软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(5):2181−2195 [doi: 10.13328/j.cnki.jos.006407]

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

COMPSPEN: 对形状性质与数据约束进行融合推理的分离逻辑求解器*

苏婉昀[1,2], 高 冲[3], 古新才[4], 吴志林[1,2]

[1](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](华为技术有限公司 编程语言实验室, 北京 100085)

[4](百度 自动驾驶技术部, 北京 100193)

通信作者: 苏婉昀, E-mail: suwy@ios.ac.cn

摘 要: 分离逻辑是经典霍尔逻辑的针对操作指针和动态数据结构的扩展, 已经广泛用于对基础软件 (比如操作系统内核等) 的分析与验证. 分离逻辑约束自动求解是提升对操作指针和动态数据结构的程序的验证的自动化程度的重要手段. 针对动态数据结构的验证一般同时涉及形状性质 (比如单链表、双链表、树等) 和数据性质 (比如有序性、数据不变性等). 主要介绍能对动态数据结构的形状性质与数据约束进行融合推理的分离逻辑求解器 COMPSPEN. 首先介绍 COMPSPEN 的理论基础, 包括能够同时描述线性动态数据结构的形状性质和数据约束的分离逻辑子集 $SLID_{\text{data}}$、$SLID_{\text{data}}$ 的可满足性和蕴涵问题的判定算法. 然后, 介绍 COMPSPEN 工具的基本框架. 最后, 使用 COMPSPEN 工具进行了实例研究. 收集整理了 600 个测试用例, 在这 600 个测试用例上将 COMPSPEN 与已有的主流分离逻辑求解器 Asterix、S2S、Songbird、SPEN 进行了比较. 实验结果表明 COMPSPEN 是唯一能够求解含有集合数据约束的分离逻辑求解器, 而且总体来讲, 能对线性数据结构上的同时含有形状性质和线性算术数据约束的分离逻辑公式的可满足性问题进行高效的求解, 另外, 也能对蕴涵问题进行求解.

关键词: 分离逻辑; 形状性质; 线性算术数据约束; 集合数据约束; 可满足性问题; 蕴涵问题; 约束求解器

中图法分类号: TP301

中文引用格式: 苏婉昀, 高冲, 古新才, 吴志林. COMPSPEN: 对形状性质与数据约束进行融合推理的分离逻辑求解器. 软件学报, 2023, 34(5): 2181–2195. http://www.jos.org.cn/1000-9825/6407.htm

英文引用格式: Su WY, Gao C, Gu XC, Wu ZL. COMPSPEN: Separation Logic Solver for Integrated Reasoning about Shape Properties and Data Constraints. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2181–2195 (in Chinese). http://www.jos.org.cn/1000-9825/6407.htm

COMPSPEN: Separation Logic Solver for Integrated Reasoning about Shape Properties and Data Constraints

SU Wan-Yun[1,2], GAO Chong[3], GU Xin-Cai[4], WU Zhi-Lin[1,2]

[1](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](Program Language Lab, HUAWEI Technologies Co. Ltd., Beijing 100085, China)

[4](Department of Automatic Driving Technology, Baidu Inc., Beijing 100193, China)

* 基金项目: 广东省科技厅新一代人工智能项目 (2018B010107004); 国家自然科学基金面上基金 (61872340); 法国 INRIA-中国科学院国际合作项目 VIP
收稿时间: 2020-08-16; 修改时间: 2021-02-04; 采用时间: 2021-06-28; jos 在线出版时间: 2021-08-03
CNKI 网络首发时间: 2022-11-15

Abstract: Separation logic is an extension of the classical Hoare logic for reasoning about pointers and dynamic data structures, and has been extensively used in the formal analysis and verification of fundamental software, including operating system kernels. Automated constraint solving is one of the key means to automate the separation-logic based verification of these programs. The verification of programs manipulating dynamic data structures usually involves both the shape properties, e.g., singly or doubly linked lists and trees, and data constraints, e.g., sortedness and the invariance of data sets/multisets. This paper introduces COMPSPEN, a separation logic solver capable of simultaneously reasoning about the shape properties and data constraints of linear dynamic data structures. First, the theoretical foundations of COMPSPEN are introduced, including the definition of separation logic fragment $SLID_{\text{data}}$ as well as the decision procedures of the satisfiability and entailment problems of $SLID_{\text{data}}$. Then, the implementation and the architecture of the COMPSPEN tool are presented. At last, the experimental results for COMPSEN are reported. 600 test cases are collected and the performance of COMPSPEN is compared with the state-of-the-art separation logic solvers, including Asterix, S2S, Songbird, and SPEN. The experimental results show that COMPSPEN is the only tool capable of solving separation logic formulae involving set data constraints, and in overall, it is able to efficiently solve the satisfiability problem of separation logic formulas involving both shape properties and linear arithmetic data constraints on linear dynamic data structures, and is also capable of solving the entailment problem.

Key words: separation logic; shape properties; linear arithmetic data constraints; set data constraints; satisfiability problem; entailment problem; constraint solvers

分离逻辑 (separation logic) 是由文献 [1–4] 在 2000 年左右提出的经典 Hoare 逻辑的一种扩展. 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, 堆结构中存在地址不相交的子堆 X 和 Y 使得 X 满足 A、Y 满足 B. 通过加入这两个算子以及相应的推导规则, 分离逻辑在对一段给定的代码进行分析与验证的过程中, 可以将分析验证过程限制在该段代码所涉及的存储单元中, 而不用提及其他的存储单元, 同时也解决了霍尔逻辑在验证操作堆的程序时指针或变量别名的问题. 因此分离逻辑能够很好地用于验证操作内存和动态数据结构的程序. 研究人员已经利用分离逻辑对实际系统中的大规模 (几万行到几十万行) 代码进行了分析与验证[4].

基于分离逻辑对操作内存和动态数据结构的程序进行验证通常需要用户给出用分离逻辑公式描述的程序的前置条件、循环不变式以及后置条件, 然后通过计算程序的最强后断言得到最后的验证条件, 即一个分离逻辑公式是否蕴涵另一个分离逻辑公式. 这被称为分离逻辑公式的蕴涵问题, 而蕴涵问题求解一般建立在可满足性问题求解的基础上, 因此一个能够自动求解这两类问题的分离逻辑公式求解器是提高操作内存和动态数据结构的程序验证的自动化程度的关键. 这方面的研究工作在近十年发展迅速, 下面对相关工作进行简要总结和介绍:

Berdine 等人[5]和 Haase 等人[6]在 2011 年提出了含有单链表归纳谓词的分离逻辑公式蕴涵问题的基于图同态的多项式时间判定算法[5], 并且实现了求解器 SeLoger[6].

Brotherston 等人开发了 SLSAT 求解器, 其通过归纳谓词的不动点抽象来求解含有用户可定义归纳谓词的分离逻辑公式的可满足性问题[7].

Iosif 等人开发了分离逻辑求解器 SLIDE, 其将分离逻辑公式用树自动机来刻画, 从而将蕴涵问题转化为树自动机的语言包含问题, 然后用树自动机工具来求解[8].

Pérez 等人开发了 Asterix 求解器, 其能够处理带长度约束的单链表的分离逻辑公式的判定问题[9].

Khoo 等人开发了分离逻辑求解器 Songbird, 其通过归纳证明和引理合成等技术来实现分离逻辑蕴涵问题求解[10].

