基于逆推算法和深度优先搜索的模型检验

数据结构大作业

王逸轩、韩啸

北京大学数学科学学院

December 20, 2017

1 问题分析
- 背景
- 解决方案

2 算法设计
- CTL 算法设计
- LTL 算法设计

3 模型展望
- 模型的评价
- 模型的推广

4 参考书目

5 结束语

Section 1

问题分析

基本术语

- 迁移状态系统 TS;

基本术语

- 迁移状态系统 TS;
- 逻辑公式 L;

基本术语

- 迁移状态系统 TS;
- 逻辑公式 L;
- 公式类型 LTL、CTL 等;

基本术语

- 迁移状态系统 TS;
- 逻辑公式 L;
- 公式类型 LTL、CTL 等;
- 公式基本算符.

问题的关联

从给定的迁移状态系统出发，我们研究不同的逻辑公式，发现本质上就是针对对象的不同。

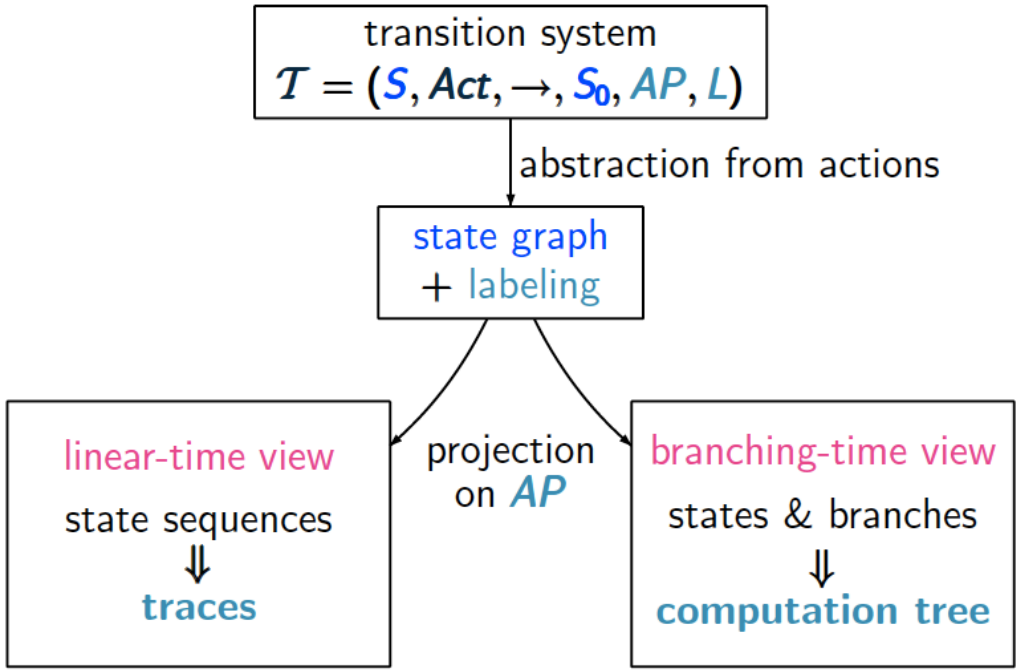


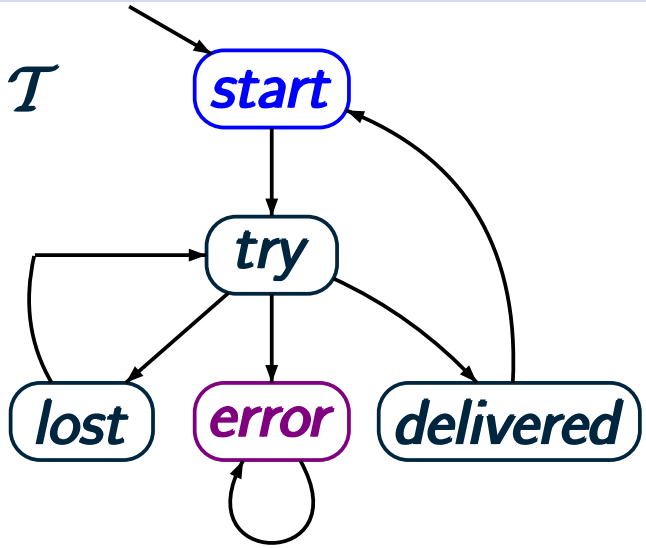
Figure: PPT 中刻画 CTL 与 LTL 模型检验的关联

两个例子

我们具体考察 CTL 与 LTL 模型检验的两个简单实例。

Example: CTL semantics

CTLSS4.1-16



$\mathcal{T} \models \exists\Diamond\forall\Box\neg start$

$\mathcal{T} \models \forall\bigcirc\exists\bigcirc\forall\Box\neg start$

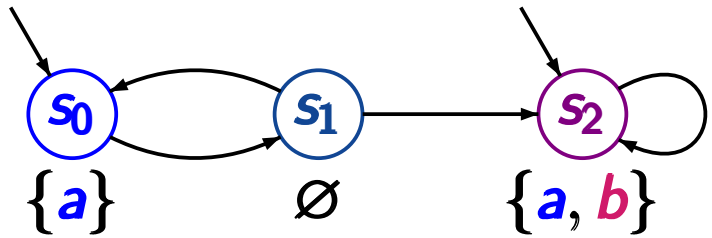
$\mathcal{T} \not\models \exists\bigcirc\forall\bigcirc\forall\Box\neg start$

$\Phi_3 = \exists\bigcirc\forall\bigcirc\forall\Box\neg start \rightsquigarrow \boxed{\exists\bigcirc error}$

$$
\begin{aligned}
Sat(\forall\Box\neg start) &= \{error\} \\
Sat(\forall\bigcirc\forall\Box\neg start) &= \{error\} \\
