LTL 公式到自动机的转换 *)

郭 建[1] 边明明[2] 韩俊岗[2]
(西安电子科技大学微电子学院 西安 710071)[1] (西安邮电学院计算机系 西安 710061)[2]

摘 要 在 LTL 公式和自动机理论的基础上,给出了一种从 LTL 公式到自动机的转换算法。该算法先简化 LTL 公式,然后再对简化的 LTL 公式转换,形成选择 Buchi 自动机。此算法与其他算法相比,具有可扩展性的优点,可以在此基础上形成属性描述语言 PSL 向自动机的转换。

关键词 模型检验,Buchi 自动机,选择 Buchi 自动机,LTL 公式

Translation from LTL Formula into Automata

GUO Jian[1] BIAN Ming-ming[2] HAN Jun-gang[2]
(Microelectronics Institute, Xidian University, Xi'an 710071, China)[1]
(Computer Department, Xi'an Institute Post and Telecommunications, Xi'an 710061, China)[2]

Abstract On the basis of introduction to LTL formula and theory of automata, we present a algorithm which can transfer LTL formulae into Buchi automata. In this algorithm, the LTL formula is simplified first, then transferred into an alternating Buchi automaton, before further transferred and then transforms it into a Buchi automaton. Compared to others, this algorithm has certain extensibility, so we can translate PSL into automata.

Keywords Model checking, Buchi automata, Alternating buchi automata, Linear temporal logic

1 引言

随着计算机日益复杂化,对于系统设计正确性的要求越来越高,一些小的错误可能导致整个系统的崩溃或者造成巨大的损失。模型检验是保证系统设计正确性的另一条重要途径。它用根据数学理论来证明所设计的系统满足系统的规范或具有所期望的性质。在不能证明所期望的性质时,则给出一个反例,以指出其设计的错误。在模型检验中,将一个属性公式转换成自动机是其核心,目前有各种各样的算法,都适合与不同的情况。

在基于自动机理论的模型检测方法中,首先是将抽象出的系统模型用 Büchi 自动机来表示,然后将需要验证的属性用一个 LTL 公式来描述,并将该公式取反后转化为 Büchi 自动机,最后检查系统自动机的接受语言是否被包含在性质自动机的接受语言中。如果是,则说明此系统具有 LTL 公式所描述的性质;反之则没有。本文将给出一个从 LTL 公式到自动机的转换算法。此算法与其它算法相比,具有可扩展性的优点。可以在此基础上形成属性描述语言 PSL 到自动机的自动转换。

2 基本概念

2.1 LTL 公式

定义 1[1] 设 AP 是原子命题集合,则 LTL 公式规定如下:

1)如果 $p\in AP$,则 p 是 LTL 公式;

2)设 f 和 g 是 LTL 公式,则 $\neg f$, $f\vee g$, $f\wedge g$, Xf, Ff, Gf, fUg, fRg 也是 LTL 公式。

其中 $\neg$, $\vee$, $\wedge$ 是逻辑非、逻辑或、逻辑与,而 X, F, G, U, R 是时态算子,表示与时间有关的一些特性。

2.2 Alternating Buchi 自动机

定义 2[1] 一个 alternating Buchi 自动机是一个五元组 $A=(\Sigma, S, \{s0\}, \rho, F)$,其中,

Σ: 是非空的有限字母表;

S: 是非空的有限状态集;

$s0\in S$: 是初始状态;

$\rho: S\times\Sigma\rightarrow\beta+(S)$是一个转移函数,

F: 可接受状态集合。

3 总体设计

从 LTL 公式到 Alternating Buchi 自动机的转换总体设计框图如图 1 所示。

程序从文件中读入一个 LTL 公式(一个文件中只允许有一个公式),先对这个公式进行词法分析,提取公式中的运算符与原子命题,如果出现其他非法符号程序提示有错,并要求重新输入。如果没发现错误,再进行语法分析,如果发现有错,如:输入 a && 时,提示 && 符号语法错误,若没有发现错误,则形成一个语法树。下一步对这个语法树进行化简,再根据化简后的语法树构造一个 Alternating Buchi 自动机,在对其进行化简,最后输出自动机。