Enea 等人开发了求解器 Spen, 其构造分离逻辑公式的布尔抽象来解决可满足性问题; 另外其将分离逻辑公式使用树来编码, 同时使用树自动机来刻画归纳谓词, 从而将蕴涵问题转化为树自动机的成员判定问题, 最后使用树自动机工具来求解[11].

Le 等人开发了求解器 S2S, 其同样构造分离逻辑公式的布尔抽象来求解可满足性问题, 而且使用循环证明 (cyclic proof) 的技术来解决蕴涵问题[12]. S2S 在分离逻辑求解器竞赛 SL-COMP 2019 中排名第一.

动态数据结构一般同时包括形状性质和数据性质. 形状性质是指与堆的形状相关的性质, 比如链表、树等, 而数据性质是指存储在堆中的数据之间的性质, 比如长度、有序性等. 上述求解器中, SeLoger、SLSAT 和 SLIDE 只能对动态数据结构的形状性质进行推理, 而 Asterix、Songbird、S2S 和 Spen 能够对形状性质和数据性质同时进

行推理, 但是它们要么是基于启发式算法 (比如 Songbird、S2S 和 Spen), 没有完备性保证, 要么是集中在单链表上 (比如 Asterix), 描述的形状性质较弱. 而且, 这些工具均无法对更复杂的数据性质, 比如集合数据性质, 进行推理. 而集合数据性质对于操作动态数据结构的程序的验证, 比如排序算法中排序前后数据集的不变性, 是非常重要的. 因此, 目前还缺乏对动态数据结构的形状性质 (包括单链表、双链表、树结构等) 和数据性质 (包括有序性、数据集不变性、长度性质等) 同时进行推理, 而且有完备性保证的分离逻辑求解器.

本论文介绍分离逻辑求解器 COMPSPEN. COMPSPEN 能够对线性数据结构的形状性质 (包括单链表、双链表等) 和数据性质 (包括线性算术数据约束、集合数据约束) 同时进行推理, 而且有完备性保证, 更重要的是, 实验结果表明在分离逻辑求解器竞赛的测试用例集上, COMPSPEN 与已有工具相比, 具有良好的求解能力, 在部分测试用例集上, 表现出了比较出色的求解效率, 而且是目前为止, 唯一能对集合数据性质进行自动推理的分离逻辑求解器. 这里需要指出的是, 本论文的主要目的是对 COMPSPEN 工具进行总体介绍, 与文献 [13,14] 有着本质性的区别, 具体区别和联系如下: COMPSPEN 工具的实现基于文献 [13,14] 中的判定算法, 但文献 [13,14] 均是理论文章, 并没有包括工具实现和实验部分. 而本论文为了方便读者, 将会对文献 [13,14] 中的判定算法进行简单介绍和回顾.

本文第 1 节介绍 COMPSPEN 所使用的能够对形状性质与数据性质同时进行描述的抽象分离逻辑 $SLID_{\text{data}}$, 其中数据约束没有具体给定. 在第 1 节中, 我们同时定义了对 $SLID_{\text{data}}$ 中的抽象数据约束进行具体化得到的两个逻辑 $SLID_{\text{int}}$ 和 $SLID_{\text{set}}$, 分别可以对线性算术数据约束和集合数据约束进行描述. 第 2 节简要介绍 $SLID_{\text{int}}$ 和 $SLID_{\text{set}}$ 的可满足性问题与蕴涵问题的判定算法. 第 3 节介绍 COMPSPEN 工具架构. 第 4 节对实验情况进行介绍. 第 5 节进行总结.

1 带线性可组合归纳谓词与数据约束的分离逻辑 $SLID_{\text{data}}$

我们考虑地址和地址中存储的数据. 我们用 $\mathbb{L}$ 表示地址集, $\mathbb{D}$ 表示数据集. 而且, 我们用 $l,l',\ldots$ 表示地址常量, 用 $E,F,X,Y,\ldots$ 表示地址变量. 我们用 $d,d',\ldots$ 和 $x,y,\ldots$ 分别表示 $\mathbb{D}$ 上的数据常量和变量. 我们用 $2^{\mathbb{D}}$ 表示 $\mathbb{D}$ 的幂集, 用 $D,D',\ldots$ 和 $S,S',\ldots$ 分别表示 $2^{\mathbb{D}}$ 上的常量和变量. 我们将指针指向的一块区域分为若干个地址域和若干个数据域, 每个地址域中存储一个地址, 每个数据域中存储一个数据, 比如在常见的数据结构单链表中, 每个指针指向的区域包含一个地址域 next 和一个数据域 data, 分别存储下一个节点的地址和当前节点的数据. 我们用 $\mathcal{F}$ 表示地址域的集合, 用 $\mathfrak{f},\mathfrak{f}',\ldots$ 表示地址域, 即 $\mathcal{F}$ 的元素, 用 $\mathcal{D}$ 表示数据域的集合, 用 $\mathfrak{d},\mathfrak{d}',\ldots$ 表示数据域, 即 $\mathcal{D}$ 的元素.

我们假设数据约束可以使用数据集 $\mathbb{D}$ 上的某个逻辑理论 $\mathcal{L}$ 进行描述. 另外, 我们在分离逻辑的归纳谓词的定义中, 会对数据约束进行限制, 使用 $\mathcal{L}$ 的一个子理论 $\mathcal{L}'$. 下面我们首先不对 $(\mathbb{D},\mathcal{L},\mathcal{L}')$ 进行具体化, 给出 $SLID_{\text{data}}$ 的抽象定义. 然后, 我们对 $SLID_{\text{data}}$ 进行实例化, 给出使用不同数据约束的几个具体分离逻辑子集的定义.

定义 1. 分离逻辑 $SLID_{\text{data}}$. 分离逻辑 $SLID_{\text{data}}$ 中的公式 $\varphi \equiv \Pi \wedge \Delta \wedge \Sigma$, 其中 Π 是纯公式 (pure formula), Δ 是数据公式 (data formula), Σ 是空间公式 (spatial formula). 公式的规则如下:

$$\begin{cases} \Pi ::= E = F | E \neq F | \Pi \wedge \Pi \\ \Delta \in \mathcal{L} \\ \Sigma ::= emp | E \mapsto \rho | P(E,\vec{\alpha};F,\vec{\beta};\vec{\xi}) | \Sigma * \Sigma \\ \rho ::= (\mathfrak{f},X) | (\mathfrak{d},d) | \rho,\rho \end{cases}$$

其中, $\mathfrak{f} \in \mathcal{F}$, $\mathfrak{d} \in \mathcal{D}$. 公式 Σ 中的 $E \mapsto \rho$ 为指针原子, $P(E,\vec{\alpha};F,\vec{\beta};\vec{\xi})$ 为归纳谓词, 其中 E, $\vec{\alpha}$ 为起始参数, F, $\vec{\beta}$ 为终止参数, $\vec{\xi}$ 为静态参数, 这些参数要么为地址类型 $\mathbb{L}$, 要么为数据类型 $\mathbb{D}$, 起始参数与终止参数是对称的, 特别地, 这两类参数的个数与类型一一对应. 而*为分离合取算子, 直观来讲, $\Sigma_1 * \Sigma_2$ 表示堆可以分成地址不相交的两个子堆, 分别满足公式 Σ_1 和 Σ_2.

