计算机科学2004Vol.31№.2

基于动作推导引擎下的故障检测方法

林才彪 李 磊
(中山大学软件研究所 广州510275)

摘 要 本文根据文[2]介绍的软件管理者方法,提出了基于动作推导引擎的软件管理者方法。软件管理者单元是一种自动的实时监控软件故障的软件工具,适用于实时软件系统特别是通信类软件系统的故障侦测。它通过监控目标系统的输入和输出,使用获得的输入和目标系统的管理模型推算出对应此输入序列的期望输出值,与目标系统的实际输出做比较,如果实际输出没有在期望输出集中,则管理者单元断定目标系统出现错误。

基于动作推导引擎的软件管理者方法,采用了动作推导引擎产生的目标系统管理模型,使该管理模型独立于软件管理者方法。可以实现管理模块与动作推导引擎同步实时更新,而不会导致软件管理者单元的改动。

关键词 动作推导引擎,故障检测,软件故障管理

A Failiue Detection Method Based on Action Derivation Engine

LIN Cai-Biao LI Lei
(Institute of Software, Zhongshan University, Guangzhou 510275)

Abstract This paper uses a ADE-based software supervisor[2] to automatic runtime detect real-time software system failure. Software supervisor is an unit that monitors the inputs and outputs of a target software system and reports discrepancies between specified and observed behaviors as failures. ADE-based software supervisor uses Action Derivation Engine as the generator of the specification language which is used by the supervisor. Action Derivation Engine is a way that engine searches corresponding arithmetic and executes based on the input of statement and action gathers, so that a specific functional software system is completed. the supervisor model can be independent from the supervisor unit if we use ADE as a specification generator. With the introduction of the idea of engine into the process of program development and failure detection. we can achieve program automatically.

Keywords Action derivation engine, Failure detection, Saftware failure supervision

1 背景知识

目前,软件故障管理几乎是全靠人工对软件系统进行故障的监测和恢复,就算能够自动地监测故障,也必须不断收集新的故障描述符,然后根据这些描述符来判断是否有故障。这些收集工作大部分是由人工来完成。这种严重依赖于人力的干涉很不利于软件可靠性工程的发展。其他科学类学科和工程类学科的发展历史表明,使用工具来代替人力是该学科的成熟标志之一。

故障侦测主要有两种方式,一种是基于故障描述符的故障侦测,它是通过定义故障的症状和范围,通过这些症状来判断系统是否出现错误。其主要缺点是需要人工对故障描述符进行收集,对领域有极强的依赖性,哪怕是对系统添加或者更换一个元件,就必须重新构造和添加新的故障描述。另外一种是基于目标系统形式化规范说明的故障侦测,它是通过观察目标系统实际发生的行为是否规范中描述的行为一致来决定是否出现故障,其优点是不需要很多人工干涉,适用范围较广,但是依赖于目标系统的形式化描述。文[2]中提出的软件管理者方法就是一种基于系统形式化描述的故障侦测软件工具。

本文利用文[2]所提出的软件管理者方法,结合我们中山大学软件所研究并在语音服务系统上实际应用的动作推导引擎通信软件系统设计方法,提出一种基于动作推导引擎的软件管理者方法。该方法用来对通信类软件进行实时的故障监控。其主要特点是通过动作推导引擎产生管理者单元所需的管理模型,使管理模型从管理者单元独立出来,可以很方便地对两者进行改进,而不会产生大的变动。

软件管理者是一种自动的软件故障实时监测方法,属于一种黑盒方法。也就是说,它不需要知道系统内部各状态和动作是如何实现,只需知道输入输出,以及各种状态的转移关系,就可以判断系统是否出错。软件管理者单元拥有目标系统的一个逻辑模型,该模型源自于软件的需求规范说明。通过监控系统的输入输出,管理者单元就可以知道目标系统的实际行为是否与它的管理模型所描述的行为是否一致,如果实际行为没有出现在期望行为集中,管理者单元就断定目标系统出现故障。