4 LTL 公式到 Alternating Buchi 自动机的转换

4.1 从 LTL 公式到语法树的形成

在计算机内 $\neg$, G, F, X, U, R, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$分别用!, [], <>, X, U, V, &&, ||, $\rightarrow$, $\leftrightarrow$表示。其中单目运算符有 $\neg$, G, F, X,放在语法树的左子树上,它们的优先级比双目运算符高,它们之间是相同的优先级,双目运算符有 U, R, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$,其中 U 和 R 的优先级小于$\wedge$, $\vee$, $\rightarrow$和 $\leftrightarrow$。

*)国家自然科学基金(NO. 90607008)资助;陕西省教育厅项目(07JK373)资助。郭 建 副教授,博士生。

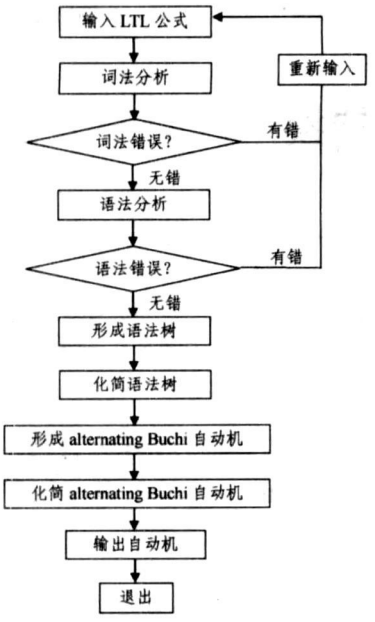

图 1　从 LTL 公式到 Alternating Buchi 自动机转换的总体设计

如:$\theta = \neg(GF\ p \rightarrow G(p \rightarrow F\ r))$形成的语法树:

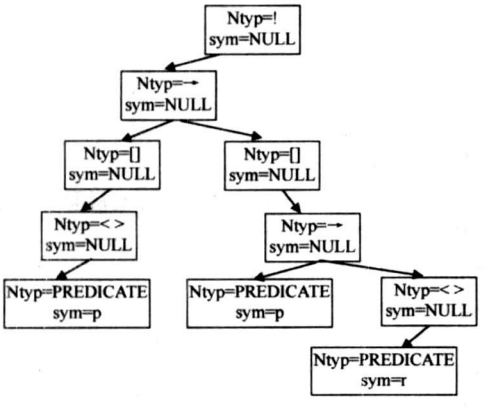

图 2　$\theta = \neg(GF\ p \rightarrow G(p \rightarrow F\ r))$形成的语法树

形成语法树的过程为:

先读入一个词 !,发现它是一个单目运算符,形成这个词的结点,Ntyp= ! sym= NULL。读入(,放入括号匹配栈中,再读入一个词[],单目运算符,形成这个词的结点,Ntyp=[],sym= NULL,连接在 ! 的左子树上。再读入一个词〈〉,它是单目运算符,再对它形成一个结点,连接在[] 的左子树上。再读入一个词 p,是原子命题,建立结点 Ntyp= PREDICATE sym= p,连接在〈〉的左子树上。再读入→ 时,发现是一双目运算符,则将前面所形成的树连接在→ 的左子树上,根结点指针指向→所形成的结点。再读入[],比 →的优先级低,则将它的结点连接在→ 的右子树上。读入(,放入括号匹配栈中,再读入词 p,将 p 所形成的结点连接在[] 的左子树上。再读入 →,发现其优先级高于[],则将它连接在[] 左子树上,再将 p 连接在 →的左子树上。再读入 →,比 →优先级低,则将其连接在 →的右子树上。再读入 r,连接在〈〉→的左子树上。再读入两个),栈弹出,栈为空,括号匹配成功。到此,语法树形成,返回语法树根结点指针。