$SLID_{\text{data}}$ 支持的归纳谓词是线性可组合归纳谓词, 它的定义包含一个基本规则和一个归纳规则:

基本规则: $R_0 : P(E,\vec{\alpha};F,\vec{\beta};\vec{\xi}) ::= E = F \wedge \vec{\alpha} = \vec{\beta} \wedge emp$

归纳规则: $R_1 : P(E,\vec{\alpha};F,\vec{\beta};\vec{\xi}) ::= \exists \vec{X} \exists \vec{S} \exists \vec{x}.\ \Delta \wedge E \mapsto \rho * P(Y,\vec{\gamma};F,\vec{\beta};\vec{\xi})$

R_1 满足以下约束:

(1) $F,\vec{\beta}$ 中的变量不会出现在 Δ 或者 $E \mapsto \rho$ 中;

(2) $P(Y,\vec{\gamma};F,\vec{\beta};\vec{\xi})$ 和 ρ 中的变量不会重复出现;

(3) $\vec{\alpha}\cup\vec{\xi}\cup\vec{X}$ 中的地址变量全部出现在 ρ 中;

(4) $Y\in\vec{X}$ 并且 $\vec{\gamma}\subseteq\{E\}\cup\vec{X}\cup\vec{S}\cup\vec{x}$;

(5) $\Delta\in\mathcal{L}'$.

此外, 我们用 $Flds(P)$ 表示在 P 的归纳定义中出现的所有域 (包括地址域和数据域) 的集合. 同样的用 $Flds(a)$ 表示空间原子 a 中的域的集合. 我们假设 φ 中最多包含一个归纳谓词, 假设为 P, 而且对于 Σ 中的每一个指针原子 a 都有 $Flds(a)=Flds(P)$. 最后, 有可能我们还需要对数据约束 Δ 进行合适的限制.

$SLID_{\text{data}}$ 公式解释在一个二元组 (s, h) 上, 其中 s 是对地址和数据变量的赋值, 而 $h:\mathbb{L}\to\mathbb{L}\cup\mathbb{D}$ 描述堆的结构. 特别地, $(s,h)|=\Sigma_1*\Sigma_2$ 当且仅当 h 可以分为 h_1 和 h_2, 使得 $dom(h_1)\cap dom(h_2)=\varnothing$, $(s,h_1)|=\Sigma_1$, 且 $(s,h_2)|=\Sigma_2$. 另外, 归纳谓词的语义用最小不动点来定义. 更多具体语义定义可以参考文献 [13,14].

以上是 $SLID_{\text{data}}$ 的抽象定义, 下面我们对 $SLID_{\text{data}}$ 中的 $(\mathbb{D},\mathcal{L},\mathcal{L}')$ 进行具体化, 考虑从 $SLID_{\text{data}}$ 导出的两种具体分离逻辑子集 $SLID_{\text{int}}$ 和 $SLID_{\text{set}}$.

定义 2. 分离逻辑 $SLID_{\text{int}}$[13,15]. $SLID_{\text{int}}$ 通过将 $SLID_{\text{data}}$ 中的 $(\mathbb{D},\mathcal{L},\mathcal{L}')$ 具体化为 $(\mathbb{Z},QFPA,DB)$ 而得到, 其中:

● $\mathbb{Z}$ 为整数集.

● $QFPA$ 为不带量词的 Presburger 算术公式, 具体来讲, $QFPA$ 由以下规则定义,

$$\begin{cases} t ::= n|x|t+t|t-t \\ \varphi ::= t\sim t|\varphi\vee\varphi|\varphi\wedge\varphi|\neg\varphi \end{cases},$$

其中, n 是整数常量, $\sim\in\{=,\leqslant,\geqslant\}$.

● DB 为差限约束, 具体来讲, DB 由以下规则定义:

$$\varphi ::= true|x\sim n|x\sim y+n|\varphi\wedge\varphi,$$

其中, n 是整数常量, $\sim\in\{=,\leqslant,\geqslant\}$.

另外, 我们还要求归纳谓词的归纳规则中的数据约束 满足以下条件:

● Δ 是以下原子命题公式的合取, $\alpha_i\ o\ n$, $\alpha_i\ o\ \xi_j+n$, $\alpha_i\ o\ \gamma_i+n$, 这里 $1\leqslant i\leqslant|\vec{\alpha}|=|\vec{\gamma}|$, $1\leqslant j\leqslant|\vec{\xi}|$; Δ 的这种形式意味着归纳规则中不含有整数集合变量.

● 对于每一个数据类型为 $\mathbb{Z}$ 的变量 α_i, 或者 α_i 出现在 ρ 中, 或者 Δ 含有形如 $\alpha_i\ o\ \gamma_i+n$ 的原子公式.

例 1: $SLID_{\text{int}}$ 公式可以用于定义带整数数据约束的常见线性数据结构, 比如带长度约束的双链表片段可以用归纳谓词 $ldllseg$ 来描述:

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

其中, E 和 F 表示双链表的首尾地址, 而 P 和 F 分别代表 E 和 F 的 $prev$ 域指向的地址, 而 l 和 l' 表示首尾对应的长度参数 (直观来讲, $l-l'$ 表示双链表片段的长度).

定义 3. 分离逻辑 $SLID_{\text{set}}$[14,16]. $SLID_{\text{set}}$ 通过将 $SLID_{\text{data}}$ 中的 $(\mathbb{D},\mathcal{L},\mathcal{L}')$ 具体化为 $(\mathbb{Z},PS,DBS)$ 而得到, 其中:

● PS 是 Presburger 算术的不带量词的集合约束扩展, 具体来讲, 其公式集合由如下规则定义:

$$\begin{cases} \Phi ::= T_s\asymp T_s|T_m\sim 0|\Phi\wedge\Phi|\neg\Phi \\ T_s ::= \varnothing|S|\{T_i\}|T_s\cup T_s|T_s\cap T_s|T_s\backslash T_s \\ T_m ::= n|x|\max(T_s)|\min(T_s)|T_m+T_m|T_m-T_m \\ T_i ::= n|x|\max(T_s)|\min(T_s) \end{cases},$$

其中, $\asymp\in\{=,\subseteq,\supseteq,\subset,\supset\}$, $\sim\in\{=,\leqslant,\geqslant\}$, T_s 是集合项, T_m 是整数项, T_i 是不带算术操作的整数项.

● DBS 是差限集合约束, 具体由如下规则定义:

$$\begin{cases} \varphi ::= S=S'\cup T_s|T_i\sim T_i+n|\varphi\wedge\varphi \\ T_s ::= \varnothing|\{\min(S)\}|\{\max(S)\} \\ T_i ::= \max(S)|\min(S) \end{cases}.$$

这意味着归纳规则中不含有整数变量. 而且, 我们还要求归纳谓词的归纳规则中的数据约束满足以下条件:

● 对于 Δ 中的每一个原子公式, 存在 i 使得其中的变量来自于集合 $\{\alpha_i, \gamma_i\}$.

