规则库冗余性控制策略的研究*

宗成庆 陈肇雄 黄河燕

(中国科学院计算技术研究所机译中心 北京 100080)

摘要 冗余性控制是研究知识库组织、管理和维护中的一个问题. 本文通过对智能型机译系统中规则知识表示方法的分析,提出了将冗余规则划分为显式冗余规则和隐式冗余规则分别予以处理的思想,给出了显式冗余规则的判别算法和部分隐式冗余规则的检测标准,并提出了控制机译系统规则库冗余性的基本原则.

关键词 机器翻译,规则库,知识库,SC文法,冗余性.

知识冗余性是指一个知识从某知识集中删除既不缩小也不扩大由该知识集采用任何方法所能衍生出的知识的集合. 即删除前后所能衍生出的知识集合相等.[1]在一个实际应用系统中,要保持知识集合的无冗余性,尤其当知识集合较大时,是相当困难的.

在机器翻译系统中,词典库和语法规则库构成了知识库的2大部分. 而机译系统的开发是一个工作量浩大的工程项目,规模庞大的规则库和词典库需要同时组织各方面的专家和技术人员共同协作完成,尤其是在系统调试阶段,需要不断地对规则库进行必要的增删和修改,因此,保持规则库的精练和简洁,避免由于规则冗余而引起系统效率低下、维护代价增加等不利后果,是规则库组织和管理中首先必须解决的问题.

尽管已有结果表明,文法的规则冗余是不可判定的.[2]但是,一旦允许冗余规则无限度地存在,势必给系统维护和规则调度等处理机制带来不利影响. 因此,在一个实用机译系统的研究和开发过程中,如何根据规则描述形式的具体特点,有针对性地发现和消除一些潜在的冗余规则,将冗余规则的数目控制在一个适当的范围,则是一种非常现实的考虑.

智能型机器翻译系统采用SC文法[3,4]描述语法规则. 本文在分析SC文法规则形式特点的基础上,提出将冗余规则划分为显式冗余规则和隐式冗余规则分别予以处理的思想,给出了显式冗余规则的判别算法和部分隐式冗余规则的检测标准,并提出了控制机译系统规则库冗余性的基本原则.

---

* 作者宗成庆,1963年生,博士生,讲师,主要研究领域为机器翻译,人工智能,数据库技术. 陈肇雄,1961年生,博士,研究员,主要研究领域为机器翻译,人工智能. 黄河燕,女,1963年生,博士,研究员,主要研究领域为机器翻译,面向对象程序设计,大型智能应用系统.

本文通讯联系人:宗成庆,北京 100080,中国科学院计算技术研究所机译中心

本文1995-12-18收到修改稿

1 基本定义和符号表示

1.1 SC文法

关于SC文法的基本定义和功能描述已有不少的论文论述(详见文献[3,4]),在此不再赘述.以下仅给出SC文法规则的表示形式:

$$\alpha_1 \ldots \alpha_n \rightarrow C^* \mid \beta_1 \ldots \beta_m, \gamma^*$$

这里 $\alpha_1 \ldots \alpha_n$ 和 $\beta_1 \ldots \beta_m$ 均为源语言结构成分或意段标记名称,在不产生歧义的情况下分别记作 α^* 和 β^*.其中 α^* 为待约成分,β^* 为归约后成分,且 $|\alpha^*| \geqslant |\beta^*|$;$\gamma^*$ 为目标语言解释;C^* 为归约条件.该规则表示:α^* 在满足条件 C^* 的情况下被归约成 β^*,译义转换为 γ^*.条件 C^* 可以为空,表示无条件满足.归约树的根结点 S 为句子标志.

1.2 冗余规则

文法 G 的规则集为 ρ,若$\exists\ r \in \rho$ 和以 $\rho' = \rho - \{r\}$ 为规则集的文法 G',且 $L(G) = L(G')$,则 r 为冗余规则,G 为规则冗余文法.

