软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(8):2769−2781 [doi: 10.13328/j.cnki.jos.006597]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种利用非确定规划的 LTL 合成方法*

陆 旭[1,2], 于 斌[1,2], 田 聪[1,2], 段振华[1,2]

[1](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

[2](综合业务网理论及关键技术国家重点实验室(西安电子科技大学), 陕西 西安 710071)

通信作者: 于斌, 田聪, 段振华 E-mail: byu@xidian.edu.cn, {ctian, zhhduan}@mail.xidian.edu.cn

摘 要: LTL 合成(linear temporal logic synthesis)是程序合成(program synthesis)的一类重要子问题, 旨在自动构建一个控制器(controller), 且要求该控制器和环境(environment)的行为交互满足给定的 LTL 公式. 一般来说, 可以将 LTL 合成定义为二人博弈(two-player game)问题, 博弈的双方是环境和控制器, 该问题的解称为合成策略. 近年来, 有研究从理论角度讨论了 LTL 合成与非确定规划(non-deterministic planning)的相关性. 基于此, 提出了一种新的利用非确定规划求解 LTL 合成问题的方法, 并证明了方法的正确性和完备性. 具体而言, 首先获得 LTL 公式对应的 Büchi 自动机, 结合二人博弈定义, 将 LTL 合成问题转换为完全可观测的非确定规划模型; 然后交由高效规划器求解. 通过实验结果说明: 与其他 LTL 合成方法相比, 提出的基于规划的合成方法在解质量方面具有较大的优势, 能够获得规模较小的合成策略.

关键词: 二人博弈; Büchi 自动机; LTL 合成; 非确定规划

中图法分类号: TP311

中文引用格式: 陆旭, 于斌, 田聪, 段振华. 一种利用非确定规划的 LTL 合成方法. 软件学报, 2022, 33(8): 2769–2781. http://www.jos.org.cn/1000-9825/6597.htm

英文引用格式: Lu X, Yu B, Tian C, Duan ZH. LTL Synthesis via Non-deterministic Planning. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2769–2781 (in Chinese). http://www.jos.org.cn/1000-9825/6597.htm

LTL Synthesis via Non-deterministic Planning

LU Xu[1,2], YU Bin[1,2], TIAN Cong[1,2], DUAN Zhen-Hua[1,2]

[1](School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

[2](State Key Laboratory of Integrated Service Networks (Xidian University), Xi'an 710071, China)

Abstract: LTL synthesis is an important sub-class of program synthesis. The process of LTL synthesis is to automatically build a controller which interacts with the environment, where the objective is to make the interactive behaviors satisfy a given LTL formula. Generally speaking, LTL synthesis problem is often defined as a two-player game, one player is environment, and the other is controller. The solution of the problem is called synthesis policy. Recently, researchers have investigated that there exists close correspondence between LTL synthesis and non-deterministic planning from a theoretical point of view. This paper presents a novel LTL synthesis approach exploiting non-deterministic planning techniques. Moreover, the correctness and the completeness of the approach is proved formally. Concretely, at first LTL formulas are converted into Büchi automata, then the automata with the two-player game definition of LTL synthesis are translated into full-observable non-deterministic planning models which can be directly fed to existing effective planners. The experimental results show that planning based LTL synthesis has significant advantage over other approaches in improving the quality of solutions, i.e., the size of the obtained policies is much smaller.

* 基金项目: 国家自然科学基金(61806158, 61732013, 62172322, 62002290); 中国博士后科学基金(2019T120881, 2018M643585); 国家重点研发计划(2018AAA0103202); 陕西省重点科技创新团队(2019TD-001); 陕西省自然科学基础研究计划(2021JQ-208)

本文由"形式化方法与应用"专题特约编辑陈立前副教授、孙猛教授推荐.

收稿时间: 2021-09-02; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

Key words: two-player game; Büchi automata; LTL synthesis; non-deterministic planning

程序合成(program synthesis)的概念最早由美国数学家、逻辑学家 Alonzo Church 于 1957 年提出[1], 一直以来都是计算机科学领域的难解问题. 简单地说, 程序合成是一种针对用户意图构造出相关程序的软件开发活动, 而用户意图可以采用形式化或半形式化规约, 甚至是自然语言来描述. 如果意图表达为线性时序逻辑(linear temporal logic, LTL)公式, 则称此类问题为 LTL 合成(LTL synthesis), 通常看作程序合成的一种子类. 一般来说, 可以将 LTL 合成定义为二人博弈(two-player game)问题, 博弈的双方是环境(environment)和控制器(controller). 控制器的行为被智能体(agent)操控, 根据环境变化做出反应, 但智能体无法干涉环境的行为. LTL 合成的解(称为合成策略)是使得控制器能够获胜的策略, 目的是令环境和控制器的所有交互行为序列满足给定的 LTL 公式. 1989 年, Pnueli 和 Rosner 证明了 LTL 合成的计算复杂度为 2EXPTIME-complete[2]. 由于复杂度较高, 学者们通过研究 LTL 子集的合成来获得较低的复杂度, 进而提高合成效率. 其中, 针对一类子集 GR(1) (generalized reactivity(1)), 学者们提出了复杂度为多项式级别($\mathbb{O}(n^3)$)的算法, 用于检查 GR(1)公式是否可实现(realizable), 并给出了合成策略[3]. 目前存在许多有效的 LTL 合成工具, 如 Acacia+[4], Lily[5], SynKit[6]和 ltlsynt[7]等.

近期涌现出许多工作, 通过借鉴高效的符号化技术求解 LTL 合成问题. 文献[8]致力于研究安全 LTL(不含 until 操作符的 LTL 公式)合成, 并提出一种构建符号化安全自动机(deterministic safety automata)的方法, 从而利用基于传统的 BDD (binary decision diagram)技术直接求解. Bohy 等人提出了一种基于反链符号化增量算法求解安全博弈问题, 而安全博弈问题刚好可以由 LTL 合成问题归约而来[4]. 在求解过程中, LTL 公式尽可能地被转换为确定 Büchi 自动机(deterministic Büchi automata). 该方法的优势在于可以获得更紧凑的策略, 并可对规模较大的 LTL 合取公式进行组合求解; 而缺点在于不是所有 LTL 公式都存在对应的确定 Büchi 自动机, 因此存在一定的局限性.

LTL_f 是 LTL 的一个变种, 二者的区别在于, 前者的语义模型限制为有穷[9]. 目前也存在一部分研究工作聚焦 LTL_f 合成问题. Giacomo 和 Vardi 从理论的角度证明了一个重要结论, 即 LTL_f 合成的计算复杂度与 LTL 合成相同, 前者并没有因为有穷模型而使得复杂度降低[10]. Zhu 等人提出了一种基于符号化的框架, 将 LTL_f 合成问题归约为 DFA (deterministic finite automata)博弈, 并进一步表示为布尔公式, 然后利用布尔分析生成策略[11]. LTL_f 合成还在机器人领域得到实际应用. 文献[12]提出了一种基于 BDD 的组合方法, 将 LTL_f 合成应用于有限视野任务, 能够合成 UR5 机器人的任务策略. 文献[13]在资源有限条件下研究如何生成机器人策略, 采用 LTL_f 描述有限任务, 并通过将 LTL_f 合成归约为定量博弈的技术路线进行求解. 此外, 最新的工作着重在不同假设的前提下研究 LTL_f 合成问题, 如公平性和稳定性[14]、LTL 条件[15]等.

近年来, 有研究成果表明, 智能规划(automated planning)[16]与 LTL 合成有着紧密的联系, 从而开辟了一种 LTL 合成的新途径. 具体而言, LTL 合成与智能规划的一类子问题, 即非确定规划(non-deterministic planning)存在很多相似之处. Camacho 等人将 LTL_f 合成问题转换为非确定规划问题, 融合了动态状态抽象和惰性自动机确定化技术, 不仅能够给出合成策略, 在合成不可实现(unrealizable)的情况下, 还能够给出相应证明[17]. 文献[18]将目标为 LTL 公式的非确定规划描述为二人博弈, 基于此构建起 LTL 合成和非确定规划的关系, 使得二者可以很容易地进行相互转换. 本文借鉴了文献[19]提出的转换方法, 并对其进行改进. 关键的区别在于: 本文提出的方法在规划过程中仅模拟自动机的一条路径, 而文献[19]则同时保留多条路径. 两种方法的详细比较将在后续实验章节阐述. 与前述工作不同, D'Ippolito 等人从相反的思路研究非确定规划与 LTL 合成的关系, 将非确定规划编码为 LTL 合成问题, 并利用合成技术进行规划求解[20]; Bonet 等人将广义规划(generalized planning)问题规约为 LTL 合成问题[21]. 此外, 还有工作研究其他类型时序逻辑的合成问题, 如采用 STL (signal temporal logic)描述合成意图[22]. 其中, STL 能够表达实值信号的时序性质, 一般可用于分析混成系统或混合信号电路.

