ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2000/00(00)-0000-00
doi: 10.3778/j.issn.1673-9418.

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

带归纳定义的分离逻辑公式求解器 COMPSPEN

高冲 [1,2+], 古新才 [1,2],

1. 中国科学院软件研究所 计算机科学国家重点实验室 北京 100190
2. 中国科学院大学 北京 100049

+. 通讯作者：电话：010-62661616，传真：010-62661616，电子邮件：gaochong@ios.ac.cn

COMPSPEN: A Solver For Separation Logic with Inductive Definitions

GAO Chong[1,2+], GU Xincai[1,2]

1. State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, 100190.
2. University of Chinese Academy of Sciences, Beijing, 100049.
+ Corresponding author: Phn: +86-10-6266-1616, Fax: +86-10-6266-1616, E-mail: gaochong@ios.ac.cn

GAO Chong, Gu Xincai. COMPSPEN: A Solver For Separation Logic with Inductive Definitions. Journal of Frontiers of Computer Science and Technology, 2000, 0(0): 1-000.

Abstract: Separation logic was proposed by Reynolds, O'Hearn, Ishtiaq, and Yang around 2000, to formally reason about the behaviors of programs manipulating pointers and dynamic data structures. The state-of-the-art solvers for separation logic usually use heuristics and are unable to guarantee their completeness. In their IJCAR 2016 paper, Gu, Chen and Wu proposed sound and complete decision procedures for a fragment of separation logic with inductive definitions that is capable of specifying the shape properties and data constraints of linear data structures. The tool COMPSPEN implemented the decision procedures proposed by Gu, Chen and Wu, based on the SMT framework. This article describes the implementation details of COMPSPEN, as well as some case studies and experimental results.

*The **** Foundation of China under Grant No.****, **** (基金中文完整名称); the **** Foundation of China under Grant No.****, **** (基金中文完整名称). [需中英文齐全].

Received 2000-00, Accepted 2000-00.

Key words: Separation logic; linear data structures; satisfiability; entailment; decision procedures

摘 要: 分离逻辑是 2000 年左右由 Reynolds、O'Hearn、Ishtiaq 和 Yang 提出的用于对含有指针和动态数据结构的程序进行分析与验证的编程逻辑。目前已有的分离逻辑公式求解器一般都是启发式的算法，无法保证结果的完备性。Gu, Chen 和 Wu 最近在 IJCAR 2016 的文章中，提出了一种能同时描述线性数据结构的形状性质和数据约束的分离逻辑子集，并针对该子集的可满足性问题与蕴涵问题设计了可靠完备的判定算法。COMPSPEN 工具基于 SMT 的框架，实现了 Gu, Chen 和 Wu 提出的算法。本文主要介绍 COMPSPEN 的实现细节与相关实例研究。

关键词: 分离逻辑；线性数据结构；可满足性问题；蕴涵问题；判定算法

文献标志码：A 中图分类号：****

1 引言

分离逻辑（separation logic）是由Reynolds、O'Hearn、Ishtiaq和Yang在2000年左右提出的对经典的用于程序验证的Hoare逻辑的一种扩展[1–3]，目的是对操作动态数据结构的程序的行为进行局部的推理。分离逻辑在经典的Hoare逻辑的基础上加入了两个算子：分离合取(separating conjunction) "*"与分离蕴涵(separating implication) "-*"。引入这两个算子的目的是对堆结构的相互独立的部分分别进行推理，例如A*B的解释是，堆结构中存在不相交的存储单元的集合X和Y使得X满足A、Y满足B。分离逻辑的最大优点是在对一段给定的代码的分析与验证过程中，可以将分析验证过程限制到该段代码所涉及的存储单元，而不用提及其它的存储单元。由于分离逻辑的局部推理的优点，我们可以将对整个堆的性质的推理分解成对堆结构的各部分的性质的推理。研究人员已经利用分离逻辑的这个优点来对实际系统中的大规模（几万行甚至几十万行）代码进行分析与验证[4]。

分离逻辑最近十多年在程序分析与验证学术界变成了一个研究热点。分离逻辑提出来的初衷是用于对操作动态数据结构的程序的行为进行手动的证明与推理，但是近期的研究工作集中在如何基于分离逻辑来对程序的行为进行自动化的分析与验证方面。而在基于分离逻辑对程序的行为进行分析与验证的过程中，针对分离逻辑公式的可满足性问题和蕴涵问题的判定算法与求解器处于非常核心的地位。动态数据结构一般同时包括形状性质