显然,重复规则是最明显的冗余规则.此外,我们注意到,在机译系统中冗余规则通常以2种不同的形式存在.一种情况是,某些规则在任何句子的分析中始终不能被引用,这些规则的存在对整个系统无任何益处.另一种情况是,存在一些规则对系统分析句子的范围无任何妨碍,即存在与否对文法识别句子的集合不产生任何影响,但是这些规则在一定条件下可能被激活,并且对系统的执行效率产生直接的影响.根据检查方法和处理手段的不同,我们将这2类冗余规则分别称之为显式冗余规则和隐式冗余规则,并分别予以处理.以下分别给出这2类冗余规则的不严格定义.

1.3 显式冗余规则

设SC文法 G 的归约规则集为 ρ,若$\exists\ r \in \rho, \forall\ w \in L(G)$,在形成的以 S 为根结点的任何语法分析树中,r 均没有被采用,则 r 为显式冗余规则.

显然,机译规则库中不符合语法现象的规则大多数为显式冗余规则.另外,还有一类规则尽管符合语法规定,但由于规则库庞大或规则设计者的疏漏,或由于不适当地删除某一规则而引起相关规则之间缺乏应有的连贯性,致使在分析过程中尽管这些规则能够被调用,但由于无法形成最终的语法分析树而被放弃,实际上,这类规则可能为假冗余规则.根据上述定义,它们也被视为显式冗余规则.

1.4 隐式冗余规则

若有SC文法 G,$\exists\ w \in L(G)$ 和$\exists\ r \in \rho$,在条件满足时 r 可能被激活并形成 w 的句法分析树,但是通过 r 归约所能得到的任何句型均可由 $\rho - \{r\}$ 条规则归约得到,则称 r 为隐式冗余规则.

2 显式冗余规则的判别算法

显式冗余规则存在于规则库中,对规则调度模块和语法分析机制的执行效率均会产生直接的影响,但其判别过程是相对直观的,并且可以由系统维护机制自动完成.以下针对SC文法规则的形式特点,给出一个显式冗余规则的判别算法.

算法说明:输入:以 SC 文法描述的规则库中全部的语法规则;

输出:冗余规则和相应的信息;

符号表示:V:非显式冗余规则标记;X:显式冗余规则标记;

算法描述:

(1)将右部归约后成分仅为 S 的所有规则注以 V 标记;

(2)将规则库中所有带 V 标记的规则构成子集 S_1,所有不带标记的规则构成子集 S_2,然后执行如下操作:

①如果 $S_2=\varnothing$,转(4);如果 $S_2\neq\varnothing$ 但 $S_1=\varnothing$,则转(3),否则,即 $S_1\neq\varnothing$ 且 $S_2\neq\varnothing$,执行下一步操作;

②从集合 S_1 中任取一条规则 $R_i:\eta_i^*\rightarrow C_i^*|\zeta_i^*,\gamma_i^*$,在 S_2 集合中找出所有右部归约后成分与 η_i^* 相同或包含于 η_i^* 的所有规则,构成集合 S_3,即:

$$S_3=\{R_k|R_k:\alpha_k^*\rightarrow C_k^*|\delta_k^*,\gamma_k^*,\text{且 }\delta_k^*=\eta_i^*\text{ 或 }\eta_i^*=\eta_{i1}\ldots\delta_k^*\ldots\eta_{in}\}$$

令 $S_1=S_1-\{R_i\}$;

③如果 $S_3=\varnothing$,转(2)①,否则,令 $S_2=S_2-S_3$,然后将 S_3 中的所有规则注以 V 标记,并将 S_3 并于 S_1,即 $S_1=S_1\cup S_3$,然后令 $S_3=\varnothing$,转(2)①.

(3)将 S_2 中的全部规则注以 X 标记;

(4)给出结果信息:如果 $S_2=\varnothing$,则规则库中无显式冗余规则,否则,所有被加注 X 标记的规则均为显式冗余规则;

(5)算法结束.