4.2　语法树的简化

在一个 LTL 公式进行转化成自动机时,要对公式进行简化,简化的等式如下:

$tt \equiv p \vee \neg p$, $ff \equiv \neg tt$, $\varphi 1 \wedge \varphi 2 \equiv \neg(\neg\varphi 1 \vee \neg\varphi 2)$

$\varphi 1\ R\ \varphi 2 \equiv \neg(\neg\varphi 1\ U\ \neg\varphi 2)$　$F\varphi \equiv tt\ U\ \varphi$　$G\varphi \equiv ff\ R\ \varphi \equiv \neg F \neg\varphi$

将公式转化成只含有 U,F,R,X,¬(非),∧ 的最简形式。

如:$\theta = \neg(GF\ p \rightarrow G(p \rightarrow F\ r))$的最简形式是

$(tt\ U((ff\ V\ !r)\ \&\&p)\ \&\&(ff\ V\ (tt\ U\ p))$

有一些化简在形成语法树的时候就可进行了,如:G false == false, G G p== G p, X true = true, X false = false, F true == true, F F p== F p, F(pU q) = F q, G(pV q) = G q

剩下的是在语法树遍历的时候进行化简,化简的时候对有不同的 ntyp 值的结点进行分类化简。对上面的语法树化简为:

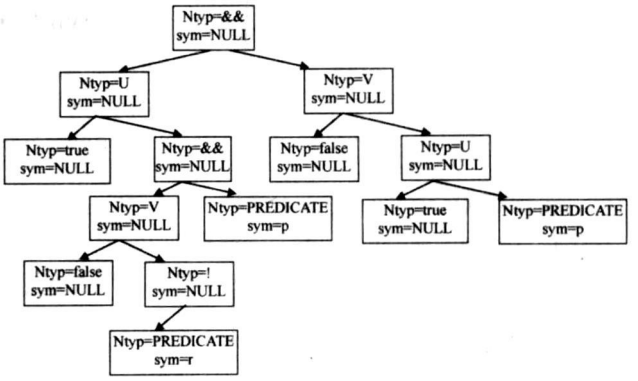

图 3　$(tt\ U((ff\ V\ !r)\ \&\&p)\ \&\&(false\ V\ (true\ U\ p))$化简后的语法树

4.3　从语法树到 Alternating Buchi 自动机的转换

设一个 Alternating Buchi 自动机 A:=(Σ, S, $s0$, ρ, F),那么对于给定的一个 LTL 公式 ξ,按照下列方法转换成一个自动机 A;

字母表为 2^{Prop};

状态集 Q 包含所有有 ξ 的子公式和其子公式的否定式;

开始状态 $s0$ 就是 ξ 公式本身;

终止状态集 F 的可接受状态包括所有在 S 中类似 ¬(3 U ξ)的子公式。

下面给出 ρ 是转移函数:

True = ¬False;

$(\alpha \wedge \beta) = \neg(\neg\alpha \vee \neg\beta)$;

$(\alpha \vee \beta) = \neg(\neg\alpha \wedge \neg\beta)$;

$\rho(p, a) =$ True 若 $p \in a$;

$\rho(\neg p, a) =$ False 若 $p \in a$;

$\rho(\eta \wedge \theta, a) = \rho(\eta, a) \wedge \rho(\theta, a)$;

$\rho(\neg\eta, a) = \neg\rho(\eta, a)$;

$\rho(X\eta, a) = \eta$;

$\rho(\eta\ U\ \theta, a) = \rho(\theta, a) \vee (\rho(\eta, a) \wedge \eta\ U\ \theta)$;

$\rho(\eta\ R\ \theta, a) = \rho(\theta, a) \wedge (\rho(\eta, a) \vee \eta\ R\ \theta)$;

例如:$\theta = \neg(GF\ p \rightarrow G(p \rightarrow F\ r))$的最简形式是

$(tt\ U((ff\ V\ !r)\ \&\&p)\ \&\&(ff\ V\ (tt\ U\ p))$形成的自动机为:

