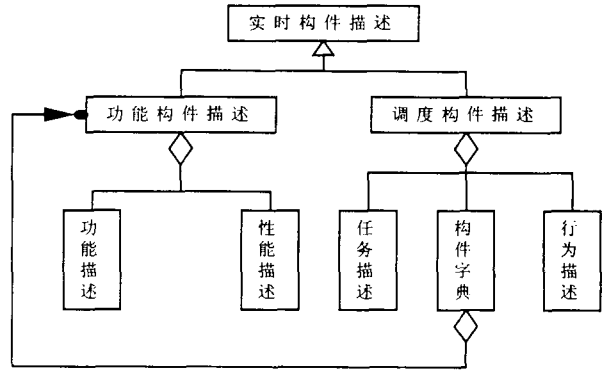

图 1 实时构件描述结构图

下面分别定义任务描述、构件字典和行为描述，并以一个简化的水泵控制系统[10]给出示例，该系统具有以下功能：水位监视器每 20ms 从水位传感器读取一次数据，如果水位过高或是过低，则向水泵控制器发出开启或是关闭命令，在接到命令后，水泵控制器要在 10ms 内完成相应操作。系统控制流程见图 2。

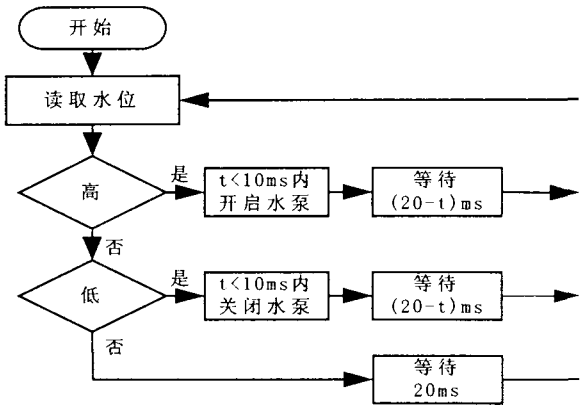

图 2　水泵控制系统流程

3.1　任务描述

任务描述刻画调度构件所代表的实时任务必须具有什么样的特征，是对调度构件行为的一个断言。任务描述的作用是检查调度构件描述的一致性，即行为描述是否满足任务描述提出的约束，这一检查可以通过 Timed CSP 的形式化推导完成[10]。

定义 10(任务描述)　任务描述刻画了调度构件所执行的实时任务要完成什么样的功能。任务描述采用 Timed CSP 中进程描述(Process Specification)的方法。

例：水泵控制系统的任务描述为：

$F_1 = \forall t \cdot ((t, \text{water-low})) \text{ in } s \Rightarrow \text{PUMP-STATUS} = \text{pump-is-off} \vee \text{pump-is-off} \in \sigma(s \uparrow (t, t+10])$,

$F_2 = \forall t \cdot ((t, \text{water-hi})) \text{ in } s \Rightarrow \text{PUMP-STATUS} = \text{pump-is-on} \vee \text{pump-is-on} \in \sigma(s \uparrow (t, t+10])$,

$F^* = F_1 \wedge F_2$。

3.2　构件字典

定义 11(构件字典)　构件字典保存调度构件需要的所有功能构件的信息。功能构件信息是一个二元组：〈功能构件名，功能构件描述〉。

在创建调度构件时，对于每一个功能构件名，查构件字典得到其描述，然后利用功能构件描述检索并获取所需要的构件实例。

例：在水泵控制系统中，涉及水泵控制器 PumpController 和水位传感器 WaterSensor 两个功能构件，为简化起见，假设只需要计算时间 t 一个 QoS 要素，且忽略方法的计算时间。WaterSensor 在构件字典中的记录为：

〈WaterSensor, {method GetLevel()→*result*:*real* ensures *result* = *currentWaterLevel* performance {(t, t=0)}}〉