本文提出了一种新的基于智能规划技术的 LTL 合成方法, 其核心思想是: 在二人博弈定义的前提下, 将

LTL 合成转换为非确定规划模型. 博弈双方的行为均由规划动作模拟. 在此过程中, 需要将 LTL 公式转换为非确定 Büchi 自动机. 相比其他求解方法, 本文提出的方法可以直接利用现有高效规划器, 并且获得的合成策略更为紧凑, 具备一定的实际应用价值.

本文第 1 节简要介绍 LTL、LTL 合成以及非确定规划的基本概念. 第 2 节详细给出从 LTL 合成到非确定规划模型的具体转换方法, 将 LTL 合成过程变为规划求解过程, 这也是本文的主要贡献. 第 3 节通过实验对比, 证明本文所提方法的可行性和有效性. 最后总结全文, 并对未来可能的研究方向进行初步探讨.

1 技术背景

1.1 线性时序逻辑LTL

LTL 是一种模态逻辑, 在命题逻辑的基础上引入时序操作符 *next*(○)和 *until*(**U**), 用于描述线性时间性质. 令 P 表示原子命题集合, LTL 公式ϕ的语法定义如下(其中, $p\in P$):

$$\phi::=p|\neg\phi|\phi_1\wedge\phi_2|\bigcirc\phi|\phi_1\mathbf{U}\phi_2.$$

LTL 的语义模型定义为无限状态序列$\sigma=s_0,s_1,\ldots$, 其中, 任意 s_i 为 P 的一个子集. true, false 和其他命题逻辑操作符$\vee$, $\rightarrow$, $\leftrightarrow$可以由ϕ中定义的基本操作符导出. LTL 公式的可满足性定义如下:

$\sigma,i\models p$ iff $p\in P$;

$\sigma,i\models\neg\phi$ iff not $\sigma,i\models\phi$;

$\sigma,i\models\phi_1\wedge\phi_2$ iff $\sigma,i\models\phi_1$ and $\sigma,i\models\phi_2$;

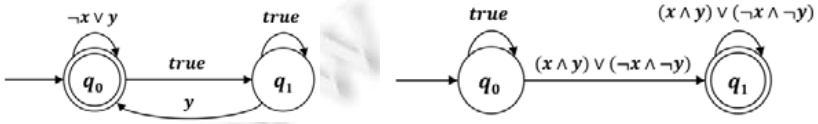
$\sigma,i\models\phi_1\mathbf{U}\phi_2$ iff $\exists j\geqslant i$ such that $\sigma,j\models\phi_2$, and $\sigma,k\models\phi_1$ for each $i\leqslant k\leqslant j-1$.

我们称σ满足ϕ, 表示为$\sigma\models\phi$, 当且仅当$\sigma,0\models\phi$. 简单地说, ○ϕ描述ϕ将在下一状态成立; $\phi_1\mathbf{U}\phi_2$ 的含义是直到ϕ_2 成立之前, ϕ_1 一直成立. 其他时序操作符, 如 *eventually*(◊)和 *always*(□)定义为: $\Diamond\phi\triangleq\text{true}\mathbf{U}\phi$, $\Box\phi\triangleq\neg\Diamond\neg\phi$.

定义 1. 非确定 Büchi 自动机(non-deterministic Büchi automata, NBA)是一个五元组: $B=(Q,\Sigma,\delta,q_0,F)$, 其中, Q 是自动机的状态集合, Σ是字母表, $\delta\subseteq Q\times\Sigma\times Q$ 是迁移关系, q_0 是初始状态, $F\subseteq Q$ 是接受状态集合.

P 的字集合表示为 $Lits(P)=P\cup\{\neg p|p\in P\}$. 一般地, 可令$\Sigma=2^{Lits(P)}$. B 路径$\gamma=q_0,q_1,\ldots$是一个无限状态序列, 定义在字序列$\sigma=s_0,s_1,\ldots$之上, 满足对于任意 $i\geqslant 0$, 有$(q_i,s_i,q_{i+1})\in\delta$. 如果一条路径中出现无限多个接受状态, 则称该路径是可接受的. 如果存在 B 的一条可接受路径定义在σ之上, 则称 B 接受σ. 给定 LTL 公式ϕ, 我们可以构造NBA B_ϕ, 使得B_ϕ接受σ当且仅当$\sigma\models\phi$. 在最坏情况下, 构造B_ϕ的时间复杂度随着ϕ大小的增加(公式的大小由其包含的操作符和变量的数量决定)呈指数级增长[23].

例 1: 图 1 展示了 LTL 公式$\Box(x\leftrightarrow\Diamond y)$和$\Diamond\Box(x\leftrightarrow y)$对应的 NBA. 自动机的状态表示为圆(双环圆是接受状态), 迁移表示为箭头. 需要注意的是, 这里用"$\vee$"操作符简化了迁移的描述. 例如, 迁移$(q_0,\neg x\vee y,q_0)$可以看作由两条迁移$(q_0,\neg x,q_0)$和(q_0,y,q_0)组合而成.



(a) $\Box(x\leftrightarrow\Diamond y)$对应的 Büchi 自动机 (b) $\Diamond\Box(x\leftrightarrow y)$对应的 Büchi 自动机

图 1 NBA 示例

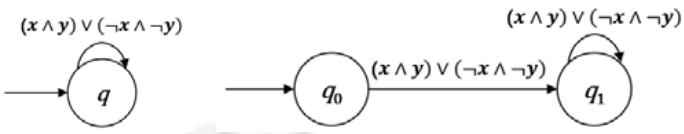
1.2 LTL合成

LTL 合成的定义最早由 Pnueli 和 Rosner 给出[2], 他们将 LTL 合成描述为二人博弈, 博弈的双方分别为环境和控制器. 博弈过程包括一个无限的回合序列, 每个回合首先由环境执行一个动作, 然后智能体为控制器选择另一个动作应对(控制器也可先于环境执行). 动作的任务是对一些变量赋值. 无论环境如何选择动作, 如果环境和控制器交互产生的状态序列满足给定的 LTL 公式, 则称控制器存在获胜策略. LTL 合成的形式化定义如下:

定义 2. LTL 合成问题是一个三元组$\mathcal{L}=(X,Y,\phi)$, 其中, $X=\{x_1,\dots,x_n\}$为环境变量集合, $Y=\{y_1,\dots,y_n\}$为控制器变量集合, 且 X 和 Y 不相交(任意变量 $x\in X\cup Y$ 为命题变量), ϕ为 LTL 公式. 如果存在一个函数 $f:(2^X)^*\to 2^Y$, 对于任意 X 的子集序列 $X_1,X_2,\dots$和$\sigma=(X_1\cup f(X_1)),(X_2\cup f(X_2)),\dots$, 使得$\sigma\models\phi$, 且$\phi$中所含变量属于 $X\cup Y$, 则称ϕ是可实现的. 其中, X_i 为成立的环境变量集合, $X\backslash X_i$ 为不成立的环境变量集合. $f(X_1,X_2,\dots)$和 $Y\backslash f(X_1,X_2,\dots)$的含义类似.

直观地说, LTL 合成要求无论环境如何选择(序列 $X_1,X_2,\dots$), 控制器都能找到相应的策略(f), 保证此博弈模型满足ϕ. 实际上, LTL 合成问题就对应求解函数 f. 若存在 f, 称 LTL 合成问题为可实现的; 否则, 称其为不可实现的.

例 2: 令$\mathcal{L}_1=(\{x\},\{y\},\square(x\to\Diamond y))$, $\mathcal{L}_2=(\{x\},\{y\},\Diamond\square(x\leftrightarrow y))$为 LTL 合成问题. 一般情况下, 可以用自动机形式描述合成策略. 图 2(a)和图 2(b)分别给出了$\mathcal{L}_1$与$\mathcal{L}_2$的一种合成策略. 容易验证, 图 2 所示的任意无限序列均满足给定公式.