3　部分隐式冗余规则的判别与消除

相对而言,隐式冗余规则的判别是困难的,并且通常需要与整个系统的执行效率等其它因素统筹考虑,因此,这类冗余规则的消除具有一定的复杂性. 重复规则作为一种特殊的隐式冗余规则,其判别和消除方法都是简单的. 在这里,我们针对智能型机译系统的具体特点,给出 4 种隐式冗余规则的判别和消除方法. 下列判别过程均假定规则描述信息为最小粒度.

3.1　消除条件包含引起的冗余规则

设有规则

$$\alpha^*\rightarrow C^{*1}|\beta^*,\gamma^* \tag{1}$$

$$\alpha^*\rightarrow C^{*2}|\beta^*,\gamma^* \tag{2}$$

且

$$C^{*1}=C_1\ldots C_kC_{11}\ldots C_{1m}$$

$$C^{*2}=C_1\ldots C_kC_{21}\ldots C_{2n}$$

如果 $m=0,n\neq0$,表明(2)式比(1)式具有更严格的条件约束,并且(2)式的归约条件包含(1)式应满足的条件,因此,(2)式为冗余规则,消除(2)式,保留(1)式. 反之,若 $m\neq0,n=0$,则(1)式为冗余规则,消除(1)式,保留(2)式. 如果 $m\neq0,n\neq0$,则(1)、(2)2 条规则合并为 1 条:$\alpha^*\rightarrow C^{*1}\vee C^{*2}|\beta^*,\gamma^*$.

3.2　消除传递性导致的冗余规则

设有规则

$$\alpha^*\rightarrow C^{*1}|\beta^*,\gamma^{*1} \tag{3}$$

$$\beta^*\rightarrow C^{*2}|\zeta^*,\gamma^{*2} \tag{4}$$

$$\alpha^*\rightarrow C^{*1}\wedge C^{*2}|\zeta^*,\gamma^{*1}\&\gamma^{*2} \tag{5}$$

则(5)式为冗余规则. 待约成分 α^* 在条件 C^{*1} 前提下,可归约为 β^*,目标语言解释为 γ^{*1},而 β^* 在条件 C^{*2} 前提下,又能归约为 ζ^*,目标语言解释为 γ^{*2}. 因此,如果条件 C^{*1} 和 C^{*2} 同时成立时,α^* 必能归约为 ζ^*,且解释为 γ^{*1} 和 γ^{*2},即(5)式为(3),(4)2 式的必然结果.

3.3　消除不一致性引起的冗余规则

设有规则 $$\alpha^* \rightarrow C^* \mid \beta^{*1}, \gamma^* \tag{6}$$

$$\alpha^* \rightarrow C^* \mid \beta^{*2}, \gamma^* \tag{7}$$

如果 $\beta^{*1} \neq \beta^{*2}$，并且 $\beta^{*1} \overset{*}{\Rightarrow} \beta^{*2}$ 或 $\beta^{*2} \overset{*}{\Rightarrow} \beta^{*1}$ 均不成立，则(6)、(7)之一为冗余规则. 类似地，

$$\alpha^* \rightarrow C^* \mid \beta^*, \gamma^{*1} \tag{6'}$$

$$\alpha^* \rightarrow C^* \mid \beta^*, \gamma^{*2} \tag{7'}$$

(6′)、(7′)之一为冗余规则. 实际上，(6)、(7)和(6′)、(7′)规则的同时存在已经破坏了规则的一致性原则. 对于这类冗余规则的消除应该同规则一致性(相容性)和完整性约束检查结合起来.