3.3　行为描述

行为描述是调度构件描述的核心内容，刻画了调度构件怎样完成任务描述指定的实时任务。行为描述以 Timed CSP 表达，具有精确的语义。通过分析调度构件的行为描述，针对特定的硬件环境可以自动生成构件实例，这是非形式化的构件描述所无法做到的。

定义 12(行为描述)　行为描述是由一系列模块进程通过同步并行组装而构成的，这些模块进程的并行组装代表了实时任务执行的逻辑模型。

定义 13(模块进程)　模块进程是一个 Timed CSP 进程，代表了实时系统中的一个逻辑单元。模块进程中的 CSP 事件是对功能构件方法的一次调用。

设 P 和 Q 都是模块进程，模块进程的构造有以下三种方式：

1)基本进程：STOP 或 SKIP，前者表示不能执行任何操作的功能构件(处于停机状态)，后者表示成功结束然后停机的功能构件，即 SKIP≡√→STOP。

2)前缀：记为 $\alpha \rightarrow P$，表示执行过 α 后，变为和 P 等价。前缀还可以带有时间属性，记为 $\alpha \xrightarrow{t} P$，表示执行过 α 后，经过 t 时间，变为和 P 等价。其中 α 对应功能构件的一个方法。

3)组装：(1)顺序组装：记为 $P;Q$，表示先执行 P，P 成功完成后再执行 Q。(2)外部选择组装：记为 $P \square Q$，由外部环境需求决定执行 P 还是执行 Q。(3)内部选择组装：记为 $P \sqcap Q$，由 $P \sqcap Q$ 所在进程，而不是外部环境决定执行 P 还是执行 Q。(4)超时选择组装，记为 $P \overset{t}{\triangleright} Q$，表示如果在 t 内 P 开始执行，则执行 P，否则执行 Q。Q 通常用于超时处理。(5)同步并行组装：记为 $P|[C]|Q$，设 α 是 P 或 Q 中的一个功能构件方法，如果 $\alpha \in C$，在 P 和 Q 都就绪时 α 才可执行，否则 α 可自由执行。(6)中断组装：记为 $P \Delta Q$，首先执行 P，如果 Q 开始执行，则中止 P 的执行。Q 通常用于故障处理和恢复。

即，模块进程 $P ::= \text{STOP} | \text{SKIP} | \alpha \rightarrow P | \alpha \xrightarrow{t} P | P;P | P \square P | P \sqcap P | P \overset{t}{\triangleright} P | P|[C]|P | P \Delta P$。显然，模块进程的定义是递归的。

例：在上文提到的水泵控制系统中，包括以下四个模块进程：

Ⅰ)水泵模块进程 P

$P = pump\text{-}is\text{-}off \rightarrow P_0$

$P_0 = pump\text{-}on \xrightarrow{t1} pump\text{-}is\text{-}on \rightarrow P_1 \square pump\text{-}off \rightarrow P_0$

$P_1 = pump\text{-}on \rightarrow P_1 \square pump\text{-}off \xrightarrow{t1} pump\text{-}is\text{-}off \rightarrow P_0$

其中：$t_1 = 10\text{ms}$

$pump\text{-}on \overset{\text{def}}{=} \text{PumpController. SendCmd(ON)}$, $pump\text{-}off \overset{\text{def}}{=} \text{PumpController. SendCmd(OFF)}$

$pump\text{-}is\text{-}on \overset{\text{def}}{=} \text{PumpController. GetStatus ( ) = ON}$, $pump\text{-}is\text{-}off \overset{\text{def}}{=} \text{PumpController. GetStatus() = OFF}$

Ⅱ)水泵控制器模块进程 PC

$PC = turn\text{-}on\text{-}pump \xrightarrow{\tau 3} pump\text{-}on \xrightarrow{\tau 4} PC \square turn\text{-}off\text{-}pump \xrightarrow{\tau 3} pump\text{-}off \xrightarrow{\tau 4} PC \square no\text{-}change \xrightarrow{\tau 3} WAIT\tau 4;\ PC \square WAIT\ (\tau 3 + \tau 4);\ PC$