(a) $\mathcal{L}_1$ 的合成策略 (b) $\mathcal{L}_2$ 的合成策略

图 2 合成策略示例

1.3 非确定规划

定义 3. 完全可观测的非确定规划(fully observable non-deterministic planning, FOND)问题是一个四元组: $\mathcal{P}=(P,I,G,A)$, 其中, P 是事实(fluent)集合, $I\subseteq P$ 是初始状态, $G\subseteq P$ 是目标条件, A 是动作集合. 完全可观测的含义是所有事实的真值在每个状态下都是可以确定的, 凡是未出现在状态中的事实均不成立.

每个动作 $a\in A$ 包含两个元素$(pre(a),eff(a))$, 其中, $pre(a)\subseteq Lists(P)$是 a 的前置条件, $eff(a)$是 a 的效果. 需要强调的是: $eff(a)$是非确定的, 即可能存在多个互斥的子效果. 每个子效果 $e\in eff(a)$是一个条件效果集合, 形如 $C\triangleright l$, 其中, $C\subseteq Lists(P)$并且 $l\in Lists(P)$. 给定状态 $s\subseteq P$ 和事实 $p\in P$, s 满足 p 当且仅当 $p\in s$, s 满足$\neg p$ 当且仅当 $p\notin s$. 此外, s 满足字集合 L 当且仅当 s 满足任意 $l\in L$. 下面将举例说明非确定效果的含义. 在经典的规划问题 Tireworld 中, 汽车在两点之间移动采用一个动作模拟, 该动作包含两个非确定子效果: 一个是到达目的地后轮胎气充足, 另一个是到达目的地后轮胎亏气. 在求解时, 需要同时考虑两个子效果出现的情况, 即: 若轮胎气充足时继续行驶, 若轮胎亏气时则补气后再行驶.

如果状态 s 满足 $pre(a)$, 则称动作 a 在 s 下是可执行的. s'是 a 在 s 下执行的结果, 且 $s'=s\backslash\{p|(C\triangleright\neg p)\in e,s$ 满足 $C\}\cup\{p|(C\triangleright p)\in e,s$ 满足 $C\}$. 策略$\mathbb{P}$是将状态映射到动作的部分函数, 满足: 如果$\mathbb{P}(s)=a$, 则 a 在 s 下是可执行的. $\mathbb{P}$在 s 下的执行π是一个有限或者无限序列 $s_0,a_0,s_1,a_1,\dots$, 其中, $s_0=s$, 每个“状态-动作-状态”子序列 s,a,s'满足$\mathbb{P}(s)=a$, 且 s'是 a 在 s 下的执行结果. 若去除执行的所有动作, 则可以获得一个状态序列, 称执行产生该状态序列. 同理, 若去除执行的所有状态, 称执行产生动作序列.

如果有限执行π的最终状态 s 满足字集合 L, 则称π达到 L. 如果存在无限执行π的某个状态 s 在π中出现无限多次, 且 s 满足 L, 则称π达到 L. 无限执行π是公平的(fair)当且仅当如果 s, a 在π中出现无限多次, 则 s,a,s'也在π中出现无限多次, 其中, s'是 a 在 s 下的任意执行结果. 在这种定义下, 意味着有限执行是公平的. 策略$\mathbb{P}$是$\mathcal{P}$的一个强循环解(strong-cyclic plan)当且仅当$\mathbb{P}$在 I 下的每个公平执行均达到 G. FOND 问题还有强解(strong plan)和弱循环解(weak-cyclic plan)的定义, 但本文并未涉及[24].

2 LTL 合成的 FOND 模型

LTL 合成是否可实现, 与 FOND 问题是否存在强循环解有着紧密的联系. 本节主要介绍一种转换方法,

将 LTL 合成的二人博弈定义转换为 FOND 模型.

2.1 从LTL合成到非确定规划的转换

给定一个 LTL 合成问题$\mathcal{L}=(X,Y,\phi)$. 首先对$\mathcal{L}$进行预处理, 去除 X 和 Y 中不在ϕ出现的变量. 这样做可以简化问题, 因为这些变量对于合成ϕ是无关的, 其真值的变化不影响ϕ的任意模型. 接着, 将$\mathcal{L}$转换为标准的 FOND 问题模型$\mathcal{P}=(P,I,G,A)$. 最后, 直接利用 FOND 规划器求解$\mathcal{P}$. 所得的强循环解需要进一步变形才能得到$\mathcal{L}$的合成策略, 但是这个步骤比较直观, 可以很容易地实现.

这里重点阐述$\mathcal{L}$到$\mathcal{P}$的转换过程, 我们将其称为$\mathcal{L}2\mathcal{P}$. 在此过程中, ϕ需要被转换为等价的 NBA $B_\phi=(Q,\Sigma,\delta,q_0,F)$. 结合$\mathcal{L}$和 B_ϕ, 通过模拟环境和控制器的博弈过程以及 B_ϕ的迁移约束, 进而得到 FOND 模型. B_ϕ的每个状态由与之同名的事实变量模拟, B_ϕ的每个迁移由一个规划动作模拟. 在规划过程中, 规划状态 s 需满足: 事实变量 q(对应自动机状态 q)成立当且仅当存在B_ϕ的某条路径定义在σ之上, 且路径的最后一个状态为 q, 其中, σ为从初始状态到 s 的规划搜索状态序列. 博弈的过程在规划模型中主要由两个交替变换的模式来模拟, 即“环境模式”和“自动机模式”. 前者模拟环境的行为, 该模式的所有行为是非确定的且无法预知和控制; 后者通过同步自动机的迁移模拟控制器的行为, 该模式下的每个行为对应一个迁移. 此外, 还存在另外一个“记录模式”, 目的是记录当前激活的自动机状态, 下文将会具体解释.

下面将详细说明转换的技术细节. 转换后 P 的定义如下.

- 事实集合 P

$$P=\{prev\_q,q|q\in Q\}\cup\{env\_mode,aut\_mode,record\_mode\}\cup\{turn_i|1\leqslant i\leqslant|X|\}\cup\{v_x,v_{\neg x}|x\in X\cup Y\}\cup\{goal\}.$$

自动机的状态是否成立, 由其前序状态以及两状态之间的迁移条件决定. 因此, 针对每个自动机状态 $q\in Q$, 引入两个事实变量 $prev\_q$ 和 q 与之对应, 分别表示自动机状态 q 在上一状态是否成立以及在当前状态是否成立. 事实变量 env_mode,aut_mode 和 $record\_mode$ 用于标记规划所处的模式: 分别表示环境模式、自动机模式和记录模式. 记录模式主要为后续规划记录此时成立或激活的自动机状态. 3 个模式依次交替变换. 事实变量 $turn_i$ 的作用将在介绍动作集合时给出. 对于每个变量 $x\in X\cup Y$, 采用两个事实变量 $v_x, v_{\neg x}$ 分别显式描述将 x 赋值为真或假(值得注意的是, 根据等价关系 $x\equiv\neg\neg x$, 可以得出 $v_x\equiv\neg v_{\neg x}$). $goal$ 用于标记规划目标.

- 动作集合 A

$$A=\{assign_x|x\in X\}\cup\{trans_t|t\in\delta\}\cup\{record_q|q\in Q\}.$$

(1) 环境模式下可执行的动作集合为$\{assign_x|x\in X\}$. 由于环境的行为是智能体不可控的, 环境有可能任意改变变量的值, 因此采用一系列非确定动作来模拟这种不确定性. 假设 $X=\{x_1,x_2,\ldots,x_{|X|}\}$, 其中每个变量 $x_i\in X$ 对应一个非确定动作 $assign_{x_i}$, 定义如下:

$$pre(assign_{x_i})=\{env\_mode,turn_i\}$$

$$eff(assign_{x_i})=oneof(\{v_{x_i},\neg v_{\neg x_i}\},\{\neg v_{x_i},v_{\neg x_i}\})\cup\begin{cases}\{turn_{i+1},\neg turn_i\}, & \text{if } i<|X|\\ \{aut\_mode,\neg env\_mode,\neg turn_i\}, & \text{if } i=|X|\end{cases}$$

所有变量按照一定顺序依次被赋值. $turn_i$ 在 $assign_{x_i}$ 的前置条件中出现, 说明 $turn_i$ 成立时 $assign_{x_i}$ 才能执行. $assign_{x_i}$ 的效果包含一个非确定效果, 用关键字 $oneof$ 表示, 模拟将 x_i 赋值为真或假. 当最后一个变量 $x_{|X|}$ 执行结束后, 从环境模式切换为自动机模式.