Sat(\exists\bigcirc\forall\bigcirc\forall\Box\neg start) &= \{error, try\}
\end{aligned}
$$

Which formulas hold for $\mathcal{T}$?

LTLSF3.1-11



$AP = \{a, b\}$

$\mathcal{T} \models a$ as $s_0 \models a$ and $s_2 \models a$

$\mathcal{T} \not\models \Diamond\Box a$ as $s_0\, s_1\, s_0\, s_1 \ldots \not\models \Diamond\Box a$

$\mathcal{T} \models \Diamond\Box b \vee \Box\Diamond(\neg a \wedge \neg b)$ as $s_2 \models b$, $s_1 \not\models a, b$

$\mathcal{T} \models \Box(a \rightarrow (\bigcirc\neg a \vee b))$ as $s_2 \models b$, $s_0 \models \bigcirc\neg a$

待解决的问题

1. 刻画迁移状态系统 TS;

待解决的问题

1. 刻画迁移状态系统 TS;
2. 刻画逻辑公式 L;

待解决的问题

1. 刻画迁移状态系统 TS;
2. 刻画逻辑公式 L;
3. 建立对公式的一些运算法则（如取子式、递推等等）;

待解决的问题

1. 刻画迁移状态系统 TS;
2. 刻画逻辑公式 L;
3. 建立对公式的一些运算法则（如取子式、递推等等）;
4. 针对不同的公式类型，设计算法.

问题难点

1. 如何刻画初始的输入信息;
2. CTL 可以利用朴素的逆推方法解决，如何定位子串;
3. LTL 中逆推算法可能无限循环，找到替代方案处理“无穷”的问题;
4. 通过逻辑筛选，简化枚举次数，降低时间复杂度.

处理问题的思想

1. 用图结构储存迁移状态系统 TS;
2. 用字符串存储逻辑公式 L，并假设已指明逻辑类型并添加括号;
3. 对 CTL 公式，利用建立的运算法则，逆推判断;
4. 对 LTL 公式，利用公式的所有子串的满足情况构建新的图，深度优先搜索举例.