其中：$\tau 3 + \tau 4 = 10\text{ms}$

$turn\text{-}on\text{-}pump \overset{\text{def}}{=} \text{PumpController. TurnOn ( )}$, $turn\text{-}off\text{-}pump \overset{\text{def}}{=} \text{PumpController. TurnOff()}$

$no\text{-}change \overset{\text{def}}{=} \text{PumpController. KeepStat ( )}$, $pump\text{-}on$ 和 $pump\text{-}off$ 定义同上。

Ⅲ)水位传感器模块进程

$S = water\text{-}ok \xrightarrow{t2} W_{OK}$

$W_{OK} = water\text{-}lo \xrightarrow{t2} W_{HI} \square water\text{-}hi \xrightarrow{t2} W_{HI} \square water\text{-}ok$

$\xrightarrow{t2} W_{OK}$

$W_{LO} = water\text{-}lo \xrightarrow{t2} W_{LO} \square water\text{-}ok \xrightarrow{t2} W_{OK}$

$W_{HI} = water\text{-}hi \xrightarrow{t2} W_{HI} \square water\text{-}ok \xrightarrow{t2} W_{OK}$

其中：$t_2 = 20ms$，

$water\text{-}lo \overset{def}{=} WaterSensor.GetLevel() = LOW$，$water\text{-}hi \overset{def}{=} WaterSensor.GetLevel() = HIGH$，

$water\text{-}ok \overset{def}{=} WaterSensor.GetLeval = OK$

Ⅳ）监控模块进程

$M = water\text{-}hi \xrightarrow{\tau 1} turn\text{-}on\text{-}pump \xrightarrow{\tau 2} M \square water\text{-}lo \xrightarrow{\tau 1} turn\text{-}off\text{-}pump \xrightarrow{\tau 2} M \square water\text{-}ok \xrightarrow{\tau 1} no\text{-}change \xrightarrow{\tau 2} M$

其中：$\tau 1 + \tau 2 = 20ms$，*water-hi*、*water-lo*、*water-ok*、*turn-on-pump*、*turn-off-pump*、*no-change* 定义同上。

对模块进程进行同步并行组装，即可完成系统的行为描述：

$PUMP = PC |[I_P]| P$，$I_P = \{pump\text{-}on,\ pump\text{-}off\}$

$WATER = S |[I_w]| M$，$I_w = \{water\text{-}lo,\ water\text{-}ok,\ water\text{-}hi\}$

$SYSTEM = PUMP |[I_{pw}]| WATER$，$I_{pw} = \alpha P \cap \alpha W = \{turn\text{-}on\text{-}pump,\ turn\text{-}off\text{-}pump,\ no\text{-}change\}$

通过分析行为描述，可以生成在特定硬件环境中执行的调度构件，例如每个模块进程可以处理为一个操作系统线程，模块进程间的同步并行组装通过线程间同步实现。又如外部选择组装可以处理为条件选择语句，上文中的监控模块进程可以翻译为：

```
while(true){
        switch(WaterSensor.GetLevel()){
        case OK: PumpController.KeepStat(); break;
        case HIGH: PumpController.TurnOn(); break;
        case LOW: PumpController.TurnOff(); break;
        }
        ...
}
```

实时构件描述的主要目的之一是使调度构件的生成自动化，下一节讨论构件描述在实时 CORBA 框架内的应用。

4 在实时 CORBA 中的应用

在 RT-CORBA 体系结构中，客户端程序在使用一个对象之前，必须先获得对象的引用，通常是客户程序先直接获取一个工厂对象引用，以后的对象引用通过工厂对象获得。工厂对象是指那些能够返回其他对象引用的对象，它可能返回一个新的对象，也可能返回一个已有的对象。