还有其他环境模式的动作定义方法, 只需定义一个动作 $assign_X$, 其效果中包含 $2^{|X|}$个非确定子效果. 然而, 这种定义可能会导致效果数量爆炸, 只适用于$|X|$较小的情况.

$$pre(assign_X)=\{env\_mode\}$$

$$eff(assign_X)=\{aut\_mode,\neg env\_mode\}\cup oneof(\{v_{x_1},\neg v_{\neg x_1},v_{x_2},\neg v_{\neg x_2},\ldots,v_{x_{|X|}},\neg v_{\neg x_{|X|}}\},\{v_{x_1},\neg v_{\neg x_1},v_{x_2},\neg v_{\neg x_2},\ldots,\neg v_{x_{|X|}},v_{\neg x_{|X|}}\},\ldots,\{\neg v_{x_1},v_{\neg x_1},\neg v_{x_2},v_{\neg x_2},\ldots,\neg v_{x_{|X|}},v_{\neg x_{|X|}}\})$$

(2) 自动机模式下的动作模拟 Y 中变量赋值以及自动机迁移. 每次更新自动机的迁移, 可以看作智能体观察 X 中变量赋值后所做出的反应(对 Y 中变量进行赋值). 智能体决定选择执行哪个迁移动作, 并确保不违反激

活迁移的条件. 针对每个迁移 $t=(q_i,guard(t),q_j)\in\delta$, $guard(t)\in 2^{Lits(P)}$, 定义一个规划动作 $trans_t$:

$$pre(trans_t)=\{aut\_mode, prev\_q_i\}\cup\{\neg v_{\neg l}\mid l\in guard(t)\}$$

$$eff(trans_t)=\{record\_mode,\neg aut\_mode\}\cup\{q_j,\neg prev\_q_i\}\cup\{v_l\mid l\in guard(t)\}$$

当前序状态下 q_i 成立, 即 $prev\_q_i$ 成立, 且 $trans_t$ 的前置条件不违反 t 的迁移条件$\{\neg v_{\neg l}|l\in guard(t)\}$, $trans_t$ 才存在执行的可能性. $trans_t$ 的效果包括设置 q_j 为真, 并设置 $guard(t)$的所有事实变量条件成立. 注意, 在每个状态下只保持一个活动的自动机状态. 也就是说, 只模拟记录自动机的一条路径. 在此模式下执行一个动作后, 切换至记录模式.

(3) 每个自动机状态 $q\in Q$ 对应记录模式下的一个动作 $record_q$, 其前置条件要求 q 成立. 执行某一 $record_q$ 动作后, 重新切换至环境模式. $record_q$ 的效果重置所有 $x\in X\cup Y$, 并为后续规划记录 q 为活动状态($prev\_q$ 置为真). 当 q 为接受状态时, 此时效果包含两个不确定子效果, 其中: 一个子效果包含目标 $goal$ 和 $prev\_q$; 另一个子效果与 q 为非接受状态时相同, 即 $prev\_q$. 将 $goal$ 加入非确定效果的做法, 使得求解时必须多次执行 $record_q$ 才能使得经过所有子效果的路径到达目标, 从而模拟 B_ϕ的无限可接受路径:

$$pre(record_q)=\{record\_mode,q\}$$

$$eff(record_q)=\{env\_mode,\neg record\_mode,turn_1,\neg q\}\cup\{\neg v_x,\neg v_{\neg x}\mid x\in X\cup Y\}\cup\begin{cases}prev\_q, & \text{if } q\notin F\\ oneof(\{prev\_q\},\{goal,prev\_q\}), & \text{if } q\in F\end{cases}$$

初始状态 I 和目标 G: 初始状态设置模式为环境模式, 且令 $turn_1$ 成立, 并令 B_ϕ的初始状态成立, 即 $I=\{env\_mode,turn_1,prev\_q_0\}$. 将 $goal$ 设为规划目标, 即 $G=\{goal\}$.

例 3: 令$\mathcal{L}_1=(\{x\},\{y\},\square(x\rightarrow\Diamond y))$为例 2 的 LTL 合成问题, $\square(x\rightarrow\Diamond y)$的 NBA 如图 1(a)所示. 根据前述转换方法, 事实集合 $P=\{prev\_q_0,q_0,prev\_q_1,q_1\}\cup\{env\_mode,aut\_mode,record\_mode\}\cup\{turn_1\}\cup\{v_x,v_{\neg x},v_{xy},v_{\neg y}\}\cup\{goal\}$. 动作集合定义如下:

环境模式只包含一个动作 $assign_x$:

$$pre(assign_x)=\{env\_mode,turn_i\}$$

$$eff(assign_x)=\{aut\_mode,\neg env\_mode,\neg turn_1\}\cup oneof(\{v_{x_i},\neg v_{\neg x_i}\},\{\neg v_{x_i},v_{\neg x_i}\})$$

自动机模式共有 5 个迁移, 分别为 $t_1=(q_0,\neg x,q_0)$, $t_2=(q_0,y,q_0)$, $t_3=(q_0,\text{true},q_1)$, $t_4=(q_1,y,q_0)$, $t_5=(q_1,\text{true},q_1)$, 对应以下 5 个动作:

$$pre(trans_{t_1})=\{aut\_mode,prev\_q_0,\neg v_x\},\quad eff(trans_{t_1})=\{record\_mode,\neg aut\_mode,q_0,\neg prev\_q_0,v_{\neg x}\};$$

$$pre(trans_{t_2})=\{aut\_mode,prev\_q_0,\neg v_{\neg y}\},\quad eff(trans_{t_2})=\{record\_mode,\neg aut\_mode,q_0,\neg prev\_q_0,v_y\};$$

$$pre(trans_{t_3})=\{aut\_mode,prev\_q_0\},\quad eff(trans_{t_3})=\{record\_mode,\neg aut\_mode,q_1,\neg prev\_q_0\};$$

$$pre(trans_{t_4})=\{aut\_mode,prev\_q_1,\neg v_{\neg y}\},\quad eff(trans_{t_4})=\{record\_mode,\neg aut\_mode,q_0,\neg prev\_q_1,v_y\};$$

$$pre(trans_{t_5})=\{aut\_mode,prev\_q_1\},\quad eff(trans_{t_5})=\{record\_mode,\neg aut\_mode,q_1,\neg prev\_q_1\}.$$

记录模式有两个动作 $record_{q_0}$ 和 $record_{q_1}$:

$$pre(record_{q_0})=\{record\_mode,q_0\},eff(record_{q_0})=\{aut\_mode,\neg record\_mode,\neg q_0,oneof(\{prev\_q_0\},\{goal,prev\_q_0\}),\neg v_x,\neg v_{\neg x},\neg v_y,\neg v_{\neg y}\};$$

$$pre(record_{q_1})=\{record\_mode,q_1\},eff(record_{q_1})=\{aut\_mode,\neg record\_mode,\neg q_1,prev\_q_0,\neg v_x,\neg v_{\neg x},\neg v_y,\neg v_{\neg y}\}.$$

根据上述 FOND 模型获得的解, 可以表示为图 3 所示的图形. 这里, 有效的迁移为 $assign_x$ 和 $trans_{t_1}$. 在进行合成策略转换时, 虽然 $assign_x$ 和 $trans_{t_1}$ 顺序执行, 但实际上二者应为同时发生的动作, 因为二人博弈的定义规定环境变量和控制变量的变化是同步的.

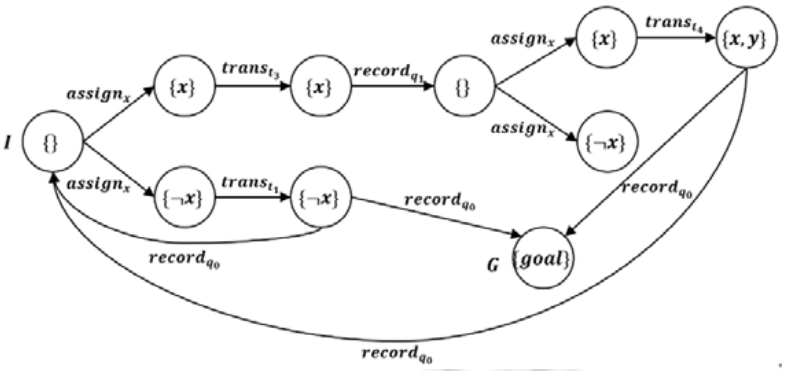


图 3 $\mathcal{L}_1$ 转换为 FOND 模型后的规划解

2.2 理论证明

定理 1. 若$|X|+|Y|<<|B_\phi|$, 则$\mathcal{L}2\mathcal{P}$转换的时间复杂度与$|B_\phi|$线性相关.