例 2: $SLID_{\text{set}}$ 公式中可以用于定义带集合数据约束的常见线性数据结构, 比如带集合约束的有序双链表片段可以用归纳谓词 *sdllseg* 来描述:

$$\begin{cases} sdllseg(E,P,S;F,L,S') ::= E=F \wedge P=L \wedge S=S' \\ sdllseg(E,P,S;F,L,S') ::= \exists X,S''.\ S=S'' \cup \{\min(S)\} \\ \qquad\qquad \wedge E \mapsto ((next,X),(prev,P),(data,\min(S))) * sdllseg(X,E,S'';F,L,S') \end{cases}.$$

这里 S 和 S' 表示首尾地址对应的集合参数, 直观来讲, 我们有 $S' \subseteq S$, 其中 S' 表示从 F 开始的双链表所存储的数据集合, 而 S 表示从 E 开始的双链表所存储的数据集合.

2 可满足性问题和蕴涵问题判定算法简介

COMPSPEN 对 $SLID_{\text{int}}$ 与 $SLID_{\text{set}}$ 公式的可满足性问题和蕴涵问题的求解基于文献 [13−16] 中所提出的判定算法, 下面对这些判定算法进行简要介绍.

2.1 $SLID_{\text{int}}$ 可满足性问题判定算法[13,15]

对于 $SLID_{\text{int}}$ 公式 φ 的可满足性问题, 通过构造 φ 的线性算术抽象 $Abs(\varphi)$, 将 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. 而 $Abs(\varphi)$ 的求解可以使用 SMT 求解器比如 CVC4、Z3 来进行.

$Abs(\varphi)$ 的构造方法如下: 假设 $\varphi = \Pi \wedge \Delta \wedge \Sigma$, 而且 $\Sigma = a_1 * \ldots * a_n$. 则 $Abs(\varphi) \equiv \Pi \wedge \Delta \wedge \bigwedge_i Abs(a_i) \wedge \varphi_*$, 其中 $\bigwedge_i Abs(a_i)$ 是对各个空间原子的抽象, φ_* 是对分离合取算子*的抽象. 而且, 对空间原子 a_i 所涉及的地址变量 E 引入布尔变量 $[E, i]$ 来表示 E 是否在 a_i 所对应的堆中被分配. 而且, 对归纳原子 $a_i = P(E,\vec{\alpha};F,\vec{\beta};\vec{\xi})$ 进行抽象时, 需要计算归纳谓词 P 的数据约束传递闭包, 为此我们引入整数变量 k_i 来表示 $P(E,\vec{\alpha};F,\vec{\beta};\vec{\xi})$ 匹配 a_i 所代表的堆时 P 的归纳规则所需要展开的次数.

例 3: 我们以公式 $\varphi = E_1 = E_4 \wedge x_1 \geqslant x_2 + 1 \wedge ldllseg(E_1,E_3,x_1;E_2,E_4,x_2)$ 为例来说明 $Abs(\varphi)$ 的构造.

$$\begin{aligned} Abs(\varphi) =& E_1 = E_4 \wedge x_1 \geqslant x_2 + 1 \\ & \wedge ((\neg[E_1,1] \wedge E_1 = E_2 \wedge E_3 = E_4 \wedge x_1 = x_2 \wedge k_1 = 0) \\ & \vee ([E_1,1] \wedge [E_4,1] \wedge E_1 = E_4 \wedge k_1 = 1 \wedge x_1 = x_2 + k_1) \\ & \vee ([E_1,1] \wedge [E_4,1] \wedge E_1 \neq E_4 \wedge k_1 \geqslant 2 \wedge x_1 = x_2 + k_1)). \end{aligned}$$

这里我们在构造 $Abs(a_1)$ 时引入了布尔变量 $[E_1,1]$ 和 $[E_4,1]$, 以及整数变量 k_i. 谓词原子的抽象分为 3 种情况, $k_i = 0$、$k_i = 1$、$k_i = 2$. 其中 $k_i = 1$ 时, 根据归纳定义, 我们有 $E_1 = E_4$, 而 $k_i \geqslant 2$ 时, $E_1 \neq E_4$. 由于只有一个空间原子, 所以 φ_* 为空.

2.2 $SLID_{\text{int}}$ 蕴涵问题判定算法[13,15]

对于蕴涵问题 $\varphi \vDash \psi$, 我们从 φ 和 ψ 分别构造图 G_φ 和图 G_ψ, 然后将蕴涵问题归结为 G_φ 到 G_ψ 的图同态问题.

首先要验证 $Abs(\varphi) \vDash \exists \vec{Z}.Abs(\psi)$ 成立, 这里 $\vec{Z}$ 是构造 $Abs(\psi)$ 时所引入的布尔和整数变量的元组. 如果该结论不成立, 则蕴涵不成立. 否则, 构造公式对应的图 G_φ 和图 G_ψ, 接着将 G_φ 化简为多个近有向无环图, 即满足其中每个连通分支最多含有一个环的图, 这些图被称为 G_φ 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. 最后对 G_φ 的每个分配方案 G_{AP}, 判定是否存在从 G_ψ 到 G_{AP} 的某种同态, 即对于图 G_ψ 中的每条边 $e \in R_\psi$, 在 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中的边的集合不相交, 所有的边对应的路径集合恰好覆盖 G_{AP} 中所有边.

例 4: 考虑蕴涵问题 $\varphi \vDash \psi$, 这里

$$\begin{aligned} \varphi =& ldllseg(E_1,E_1',x_1;E_3,E_3',x_3) * ldllseg(E_2,E_2',x_2;E_4,E_4';x_4) \\ & * ldllseg(E_3,E_3',x_3;E_4,E_4',x_4) * ldllseg(E_4,E_4',x_4;E_3,E_3';x_3) \\ & * ldllseg(E_3,E_3',x_3;E_5,E_5',x_5) * ldllseg(E_5,E_5',x_5;E_3,E_3';x_3) \\ & * ldllseg(E_4,E_4',x_5;E_6,E_6',x_6), \end{aligned}$$

而且, $\psi = dllseg(E_1, E_1'; E_6, E_6') * dllseg(E_2, E_2'; E_4, E_4')$. 这里 *dllseg* 是从 *ldllseg* 中去掉参数 l 和 l' 得到的归纳谓词. 我们首先构造图 G_φ (图 1) 和 G_ψ (图 2), 这里图上的节点对应于谓词原子的首尾地址, 而边的标号为去掉首尾地址的谓词原子.

对应的分配方案 *AP* 如图 3 (假设 C_1 非空)、图 4 (假设 C_2 非空)、图 5 (假设 C_1 和 C_2 都为空) 所示. 我们以 G_{AP_1} 为例来说明. 假设 C_1 非空, 则从 *ldllseg* 的定义可知, E_4' 和 E_3' 必定在 C_1 对应的子堆中被分配. 而 E_3' 在从 E_1 到 E_3 的边上出现, 类似地, E_4' 在从 E_2 到 E_4 的边上出现, 因此这两条边对应的子堆必定为空. 最后, 我们得到 G_{AP_1} 实际上是刚好包含一个环的图 (图 3).