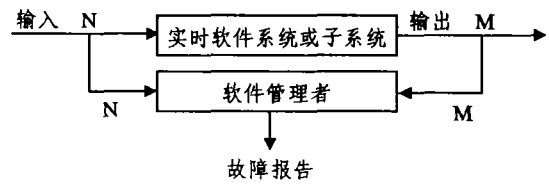

图1

对于软件管理者方法来说,主要存在两个挑战。第一是如何对系统规范的不确定性进行管理;第二是在一个故障被检

测出来后应该如何保证管理的连续性。

不确定性允许对于一个给定的输入序列有几个合法的行为产生。管理者单元必须能够考虑到对应于特定输入的所有合法的行为。这样才能避免把正常状态诊断为故障的现象。

当一个故障被检测到后，管理者单元对于目标系统所处的实际状态知之甚少，它必须立刻辨认出目标系统的状态才能够开始下一个故障的侦测。

本文第二部分是对软件管理者方法的难点进行详细的分析，然后介绍动作推导引擎。第三部分提出基于动作推导引擎的软件管理者方法以及基本结构；最后是总结和展望。

2 基于动作推导引擎的管理者自动故障检测方法

2.1 软件管理者方法的难点和构架

2.1.1 *软件管理需满足的要求* 在本文的背景知识介绍中，我们说过一个管理者要使用到目标系统的一个逻辑模型(以下称为管理模型)，该模型产生于目标系统的规范说明。

令 S_i 表示管理模型的一个全局的合法的状态，Q_x 为目标系统的合法状态。状态 S_i 与目标系统的 j 个状态 $\sum_i=\{Q_{i1},Q_{i2},\cdots,Q_{ij}\}(j>0)$ 相关联。$\sum_i$ 表示的状态组必须满足以下的两个条件：

1. $\forall i,j,k,i\neq k, Q_{ij}\in\sum_i\rightarrow Q_{ij}!\in\sum_k$；

2. 如果一个状态迁移过程 $S_i\rightarrow S_i$，被刺激 I(启动或将启动一个状态的迁移过程的信号)所启动，产生了一个可能是空的有序的反应 O(产生于一个状态迁移过程的信号)的集合；那么相应的刺激 I 所引起的状态迁移过程，或者说是状态集迁移过程，这个过程起始于状态 $Q_{im}\in\sum_i$ 和结束于状态 $Q_{in}\in\sum_j$ 必须也产生反应 O。

一个软件管理单元的两个要求如下：

1. 对于目标系统的状态转移，$\sum_i\rightarrow\sum_j\rightarrow\sum_k\rightarrow\cdots$ 支持相关的状态转移 $S_i\rightarrow S_j\rightarrow S_k\rightarrow\cdots$

2. 对于一个给定序列的输入，我们可以计算出管理模块所产生的期望输出集，并在获得目标系统的实际输出后进行比较，如果实际输出没有包含在期望输出集中，管理模块就得出结论，目标系统出现故障。

2.1.2 *软件管理者单元的组成* 软件管理者由两个部分组成，一个是实现故障侦测机制组件，一个是状态重定位组件(见图2)。管理者单元在运行时，任何时间只有其中之一是处于活动状态。

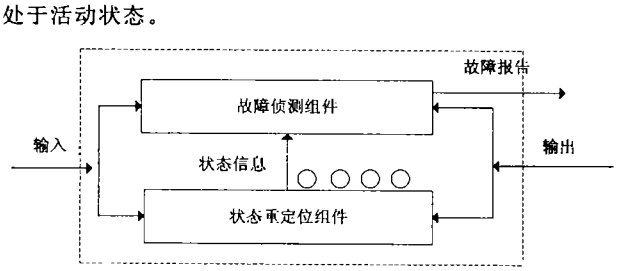

图2 软件管理单元结构

当处于正常运行期时，故障侦测机制组件监控着目标系统，一旦出现故障，它就给出故障信息，接着停止运作，把单元的控制权交给状态重定位组件。状态重定位组件负责目标系统在出现故障后的状态确认，如果可以辨认出目标系统的状态，则通知并激活故障侦测机制组件，把控制权交给故障侦测机制组件。管理者单元的这种运行机制可以用图3来清楚地表达。

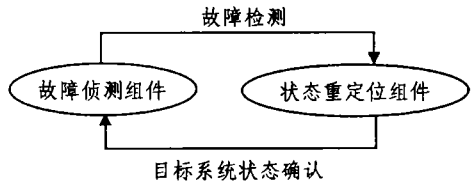

图3 软件管理者单元的运行模式

2.2 动作推导引擎

我们在语音服务系统中引入了动作推导引擎 ADE (Action Derivation Engine)。ADE 方法是以动作逻辑为输入语言的一种新颖的程序设计方法。动作推导引擎提供一个输入接口，通过这个接口可以把系统状态之间的逻辑推导信息用说明性语言输入到推导引擎的动作库中，而动作推导引擎中的状态控制器根据这些逻辑推导信息接受系统的外部事件并进行相应的系统状态转换，同时调用算法库中相应的算法对状态进行处理，从而完成系统的所需功能。