证明: 按照传统的定义, $|B_\phi|=|Q|+|\delta|$. 令 LTL 合成问题$\mathcal{L}=(X,Y,\phi)$, $\mathcal{L}$经过转换得到的 FOND 模型为$\mathcal{P}=(P,I,G,A)$. $\mathcal{L}2\mathcal{P}$转换主要包括事实集合转换、动作集合转换以及初始状态和目标转换. 假设转换每个变量的复杂度为常量 1, 各转换的时间复杂度分析如下.

- 事实集合转换复杂度: $|\{prev\_q,q|q\in Q\}|+|\{env\_mode,aut\_mode,record\_mode\}|+|\{turn_i|1\leqslant i\leqslant|X|\}|+|\{v_x, v_{\neg x}|x\in X\cup Y\}|+|\{goal\}|=2|Q|+3|X|+2|Y|+4$;
- 动作集合转换复杂度:第 1 类环境模式动作转换的时间复杂度为$|\{assign_x|x\in X\}|=9|X|$, 第 2 类自动机模式动作转换的时间复杂度为$\{trans_t|t\in\delta\}=(6+2|X|+2|Y|)|\delta|$, 第 3 类记录模式动作转换的时间复杂度为$|\{record_q|q\in Q\}|=(9+|X|+|Y|)|Q|$.
- 初始状态和目标转换复杂度: 该转换的时间复杂度为常量.

已知$|X|+|Y|<<|B_\phi|$, 因此定理的结论成立. □

定理 2(正确性). 若$\mathcal{P}$是 LTL 合成问题$\mathcal{L}$经$\mathcal{L}2\mathcal{P}$转换得到的 FOND 模型, 则$\mathcal{P}$的强循环解对应$\mathcal{L}$的合成策略.

证明: 假设$\mathcal{P}=(P,I,G,A)$, $\mathcal{L}=(X,Y,\phi)$, ϕ的 NBA 为 B_ϕ. 首先证明$\mathcal{P}$的强循环解$\mathbb{P}$的任意执行产生的序列满足ϕ. 容易证明, $\mathbb{P}$的任意执行产生的序列一定是无限的. 采用反证法, 假设$\mathbb{P}$的某一执行是有限的. 根据 $G=\{goal\}$, 可知存在动作 $record_q$ 的执行, 使得其中一个非确定子效果 $goal$ 成立. 然而, 仍然存在另一个不含有 $goal$ 的非确定子效果无法到达目标. 因此, $\mathbb{P}$的任意执行一定是无限的, 且存在某 $record_q$ 的无限多次执行. 一般地, 令执行产生的序列为σ. 再根据自动机模式的动作定义, 每次动作执行都符合 B_ϕ的某迁移, 由此可以找到定义在σ之上的一条可接受路径. 因此, $\mathcal{P}$的强循环解$\mathbb{P}$的任意执行产生的序列满足ϕ. 我们可以对$\mathbb{P}$稍加改动, 便可得到$\mathcal{L}$的合成策略, 包括: (1) 去除所有 $record_q$ 动作; (2) 去除 $X\cup Y$ 之外的所有辅助变量; (3) 假设 $X=\{x_1,x_2,\ldots,x_{|X|}\}$, 将所有动作序列 $assign_{x_1},assign_{x_2},\ldots,assign_{x_{|X|}},trans_t$ 的执行统一为一个动作, 该动作的前置条件为 $assign_{x_1}$ 的前置条件, 执行效果为的 $trans_t$ 执行结果, 没有中间执行过程.

定理结论得证. □

定理 3(完备性). 若$\mathcal{P}$是 LTL 合成问题$\mathcal{L}$经过$\mathcal{L}2\mathcal{P}$转换得到的 FOND 模型, 则$\mathcal{L}$的任意合成策略均可由$\mathcal{P}$的强循环解得到.

证明: 假设$\mathcal{P}=(P,I,G,A)$, $\mathcal{L}=(X,Y,\phi)$, $X=\{x_1,x_2,\ldots,x_{|X|}\}$, ϕ的 NBA 为 $B_\phi=(Q,2^{Lits(X\cup Y)},\delta,q_0,F)$. 令 f 是$\mathcal{L}$的合成策略, 对于任意序列$\sigma=(X_1\cup f(X_1)),(X_1\cup f(X_1,X_2)),\ldots$, 其中, $X_i\subseteq X$, $f(X_1,X_2,\ldots,X_i)\subseteq Y$. 已知$\sigma\models\phi$, 则存在 B_ϕ的一条路径 $q_0,q_1,\ldots$, 使得任意$(q_i,(X_i\cup f(X_1,X_2,\ldots,X_i)),q_{i+1})\in\delta$. 类似定理 2 中的策略修改方法.

(1) 将 X_i 和 $f(X_1,X_2,\ldots,X_i)$分解为一系列动作 $\tau_i=assign_{x_1},assign_{x_2},\ldots,assign_{x_{|X|}},trans_t$ 的最终效果, 其中,

$t=(q_i,(X_i\cup f(X_1,X_2,\ldots,X_i)),q_{i+1})$. 对于 $x_i\in X_i$, 取 $assign_{x_i}$ 的非确定子效果 $\{v_{x_i},\neg v_{\neg x_i}\}$; 而对于 $x_i\notin X_i$, 取 $assign_{x_i}$ 的非确定子效果 $\{\neg v_{x_i},v_{\neg x_i}\}$;

(2) 在 τ_i 后增加动作 $record_{q_{i+1}}$, 因此, $\tau_0,record_{q_1},\tau_1,record_{q_2},\ldots$ 可看作某执行产生的动作序列. 我们可将所有 σ 对应的动作序列组合得到 $\mathbb{P}$. 根据 X_i 的任意性($X_i\in 2^X$), $\mathbb{P}$ 的所有执行产生的状态序列均满足 ϕ, 因此 $\mathbb{P}$ 是 $\mathcal{P}$ 的强循环解.

定理结论得证. □

2.3 方法对比

本节提出的 $\mathcal{L}2\mathcal{P}$ 转换方法改进了 Camacho 等人提出的转换方法[19], 记为 $\mathcal{L}2\mathcal{P}(\mathcal{C})$. 下面将对 $\mathcal{L}2\mathcal{P}(\mathcal{C})$ 作简要介绍, 并分析二者异同. $\mathcal{L}2\mathcal{P}(\mathcal{C})$ 转换后的规划模型定义如下.

- 事实集合 P

$$P=\{q,q^s,q^t,q^{st}|q\in Q\}\cup\{env\_mode,aut\_mode,can\_switch,can\_accept\}\cup\{turn_i|1\leqslant i\leqslant|X|\}\cup\{v_x,v_{\neg x}|x\in X\cup Y\}\cup\{goal\}.$$

针对每个自动机状态 q, 有 4 个事实变量与之对应, 主要目的在于, 需要同时记录 NBA 的多条路径. q 与 q^s, q^t 与 q^{st} 的关系类似于 $\mathcal{L}2\mathcal{P}$ 中的 $prev\_q$ 与 q. q 和 q^t 记录前序自动机状态, q^s 和 q^{st} 记录当前自动机状态. $\mathcal{L}2\mathcal{P}(\mathcal{C})$ 的模式只包含环境模式和自动机模式. can_switch 用于表示 X 中的变量赋值结束, can_accept 用于标记存在一条路径到达接受状态.

- 动作集合 A

$$A=\{assign_x|x\in X\}\cup\{trans_t|t\in\delta\}\cup\{switch2aut,switch2env,accept\}.$$

(1) 环境模式下的动作与 $\mathcal{L}2\mathcal{P}$ 基本相同, 最后一个变量 $x_{|X|}$ 赋值后执行 $switch2aut$ 动作, 其效果将 q 和 q^t 分别变为 q^s 和 q^{st}. 若自动机的节点数目过多, $switch2aut$ 的效果将会出现过多的情况, 从而使规划器无法处理:

$$pre(assign_{x_{|X|}})=\{env\_mode,turn_{|X|}\}$$

$$eff(assign_{x_{|X|}})=oneof(\{v_{x_{|X|}},\neg v_{\neg x_{|X|}}\},\{\neg v_{x_{|X|}},v_{\neg x_{|X|}}\})\cup\{can\_switch,\neg turn_{|X|}\}$$

$$pre(switch2aut)=\{env\_mode,can\_switch\}$$

$$eff(switch2aut)=\{aut\_mode,\neg env\_mode,\neg can\_switch\}\cup\{q\rhd\{q^s,\neg q\},q^t\rhd\{q^{st},\neg q^t\}|q\in Q\}$$