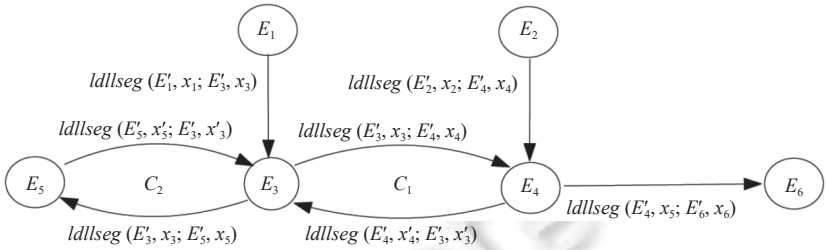


图 1 G_φ

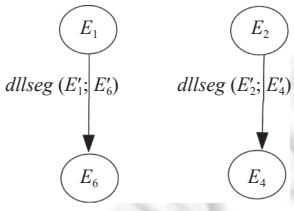


图 2 G_ψ

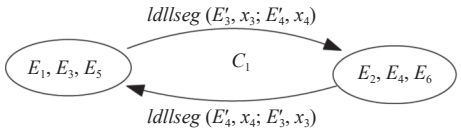


图 3 G_{AP_1}

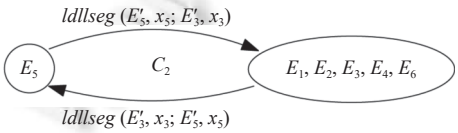


图 4 G_{AP_2}

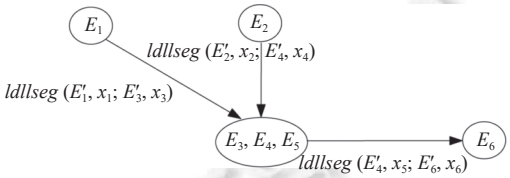


图 5 G_{AP_3}

让我们来考虑是否存在从 G_ψ 到 G_{AP_1} 的图同态. 对于 G_ψ 中从 E_1 到 E_3 的边, 必须分配 G_{AP_1} 中从 E_1, E_3, E_5 到 E_2, E_4, E_6 的边. 由于在 G_φ 中, E_2 和 E_4 处于同一个节点 E_2, E_4, E_6 中, 因此对于 G_ψ 中从 E_2 到 E_4 的边, 要么分配空路径, 或者分配 C_1. 对于前者, 则同态没有覆盖 G_φ 中的所有边, 而对于后者, 则存在一条边被分配到两条不同的路径中. 从而, 不存在从 G_ψ 到 G_{AP_1} 的同态, 所以 $\varphi \mid = \psi$ 不成立.

2.3 $SLID_{set}$ 可满足性问题判定算法[14,16]

类似于 $SLID_{int}$, 我们也是通过构造 $SLID_{set}$ 分离逻辑公式 φ 的抽象 $Abs(\varphi)$ 来求解可满足性问题. 但是, $SLID_{set}$ 含有集合约束, 因此 $Abs(\varphi)$ 的计算以及求解相对更加复杂.

$Abs(\varphi)$ 计算的关键是对归纳谓词原子进行抽象, 这需要计算用 *DBS* 公式 $\Delta(\vec{S}, \vec{S}')$ 表示的关系的传递闭包, 这里 Δ 是归纳谓词的归纳规则中的数据约束, 而且 $\vec{S}$ 和 $\vec{S}'$ 是对偶的 (即长度一样).

我们以 *sdllseg* 为例来说明传递闭包的计算. 根据 *sdllseg* 的归纳定义, 我们知道归纳规则中的数据约束为 $\Delta(S, S') = S = S' \cup \{\min(S)\}$, 从而其传递闭包为 $\Delta(S, S') = S' \subseteq S \wedge \max(S \setminus S') \leqslant \min(S')$.

由于 *sdllseg* 归纳规则中的集合数据约束相对比较简单, 因此 $\Delta(S, S')$ 传递闭包的计算也比较简单, 但是对于

更一般的集合约束, 比如 $\Delta(S,S')$ 同时包含形如 $\min(S')\geqslant\min(S)+n$ 的约束, 则 $\Delta(S,S')$ 的计算将需要进行比较详细的分情况讨论.

得到 $Abs(\varphi)$ 之后, 我们需要判定 $Abs(\varphi)$ 的可满足性, 其大致思路如下:

(1) 首先将定义在整数集合 $\mathbb{Z}$ 上的公式 $Abs(\varphi)$ 编码为定义在自然数集合 $\mathbb{N}$ 上的公式 $Abs'(\varphi)$.

(2) 然后将 $Abs'(\varphi)$ 公式规范化, 将其变为析取范式.

(3) 最后为规范化后的公式中的每个析取分量构造一个 Presburger 自动机 (PA), 这个自动机捕获析取分量对应的所有模型, 从而将 $Abs(\varphi)$ 的可满足性问题归约为 PA 的非空性问题进行判定. 而 PA 自动机的构造及其非空性问题求解均可以使用一元二阶逻辑公式求解器 MONA[17]来进行.

$SLID_{\text{set}}$ 的可满足性问题判定算法的更多细节请参见文献 [14,16].

2.4 $SLID_{\text{set}}$ 蕴涵问题判定算法[14,16]

$SLID_{\text{set}}$ 蕴涵问题 $\varphi\vDash\psi$ 的判定算法与 $SLID_{\text{int}}$ 蕴涵问题的判定算法类似, 都是构造公式对应的图 G_φ 和 G_ψ, 然后生成 G_φ 所有可行的分配方案 G_{AP}, 最后对于每个分配方案 G_{AP}, 判断 G_ψ 是否存在到 G_{AP} 的图同态映射. 蕴涵问题的求解过程需要进行可满足性判断.

3 工具实现

COMPSPEN 是采用 C++语言开发的, 它的整体框架如图 6 所示, 主要包含 3 个模块: 解析模块、预处理模块和求解模块.

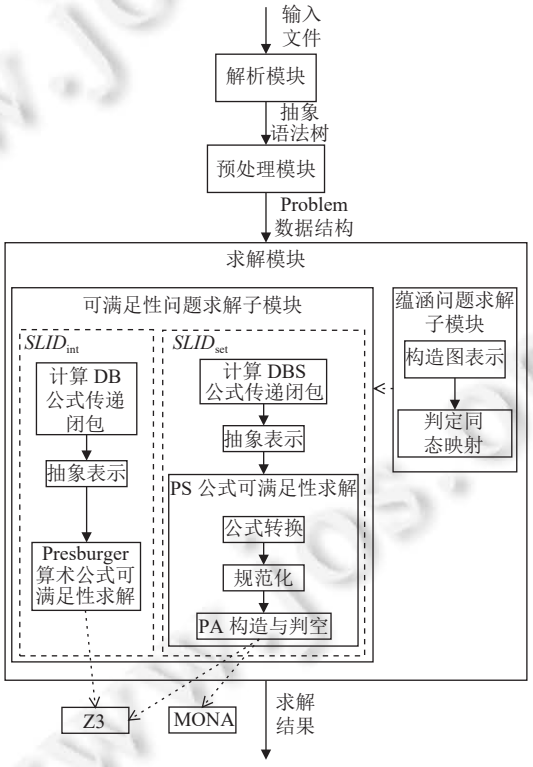


图 6 COMPSPEN 整体框架

输入文件遵循 SMT-LIB 2[18]. 解析模块对输入文件进行语法解析, 得到抽象语法树. 预处理模块从抽象语法树整理提取出求解所需要的公式信息存入到名为“Problem”的数据结构中. 求解模块包括可满足性问题求解和蕴涵

问题求解两个子模块, 根据 Problem 数据结构存储的具体信息选择不同的子模块进行求解. 求解模块可以解决 $SLID_{\text{int}}$ 和 $SLID_{\text{set}}$ 两类问题. 求解模块如果求解成功, 最后输出求解结果. 下面对各个模块进行具体介绍.