1:!r, 2:ff, 3:ff V !r, 4: p, 5:(ff V !r) &&p, 6:tt, 7:tt U((ff V !r) &&p), 8:tt U p, 9:ff V(tt U p), S:(tt U((ff V !r) &&p) &&(ff V(tt U p)),

10: 终止状态。

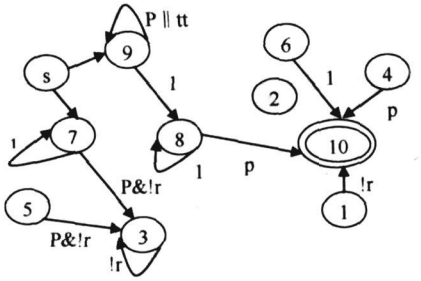

图 4 （tt U (((ff V !r) &&p) &&(ff V (tt U p)) 形成的自动机

自动机形成过程:

对语法树进行遍历可以得到两个重要的数据: 状态个数与原子命题个数。这样就可以得到字母表和状态集。

根据定理规定开始状态 S 是公式 θ，从语法树根结点开始的两个子树，分别遍历到树的最顶叶子结点，分为两种情况: 类似 P 与 !P 的结构，这样形成第一个状态 !r，与 !r 是兄弟的 ff 形成第二个状态，然后它们与它们的根成为第三个状态，就这样语法树的左子树依次可形成 1: !r，2: ff，3: ff V !r，4: p，5: (ff V !r) &&p，6: tt，7: tt U ((ff V !r) &&p) 这七个结点，右子树分别形成 tt，p，8: tt U P，9: ff V (tt U p) 这四个结点，第十个结点就是 S 了，再下一步是将相同的状态合并，对 label[] 和 sym_table[] 进行初始化，最终形成十个状态。

完成所有结点的转换关系都计算完成并记录在转移表 ransition 中，Alternating Buchi 自动机就建好了。

4.4 Alternating Buchi 自动机的化简

在转换关系 transition 中已经存储了已建好的自动机，从 transition[0] 开始，检查所有可以到达的状态，保留可到达状态，将不可到达状态删除。这样我们完成 Alternating Buchi 自动机的简化。

例如: θ= ¬(GF p→ G(p→ F r))所形成的自动机简化成:

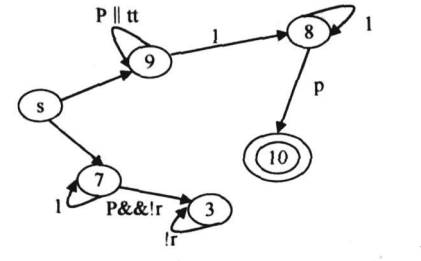

图 5 θ= ¬(GF p→ G(p→ F r))所形成的简化自动机

从 S 开始可到达的状态有 9，8，7，3，其他状态都删除。这样我们简化了 Alternating Buchi 自动机。

5 实验结果

5.1 运行结果

以θ= ¬(GF p→ G(p→ F r))为输入，程序的运行结果是:

```
formula: ! ([]<> p=>[] (p=><> r))
    / * Normlzd: ((true U ((false V !((r))) && (p))) && (false V (true U (p)))) */
Alternating automaton before simplification
init: ((true U ((false V !((r))) && (p))) && (false V (true U (p))))
1 => {7, 9}
state 9: (false V (true U (p)))
P-> {9}
1 => {8, 9}
state 8: (true U (p))
1 => {8}
P-> {}
state 7: (true U ((false V !((r))) && (p)))
1 => {7}
P & ! r => {3}
state 6: true
1 => {}
state 5: ((false V !((r))) && (p))
P & ! r => {3}
state 4: (p)
P-> {}
state 3: (false V !((r)))
! r => {3}
state 2: false
state 1: !((r))
! r => {}
Alternating automaton after simplification
init: ((true U ((false V !((r))) && (p))) && (false V (true U (p))))
1 => {7, 9}
state 9: (false V (true U (p)))
P-> {9}
1 => {8, 9}
state 8: (true U (p))
1 => {8}
P-> {}
state 7: (true U ((false V !((r))) && (p)))
1 => {7}
P & ! r => {3}
state 3: (false V !((r)))
! r => {3}
```