3.4 消除语义变化引起的冗余规则

设有规则 $$\alpha_1^1 \ldots \alpha_n^1 \rightarrow C^* \mid \beta^*, \gamma^* \tag{8}$$

$$\alpha_1^2 \ldots \alpha_n^2 \rightarrow C^* \mid \beta^*, \gamma^* \tag{9}$$

其中 $\alpha_1^1 \ldots \alpha_n^1$ 和 $\alpha_1^2 \ldots \alpha_n^2$ 中各成分的语义信息分别为 S_i^1 和 $S_i^2 (i \in [1,n])$. 如果有 $S_i^1 \subset S_i^2$，则规则(8)冗余. 反之，如果 $S_i^1 \supset S_i^2$，则规则(9)冗余. 否则，$S_i^1 \neq S_i^2$，则(8)、(9)2 规则可合并为 1 条：$\alpha_1 \ldots \alpha_n \rightarrow C^* \mid \beta^*, \gamma^*$，其中语义信息 $S_i = S_i^1 \cup S_i^2, 1 \leqslant i \leqslant n$.

由于隐式冗余规则的确定和消除存在固有的复杂性，而且，由于不同系统所采用的规则描述形式不同，因此，冗余规则的判定和消除方法也将随着规则表示形式的不同而有所差异，而且，通常情况下需要人工参与才能完成.

4 关于冗余性控制的讨论

规则库具有一定规模时，其冗余性的存在是不可避免的，但又无法完全识别和消除所有的冗余规则，甚至有时为了某种需要，故意保留部分冗余规则. 那么，如何合理地控制冗余规则的存在，则是规则设计者和系统维护者共同关注的问题. 本小节针对机译系统的具体特点，提出 4 条控制冗余规则的基本原则.

4.1 对称标点的过程化处理

在自然语言中，有些标点符号由对称的 2 个部分构成，如引号、括号等. 在 2 个对称部分之间的内容具有相对的独立性，而标点符号本身并不影响符号之间的内容，因此，对于这类标点完全可以采用过程化处理方法，在分析程序中对其单独解释说明，甚至不予处理而直接将其照搬到目标语言结构中，而不必使用额外带标点的规则. 这样，可以减少一定数量的附加规则，这些规则尽管从严格的定义上讲不属于冗余规则，但是，它们的确对规则库造成一定的负担，从某种意义上讲，它们已经构成了冗余. 因此，对称标点的过程化处理，也应作为控制规则库规模的一种措施.

4.2 隐式冗余规则不一定为摈弃规则

从理论上讲，所有的冗余规则都应该从规则库中消除，但是，对某些规则而言，如果它们被引用的频率较高，一旦将其从规则库中消除，由该规则所完成的归约必须由另外被认为是非冗余的一组规则才能实现，如 3.2 中的规则(5)，这一条规则相当于(3)、(4)2 条规则的作用，但又不能完全替代规则(3)和(4)，一旦将其消除，由该规则一步完成的归约必须由(3)、(4)2 条规则至少 2 步才能实现. 因此，在实际系统的实现中，考虑到系统执行效率等因素，

可有意保留这样的冗余规则,以牺牲少量的存储空间来换取较好的执行效率.

一种简单的方法是通过统计测试的结果决定是否保留或删除相应的规则.例如,以 3.2 中的规则(3)～(5)为测试对象,由语言专家提供 n 个与该组规则相关的源语言句子(n 要足够大),然后运行系统,完成这 n 个句子的翻译,测得系统的运行时间为 t_1,然后屏蔽规则(5),同样完成这 n 个句子的翻译,测得系统的执行时间为 t_2,比较测试结果,如果 $t_2>t_1$ 且 $(t_2-t_1)/t_1>k$(给定的阈值),则保留规则(5);否则,消除规则(5).这种方法着重考虑了系统的运行效率,而如果以存储空间或其它因素为优先考虑条件,则可以采用其它方法决定冗余规则的删除与否.

4.3 谨慎引入单规则

在机译系统中,大部分语法现象可以通过通用规则解决,对于一些特殊语法现象必须通过特殊规则处理,而这些特殊规则中相当一部分是单规则(即没有其它规则与该规则拥有相同的左部).单规则对于处理特殊的语言现象有独特的效果,但从规则库的维护角度出发,为了控制规则库的规模,却应尽量避免使用单规则而采用通用规则处理特殊语法现象.因此,这个问题同 4.2 中的问题一样,需要综合考虑单规则的使用频率和系统的执行效率等相关因素,甚至在特定情况下可以考虑规则处理与过程处理相结合的方法.总之,既要确保单规则的引入不破坏原有规则的一致性和完整性,又要尽量减少冗余规则以提高系统运行的效率.