(2) 虽然仍是每个迁移对应一个动作, 自动机模式下的动作与 $\mathcal{L}2\mathcal{P}$ 区别很大. 当迁移的后继状态为接受状态时, 将 can_accept 设置为真; 否则, 如果前序状态 q_i^{st} 成立, 则令 q_j^t 成立:

$$pre(trans_t)=\{aut\_mode,q_i^s\}\cup\{\neg v_{\neg l}|l\in guard(t)\}$$

$$eff(trans_t)=\{q_j\}\cup\{v_l|l\in guard(t)\}\cup\begin{cases}\{q_i^{st}\rhd q_j^t\}, & \text{if } q_j\notin F\\ \{can\_accept\}, & \text{if } q_j\in F\end{cases}$$

从自动机模式切换到环境模式, 可以通过两个动作 $switch2env$ 和 $accept$. $switch2env$ 将所有条件(除了自动机状态变量)置为初值. $accept$ 的效果有对自动机状态变量的处理, 并有非确定子效果 $goal$, 使得可以找到访问无限多次接受状态的无限执行. $accept$ 也存在效果过多的潜在问题. 更为详细的解释可参考文献[19]:

$$pre(switch2env)=\{aut\_mode\}$$

$$eff(switch2env)=\{env\_mode,\neg aut\_mode\}\cup\{turn_1\}\cup\{\neg q^s,\neg q^{st}|q\in Q\}\cup\{\neg v_x,\neg v_{\neg x}|x\in X\cup Y\}$$

$$pre(accept)=\{aut\_mode,can\_accept\}$$

$$eff(accept)=oneof(\{goal\},\{turn_1\}\cup\{env\_mode,\neg aut\_mode,\neg can\_accept\}\cup\{q^s\rhd\neg q^s,q^{st}\rhd\neg q^{st}|q\in Q\})\cup$$
$$\{q\wedge q^t\rhd\{\neg q,\neg q^t\}|q\in(Q\setminus F)\}\cup\{q\wedge\neg q^t\rhd q^t|q\in Q\}\cup\{\neg v_x,\neg v_{\neg x}|x\in X\cup Y\}$$

由于 $\mathcal{L}2\mathcal{P}(\mathcal{C})$ 没有记录模式, 其转换工作量略少于 $\mathcal{L}2\mathcal{P}$. 然而在下一节我们将会看到: 由于 $switch2aut$ 和 $accept$ 动作的效果过多, 导致规划器无法对其解析; 而 $\mathcal{L}2\mathcal{P}$ 把这种复杂性简化分布到多个动作中, 不会产生上述情况. 此外, $\mathcal{L}2\mathcal{P}(\mathcal{C})$ 只保留部分路径而不是全部路径, 因此该方法不是完备的.

例 4: 根据 $\mathcal{L}2\mathcal{P}(\mathcal{C})$ 转换方法, 将例 2 的 LTL 合成问题 $\mathcal{L}_1$ 转换为 FOND 模型.

- 事实集合:

$$P=\{q_0,q_0^s,q_0^t,q_0^{st},q_1,q_1^s,q_1^t,q_1^{st}\mid q\in Q\}\cup\{env\_mode,aut\_mode,can\_switch,can\_accept\}\cup\{turn_1\}\cup\{v_x,v_{\neg x},v_y,v_{\neg y}\}\cup\{goal\}$$

- 动作集合定义如下:

环境模式有动作 $assign_x$ 和动作 $switch2aut$:

$$pre(assign_x)=\{env\_mode,turn_1\},\quad eff(assign_x)=\{can\_switch,\neg turn_1\}\cup oneof(\{v_{x_i},\neg v_{\neg x_i}\},\{\neg v_{x_i},v_{\neg x_i}\});$$

$$pre(switch2aut)=\{env\_mode,can\_switch\},\quad eff(switch2aut)=\{aut\_mode,\neg env\_mode,\neg can\_switch\}\cup\{q_0\rhd\{q_0^s,\neg q_0\},q_0^t\rhd\{q_0^{st},\neg q_0^t\},q_1\rhd\{q_1^s,\neg q_1\},q_1^t\rhd\{q_1^{st},\neg q_1^t\}\}.$$

自动机模式共有 7 个迁移, 包括 5 个迁移 $t_1=(q_0,\neg x,q_0)$, $t_2=(q_0,y,q_0)$, $t_3=(q_0,\text{true},q_1)$, $t_4=(q_1,y,q_0)$, $t_5=(q_1,\text{true},q_1)$ 对应的动作以及 $switch2env$ 和 $accept$:

$$pre(trans_{t_1})=\{aut\_mode,q_0^s,\neg v_x\},\quad eff(trans_{t_1})=\{q_0,can\_accept,v_{\neg x}\};$$

$$pre(trans_{t_2})=\{aut\_mode,q_0^s,\neg v_{\neg y}\},\quad eff(trans_{t_2})=\{q_0,can\_accept,v_y\};$$

$$pre(trans_{t_3})=\{aut\_mode,q_0^s\},\quad eff(trans_{t_3})=\{q_1,q_0^{st}\rhd q_1^t\};$$

$$pre(trans_{t_4})=\{aut\_mode,q_1^s,\neg v_{\neg y}\},\quad eff(trans_{t_4})=\{q_0,can\_accept,v_y\};$$

$$pre(trans_{t_5})=\{aut\_mode,q_1^s\},\quad eff(trans_{t_5})=\{q_1,q_1^{st}\rhd q_1^t\};$$

$$pre(switch2env)=\{aut\_mode\},\quad eff(switch2env)=\{env\_mode,\neg aut\_mode,can\_accept,turn_1,\neg q_0^s,\neg q_0^{st},\neg q_1^s,\neg q_0^{st},\neg v_x,\neg v_{\neg x},\neg v_y,\neg v_{\neg y}\};$$

$$pre(accept)=\{aut\_mode,can\_accept\},\quad eff(accept)=oneof(\{goal\},\{env\_mode,\neg aut\_mode,\neg can\_accept,turn_1,\neg q_0^s\rhd\neg q_0^s,\neg q_1^s\rhd\neg q_1^s,\neg q_0^{st}\rhd\neg q_0^{st},\neg q_1^{st}\rhd\neg q_1^{st},q_0\wedge q_0^t\rhd\{\neg q_0,\neg q_0^t\},q_0\wedge\neg q_0^t\rhd q_0^t,q_1\wedge\neg q_1^t\rhd q_1^t,\neg v_x,\neg v_{\neg x},\neg v_y,\neg v_{\neg y}\}).$$

假设每个变量的空间复杂度为 1, 下面将分析$\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$中某些关键动作的空间复杂度. 二者最重要的区别在于前者的记录模式以及后者的 $switch2env$ 和 $accept$ 动作. $\mathcal{L}2\mathcal{P}$中每个动作的空间复杂度为 8+2(|X|+|Y|), 而$\mathcal{L}2\mathcal{P}(\mathcal{C})$中 $switch2env$ 动作的空间复杂度为 5+2|Q|+2(|X|+|Y|), $accept$ 动作的空间复杂度为 7+11|Q|+2(|X|+|Y|). 从单个动作的角度看, $\mathcal{L}2\mathcal{P}(\mathcal{C})$的两个动作空间复杂度与自动机规模相关. 尤其在自动机规模较大的情况下, 将会导致动作的复杂度过高; 相反, $\mathcal{L}2\mathcal{P}$中所有动作的空间复杂度与自动机规模无关.

3 实验与分析

本节主要通过实验比较本文所提方法与其他方法的优劣, 选取的基准测试集来自最近一届国际合成竞赛 SYNTCOMP20 (http://www.syntcomp.org/syntcomp-2020-results/). 实验选取的对比合成工具 ltlsynt[7]是参赛工具之一, ltlsynt 将输入的 LTL 公式转换为确定奇偶自动机(deterministic parityautomata), 而后者等价于奇偶博弈(parity game), 并采用 Zielonka 提出的递归算法求解[25]. 对于其他合成工具, 如 Acacia+[4], Lily[5], SynKit[6]等, 它们的求解效率远低于 ltlsynt, 本文仅将 SynKit 作为另一个现有合成工具进行实验对比. SynKit 将 LTL 合成规约为安全博弈问题, 并建立了该问题与多种自动机(universal k-coBüchiWord (UkCW) automata and non-deterministic k-Büchi word (NkBW) automata)之间的关系. 实验采用的 FOND 规划器为非确定规划的著名工具 PRP[26]. LTL 公式转换为 NBA 的过程利用 LTL2BA[27]实现. 我们开发了$\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$转换工具, 并在实验中列出二者的运行效果. 实验运行的平台配置为 Ubuntu 16.04 操作系统, Intel i7 CPU, 8 GB 内存.