3.1 解析模块

输入文件一般包含归纳谓词定义、数据类型和变量定义以及分离逻辑公式的描述. 解析模块读取输入文件, 进行词法解析和语法解析, 构造抽象语法树. 对于新定义的数据类型以及变量, 检查是否重定义并保存在相应的数据结构中; 而对于分离逻辑公式, 检查其中各种函数的参数个数以及参数类型是否符合定义.

下面以两个例子来说明输入文件的结构.

例 5: 考虑以下 $SLID_{\text{int}}$ 蕴涵问题:

$$E_1 = P_2 \wedge x_1 = x_2 + 1 \wedge E_1 \mapsto ((next, E_2), (prev, P_1)) * ldllseg(E_2, P_2, x_2; E_3, P_3, x_3) \models ldllseg(E_1, P_1, x_1; E_3, P_3, x_3)$$

其对应的 SMT-LIB 2 格式输入文件为:

```
(set-logic QF_SLID_INT)
(declare-sort RefDll_t 0)
(declare-datatypes ((Dll_t 0)) (((c_Dll_t (next RefDll_t) (prev RefDll_t) ))))
(declare-heap (RefDll_t Dll_t) )
(define-fun-rec ldllseg ((E RefDll_t) (P RefDll_t) (len1 Int) (F RefDll_t) (L RefDll_t) (len2 Int)) Bool
  (or
    (and (= E F) (= P L) (= len1 len2) (_ emp RefDll_t Dll_t))
    (exists ((X RefDll_t) (len3 Int))
      (and (= len1 (+ len3 1 ) )
        (sep
          (pto E (c_Dll_t X P ))
          (ldllseg X E len3 F L len2 )
 )))))
(declare-const E1 RefDll_t) (declare-const E2 RefDll_t) (declare-const E3 RefDll_t)
(declare-const P1 RefDll_t) (declare-const P2 RefDll_t) (declare-const P3 RefDll_t)
(declare-const x1 Int) (declare-const x2 Int)
(declare-const x3 Int) (declare-const x4 Int)
(assert (and
  (= E1 P2) (= x1 (+ x2 1 ) )
  (sep
    (pto E1 (c_Dll_t E2 P1 ))
    (ldllseg E2 P2 x2 E3 P3 x3 )
 )))
(assert (not (ldllseg E1 P1 x1 E3 P3 x3 )))
(check-sat)
```

该输入文件涉及 8 种命令:

(1) set-logic: 设置逻辑, 这里设置的逻辑是 QF_*SLID*_INT, 对应于前述分离逻辑子集 $SLID_{\text{int}}$. QF_*SLID*_INT 逻辑理论使用 3 个初始理论, 即整数理论 Ints、布尔理论 Bools、分离逻辑理论 SepLogicTyped. 整数理论 Ints 和布尔理论 Bools 是几乎每个 SMT 求解器都支持的逻辑理论. SepLogicTyped 理论的定义如下:

```
(theory SepLogicTyped
```

```
:funs(    (emp bool)
          (sep bool bool bool :left-assoc)
          (par (L D) (pto L D Bool))
          (par (L) (nil L))
))
```

其中, L 表示地址类型, D 表示堆单元的类型, 他们将由输入文件中的 decare-heap 命令进行具体指定. SepLogicTyped 给出了 emp 原子、pto 指针原子以及 nil 地址的定义, 并说明了用 sep 来表示分离合取算子.

(2) declare-sort 定义新类型: 通过这个命令引入新的 RefDll_t 类型来表示地址.

(3) declare-datatypes 定义代数数据类型 (algebraic data types): 通过这个命令定义 Dll_t 数据类型来表示堆中的内存单元格局. Dll_t 数据类型用 c_Dll_t 来命名, 包含均为 RefDll_t 类型的 next 和 prev 属性.

(4) declare-heap 定义堆的类型: 堆包含一个 RefDll_t 类型的地址和 Dll_t 类型的内存单元格局.

(5) define-fun-rec 以递归函数的形式给出描述带长度约束的双链表片段的 *ldllseg* 归纳谓词的定义.

(6) declare-const 定义分离逻辑公式种所用到的变量, 包括 RefDll_t 类型的变量 E1、E2、E3、P1、P2、P3, 以及 Int 类型的变量 x1、x2、x3、x4.

(7) assert 给出分离逻辑公式的描述, 这里有两个 assert, 分别给出蕴涵问题的前提和结论. 直观来讲, 这里通过将结论取反的方式来将蕴涵问题编码为可满足性问题. 因此蕴涵成立当且仅当该可满足性问题答案为否定.

(8) check-sat 命令, 表示已经完整给出了可满足性问题的定义, 现在可以开始可满足性问题的求解.

例 6: 考虑以下 $SLID_{\text{set}}$ 可满足性问题:

$$E_1 = E_4 \wedge \min(S_1) < \min(S_2) - 1 \wedge sdllseg(E_1, E_3, S_1; E_2, E_4, S_2)$$

其 SMT-LIB 2 格式输入文件为:

```
(set-logic QF_SLID_SET)
(declare-sort RefDll_t 0)
(declare-datatypes ((Dll_t 0)) (((c_Dll_t (next RefDll_t) (prev RefDll_t) (data Int) ))))
(declare-heap (RefDll_t Dll_t) )
(define-fun-rec sdllseg ((E RefDll_t) (P RefDll_t) (S SetInt) (F RefDll_t) (L RefDll_t) (S1 SetInt)) Bool
  (or
    (and (= E F) (= P L) (= S S1) (_ emp RefDll_t Dll_t))
    (exists ((X RefDll_t) (S2 SetInt))
      (and
        (= S (setunion S2 (set (min S))))
        (sep
          (pto E (c_Dll_t X P (min S)))
          (sdllseg X E S2 F L S1)
)))))
(declare-const E1 RefDll_t) (declare-const E2 RefDll_t) (declare-const E3 RefDll_t) (declare-const E4 RefDll_t)
(declare-const S1 SetInt) (declare-const S2 SetInt)
(assert (and (= E1 E4) (< (min S1) (- (min S2) 1)) (sdll E1 E3 S1 E2 E4 S2)))
(check-sat)
```

该输入文件的格式与例 6 类似, 区别在于该文件设置逻辑为 QF_SLID_SET.QF_SLID_SET 使用 4 个初始理论, 即 SepLogicTyped、Ints、Bools、和 SetInts, 其中 SetInts 是整数集合理论, 定义如下:

```
(theory SetInts
 :sorts ( (SetInt 0) )
```

```
:funs (   (emptyset SetInt)
          (set Int SetInt)
          (setunion SetInt SetInt SetInt :left-assoc)
          (setintersect SetInt SetInt SetInt :left-assoc)
          (setminus SetInt SetInt SetInt)
          (subset SetInt SetInt Bool)
          (psubset SetInt SetInt Bool)
          (belongsto Int SetInt Bool)
          (min SetInt Int)
          (max SetInt Int))
)
```

其他命令的使用与例 6 类似. 而且该文件只包括一个 assert 命令, 因此是可满足性问题.