Section 2

算法设计

Subsection 1

CTL 算法设计

逆推算法

算法的指导思想

本质上的思想是找到每个逻辑公式对应的满足该公式的节点集合。
解决了最简单的公式对应的节点集合之后，就可以对于复杂的公式，利用逆推算法，脱去最外层的括号，找到相应的节点集合。
最后通过原始逻辑公式 L 找到的相应节点集合，判断迁移状态系统 TS 的初始节点是否在该集合中即可。

逆推算法

需要注意的问题

尽管我们结合前面的分析，知道逆推算法能够有效的解决 CTL 模型检验问题，但我们还需要注意处理以下的细节。

1. 括号匹配：我们在逆推时需要定位最左侧括号对应的反括号，进而分割出完整的子式语句;
2. 最外层算符的讨论：我们在逆推时需要特别注意，最外侧为 $\forall$、$\exists$ 这两个算符时，要连同下一个算符一起考虑。

Subsection 2

LTL 算法设计

Recall: nondeterministic Büchi automata

LTLMC3.2-DEF-NBA

NBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

- Q finite set of states
- Σ alphabet ⟵ here: $\Sigma = 2^{AP}$
- $\delta : Q \times \Sigma \to 2^Q$ transition relation
- $Q_0 \subseteq Q$ set of initial states
- $F \subseteq Q$ set of final states, also called accept states

accepted language $\mathcal{L}_\omega(\mathcal{A}) \subseteq \Sigma^\omega$ is given by:

$$\mathcal{L}_\omega(\mathcal{A}) \stackrel{\text{def}}{=} \text{set of infinite words over } \Sigma \text{ that have an accepting run in } \mathcal{A}$$

利用 NBA 解决 LTL 模型检验

克服逆推算法带来的“无穷”的问题

如前所述，朴素的逆推算法难以解决 LTL 模型检验的问题，我们参阅*Introduction to Model Checking*这门课程的 PPT 资源，利用 NBA 找到了以下代替方案：
我们把逻辑公式 S 的所有子串给记录下来，然后这些子串形成一个集合，我们考察这个集合生成的子集族 Q，子集族中每个元素代表着相应的子串得到了满足，而余下的子串没有得到满足。我们考察一个新的图 G_1，节点集合是由原来的节点集合 V 与新得到的子集族 Q 形成的的笛卡尔积，我们设法把原来的迁移状态系统 TS 存在反例的问题转化为新图 G_1 存在一条满足某种性质的路径的问题，这样一来通过对 G_1 的深度搜索即可完成结论。

理论支持

首先我们需要说明原来的迁移状态系统 TS 存在反例，就等价于新图 G_1 中存在某种无穷长的路径。这在 PPT 的如下定理中得到佐证。

Claim: If $B_0 \xrightarrow{A_0} B_1 \xrightarrow{A_1} B_2 \xrightarrow{A_2} \ldots$ is a path in $\mathcal{G}$ s.t.

$$\forall F \in \mathcal{F} \ \overset{\infty}{\exists} j \geq 0.\ B_j \in F$$

then for all formulas $\psi \in cl(\varphi)$:

$$\psi \in B_0 \quad \text{iff} \quad A_0 A_1 A_2 \ldots \models \psi$$

其次，我们说明给出的反例一定可以具有无穷循环的回路的形式。这个事实我们通过回路的拼接可以说明。

核心算法的描述

结合之前的理论支持，我们把寻找反例转化为找到在 G_1 中的一个无穷路径，其中有限步之后形成回路的无穷循环，这个回路满足所有出现的无穷算符。最后一步，我们通过深度优先搜索来实现。

具体算法的设计

我们简明扼要地给出算法的步骤：

1. 简化输入的 LTL 公式，用 U “直到”算符来表示 W “弱直到”、D “最终满足”、S “一直满足”等算符

具体算法的设计

我们简明扼要地给出算法的步骤：

1. 简化输入的 LTL 公式，用 U “直到”算符来表示 W “弱直到”、D “最终满足”、S “一直满足”等算符
2. 找到那些出现 U “直到”算符，即需要之后回路判断的子串

具体算法的设计

我们简明扼要地给出算法的步骤：

1. 简化输入的 LTL 公式，用 U “直到”算符来表示 W “弱直到”、D “最终满足”、S “一直满足”等算符
2. 找到那些出现 U “直到”算符，即需要之后回路判断的子串
3. 先进行一步筛查来判断哪些图 G_1 中的点连有边，即原图 G 中哪些状态可以相互转化

具体算法的设计

我们简明扼要地给出算法的步骤：

1. 简化输入的 LTL 公式，用 U “直到”算符来表示 W “弱直到”、D “最终满足”、S “一直满足”等算符
2. 找到那些出现 U “直到”算符，即需要之后回路判断的子串
3. 先进行一步筛查来判断哪些图 G_1 中的点连有边，即原图 G 中哪些状态可以相互转化
4. 最后对于图 G_1 进行深度优先搜索，判断是否存在满足反例条件的回路