动作逻辑中的语法概念如项、原子公式及公式等模仿一般多类逻辑的类似定义。

设 $\mathscr{L}$ 表示所有一阶逻辑公式的集合。给定一个状态类型的项 t，定义 $\mathscr{L}_t$ 如下：

$\mathscr{L}_t$ 中的公式是 $\mathscr{L}$ 中公式的一个子集，其每个成员中除 t 外不包含其它类型为 s 的项，不对状态变元使用量词，不包含关系符号 Poss 和<。形式地它是满足如下条件的最小公式集合：

(1)如果 $\phi\in\mathscr{L}$ 不含有任何状态类型的项，则 $\phi\in\mathscr{L}_t$；

(2)对于每个有 n 个操作对象的动态关系 F 以及类型 o 的项 $x_1,\cdots,x_n,F(x_1,\cdots,x_n,t)\in\mathscr{L}_t$；

(3)如果 $\varphi,\varphi\in\mathscr{L}$，那么就有以及都属于 $\neg\phi,\phi\wedge\varphi,\phi\vee\varphi,\phi\rightarrow\varphi,\phi\leftrightarrow\varphi,\forall x\phi,\exists x\phi,\forall a\phi$ 以及 $\exists a\phi$ 都属于 $\mathscr{L}_t$，这里 x，a 分别具有类型 o 和 a。

在动作逻辑中，一个基本的动作理论 $\mathscr{D}$ 就是一个如下的一组句子：

$\mathscr{D}=\Sigma\cup\mathscr{D}_{ss}\cup\mathscr{D}_{ap}\cup\mathscr{D}_{una}\cup\mathscr{D}_{s0}$

其中：

(1)Σ 是一组逻辑公式。它的直观含义是说所有类型为 s 的个体在二元关系<(二元关系符号<表示状态之间的先后关系，$s<S_0$ 表示 s 先于 S_0)之下构成一个分叉的时序结构，在其中有一个类型为 s 的表示起始状态的常量符号 S_0 作为起点，而 do 是后继函数(二元函数符号 do(a,s)描述动作 a 的发生使得状态 s 从 s 变成另外一个)。而且所谓的树型归纳法也包含在其中。具体地，它包含如下的公理：

$S_0\neq do(a,s)$；

$do(a_1,s_1)=do(a_2,s_2)\rightarrow(a_1=a_2\wedge s_1=s_2)$

$\forall P[(P(S_0)\wedge\forall a,s(P(s)\rightarrow P(do(a,s))))\rightarrow\forall s\,P(s)]\rightarrow(s<S_0)$

$s<do(a,s')\leftrightarrow(Poss(a,s)\wedge s\leqslant s')$(二元关系符号 Poss 表示动作 a 在状态 s 之下是可能发生的)

(2)$\mathscr{D}_{ss}$ 是一组逻辑公式。它们描述了一个类型为 a 的个体按一定的规则被执行以后所产生的效果。一般地，其中的公式具有以下的形式：

$Poss(a,s)\rightarrow(F(x,Do(a,s)))\leftrightarrow\psi_F(x,a,s))$

其中 F 是一个动态关系符号而 ψ_F 是 $\mathscr{L}_s$ 中的一个逻辑公式。

(3)$\mathscr{D}_{sp}$是一组逻辑公式。它们描述了一个类型为 a 的个体能够按一定的规则被执行的前提条件。它们一般具有以下的形式：

$$Poss((A(x),s)\leftrightarrow\psi_F(x,s)$$

其中 A 是一个动作常量，而 ψ_F 是 $\mathscr{L}_s$ 中的一个逻辑公式。

(4)$\mathscr{D}_{una}$是表示以下意义的逻辑公式：两个动作常量在两组类型为 o 的个体上执行的结果是不同的，除非它们分别是同一个动作以及同一个个体对象的序列。它们具有如下的形式：