可以看出所形成的 Alternating Buchi 自动机与理论相同，只是状态序号不同。

5.2 空间与时间开销测试

测试公式 θn = !((F $a1$∧F $a2$∧F $a3$∧F $a4$…∧F an)-> G(F b))

下表是公式规模 n 对应程序所用的内存空间(B) 与时间(S)开销。

表 1 公式规模 n 对应程序所用的内存空间(B) 与时间(S)开销

公式规模(n)	内存空间开销(B)	时间开销(S)
1	7026	0.01
2	13764	0.01
3	14004	0.01
4	27492	0.01
5	52528	0.01
6	53924	0.01
7	103936	0.01
8	104096	0.01
9	202572	0.01
10	205532	0.03
11	207164	0.04
12	403996	0.08
13	405196	0.16
14	410508	0.30
15	804436	0.63
16	804796	1.40
17	805042	3.07
18	806884	7.21
19	1593404	16.44
20	1593664	41.27
21	1604260	96.59

结束语 已经成功的完成 LTL 公式到 Alternating Buchi 自动机的转换工作，我们对公式和自动机的形成都进行了化简，这样有效地使自动机规模大大减小。在公式规模不是很大的情况下，我们的程序可以很快给出结果，并且空间开销不超过 2MB。将来的工作在此基础上扩展其公式中操作符，实

(下转第 276 页)

```
    if I_{k+1}能让 P 被经过 then I_f= I_{k+1};
      Done= true
    else k++ endif
    endwhile
  endprocedure
```

4 实例分析

4.1 应用实例

考虑到文献[1, 2]所提方法的比较，我们亦采用图 1 所示的程序。同样取 P={1, 2, 3, 4, P1$_1$, P2$_1$, P3$_1$, 7, P1$_2$, P2$_2$, P3$_2$, 6, 7, P1$_3$, 8}，I_0=(low=39, high=93, step=12, A[1]=1, …, A[100]=100)。

为简单起见，令 l, h, s 分别表示 low, high, step。由于每次执行循时，数组元素 A[i] 是变化的，故根据路径 P，可令 $X=A[i]$，$Y=A[l+s]$，$Z=A[l+2s]$。执行算法：

由于 I_0 不能使 P 通过，故继续执行算法后面的步骤。

因 P 中谓词函数 P1$_1$, P2$_1$, P3$_1$, P1$_2$, P2$_2$, P3$_2$, P1$_3$ 都是线性的，所以直接执行算法的第 2 步，构造谓词函数关于输入变量 I_0 的线性约束系统：

step2

$$\begin{cases} l+s-h<0 \\ X-Y\geqslant 0 \\ X-Y\leqslant 0 \\ l+2s-h<0 \\ X-Z\geqslant 0 \\ X-Z>0 \\ l+3s-h\geqslant 0 \end{cases}$$

step3 求解线性约束系统。

设 $a, d, f>0$, $b, c, e\geqslant 0$, 则线性约束系统可变成以下线性方程组：

$$\begin{cases} l+s-h=-a \\ X-Y=b \\ X-Y=-c \\ l+2s-h=-d \\ X-Z=e \\ X-Z=f \\ l+3s-h=g \end{cases}$$

令 $a=d=f=1$, $b=c=e=g=0$, 根据文献[5]中求解线性约束系统的方法，求线性方程组的最小二乘解，可求得 $l=1$, $h=4$, $s=1$, $X=2$, $Y=2$, $Z=1$。由于数组其它元素与路径中分支谓词函数无关，可令其与 I_0 相同，故新的输入 I_1=($low=1$, $high=4$, $step=1$, A[1]=2, A[2]=2, A[3]=1, A[4]=4, …, A[100]=100)。