3.2 预处理模块

为了求解可满足性问题和蕴涵问题, 我们需要从抽象语法树中提取出相关信息, 存入 Problem 数据结构中 (比如归纳谓词的具体定义、分离逻辑公式等) 以便后续进行求解. Problem 数据结构的设计如图 7 所示. Problem 数据结构主要包含:

- 属性 m_logic: 字符串类型, 存储逻辑理论名称, 被用于求解模块来判断使用哪个求解器进行求解;
- 属性 m_pred: Predicate 类型, 存储归纳谓词定义, 包括参数 m_pars、基本规则 m_base_rule 和归纳规则 m_rec_rule 等信息;
- 属性 m_phi: 蕴涵问题的前提;
- 属性 m_psi: 蕴涵问题的结论, 如果是可满足性问题, 则 m_psi 没有定义, 即为 null;
- 方法 *isSat*(): 根据 m_psi 是否为空来判断需要求解的是可满足性问题还是蕴涵问题.

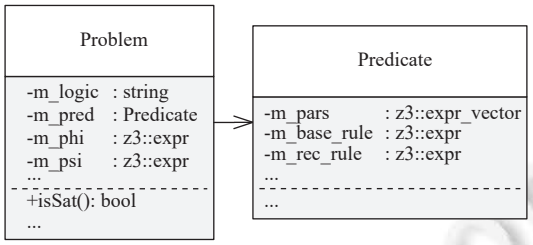


图 7 Problem 数据结构设计

另外, 为了求解模块中调用 Z3 求解器的方便, Problem 中涉及的所有分离逻辑公式都直接保存成 Z3 中的 expr 类型.

3.3 求解模块

求解模块负责完成对分离逻辑公式可满足性问题和蕴涵问题的求解. 求解模块包含两个子模块, 即 $SLID_{\text{int}}$ 求解器子模块和 $SLID_{\text{set}}$ 求解器子模块. 这两个求解器子模块都实现了解决可满足性问题的 *checkSat*() 函数和解决蕴涵问题的 *checkEntl*() 函数. 在读取 SMT-LIB 2 文件, 解析到 check-sat 命令之后, COMPSPEN 根据 Problem 数据结构中的 m_logic 属性的值来选择相应的求解器进行求解, 求解完输出 SAT/UNSAT 结果. 对于可满足性问题, SAT 表示分离逻辑公式是可满足的, UNSAT 表示不可满足; 而对于蕴涵问题, SAT 表示蕴涵不成立, UNSAT 表示蕴涵成立. 下面对这两个求解器分别进行介绍.

3.3.1 $SLID_{\text{int}}$ 求解器

$SLID_{\text{int}}$ 求解器中 *checkSat*() 函数将 Problem 数据结构中的 *m_phi* 作为参数, 返回 *m_phi* 可满足性判定的结果.

首先通过 *getAbs*() 函数构造待分离逻辑公式 *phi* 的抽象表示 *abs_phi*, 得到的 *abs_phi* 是无量词的 Presburger 算术公式, 通过调用 Z3 求解器判定 *abs_phi* 的可满足性, 并将结果进行输出. *checkSat*() 函数伪代码如下所示:

```
string SLIDintSolver::checkSat(z3::expr phi){
    abs_phi ← getAbs(phi);
    z3::solver s;
    s.add(abs_phi);
    if s.check() == z3::sat
      return "SAT";
    else
      return "UNSAT";
}
```

getAbs() 函数需要计算 3 个子公式: *data_abs* 、 *spatial_abs* 和 *spatial_star*, 分别表示 *phi* 中的纯公式、空间公式中的各个空间原子的抽象和分离合取算子语义的抽象, 其伪代码如下所示:

```
z3::expr SLIDintSolver::getAbs(z3::expr phi){
    num ← phi.num_args();
    for(int i = 0; i < num – 1; i++){
      data_itemset.push_back(phi.arg(i));
    }
    data_abs ← mk_and(data_itemset);
    spatial ← phi.arg(num-1);
    for (int i = 0; i < spatial.num_args(); i++) {
      atom ← spatial.arg(i);
      atom_abs ←getAtomAbs(atom, new_bools);
      atom_abs_set.push_back(atom_abs);
    }
    spatial _abs ← mk_and(atom_abs_set);
    spatial_star ← getSpatialStar(new_bools);
    abs←data_abs && spatial_abs && spatial_star;
    return abs;
}
```

其中, *phi* 由 *num* 个 bool 类型的参数通过合取运算连接而成, 其中前 *num* – 1 个参数组成它的纯公式部分, 最后一个参数是空间公式部分, 它由多个空间原子通过分离合取运算连接而成. *getAbs*() 函数首先取出 *phi* 的前 *num* – 1 个参数, 将他们通过 *mk_and*() 进行合取运算连接得到纯公式部分 *data_abs*. 然后取出 *phi* 的空间公式部分, 对各个空间原子进行抽象, 并用合取运算连接得到 *spatial_abs*, 对空间原子进行抽象时会产生代表对应堆是否为空的新的布尔变量, 这些变量保存在 *new_bools* 中. 对归纳谓词原子进行抽象时还需要知道归纳谓词数据约束的传递闭包, 传递闭包在预处理模块初始化 *Predicate* 对象时就可以根据它的数据约束计算出来, 作为数据成员存在 *Predicate* 对象中. 另外, 根据 *new_bools* 对分离合取算子语义进行编码得到 *spatial_star* 公式. 最终得到的 *phi* 的抽象表示为这 3 个子公式的合取.

$SLID_{\text{int}}$ 求解器中 *checkEntl*() 函数将 Problem 数据结构中的 *m_phi* 和 *m_psi* 作为参数, 返回 $m\_phi \models m\_psi$ 是否成立的结果, 其伪代码如下:

```
string SLIDintSolver::checkEntl(z3::expr phi, z3::expr psi){
    if checkSat(phi) == "UNSAT"
```

```
    return "UNSAT";
  abs_phi ← getAbs(phi);
  abs_psi ← getAbs(psi);
  z3::solver s;
  s.add( and(abs_phi,not(∃Z⃗.abs_psi)) );
  if s.check() == z3::unsat
    return "SAT";
  g_phi ← constructGraph(phi);
  g_psi ← constructGraph(psi);
  if checkAllocatingPlans(g_phi, g_psi)
    return "UNSAT";
  else
    return "SAT";
}
```

在 *checkEntl*() 函数中, 首先调用 *checkSat*() 判断 *phi* 是否可满足. 如果不可满足, 则蕴涵成立, 返回 UNSAT. 否则, 计算 *phi* 和 *psi* 的抽象表示, 并调用 Z3 求解器判断 $abs\_phi \models \vec{Z}.abs\_psi$, 即判断 $abs\_phi \wedge \neg(\exists \vec{Z}.abs\_psi)$ 是否可满足. 如果该公式可满足, 则蕴涵不成立, 返回 SAT. 否则, 构造 *phi* 和 *psi* 的图表示 *g_phi* 和 *g_psi*. 这里, 我们采用 Boost 的 Graph 库中的邻接表来表示 *g_phi* 和 *g_psi*. 最后调用 *checkAllocatingPlans* 函数来检查对于每一个 *g_phi* 的分配方案, 是否存在 *g_psi* 到该分配方案的同态. 如果存在某个分配方案, 同态不存在, 则蕴涵不成立, 返回 SAT. 否则, 蕴涵成立, 返回 UNSAT.