$$A(x)\neq A'(y)$$

$$A(x)=A(y)\rightarrow(x=y)$$

(5)$\mathscr{D}_{s0}$是一组有限的 $\mathscr{L}_{s0}$ 中的逻辑公式。称为基本动作理论的初始条件。

我们把动作逻辑看成是一种形式化的语言，并用它来表示动作推导引擎的语法。动作推导引擎的系统结构图如图4所示。

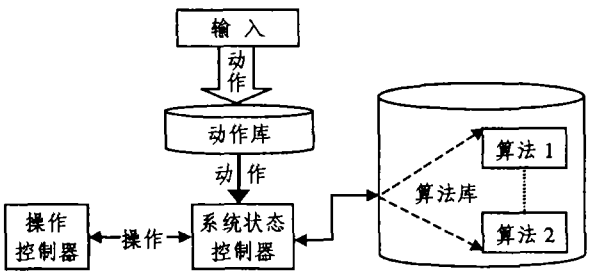

图4

我们把动作作为一个从状态集合到状态集合的映射，每一个动作通过将系统从一个特定的状态改变到另一个状态，完成特定的功能。

一个动作可以用二元运算符来表示。例如：P<a>Q 表示如果系统的当前状态由命题 P 来描述，a 是一个操作系列，将 a 中的操作一次作用到当前状态，系统的状态将发生改变，新状态由命题 Q 来描述，也可以用图形来表示。如果系统的当前状态是 S，并执行了操作 a 序列，那么状态转移到 T，S 和 T 是描述状态的命题，如图5所示。

动作推导引擎启动的时候，状态控制器将系统状态设置为初始态。这个初始态是动作推导引擎自定义的。在初始态中，系统中所有变量都处于默认值或者是初始值。

我们可以根据系统当前的状态和输入，通过动作推导引擎，演算出系统将出现的后继状态和相应的输出。这是因为我们已经知道了软件系统的规范说明，这也为本文以下软件故障自动监测部分的实现提供了客观条件和理论依据。

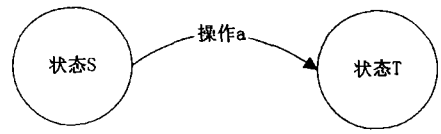

图5

3 基于动作推导引擎的管理者自动故障检测方法的结构

动作推导引擎由当前状态，采取不同的动作序列，可以转换到不同的状态，一旦知道系统当前状态和环境输入，就可以预先知道后继的状态和输出的集合。这十分吻合管理者方法的要求。而且，可以将管理模型与动作推导引擎关联起来，让动作推导引擎产生软件管理者单元所需要的目标系统管理模型，把管理模型从软件管理者单元独立出来。所以，我们采用管理者方法并结合动作推导引擎来作为自动故障检测方法。

管理者单元主要有两种实现方式，一种是以输入驱动，一种是以输出为驱动。以输入驱动的管理者单元在获得目标系统的当前输入后，根据该输入和管理模型演算出期望输出集。然后拿实际输出与期望输出集做比较，如果实际输出不在期望集中，则报告出错。以输出驱动的管理者单元则是根据实际输出推算输入，比较是否与实际输入相符来决定是否出错。本文使用的是以输入驱动的管理者方法。

前面已经说过管理者单元由两部分组成，一个是故障侦测组件，另一个是状态重定位组件。下面介绍基于动作推导引擎管理者方法的这两个组件构成。

3.1 故障侦测组件的结构

为了能够实现故障自动监测，管理者单元要有目标系统的一个管理模型，这个管理模型就是我们前面介绍的动作推导引擎产生的系统状态转换图，此图以树的形式存储(如图6所示)。故障侦测组件必须实现：(1)根据当前系统状态和输入产生一个期望输出集；(2)比较期望输出集和目标系统的实际输出。

故障侦测组件由五部分组成：包括负责产生期望输出集的路径解释器和负责确认行为为合法性的行为管理器；两个缓冲区，一个用来存储期望输出集，一个用来存储实际输出值；还有一个期望值和实际输出值的比较算法。

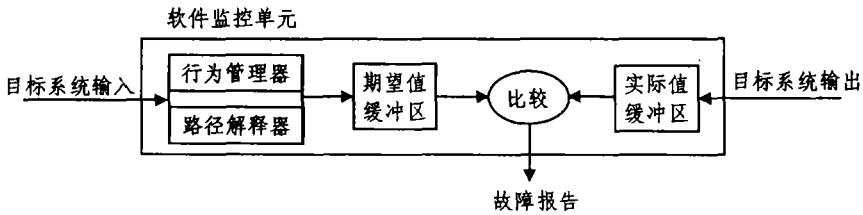

图6 故障侦测组件构成图

3.1.1 *路径解释器和行为管理器* 路径解释器的功能是由目标系统的当前状态和输入推算出接下来有可能出现的状态和输出的集合，即期望输出集。而行为管理器则确认期望输出集是否包含所有可能产生的合法的输出，以及此期望输出集是否合法。要做到这一点，路径解释器必须知道各种状态之间的转换关系，这些可以通过目标系统的管理模型得出。状态转换可以用状态转换图来表示，此状态转换图源自于动作推导引擎，它随动作推导引擎的改变而改变。