（比如链表）和数据性质（比如有序性）。目前的关于分离逻辑公式的判定算法大多集中在动态数据结构的形状性质，而较少涉及数据性质，比如[5-11]，或者虽然涉及数据性质，但大多仅仅提供不完备的启发式判定算法，比如[12-20]。

在本文中，我们聚焦在[21]中提出的带线性可组合归纳谓词的分离逻辑子集（记为$SLID_{LC}$）。$SLID_{LC}$能同时对数据结构的形状属性，即单链表、双链表、带尾指针的单双链表、和数据约束，即序关系、长度约束等同时进行描述。[21]中给出了$SLID_{LC}$子集的可满足性问题和蕴涵问题的可靠完备的判定算法。 我们对[21]中提出的判定算法进行了实现，开发了COMPSPEN原型工具。本文主要介绍COMPSPEN工具，及其相关实例研究。

本文组织如下：第2节介绍$SLID_{LC}$的定义。第3节简要介绍$SLID_{LC}$的可满足性问题和蕴涵问题的判定算法。第4节介绍COMPSPEN工具的架构和实现。第5节介绍实例研究和实验结果。

2 带线性可组合归纳谓词的分离逻辑子集（$SLID_{LC}[\mathcal{P}]$）

本节将引入带线性可组合归纳谓词的分离逻辑子集的定义。这里用$SLID_{LC}[\mathcal{P}]$表示，$\mathcal{P}$其中是归纳谓词的有限集合。在我们的问题中只考虑两种数据类型，一种是指针类型（我们用$\mathbb{L}$表示），另一种是整数类型（我们用$\mathbb{Z}$表示）。我们约定$l, l', ... \in \mathbb{L}$表示指针，$n, n', ... \in \mathbb{Z}$表示整数。$SLID_{LC}[\mathcal{P}]$用来描述数据结构，所以在$SLID_{LC}[\mathcal{P}]$中就只有两种类型的变量。$\mathcal{F}$表示指针域，$\mathcal{D}$表示数据域。我们假定$LVars$ 表示指针变量的集合，用大写字母$E, F, X, Y, ...$表示，$DVars$ 表示整型变量的集合，用小写字母$x, y, ...$表示。

每个$SLID_{LC}[\mathcal{P}]$公式可能含有$P(E, \alpha; F, \beta; \xi)$形式的归纳谓词。每个归纳谓词由一组规则来归纳定义。归纳谓词的参数被分为三类：源参数α，目标参数β，静态参数ξ。其中要求源参数和目标参数的个数和类型要一一匹配。即：如果α，β的长度为$l(l > 0)$，那么对每个$1 \le i \le l$， α_i 和β_i有相同的数据类型。不失一般性假定α，β的第一个参数的数据类型都为指针类型。这里用$E, \alpha;\ F, \beta$来表示P的参数。

$SLID_{LC}[\mathcal{P}]$公式由以下三种类型的公式组成：纯公式(pure formula) Π，数据公式(data formula) Δ，空

间公式(spatial formula)Σ，其规则如下：

$$\Pi ::= E = F|E \neq F|\Pi \wedge \Pi;$$
$$\Delta ::= true\ |x\ o\ c\ |x\ o\ y + c\ |\ \Delta \wedge \Delta;$$
$$\Sigma ::= emp|E \mapsto \rho|P(E, \alpha; F, \beta; \xi)|\Sigma * \Sigma;$$
$$\rho ::= (f, X)|d, x|\rho, \rho;$$

其中：$o \in \{=, \leq, \geq\}$,c是整型常量，$P \in \mathcal{P}, f \in \mathcal{F}, d \in \mathcal{D}$。公式Σ中的原子$emp, E \mapsto \rho, P(E, \alpha; F, \beta; \xi)$, 其中$E \mapsto \rho$叫指针原子(point-to atoms)，$P(E, \alpha; F, \beta; \xi)$叫做谓词(predicate atoms)，$E$叫做指针和谓词的根(root)。

现在定义带归纳谓词和数据约束的分离逻辑的语法：

基本规则:

$R_0: P(E, \alpha; F, \beta; \xi) ::= E = F \wedge \alpha = \beta \wedge emp$

归纳规则:

$R_1: P(E, \alpha; F, \beta; \xi) ::= \exists X \exists x. \Delta \wedge E \mapsto \rho * P(Y, \gamma; F, \beta; \xi)$