为实现调度构件的自动生成，可在服务端 ORB 实现一个特殊的工厂对象 SchedulerFactory，专门负责解析构件描述并生成调度构件实例。当客户需要一个新的调度构件时，先获取 SchedulerFactory 引用，然后向 SchedulerFactory 发送构件描述，请求创建调度对象，得到对象引用后，即可实现对实时任务的监控。

通过增加 SchedulerFactory 工厂对象，可实现调度构件的自动生成，很大程度上提高了实时系统的构件复用程度。传输调度构件描述会加重网络负担，但仅在创建时需要，如采用适当的压缩算法，不会带来过大开销。

结束语 本文提出的实时构件描述机制具有精确的语义，可以准确表达构件的功能和 QoS 特性，这对于实时构件复用是必须的。调度构件描述的自包含特性使调度构件的自动生成成为可能。我们将进一步开展相应支撑技术的研究，如构件描述生成工具、行为描述的分析和转换算法等，并在实时 CORBA 的框架内开发一个实用模型。

参 考 文 献

1 Nielsen B, Agha G. Towards Reusable Real-Time Objects. Annals of Software Engineering, 1999.7: 257～282

2 Lau K-K, Ornaghi M. A Formal Approach to Software Component Specification. In: Proc. of Specification and Verification of Component-based Systems Workshop at OOPSLA 2001

3 张涌，王渊峰，钱乐秋．一个集成式的软件构件描述框架．计算机学报，2002，25(5)

4 Zaremski A M, Wing J M. Specification Matching of Software Components. ACM Trans. Softw. Eng. Method. Oct. 1997, 6 (4): 333～369

5 李阳，吴朝晖．一种形式化构件集成语义的研究．浙江大学学报（工学版），2004，38(2)

6 Fröhlund S, Koistinen J. Quality of Service Aware Distributed Object Systems. 5th USENIX Conference on Object-Oriented Technologies and Systems(COOTS '99)

7 Zaremski A M, Wing J M. Signature Matching: a Tool for Using Software Libraries. ACM TOSEM, Apr. 1995

8 Hoare C A R. Communicating Sequential Process. Prentice-Hall international, 1985

9 Davies J. Specification and Proof in Real-Time CSP. Cambridge University Press, 1993

10 Joseph M. Real-time Systems Specification, Verification and Analysis. Prentice Hall International, 1996

（上接第 204 页）

结论 PRAM 模型上的上下文无关文法的并行识别和改进的并行语法分析方法——金字塔结构，能较好地应用于符合 CNF 的规范产生式集合环境。本文对该方法进行了修改和补充，对产生式右部的候选式（即规则）有两个以上的非终结符连接的、或者候选式中既有非终结符又有终结符的情况作了等价变换，并对转换后的文法的等价性、适应性、效率以及识别分析过程做了分析和讨论。在不增加系统太多开销的情况下，算法能较有效地进行识别和分析。

参 考 文 献

1 Gibbons A, Rytter W. Efficient parallel algorithms. Cambridge University Press, 1990

2 Rytter W, Giancarlo R. Optimal parallel parsing of bracket language. Theoretical Computer Science, 1987

3 Ra Dong-Yul, Kim Jong-Hyun. A parallel parsing algorithm for arbitrary contexe-free grammars. Information Processing Letters, 1996, 58: 87～96

4 Matsumoto Y. A parallel parsing system for natural language analysis, New Generation Comput, 1987, 15: 63～78

5 Yonezawa A, Ohsawa I. Object-oriented parallel parsing for context-free grammars. In: proc. of 12th Int. Conf on Computational Linguistics(COLING-88), 1988. 773～778

6 Wyard P J, Ninghtingale C. A single layer higher order neural net and its application to context-free grammar recognition, In: N. Sharkey. ed. Connectionists Natural Language Processing. Chapter, 1992, 8: 139～162

7 Chandwani M, Chaudhari N S. Formulation and analysis of parallel context-free recognition and parsing on a PRAM model. Parallel computing, 1996, 22: 845～868