3.1.2 *期望输出队列和实际输出队列* 期望输出队列存储了从解释器和管理器计算出来的期望输出，而实际输出

队列存储的则是目标系统实际产生的输出。使用队列的原因是我们必须保证状态转换的顺序和信号"先到先出"的特点。

3.1.3 比较算法 比较期望队列和实际值队列，删除期望队列的不匹配项，最后如果期望队列为空，则返回 false，说明系统出现故障。我们还可以利用比较的结果，确定实际输出对应的目标系统的当前状态，这样可以减少期望集的递增趋势。

3.2 状态重定位组件结构

状态重定位组件主要是在故障发生后，管理者单元未知道目标系统当前状态之前运行，它的主要功能就是定位目标系统在发生故障后系统的状态，以使后继的故障侦测得以进行。组件中的两个模块都需要用到目标系统的管理模型，该模型源自于目标系统的需求规范说明。状态重定位组件包括两部分，一是鉴别状态模块，一是状态迁移确定模块。

3.2.1 鉴别状态模块 根据目标系统的管理模型来确定目标系统当前的状态。该模块通过观察到的目标系统的往来信号，如果发现当前信号与规范说明中的某个状态相符，它就把状态和相关的动作信息传给状态迁移确定模块。

3.2.2 状态迁移确定模块 根据从鉴别状态模块传来的状态和动作信息确定状态迁移。

4 基于动作推导引擎的管理者自动故障检测方法在语音服务系统中的应用

使用动作推导引擎，要建立完善的算法库，用户向算法库中增加新的算法时，必须经过严格的论证与测试。

假定为了实现系统内部两个通道之间的通话，定义状态集合和动作集合如图7所示。

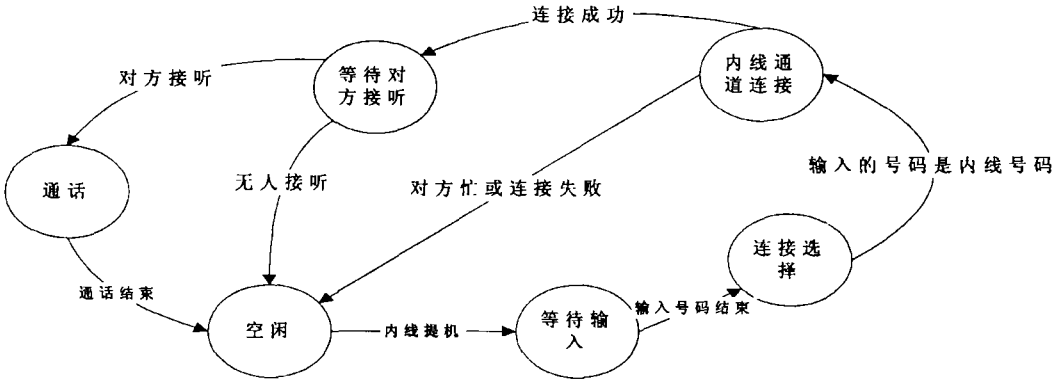

图7

实现系统内内部和外部之间的通话，也就是内线和外线之间的通话，定义状态集合和动作集合如图8所示。

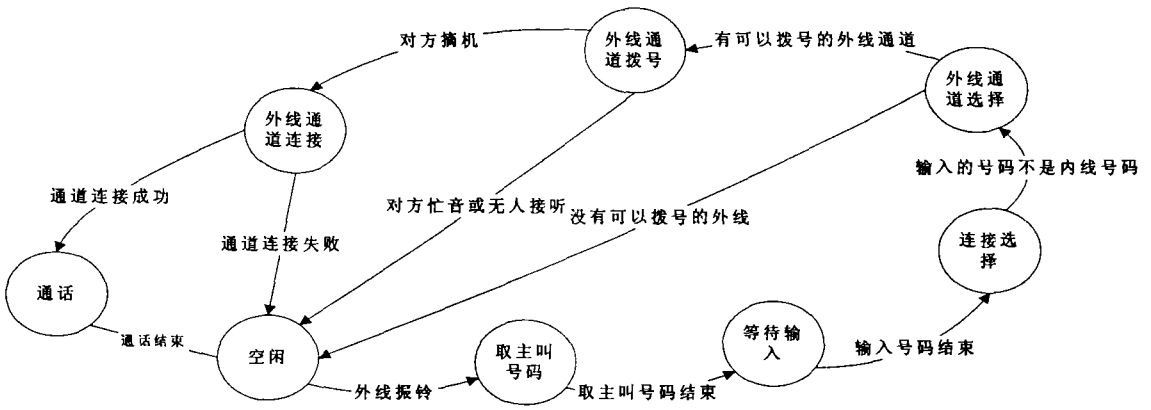

图8

将上面的状态集合和动作集合作为我们软件管理者单元的管理模型，并让此管理模型与语音服务系统的动作推导引擎关联，由动作推导引擎产生该模型，可以实现管理模型与动作推导引擎算法和状态同步更新，从软件管理者单元独立出来。

我们在语音服务系统植入代码使它运行时状态出现偏差，这些偏差软件管理单元都可以检测出来。

总结和展望 本文根据文[2]的软件管理者方法，提出了基于动作推导引擎的软件管理者方法。该方法利用目标系统的输入输出来判断其是否出错。此方法不用去收集故障描述符，根据系统的规范说明便可侦测系统故障，管理模型由目标系统的动作推导引擎产生，与管理者单元相互独立，可以很方便地对两者进行修改升级而不会出现干扰。但是软件管理方法是建立在目标软件系统的规范是正确的假设之上，它不能侦测规范中存在的错误，这是它的一个局限性。

软件管理者方法还有很多东西有待挖掘。比如故障出现后，我们只是采用故障报告，而没有好好利用所获得故障信息来进行故障诊断，进一步做到故障自动恢复。下一步的工作是研究如何利用得到的故障信息进行故障诊断。

参 考 文 献

1 Nilsson N J. 人工智能 Artifical Intelligence. 机械工业出版社

2 Hlady M, et al. An Approach to Automatic Detection of Software Failure, Bell Canada Software Reliability Lab, Waterloo Univ,