其中：

1) F,β中的变量不会出现在Δ或者E ↦ ρ中

2) Δ中的每个原子都是这种形式：$\alpha_i\ o\ c, \alpha_i\ o\ \xi_j, \alpha_i\ o\ \gamma_i + c, 1 \leq i \leq |\alpha|, 1 \leq j \leq |\xi|, c \in \mathbb{Z}, o \in \{=, \leq, \geq\}$

3) 对于$1 \leq i \leq |\alpha|$，如果α_i是数据变量，要么α_i出现在ρ里要么Δ中包含$\alpha_i\ o\ \gamma_i + c,\ c \in \mathbb{Z}$

4) $(Y, \gamma; F, \beta; \xi)$中的变量最多出现一次

5) $\alpha \cup \xi \cup X$中的地址变量全出现在ρ中，并且ρ中的变量最多出现一次

6) $Y \in X, \gamma \subseteq \{E\} \cup X \cup x$

对于一个归纳谓词$P \in \mathcal{P}$，用$Flds(P)$表示在P的归纳定义中出现的所有域的集合，用$LFlds(P)$表示在P的归纳定义中出现的所有指针域的集合，定义$PFlds(P)$表示的P主要地址域，即：如果$f \in LFlds(P)$，并且(f, Y)出现在ρ中，就称f是P的主要地址域，由此可以知道$PFlds(P)$是唯一的。同样的用$Flds(a)$表示空间原子a中的域的集合，即：如果$a = E \mapsto \rho$，则$Flds(a)$就是出现在ρ中的所有域的集合；如果$a = P(Y, \gamma; F, \beta; \xi)$就是$Flds(P)$。

用$SLID_{LC}[\mathcal{P}]$表示所有带归纳谓词和数据约束的公式$\varphi = \Pi \wedge \Delta \wedge \Sigma$的集合，这些公式满足：对于任何谓词$P_1, P_2 \in \mathcal{P}$，如果$Flds(P_1) = Flds(P_2)$，则有$PFlds(P_1) = PFlds(P_2)$；对于一个谓词$P \in \mathcal{P}$，对于$\Sigma$中的每一个谓词都是$P(Y, \gamma; F, \beta; \xi)$形式，对于Σ中的每一个指针原子$a$都有$Flds(a) = Flds(P)$。

$SLID_{LC}[\mathcal{P}]$公式解释在一个二元组(s, h)上，其中s是对变量的赋值，h是堆结构。其中归纳谓词的语义用最小不动点来定义。具体语义请参见[21]。

下面是几个在$SLID_{LC}[\mathcal{P}]$中可以定义的几个表示常见数据结构的例子：其中$lseg$表示链表，$slseg$表示有序链表，$dllseg$表示双向链表，tlseg表示带尾指针的链表，$ldllseg$表示带长度的双向链表。

$lseg(E;F) ::= E = F$
$lseg(E;F) ::= \exists X. E \mapsto (next, X) * lseg(X;F)$

$slseg(E,x;F,x') ::= E = F \wedge x = x'$
$slseg(E,x;F,x') ::= \exists X, x''. x \leq x'' \wedge E \mapsto$
$\big((next, X), (data, x)\big) * slseg(X, x''; F, x')$

$tlseg(E;F;B) ::= E = F$
$tlseg(E;F;B) ::= \exists X. E \mapsto$
$\big((next, X), (tail, B)\big) * tlseg(X;F;B)$

$dllseg(E,P;F,L) ::= E = F \wedge P = L$
$dllseg(E,P;F,L) ::= \exists X. E \mapsto$
$\big((next, X), (prev, P)\big) * dllseg(X, E; F, L)$

$ldllseg(E,P,l;F,L,l') ::= E = F \wedge P = L \wedge l = l'$
$ldllseg(E,P,l;F,L,l') ::= \exists X, l''. l = l'' + 1 \wedge E \mapsto$
$(next, X), (prev, P)) * ldllseg(X, E, l''; F, l. l')$

3 可满足性问题和蕴涵问题判定算法简介

3.1 可满足性问题求解

为了判定分离逻辑公式φ的可满足性，引入φ的抽象$Abs(\varphi)$，从而公式φ的可满足性问题转化为$Abs(\varphi)$的可满足性问题，即φ可满足当且仅当$Abs(\varphi)$可满足。