由于 I_1 可使路径 P 被经过，故算法结束。

我们使用文献[2]的方法和本文所提出的方法，在系统配置为 CPU P4 1.6G; 512M DDR; OS Red Flag 4.1 的计算机上，对上述实例分别进行了实验，其实验结果(两种方法的运行时间)见表 1。

表 1 实例的执行时间 (ms)

实验次数	1	2	3	4	5	平均时间
文献[2]方法	60	70	60	60	70	64
本文方法	30	30	30	40	30	32

4.2 结果分析

从上述实例中可看出，对于同一程序中的同一路径，其测试数据的生成，文献[1]执行了 21 次，而本文的方法仅需执行 8 次，计算量大大减少，可节省大量的资源，因而具有求解速度快、简单直观等特点。同时，迭代 1 次即可求得所需测试数据，或确保路径不可达。与文献[2]比较，虽然两者都只需执行 8 次，但本文的方法不必计算路径中的谓词片和确定输入依赖集，也不必计算谓词函数的线性算术表示、谓词残量以及构造谓词函数关于输入变量的增量的线性约束，因而计算量大大减少。从实验的情况看，文献[2]的方法求解测试数据的平均时间为 64ms，而本文的方法仅为 32ms。可见，理论分析和实验结果均表明：本文的方法明显优于文献[1, 2]的方法。

结束语 本文提出了一种新的带数组、循环的路径测试数据自动生成的方法。该方法对于给定的路径能很好地自动地生成测试数据，较好地解决了带数组、循环的路径测试数据的自动生成的问题，并能容易实现工具化。它比目前其它方法更有效。由于本方法同时考虑了路径中所有的分支谓词，所以对于线性约束路径，通过一次迭代即可求出测试数据，或确保路径不可达；对于含有非线性约束的路径，通过多次迭代亦可求出测试数据，或在相当程度上能保证路径不可达。另外，该方法同样适用于其它的语言程序的路径测试数据的自动生成。

参 考 文 献

[1] Korel B. Automated Software Test Data Generation[J]. IEEE Transactions on Software Engineering, 1990, 16(8): 870-879

[2] Neelam Gupta Aditya P. Mathur Mary Lou Soffa. Automated Test Data Generation Using An Iterative Relaxation Method [A] //ACM SIGSOFT Sixth International Symposium on Foundations of Software Engineering (FSE-6). Florida, United States: ACM, 1998: 231-244

[3] 单锦辉. 面向路径的测试数据自动生成方法研究[D]. 长沙: 国防科学技术大学研究生院, 2002

[4] Glass H, Cooper L. Sequential search: A method for solving constrained optimization problems[J]. J. ACM, 1965, 12(1): 71-82

[5] Neelam Gupta Aditya P. Mathur Mary Lou Soffa. UNA based Iterative Test Data Generation and its Evaluation[A] //14th IEEE International Conference on Automated Software Engineering (ASE'99). Cocoa Beach, Florida, USA, October 1999: 224-232

(上接第 243 页)

现 IEEE 的标准 PSL(属性描述语言)到自动机的转换。

参 考 文 献

[1] 易锦, 张文辉. 从基于迁移的扩展 Büchi 自动机到 Büchi 自动机. 软件学报, 2006, 17(4)

[2] Gastin P, Oddoux D. Fast LTL to Buchi Automata Translation, Lecture Notes In Computer Science; Vol. 2102 archive //Proceedings of the 13th International Conference on Computer Aided Verification. ISBN: 3-540-42345-1, 2001: 53-65

[3] Vardi M Y. An Automata-Theoretic Approach to Linear Temporal Logic. www.cs.rice.edu/~vardi/papers/banff94rj.ps.gz

[4] 蒋立源, 康目宁. 编译原理(第二版). 西北工业大学出版社, 2003