Section 3

模型展望

模型的优点

1. 处理 CTL 模型检验时，我们的算法在思想上和具体实现上都十分简单，程序的复杂度也很低，我们可以容易地处理大规模的模型检验运算。
2. 处理 LTL 模型检验时，我们的算法将反例化归到回路情况，能够较好地给出反例。
3. 我们将输入输出标准化，存储在文件里，判断具体的模型检验时就显得一目了然了。

模型的缺点

1. 在 LTL 模型检验中，我们的程序在时间复杂度和空间复杂度上都显得太过于复杂，深度搜索引入了近乎难以容忍的运算量，于是 LTL 的算法只能解决比 CTL 模型检验规模小的多的问题。
2. 程序的交互性做的不够好，不能达到一个较好的可视化界面的设计。

CTL^* 模型检验

定义

We saw in Theorem 6.21 on page 337 that CTL and LTL have incomparable expressiveness. An extension of CTL, proposed by Emerson and Halpern, called CTL*, combines the features of both logics, and thus is more expressive than either of them.

6.8.1 Logic, Expressiveness, and Equivalence

CTL* is an extension of CTL as it allows path quantifiers $\exists$ and $\forall$ to be arbitrarily nested with linear temporal operators such as $\bigcirc$ and U. In contrast, in CTL each linear temporal operator must be immediately preceded by a path quantifier. As in CTL, the syntax of CTL* distinguishes between state and path formulae. The syntax of CTL* state formulae is roughly as in CTL, while the CTL* path formulae are defined as LTL formulae, the only difference being that arbitrary CTL* state formulae can be used as atoms. For example, $\forall \bigcirc \bigcirc a$ is a legal CTL* formula, but does not belong to CTL. The same applies to the CTL* formulae $\exists \Box \Diamond a$ and $\forall \Box \Diamond a$. (However, $\forall \Box \Diamond a$ is equivalent to the CTL formula $\forall \Box \forall \Diamond a$.)

CTL^* 模型检验

算法

CTL^* 模型检验，在本质上是 LTL 与 CTL 模型检验算法的整合，我们利用逆推算法来拆解 CTL^* 逻辑公式，在必要的逆推步骤时利用 LTL 模型检验得到的结果。

Algorithm 27 CTL* model checking algorithm (basic idea)

Input: finite transition system TS with initial states I, and CTL* formula Φ
Output: $I \subseteq Sat(\Phi)$

for all $i \leqslant |\Phi|$ **do**
  for all $\Psi \in Sub(\Phi)$ **with** $|\Psi| = i$ **do**
    switch(Ψ):

true	:	$Sat(\Psi) := S$;
a	:	$Sat(\Psi) := \{ s \in S \mid a \in L(s) \}$;
$a_1 \wedge a_2$	:	$Sat(\Psi) := Sat(a_1) \cap Sat(a_2)$;
$\neg a$	:	$Sat(\Psi) := S \setminus Sat(a)$;
$\exists \varphi$	:	determine $Sat_{LTL}(\neg\varphi)$ by means of an LTL model-checker;
	:	$Sat(\Psi) := S \setminus Sat_{LTL}(\neg\varphi)$

    end switch
    $AP := AP \cup \{ a_\Psi \}$; (* introduce fresh atomic proposition *)
    replace Ψ with a_Ψ
    forall $s \in Sat(\Psi)$ **do** $L(s) := L(s) \cup \{ a_\Psi \}$; **od**
  od
od
return $I \subseteq Sat(\Phi)$

$TCTL$ 模型检验

探索

本质上 $TCTL$ 就是考虑到时间因素的 CTL 模型，逆推的过程中考虑时间元素即可。

$PCTL$ 模型检验

展望

$PCTL$ 中状态出现的概率也可以朴素的使用逆推来完成计算，并比较输出检验成果。

参考书目

 Christel Baier. et al. *Principles of Model Checking*. The MIT Press, 2008.

 *Introduction to Model Checking*. A course for software modeling and verification in RWTH Aachen University. 2016.

谢谢！