$Abs(\varphi)$的构造方法如下：对于公式$\varphi = \Pi \wedge \Delta \wedge \Sigma$，求$Abs(\varphi)$的关键在于计算空间原子公式的抽象表示$Abs(\Sigma)$。对于空间原子$a_i \in \Sigma$，构造抽象表示$\varphi_{a_i}$：如果$a_i = E \mapsto \rho$为指针原子则引入一个布尔变量表示 E 已经被分配；如果$a_i = P(Z_1, \mu; Z_2, \nu; \chi)$为谓词原子，则需要引入布尔变量表示 Z1 已经被分配，而且，根据 P 的归纳定义可能需要为 ai 中的其它参数引入布尔变量来表示其是否被分配。另外，需要计算对应谓词P的数据约束传递闭包$\psi_P(k_i, \alpha', \beta')$（具体细节参见[21]），进而计算其抽象表示。此外，还要计算对分离合取算子*的语义进行编码的公式φ_*。最终

$Abs(\varphi) = \Pi \wedge \Delta \wedge \wedge_i \varphi_i \wedge \varphi_*$。

例如：对于公式$\varphi = E_1 = E_4 \wedge x_1 > x_2 + 1 \wedge ldllseg(E_1, E_3, x_1; E_2, E_4, x_2)$,其抽象表示为：

$Abs(\varphi) = E_1 = E_4 \wedge x_1 > x_2 + 1 \wedge$
$((E_1 = E_2 \wedge E_3 = E_4 \wedge x_1 = x_2 \wedge k_1 = 0) \vee$
$([E_1, 1] \wedge [E_4, 1] \wedge k_1 = 1 \wedge x_1 = x_2 + k_1) \vee$
$([E_1, 1] \wedge [E_4, 1] \wedge k_1 \geq 2 \wedge x_1 = x_2 + k_1))$

3.2 蕴含问题求解

从φ和ψ分别构造图$\mathcal{G}_\varphi$和图$\mathcal{G}_\psi$，然后将蕴涵问题归结为$\mathcal{G}_\varphi$到$\mathcal{G}_\psi$的（某种）图同态问题。

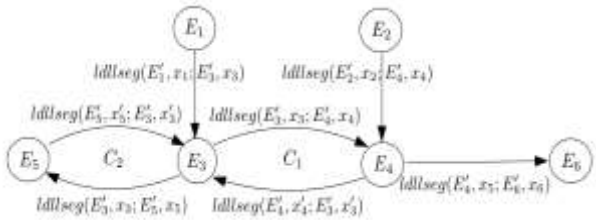
对于蕴含问题$\varphi \vDash \psi$,首先要验证$Abs(\varphi) \vDash \exists Z. Abs(\psi)$成立。如果该结论不成立，则蕴涵不成立。否则，构造公式对应的图$\mathcal{G}_\varphi$和$\mathcal{G}_\psi$，接着将$\mathcal{G}_\varphi$化简为多个近有向无环图（DAG-like，即图的每个连通分支最多含有一个环）的图（这些图称为$\mathcal{G}_\varphi$的分配方案）。化简的基本思想如下：对于含有环的每个连通分支，假设其中某个环对应非空的子堆，或者假设所有的环都为空，在这些前提下按照分离算子的语义进行化简，直到图变成近有向无环图(具体细节参见[21])。最后对每个分配方案$\mathcal{G}_{AP}$，判定是否存在从$\mathcal{G}_\psi$到$\mathcal{G}_{AP}$的某种同态。同态定义的基本思想如下：对于图$\mathcal{G}_\psi$中的每条边$e \in \mathcal{R}_\psi$，在$\mathcal{G}_{AP}$中找

到一条路径与之对应，并且保证不同的边对应的路径不相交，所有的边对应的路径恰好覆盖$\mathcal{G}_{\mathrm{AP}}$中所有边（具体定义参见[21]）。