主要从两个方面对实验结果进行分析: 一是求解时间, 二是策略的规模. 求解时间说明工具的求解效率, 策略的规模说明解的质量. 策略规模越小, 质量越高, 即说明该策略更加具备实用性. 第 1 个实验采用的测试集均为不可实现(unrealizable)的 LTL 合成问题, 即问题无解. 表 1 列出了相应的实验结果. 每个问题包含多个实例, 复杂度由低到高, 每个实例的求解时间限制在 1 小时以内. 由于问题不可实现, 所以不存在合成策略,

表 1 仅对比求解时间. 列“SYN”表示 ltlsynt 的求解时间. $\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$的运行时间包括 3 个部分: NBA 转换时间(列“NBA”)、FOND 模型转换时间(列“FOND”)和规划器 PRP 求解时间(列“PRP”). $\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$采用同样的 NBA 转换工具, 因而 NBA 的转换时间相同, 所以列“NBA”只出现 1 次, 将$\mathcal{L}2\mathcal{P}(\mathcal{C})$的 NBA 列省略. “–”表示超时或者内存不足. 总体来说, $\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$转换方法能够求解出更多的问题实例, ltlsynt 次之, SynKit 表现最差(无法求解大部分实例). 然而, $\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$方法的瓶颈在于 NBA 转换, 此过程耗费的时间较长. 在极少数情况下, NBA 的转换时间就已经超过了 ltlsynt 的合成时间, 如 full_arbiter_unreal 的 p4 实例和 prioritized_arbiter_unreal 的 p4 实例. 相对而言, FOND 模型转换的时间可以忽略不计. 在求解复杂实例时, 绝大多数 PRP 的求解时间都远远小于 NBA 转换时间, 也小于 ltlsynt 的求解时间. 对比$\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$这两种方法, 二者的 FOND 模型转换时间相近, 而后者的求解效率稍低一些. 原因在于, 关键动作 *switch2aut* 和 *accept* 的效果较为复杂.

表 1 不可实现的基准测试集运行结果

名称		$\mathcal{L}2\mathcal{P}$			$\mathcal{L}2\mathcal{P}(\mathcal{C})$		ltlsynt	SynKit
		NBA	FOND	PRP	FOND	PRP	SYN	SYN
detector_unreal	p1	0.03	0	0	0	0.02	0.03	0.46
	p2	0.04	0	0.04	0	0.06	0.04	103.02
	p3	0.16	0	0.56	0	0.6	2.9	–
	p4	1.06	0	5.56	0	6.52	–	–
	p5	11.09	0.01	54.64	0.01	65.08	–	–
	p6	201.74	0.01	594.82	0.01	687	–	–
full_arbiter_unreal	p1	0.28	0.02	0.5	0.05	5.76	0.09	–
	p2	2.57	0.02	0.5	0.05	5.78	0.14	–
	p3	85.11	0.03	0.54	0.05	5.74	4.61	–
	p4	2 211.81	0.03	0.42	0.05	5.74	211.09	–
	p5	–	–	–	–	–	–	–
	p6	–	–	–	–	–	–	–
prioritized_arbiter_unreal	p1	0.34	0.38	6.14	0.06	18	0.18	–
	p2	1.35	0.38	6.32	0.06	18.66	0.38	–
	p3	46.59	0.38	6.08	0.06	18.88	12.07	–
	p4	1 938.01	0.38	6.06	0.06	19.46	509.21	–
	p5	–	–	–	–	–	–	–
round_robin_arbiter_unreal	p1	0.03	0	0.04	0	0.02	0.04	896.86
	p2	0.05	0	0.06	0	0.02	0.05	–
	p3	0.16	0	0.04	0	0	0.09	–
	p4	1.16	0	0.02	0	0.02	0.4	–
	p5	10.7	0	0.02	0	0.02	3.41	–
	p6	119	0	0.04	0	0	36.57	–
simple_arbiter_unreal	p1	0.02	0	0	0	0.04	0.04	1.16
	p2	0.12	0.01	0.26	0.01	0.72	0.07	1 037.95
	p3	13.2	0.06	5.7	0.11	35.8	4.34	–
	p4	–	–	–	–	–	2 110.43	–
	p5	–	–	–	–	–	–	–

第 2 个实验采用的测试集均为可实现的 LTL 合成问题, 实例的 NBA 规模普遍比第 1 个实验大很多. 表 2 给出了实验结果. 列“$\mathbb{P}$”表示合成策略的大小(仅统计迁移数量), 其中, $\mathcal{L}2\mathcal{P}$和$\mathcal{L}2\mathcal{P}(\mathcal{C})$的合成策略只统计提取有效迁移后的策略规模, 即去除记录模式的动作. 针对每个实例, 所有结果中最紧凑的策略加粗显示. SynKit 的运行结果无论从求解时间和解质量来说都不尽如人意. $\mathcal{L}2\mathcal{P}(\mathcal{C})$的表现较差, 这是由于转换的 NBA 规模较大, 使得 *switch2aut* 和 *accept* 动作的复杂度进一步加大, 进而无法获得大部分实例的解, 绝大部分实例求解时内存不足. 从解的覆盖率来看, $\mathcal{L}2\mathcal{P}$并没有比 ltlsynt 求解出更多的实例. 然而, $\mathcal{L}2\mathcal{P}$能够为所有实例合成更为紧凑的策略, 更具有实际意义. 规划器一般会追求解的质量, 其内部设置的启发式算法在考虑效率的同时, 以追求较优解为更重要的目标. 求解时间的瓶颈依然是 NBA 转换, 如 prioritized_arbiter_enc 的 p4 实例, 由于 NBA 的规模过大, 使得转换后的规划模型也很庞大, 致使规划器无法求解. 在求解效率方面, ltlsynt 稍具优势, 但是与$\mathcal{L}2\mathcal{P}$差距不大.

表 2 可实现的基准测试集运行结果

名称		$\mathcal{L}2\mathcal{P}$				$\mathcal{L}2\mathcal{P}(\mathcal{C})$			ltlsynt		SynKit	
		NBA	FOND	PRP	$\mathbb{P}$	FOND	PRP	$\mathbb{P}$	SYN	$\mathbb{P}$	SYN	$\mathbb{P}$
amba_decomposed_arbiter	p1	0.07	0	0	**23**	0	0	25	0.04	44	289.15	605
	p2	0.63	0.03	0.74	**197**	0.03	–	–	0.17	644	–	–
	p3	27.8	0.13	11.28	**315**	0.15	–	–	3.84	4 084	–	–
	p4	2 994.63	0.4	84.72	**582**	0.48	–	–	319.16	23 044	–	–
	p5	–	–	–	–	–	–	–	–	–	–	–
	p6	–	–	–	–	–	–	–	–	–	–	–
amba_decomposed_lock	p1	0.02	0	0	**29**	0	0.02	**29**	0.04	**29**	0.18	264
	p2	0.05	0	0.1	**69**	0	0.02	**69**	0.09	**69**	0.44	660
	p3	0.44	0	0.32	**125**	0	0.16	**125**	0.39	**125**	1.41	1 576
	p4	20.09	0.01	1.24	**197**	0.01	–	–	19.41	**197**	11.43	4 300
	p5	757.21	0.01	3.48	**285**	0.01	–	–	724.11	**285**	214.14	14 016
	p6	–	–	–	–	–	–	–	–	–	–	–
collector_v1	p1	0.01	0	0	**13**	0	0.12	22	0.05	25	0.17	207
	p2	0.03	0	0	**35**	0	7.86	58	0.03	69	2.88	1 672
	p3	0.04	0	0.02	97	0	28.46	**66**	0.08	193	–	–
	p4	0.1	0.01	0.06	**275**	0.01	–	–	0.14	549	–	–
	p5	0.39	0.02	0.74	**793**	0.03	–	–	0.63	1 585	–	–
	p6	1.78	0.05	6.96	**2 315**	0.1	–	–	4.1	4 629	–	–
full_arbiter_enc	p1	0.05	0	0	**20**	0	0.04	25	0.09	26	0.92	308
	p2	0.96	0.06	0.92	**308**	0.19	–	–	0.65	389	–	–
	p3	15.16	0.37	14.9	**669**	6.73	–	–	69.98	4 733	–	–
	p4	–	–	–	–	–	–	–	–	–	–	–
	p5	–	–	–	–	–	–	–	–	–	–	–
	p6	–	–	–	–	–	–	–	–	–	–	–
prioritized_arbiter_enc	p1	0.06	0.01	0	**81**	0.01	–	–	0.06	100	13.52	1 428
	p2	1.08	0.1	1.52	**458**	0.22	–	–	1.61	1 202	–	–
	p3	11.05	0.46	20.66	**768**	4.21	–	–	87.89	12 253	–	–
	p4	1 067.29	6.55	–	–	251.85	–	–	–	–	–	–
	p5	–	–	–	–	–	–	–	–	–	–	–
	p6	–	–	–	–	–	–	–	–	–	–	–