Ont.; Software Reliability Engineering. 1995. In: Proc. Sixth Intl. Symposium on 10/24/1995 -10/27/1995. Oct. 1995

3 Savor T. Seviora R E. An architectural overview of a software supervisor. Bell Canada Software Reliability Lab. Waterloo Univ. Ont.; Real-Time Systems, 1996. In: Proc. of the Eighth Euromicro Workshop on 06/12/1996 -06/14/1996, Jun 1996

4 梁冰. 李磊. 动作推导引擎及其在通信软件设计中的应用计算机工程与应用(录用)

5 Savor T. Seviora R E. An approach to automatic detection of software failures in real-time systems. Bell Canada Software Reliability Lab. Waterloo Univ. Ont.; Real-Time Technology and Applications Symposium. 1997. In: Proc. Third IEEE 06/09/1997 -06/11/1997. Jun 1997

6 Alagar V S, et al. Specification-based Testing for Real-Time Reactive Systems. Dept. of Comput. Sci., Concordia Univ., Montreal. Que.; Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. In: Proc. 34th Intl. Conf. on 07/30/2000 -08/04/2000. 2000

7 Brown D B, et al. An automated oracle for software test-ing. Reliability. IEEE Transactions, 1992. 41(2)

8 吴今培, 肖健华. 智能故障诊断与专家系统. 科学出版社

(上接第154页)

表4给出了一组相关文献报道的实验结果,与已有的最好的一批系统相比,本文组块识别的性能极为接近目前的最优水平。值得一提的事,比本文稍优的系统,使用了更为复杂的多种机器学习组合的算法,结果和本文的结果没有明显的差别。而本文通过导入各种上下文信息到隐马尔科夫模型,达到了所有使用单一方法的系统的最佳结果。

表4 CoNLL-2000共享任务的测试结果

实验	Precision	Recall	$F_{\beta=1}$
文[12]	93.45%	93.51%	93.48
文[3]	94.04%	91.00%	92.50
实验4	92.05%	92.46%	92.25
文[14]	91.99%	92.25%	92.12
文[11]	92.08%	91.86%	91.97
文[16]	91.05%	92.03%	91.54
文[15]	90.63%	89.65%	90.14
实验3	89.58%	89.55%	89.57
文[18]	86.24%	88.25%	87.23
文[17]	88.82%	82.91%	85.76