4.4 冗余控制与一致性、完整性检查相结合

规则的冗余性与一致性、完整性是相互关联的,任意规则的增加都可能引起规则冗余.不慎重地消除冗余规则又可能破坏规则的一致性和完整性,或导致新的冗余规则产生,即出现虚假冗余现象.因此,规则冗余性控制必须与一致性、完整性检查相结合,采用系统检查与人工核对相互辅佐的方式,以确保规则库的一致性和完整性,并达到控制规则冗余性的目的.其处理过程可以简单地表示为图 1 所示的流程.

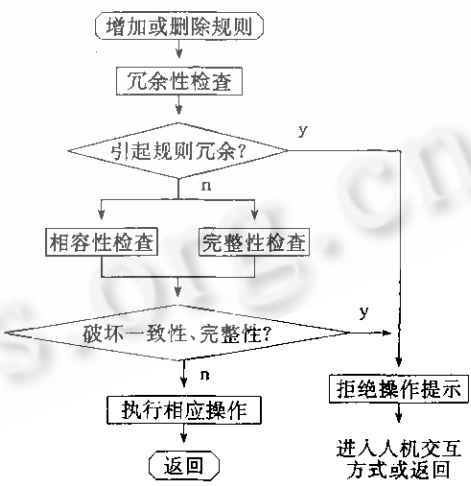

图1 冗余性处理

5 结束语

规则冗余性控制是一个非常复杂的问题.本文根据 SC 文法规则的形式特点,提出的冗余规则检查和消除算法以及控制规则冗余性的基本原则,较好地解决了智能型机译系统中规则冗余性的控制问题,为机译系统规则库的精炼和维护提供了相应的技术保障,其处理思想对其它基于规则的系统也将提供有益的参考.诸多理论和技术问题正在进一步研究和探索中.

致谢　衷心地感谢仰礼友博士在本文的撰写过程中给予的大力支持.

参考文献

1 何新贵. 知识处理与专家系统. 北京：国防工业出版社，1990. 191～216.

2 郑坚平，胡环荣，陈肇雄. 论上下无关文法冗余性. 见：陈肇雄编，机器翻译研究进展，北京：电子工业出版社，1992. 441～444.

3 陈肇雄. SC 文法功能体系. 计算机学报，1992，**15**(**11**)：801～808.

4 叶一民，陈肇雄，高庆狮等. 智能型机译规则库系统 IMT-RB/EC. 中国科学(A 辑)，1990，(**3**)：314～319.

5 王克宏，汤志忠，胡蓬. 知识工程与知识处理系统. 北京：清华大学出版社，1994. 104～120.

6 吴泉源，刘江宁. 人工智能与专家系统. 长沙：国防科技大学出版社，1995. 270～271.

7 王兵山，吴兵. 形式语言. 长沙：国防科技大学出版社，1989.

8 Revesz G E. Introduction to formal language. McGraw-Hiu Book Company, 1983.

RESEARCH ON STRATEGIES FOR CONTROL REDUNDANCY OF RULE BASE

ZONG Chengqing CHEN Zhaoxiong HUANG Heyan

(Research Center of Intelligent Machine Translation Institute of Computing Technology The Chinese Academy of Sciences Beijing 100080)

Abstract Redundancy control is an important aspect in knowledge organization, management and maintenance. According to the characteristics of SC-grammar and machine translation system, this paper presents a new idea that divides redundant rules into two kinds: obvious redundant rules and hidden redundant rules. The paper gives an algorithm for finding obvious redundant rules and discusses in detail the basic strategies that rule out some hidden redundant rules reasonably. Some basic principles to control redundancy of rule base are given also.

Key words Machine translation, rule base, knowledge base, SC-grammar, redundancy.