例如以下公式：

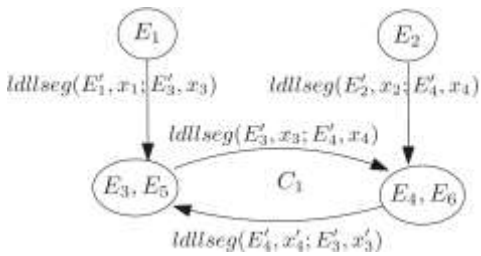
$$\begin{aligned}\varphi = & \underbrace{ldllseg(E_1,E_1';x_1;E_3,E_3';x_3)}_{a_1} * \underbrace{ldllseg(E_2,E_2';x_2;E_4,E_4';x_4)}_{a_2} * \\ & \underbrace{ldllseg(E_3,E_3';x_3;E_4,E_4';x_4)}_{a_3} * \underbrace{ldllseg(E_4,E_4';x_4';E_3,E_3';x_3')}_{a_4} * \\ & \underbrace{ldllseg(E_3,E_3';x_3;E_5,E_5';x_5)}_{a_5} * \underbrace{ldllseg(E_3,E_3';x_3';E_3,E_3';x_3')}_{a_6} * \\ & \underbrace{ldllseg(E_4,E_4';x_5;E_6,E_6';x_6)}_{a_7}.\end{aligned}$$

构造其对应的图$\mathcal{G}_{\varphi}$（仅有一个连通分支）如图表 1 $\mathcal{G}_{\varphi}$图表 1：

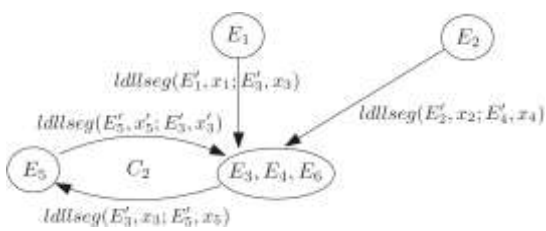


图表 1 $\mathcal{G}_{\varphi}$

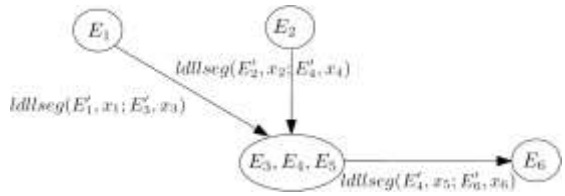
对应分配方案 AP 如图表 2（假设C_1非空），图标 3（假设C_2非空），图表 （假设C_1, C_2都为空）：



图表 2 $\mathcal{G}_{AP_1}$



图表 3 $\mathcal{G}_{AP_2}$



图表 4 $\mathcal{G}_{AP_3}$

4 工具实现

主要介绍$\mathrm{SLID}_{\mathrm{LC}}[\mathcal{P}]$逻辑公式蕴涵问题的判定工具COMSPEN及相关实验。COMSPEN工具在法国巴黎第七大学IRIF实验室验证组开发的分离逻辑求解器SPEN工具的基础上扩展而成。

4.1 解析模块

COMPSPEN输入文件基于SMTLIB2语法，SMTLIB2是SMT建模语言，其解析模块用的是开源的SMT文件解析器SMTLIB2PARSER。

4.1.1 输入文件格式说明

SMTLIB2是SMT建模语言。利用SMTLIB2可以方便地对$SLID_{LC}[\mathcal{P}]$进行定义，包括谓词的定义，可满足性问题以及蕴涵问题的描述。SMT文件中主要包含两方面的内容：线性可组合归纳谓词的定义和可满足性问题公式或蕴涵问题公式。下面就从这两方面介绍SMT文件格式。

首先是线性可组合归纳谓词的定义。谓词定义需要引入分离合取（*），域（field）等概念，这些

概念在$SLID_{LC}$ [$\mathcal{P}$]中定义，以下是SMTLIB2中对$SLID_{LC}$ [$\mathcal{P}$]的定义，其中SetRef 用来定义逻辑定义中ρ , $ssep$即逻辑中的分离链接词($*$) , pto即$E \mapsto \rho$中的$\mapsto$：

```
(theory SLRDI
      :sorts((Field 2) (SetRef 1) (Space 0) (Int 0))
      :funs((emp Space)
          (ssep Space Space :left-assoc)
          (par (A) (pto A (SetRef A) Space :left-assoc))
          (tobool Space Bool)
          (tospace Bool Space)
          (par (A B) (ref (Field A B) B (SetRef A)))
          (par (A) (sref (SetRef A) (SetRef A) (SetRef A) :left-assoc))
      )
)
```

例如形式定义为：

$ldllseg(E,P,l;F,L,l') ::= E = F \wedge P = L \wedge l = l'$

$ldllseg(E,P,l;F,L,l') ::= \exists X, l''. l = l'' + 1 \wedge E \mapsto (next, X), (prev, P)) * ldllseg(X,E,l'';F,l.l')$