4 结 论

当前, LTL 合成仍然是非常重要且具有挑战性的研究问题. 智能规划涌现出了很多有效的方法和工具, 且与 LTL 合成有着一定的相似性. 基于此, 本文提出了一种基于非确定规划的 LTL 合成方法, 试图通过利用非确定规划的高效工具求解 LTL 合成问题. 本文的主要创新点在于: 提出了一种从 LTL 合成的二人博弈定义到非确定规划模型的转换方法, 且转换的复杂度与 LTL 公式的 NBA 大小线性相关, 并证明了方法的正确性和完备性. 选取国际合成竞赛的基准测试集进行实验, 结果证明: 本文提出的方法在规划解的质量方面具有较大优势, 能够获得更为紧凑的合成策略. 未来的工作将着眼于在保证规划解质量的前提下提高合成效率, 研究在非确定规划工具中设计适合 LTL 合成过程的启发式算法.

References:

[1] Church A. Applications of recursive arithmetic to the problem of circuit synthesis. Journal of Symbolic Logic, 1963, 28(4): 289−290. [doi: 10.2307/2271310]

[2] Pnueli A, Rosner R. On the synthesis of a reactive module. In: Proc. of the 16th Annual ACM Symp. on Principles of Programming Languages (POPL'89). ACM, 1989. 179−190. [doi: 10.1145/75277.75293]

[3] Piterman N, Pnueli A, Sa'ar Y. Synthesis of reactive (1) designs. In: Emerson EA, Namjoshi KS, eds. Proc. of the 7th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2006). Springer, 2006. 364−380. [doi: 10.1007/11609773_24]

[4] Bohy A, Bruyère V, Filiot E, *et al*. Acacia+: Atool for LTLsynthesis. In: Madhusudan P, Seshia SA, eds. Proc. of the 24th Int'l Conf. on Computer Aided Verification (CAV 2012). Springer, 2012. 652−657. [doi: 10.1007/978-3-642-31424-7_45]

[5] Jobstmann B, Bloem R. Optimizations for LTL synthesis. In: Proc. of the 6th Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD 2006). IEEE Computer Society, 2006. 117−124. [doi: 10.1109/FMCAD.2006.22]

[6] Camacho A, Muise C, Baier JA, *et al*. LTL realizability via safety and reachability games. In: Lang J, ed. Proc. of the 27th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2018). 2018. 4683−4691. [doi: 10.24963/ijcai.2018/651]

[7] Michaud T, Colange M. Reactive synthesis from LTL specification with Spot. In: Proc. of the 7th Workshop on Synthesis (SYNT 2018). 2018.

[8] Zhu SF, Tabajara LM, Li JW, *et al*. A symbolic approach to safety LTLsynthesis. In: Strichman O, Tzoref- Brill R, eds. Proc. of the 13th Int'l Haifa Verification Conf. (HVC 2017). Springer, 2017. 147−162. [doi: 10.1007/978-3-319-70389-3_10]

[9] Li JW, Zhang LJ, Pu GG, *et al*. LTL_f satisfiability checking. In: Schaub T, Friedrich G, O'Sullivan B, eds. Proc. of the 21st European Conf. on Artificial Intelligence (ECAI 2014). IOS, 2014. 513−518. [doi: 10.3233/978-1-61499-419-0-513]

[10] Giacomo GD, Vardi MY. Synthesis for LTL and LDL on finite traces. In: Yang Q, Wooldridge MJ, eds. Proc. of the 24th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2015). AAAI, 2015. 1558−1564.

[11] Zhu SF, Tabajara LM, Li JW, *et al*. Symbolic LTL_f synthesis. In: Sierra C, ed. Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2017). 2017. 1362−1369. [doi: 10.24963/ijcai.2017/189]

[12] He KL, Wells AM, Kavraki LE, *et al*. Efficient symbolic reactive synthesis for finite-horizon tasks. In: Proc. of the Int'l Conf. on Robotics and Automation (ICRA 2019). IEEE, 2019. 8993−8999. [doi: 10.1109/ICRA.2019.8794170]

[13] He KL, Lahijanian M, Kavraki LE, *et al*. Reactive synthesis for finite tasks under resource constraints. In: Proc. of the 2017 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS 2017). IEEE, 2017. 5326−5332. [doi: 10.1109/IROS.2017.8206426]

[14] Zhu SF, Giacomo GD, Pu GG, *et al*. LTL_f synthesis with fairness and stability assumptions. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence (AAAI 2020). AAAI, 2020. 3088−3095. [doi: 10.1609/aaai.v34i03.5704]

[15] Giacomo GD, Stasio AD, Vardi MY, *et al*. Two-stage technique for LTLf synthesis under LTL assumptions. In: Calvanese D, Erdem E, Thielscher M, eds. Proc. of the 17th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR 2020). 2020. 304−314. [doi: 10.24963/kr.2020/31]

[16] Ghallab M, Nau D, Traverso P. Automated Planning—Theory and Practice. Elsevier, 2004. 1−635.

[17] Camacho A, Baier JA, Muise C, *et al*. Finite LTL synthesis as planning. In: de Weerdt M, Koenig S, Röger G, Spaan MTJ, eds. Proc. of the 28th Int'l Conf. on Automated Planning and Scheduling (ICAPS 2018). AAAI, 2018. 29−38.

[18] Camacho A, Bienvenu M, McIlraith SA. Towards a unified view of AI planning and reactive synthesis. In: Benton J, Lipovetzky N, Onaindia E, Smith DE, Srivastava S, eds. Proc. of the 29th Int'l Conf. on Automated Planning and Scheduling (ICAPS 2019). AAAI, 2019. 58−67.

[19] Camacho A, Baier JA, Muise C, *et al*. Bridging the gap between LTL synthesis and automated planning. In: Proc. of the Workshop Generalized Planning. 2017. 1−10.

[20] D'Ippolito N, Rodriguez N, Sardina S. Fully observable non-deterministic planning as assumption-based reactive synthesis. Journal of Artificial Intelligence Research, 2018, 61: 593−621. [doi: 10.1613/jair.5562]

[21] Blai B, Giuseppe DG, Hector G, *et al*. High-level programming via generalized planning and LTL synthesis. In: Calvanese D, Erdem E, Thielscher M, eds. Proc. of the 17th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR 2020). 2020. 152−161. [doi: 10.24963/kr.2020/16]

[22] Raman V, Donze A, Sadigh D, *et al*. Reactive synthesis from signal temporal logic specifications. In: Girard A, Sankaranarayanan S, eds. Proc. of the 18th Int'l Conf. on Hybrid Systems: Computation and Control (HSCC 2015). ACM, 2015. 239−248. [doi: 10.1145/2728606.2728628]

[23] Vardi MY, Wolper P. Reasoning about infinite computations. Information and Computation, 1994, 115(1): 1−37. [doi: 10.1006/inco.1994.1092]

[24] Geffner H, Bonet B. A concise introduction to models and methods for automated planning. In: Proc. of the Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. 1−141. [doi: 10.2200/S00513ED1V01Y201306AIM022]

[25] Zielonka W. Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science, 1998, 200(1−2): 135−183. [doi: 10.1016/S0304-3975(98)00009-7]

[26] Muise C, McIlraith SA, Beck JC. Improved non-deterministic planning by exploiting state relevance. In: McCluskey L, Williams BC, Silva JR, Bonet B, eds. Proc. of the 22nd Int'l Conf. on Automated Planning and Scheduling (ICAPS 2012). AAAI, 2012. 172−180.

[27] Gastin P, Oddoux D. Fast LTL to Büchi automata translation. In: Berry G, Comon H, Finkel A, eds. Proc. of the 13th Int'l Conf. on Computer Aided Verification (CAV 2001). Springer, 2001. 53−65. [doi: 10.1007/3-540-44585-4_6]

陆旭(1985－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为智能规划, 模型检测, 程序验证.

田聪(1981－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为可信软件的基础理论与方法, 人工智能系统安全.

于斌(1990－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为模型检测, 运行时验证.

段振华(1948－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为时序逻辑, 形式化方法, 高可信嵌入式系统.