结论 本文提出了一种基于增益的隐马尔科夫模型的文本组块分析技术,该方法通过不断增益的转换函数对训练语料进行不断优化,从而达到训练模型的不断增益的过程。该方法不需要修改原来模型的训练过程和标注过程,从而保留了原有模型的训练和标注的良好的性能,并增加了输出的标记集合,给出了更加完善的上下文模型;另一方面,实验的训练过程和标注过程都是在秒级单位内完成,这也是该模型优于其他模型的地方,比如最大熵模型、支持向量机模型等。从实验结果中发现,中文比英文的结果低10个百分点左右。主要原因可能是中文的语法结构比较灵活,存在大量的结构歧义。其次,从上面的实验也可以看出,更多的训练数据也可提高识别效果。最后,使用更多的知识能提高实验效果。下一步的工作通过导入更多的上下文信息,包括短语边界,语义,搭配和共现等信息,以提高识别的准确率和召回率。

参考文献

1 Abney S. Parsing by chunk. In Berwick, A. and Tenny, editors, Principle-Based Parsing. Kluwer, 1991

2 Erik F. Tjong Kim Sang and Sabine Buchholz Introduction to the CoNLL-2000 Shared Task: Chunking. CoNLL-2000 and LLL-2000. Lisbon, Portugal, pp. 127～132

3 Erik F, Sang T K. Text chunking by system combination. In: Proc. of CoNLL-2000 and LLL-2000. Lisbon, Portugal, 2000

4 Brants T. TnT -a statistical part-of-speech tagger. In: Proc. of the Sixth Applied Natural Language Processing (ANLP-2000), Seattle, WA, 2000

5 Ramshaw L, Marcus M. Text Chunking Using Transformation-Based Learning. In: Proc. of third Workshop on Very Large Corpora, June 1995. 82～94

6 Ratnaparkhi A. Maximum Entropy Models for Natural Language Ambiguity Resolution: [Phd. Thesis]. University of Pennsylvania, 1998

7 Merialdo B. Tagging English Text with a Probabilistic Mod-el. Computational Linguistics, 1994, 20(2): 155～171

8 Church K W. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In: Proc. of the 1st Conf. on Applied Natural Language Processing, ANLP, ACL, 1988. 136～143

9 Daelemans W, Buchholz S, Veenstra J. Memory-Based Shallow Parsing. In: Proc. of EMNLP/VLC-99, University of Maryland, USA, June 1999. 239～246

10 Argamon S, Dagan I, Krymolowski Y. A Memory-based Approach to Learning Shallow Natural Language Patterns. In: Proc. of the joint 17th Intl. Conf. on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics, COLING-ACL, Montreal, Canada, 1998. 67～73

11 Koeling R. Chunking with Maximum Entropy Models. In: Proc. of CoNLL-2000 and LLL-2000, Lisbon, Portugal, Sep. 2000

12 Kudo T, Matsumoto Y. Chunking with Support Vector Machines. In: Proc. of NAACL 2001, Pittsburgh, USA, Morgan Kaufman Publishers, 2001

13 Kudo T, Matsumoto Y. Use of Support Vector Learning for Chunk Identification. In: Proc. of CoNLL-2000 and LLL-2000, Lisbon, Portugal, Sep. 2000

14 Zhou G D, Su J, Tey T G. Hybrid Text Chunking. In: Proc. of CoNLL-2000 and LLL-2000, Lisbon, Portugal, Sep. 2000

15 Pla F, Molina A, Prieto N. Improving chunking by means of lexical-contextual information in statistical language models. In: Proc. of CoNLL-2000 and LLL-2000. Lisbon, Portugal, 2000

16 Veenstra J, van den Bosch A. Single-classifier memory-based phrase chunking. In: Proc. of CoNLL-2000 and LLL-2000. Lisbon, Portugal, 2000

17 Vilain M, Day D. Phrase parsing with rule sequence processors: an application to the share conll task. In: Proc. of CoNLL-2000 and LLL-2000. Lisbon, Portugal, 2000

18 Johansson C. A context sensitive maximum likelihood approach to chunking. In: Proc. of CoNLL-2000 and LLL-2000. Lisbon, Portugal, 2000