可表示为：

```
(set-logic QF_SLRDI)
(declare -sort dll_t 0)
(declare -fun next() (Field dll_t dll_t))
(declare -fun prev() (Field dll_t dll_t))
 (define -fun dllseg
     ((?E dll_t) (?P dll_t) (?x0 Int)
     (?F dll_t) (?L dll_t) (?x1 Int)) Space
     (tospace
          (or
               (and (= ?E ?F) (= ?P ?L) (= ?x0 ?x1) (tobool emp))
               (exists ((?X dll_t) (?x2 Int))
               (and
               (= ?x0 (+ ?x2 1))
               (tobool (ssep
               (pto ?E (sref (ref next ?X) (ref prev ?P) ))
               (dllseg ?X ?E ?x2 ?F ?L ?x1)))))))
     )
```

对于可满足问题：

$\varphi = x_1 > x_2 + 1 \wedge E_0 \mapsto \big((prev, nil), (next, E_1)\big) * ldllseg(E_1, E_3, x_1; E_2, E_4, x_2)$,可表示为：

```
(declare-fun E0 () dll_t)
(declare-fun E1 () dll_t)
(declare-fun E2 () dll_t)
(declare-fun E3 () dll_t)
(declare-fun E4 () dll_t)
(declare-fun E5 () dll_t)
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun alpha0 () SetLoc)
(assert (and
          (> x1 (+ x2 1))
          (tobool (ssep
          (pto E0 (sref (ref prev nil) (ref next E1)))
          (index alpha0 (ldllseg E1 E3 x1 E2 E4 x2) )))))
(check-sat)
```

$\psi = x_1 > x_2 + 2 \wedge ldllseg(E_0, E_5, x_1; E_2, E_4, x_2)$对于蕴含问题$\varphi \vDash \psi$的右边可表示为：

```
(assert (not (and
       (> x1 (+ x2 2))
       (tobool (ssep
          (index alpha0 (ldllseg E0 E5 x1 E2 E4 x2)
      ))))))
(check-sat)
```

4.1.2 解析过程

解析模块是工具的前端，完成从文件到逻辑结

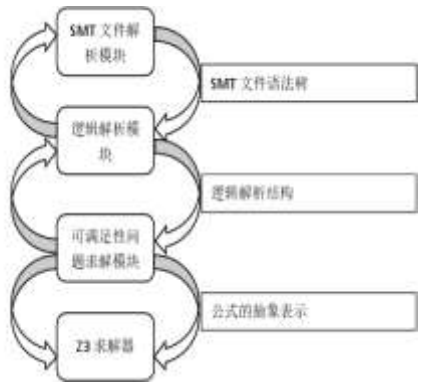
构的解析工作。SMT文件解析器通过词法分析，语法分析得到抽象语法树，然后逻辑解析模块会将抽象语法树结构进行语义分析并转换为我们需要的内部数据结构的表示,比如归纳谓词结构的表示，公式的结构表示等。

4.2 求解模块

求解模块是工具的后端，完成对公式可满足性问题或蕴含问题的求解。可满足性问题求解是利用解析模块得到数据结构，构造公式的抽象表示，然后调用SMT求解器Z3进行求解；蕴含问题求解构造公式的图形表示，检查蕴涵问题相关的逻辑公式的图结构的同态关系，这个过程会多次调用可满足性求解模块生成公式的抽象表示和Z3生成求解中间过程涉及的可满足性问题。

4.2.1 可满足性问题求解模块

$SLID_{LC}$ [$\mathcal{P}$]逻辑公式蕴涵问题$\varphi \vDash \psi$中，如果ψ部分为空，则原问题转变为可满足性问题，这时COMPSPEN会调用可满足性求解模块进行求解。此外，在求解蕴涵问题的过程中，也需要调用可满足性求解模块生成公式的抽象表示（Abs）。可满足性求解模块基于逻辑解析模块给出的逻辑解析结构，逻辑解析模块会首先调用SMTLIB2PARSER进行语法分析，得到语法树结构。可满足性求解模块的核心算法基于$SLID_{LC}$ [$\mathcal{P}$]逻辑公式的抽象表示，二者之间在可满足性问题上是等价的。如图表5，可满足性求解模块最终会生成$SLID_{LC}$ [$\mathcal{P}$]逻辑公式的抽象表示，交由Z3求解器求解。



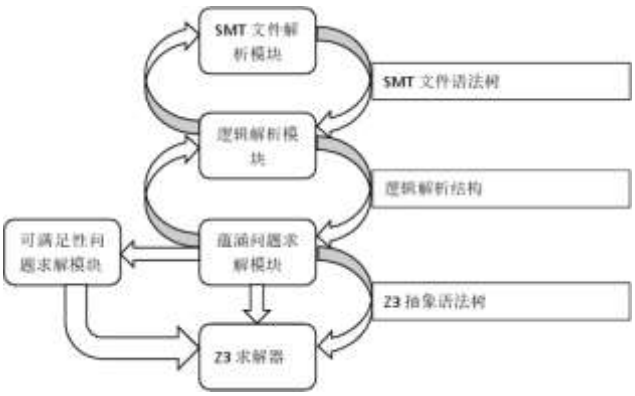
图表 5 可满足性问题求解模块

4.2.2 蕴含问题求解

与可满足性求解模块类似，蕴涵问题求解模块会调用逻辑解析模块得到其逻辑解析结构，进而进行下一步。蕴涵问题求解模块进行求解过程中可能不止一次调用可满足性求解模块生成公式的抽象表示，以及调用Z3求解器求解相关可满足性问题。图表为蕴涵问题求解模块的结构图。

关于$SLID_{LC}$ [$\mathcal{P}$]逻辑公式蕴涵问题$\varphi \vDash \psi$的判定，蕴涵问题求解模块会调用可满足性问题求解模块生成公式φ的抽象表示$Abs(\varphi)$，然后调用Z3求解器检查$Abs(\varphi) \vDash \exists Z.\, Abs(\psi)$是否成立。然后构造公

式对应的图$\mathcal{G}_{\varphi}$和$\mathcal{G}_{\psi}$，接着由SPLITTER模块计算φ的分配方案AP，蕴涵问题求解模块会检查该分配方案与图$\mathcal{G}_{\psi}$间是否存在同态映射，迭代直到检查完所有的分配方案。如果都存在同态映射，则蕴含问题成立，否则蕴含问题不成立。



图表 6 蕴含问题求解模块

5 实验

为了测试COMPSPEN的性能，进行了相关的实验。下面首先对测试用例进行简单的介绍，然后描述我们的实验环境，最后对实验结果进行简单的分析总结。

5.1 测试数据说明

实验中用到的测试用例来源于对操作动态数据结构的程序的验证过程：我们从程序和规范手动生成验证条件，即$SLID_{LC}\ [\mathcal{P}]$ 逻辑公式的可满足性问题或蕴涵问题，然后利用COMPSPEN 工具进行求解。测试用例考虑了主要的线性数据结构，包括单链表、双链表，而且同时考虑了形状性质和数据性质。使用的测试数据如表格 1被分成两组：针对单链表的测试用例，针对双链表的测试用例。

测试程序		数目
ls	sat	6
	entl	18
dll	sat	7
	entl	12

表格 1 测试数据

5.2 实验环境

- linux 操作系统(测试在 Ubuntu 17.04LTS 下完成)
- 实验系统内存：8G
- 处理器：Intel Core i7 CPU M 540@2.50GHz * 4
- C99 编译器（实验版本为 gcc）
- GNU flex >= 2.5.33
- GNU bison (实验版本为 bison 2.4.1)
- SMTLIB2 语法解析器
- Z3 SMT 求解器(实验版本为 Z3 4.4.2) ：
- boost 库(实验版本为 boost 1.58.0)

5.3 实验结果与分析

测试程序	数目	平均时间(s)
ls	sat	6	0.011
	entl	18	0.333
dll	sat	7	0.011
	entl	12	1.269

表格 2 实验结果

实验的主要目的在于测试可满足性问题和蕴

涵问题判定算法正确性与完备性。对于表格 1中的标准测试程序COMSPEN的运算结果有两个：SAT和UNSAT。对于可满足性问题，SAT 说明所验证的$SLID_{LC}$ [$\mathcal{P}$]逻辑公式可满足，而且，COMSPEN还会基于$SLID_{LC}$ [$\mathcal{P}$]逻辑公式的抽象表示给出可满足模型，UNSAT则说明所验证的公式不可满足。对于蕴涵问题，SAT说明原蕴涵问题不成立，UNSAT说明原蕴涵问题成立（蕴涵问题$p \vDash q \equiv \neg p \vee q$，则$p \wedge \neg q$如果不可满足则原蕴涵问题成立，否则不成立）。

通过实验发现，实验中的标准测试程序涵盖了实际的程序验证过程中可能涉及到大部分线性动态数据结构（单链表，双链表，带尾指针的单双链表以及有长度约束和数据约束的单双链表等），并且对于其中的可满足性问题和蕴涵问题，COMPSEN都可以给出准确的结果（COMPSEN不会给出UNKNOWN的结果，因为我们的判定算法是完备的）。表格 2中给出执行相关检查的平均运行时间，对于表格 2的实验结果，主要有两点需要说明，一是在我们所采用的标准测试程序中，*ls*中涉及的可满足性问题较多，而*dll*中大部分为蕴涵问题，二是*dll*的平均求解时间一般大于*ls*的平均求解时间，从$SLID_{LC}$ [$\mathcal{P}$]逻辑公式的抽象表示可以看出双向链表比单链表需要处理的情况多，从而导致生成抽象表示和调用Z3求解（尤其是蕴涵问题，因为需要多次调用Z3进行求解）的运行时间的差异。表格 2中，COMSPEN关于可满足性问题的平均求解时间比蕴涵问题的平均求解时间少，原因在于对于可满足性问题只需生成$SLID_{LC}$ [$\mathcal{P}$]逻辑公式的抽象表示交由Z3求解，而对于蕴涵问题，需要迭代生成分配方案，而且整个求解过程中需要不止一次调用Z3进行相关可满足性求解。

6 结束语

本文简单介绍了针对$SLID_{LC}$ [$\mathcal{P}$]（带线性可组合归纳谓词的分离逻辑子集）的可满足性问题与蕴涵问题的工具COMPSPEN。在$SLID_{LC}$ [$\mathcal{P}$]中，常见的线性数据结构的形状性质，比如单链表、双链表等，以及数据性质，比如有序性、长度约束等，都可以被描述。在将来，考虑继续完善COMPSPEN工具，并考虑对其进行扩展，考虑针对可以描述非线性数据结构的判定算法的实现，比如[22]中最近提出的针对树结构的完备的判定算法。

参考文献

[1] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In LICS,2002, 55–74.
[2] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In POPL 2001, 14–26.
[3] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. CSL 2001, 1–19.
[4] Hongseok Yang, Oukseh Lee, Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, and Peter W. O'Hearn. Scalable shape analysis for systems code. In CAV 2008, 385–398.
[5] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In FSTTCS 2004, 97–109.
[6] Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In CONCUR 2011, 235–249.
[7] Radu Iosif, Adam Rogalewicz, and Jirí Simácek. The tree width of separation logic with recursive definitions. In CADE-24, 2013, 21–38.
[8] Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In FOSSACS 2014, 411–425.
[9] Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In ATVA 2014, 201–218.
[10] Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. Compositional entailment checking for a fragment of separation logic. In APLAS 2014, 2014, 314–333.
[11] James Brotherston, Carsten Fuhs, Juan A. Navarro Pérez, and Nikos Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates. In CSL-LICS 2014, 25:1–25:10.
[12] Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Sci. Comput. Program., 2012, 77(9):1006–1036.
[13] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using SMT. In CAV 2013, 773–789.
[14] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In CAV 2014, 711–728.
[15] Ruzica Piskac, ThomasWies, and Damien Zufferey. Grasshopper - complete heap verification with mixed specifications. In TACAS 2014, 124–139.
[16] Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and Parthasarathy Madhusudan. Natural proofs for structure, data, and separation. In PLDI 2013, 231–242.
[17] Edgar Pek, Xiaokang Qiu, and P. Madhusudan. Natural proofs for data structure manipulation in C using separation logic. In PLDI 2014, 440–451.
[18] James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In CADE-23, 2011, 131–146.
[19] Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. Automating proofs of data-structure properties in imperative programs. In PLDI 2015.
[20] Constantin Enea, Mihaela Sighireanu, and ZhilinWu. On automated lemma generation for separation logic with inductive definitions. In ATVA 2015, 80–96.
[21] Xincai Gu, Taolue Chen, Zhilin Wu, A complete decision procedure for linearly compositional separation logic with data constraints, IJCAR 2016.
[22] Zhaowei Xu, Taolue Chen, Zhilin Wu, Satisfiability of Compositional Separation Logic with Tree Predicates and Data Constraints, CADE-26, 2017.