3.3.2 $SLID_{set}$ 求解器

$SLID_{set}$ 求解器中 *checkSat*() 函数的基本步骤与 $SLID_{int}$ 求解器类似, 都是先计算 *phi* 的抽象表示, 然后对抽象表示进行求解, 但是抽象表示的计算和最终求解都要更复杂一些. 这里 *checkSat*() 函数除了调用 Z3 求解器进行求解之外, 还需要调用 MONA 求解器.

$SLID_{set}$ 求解器中 *checkEntl*() 函数的步骤与 $SLID_{int}$ 求解器中类似, 区别只是它调用 $SLID_{set}$ 的可满足性问题求解函数.

4 实 验

我们使用了 600 个测试用例对 COMPSPEN 工具的性能进行了测试. 下面我们首先介绍测试用例, 然后介绍实验的具体情况, 最后对实验结果进行分析总结.

4.1 测试用例

测试用例共有 600 个, 分为以下 6 组:

- qf-shls-sat 和 qf-shls-entl: 这两组来自于分离逻辑求解器竞赛 SL-COMP 2019[19], 分别是含有单链表归纳谓词 (不含有数据约束) 的分离逻辑公式的可满足性问题与蕴涵问题的测试用例集;
- qf-shidlia-sat 和 qf-shidlia-entl: 这两组来自于分离逻辑求解器竞赛 SL-COMP 2019, 分别是含有线性数据结构自定义归纳谓词和线性算术数据约束的分离逻辑公式的可满足性问题与蕴涵问题的测试用例集;
- qf-shids-sat 和 qf-shids-entl: 这两组是我们收集整理的, 分别是含有线性数据结构自定义归纳谓词和集合数据约束的可满足性问题与蕴涵问题的测试用例集.

4.2 实验配置

- 处理器: Intel Core i5-8250U CPU@ 1.60 GHz.

- 系统内存: 2 GB.
- 操作系统: Ubuntu 16.04.
- 软件依赖: gcc5.4.0、MONA-1.4、z3 4.6.0.
- 超时设置: 每个测试用例最多 20 s.

4.3 实验结果

我们在这些测试用例集上, 将 COMPSPEN 与参加 SL-COMP 2019 的工具 Asterix、S2S、SPEN、Songbird 进行了对比. 不同的工具能够处理的测试用例集有所区别, 所以在不同的测试用例集上, 我们会根据具体情况, 将 COMPSPEN 与不同的工具进行了比较. 最终实验结果如表 1 所示, 实验结果包括可求解测试用例个数、总时间, 以及平均时间. 下面对实验结果进行分析总结.

表 1 实验结果 (超时设置: 20 s/测试用例)

测试用例集	求解器	可求解测试用例个数	总时间(s)	平均时间(s)
qf-shls-sat (110)	COMPSPEN	**110**	9.03	0.08
	Asterix	**110**	**1.21**	**0.01**
	S2S	**110**	1.38	**0.01**
	SPEN	**110**	9.72	0.08
	Songbird	**110**	12.31	0.11
qf-shls-entl (296)	COMPSPEN	**296**	1 456.41	4.92
	Asterix	**296**	**3.52**	**0.01**
	S2S	**296**	5.02	0.02
	SPEN	**296**	19.69	0.07
	Songbird	186	73.50	0.40
qf-shidlia-sat (30)	COMPSPEN	20	1.06	**0.05**
	S2S	**30**	11.16	0.37
	Songbird	14	1.46	0.10
qf-shidlia-entl (60)	COMPSPEN	47	105.19	2.24
	S2S	**60**	**8.48**	**0.14**
	Songbird	17	34.63	2.04
qf-shids-sat (58)	COMPSPEN	**58**	**44.49**	**0.77**
qf-shids-entl (46)	COMPSPEN	**46**	**454.47**	**9.88**

从表 1 我们可以得出如下结论.

- 从可求解测试用例数来讲, COMPSPEN 在 qf-shls-sat 和 qf-shls-entl 两个测试用例集上能够求解所有测试用例, 而在 qf-shidlia-sat 和 qf-shidlia-sat 测试用例集上, COMPSPEN 能够求解的测试用例数占总测试用例数的 66% 和 78% 左右, 仅次于 S2S, 大幅领先于 Songbird (46% 和 28%). COMPSPEN 不能求解的测试用例主要是不满足 $SLID_{\text{int}}$ 语法限制的测试用例, 比如归纳定义中含有多个指针原子、通过方向展开的方式来定义的链表归纳谓词等. 需要着重指出的是, COMPSPEN 是唯一支持对集合数据约束进行求解的分离逻辑求解器.
- 从求解时间上来讲, COMPSPEN 在可满足性问题上的求解时间与最好水平 (S2S) 相对来讲比较接近, 而且在含有线性算术数据约束的 qf-shidlia-sat 测试用例集上平均花费时间最少. 在蕴涵问题上, COMPSPEN 耗时较长, 可能的原因主要包括: 1) 求解蕴涵问题时, 需要多次调用可满足性模块, 而可满足性问题求解中数据约束的传递闭包的计算比较复杂; 2) 需要对最坏情况下有指数多个的分配方案及进行枚举.
- 总体来讲, S2S 的求解能力最为突出, 这也符合 S2S 在 SL-COMP 2019 竞赛中排名第一的结果, 但是 S2S 无法求解含有集合数据约束的分离逻辑公式, 而 COMPSPEN 可以求解.

5 结 论

本文介绍了分离逻辑求解器 COMPSPEN. COMPSPEN 能对线性数据结构上的形状性质和数据性质进行融合

推理, 包括单链表、双链表、含有尾指针的链表、线性算术约束 (比如有序性和长度约束)、集合数据约束 (比如数据集不变性) 等. COMPSPEN 建立在具有完备性保证的判定算法的基础上, 既支持对可满足性问题的求解, 也支持对蕴涵问题的求解. 由于实际操作动态数据结构的程序 (比如排序算法等) 一般同时涉及形状性质和数据性质, 因此 COMPSPEN 的能够对形状性质和数据性质进行可靠完备的推理的特性可以大幅度提高对操作动态数据结构的程序的验证的自动化程度, 减少验证的成本, 提升验证效率.

COMPSPEN 的不足之处首先是蕴涵问题的求解效率不够高, 我们考虑对算法进行优化; 其次, COMPSPEN 目前还不能处理非线性数据结构, 比如树结构, 我们正在探讨如何对 COMPSPEN 进行扩展以支持非线性数据结构, 比如实现针对树结构的判定算法[20]; 另外, 虽然原则上来讲, $SLID_{\text{data}}$ 框架能够支持任何数据理论的实例化, 但针对不同数据理论, 其可满足性问题和蕴涵问题的判定算法需要单独设计, 因此 COMPSPEN 目前只支持 $SLID_{\text{data}}$ 的两种理论, 即 $SLID_{\text{int}}$ 和 $SLID_{\text{set}}$, 的实例化. 未来, 我们考虑 $SLID_{\text{data}}$ 的更多实例化, 包括多重集合数据约束和序列 (list) 数据约束等.

References:

[1] Reynolds JC. Separation logic: A logic for shared mutable data structures. In: Proc. of the 17th Annual IEEE Symp. on Logic in Computer Science. Copenhagen: IEEE, 2002. 55–74. [doi: 10.1109/LICS.2002.1029817]

[2] Ishtiaq SS, O'Hearn PW. BI as an assertion language for mutable data structures. ACM SIGPLAN Notices, 2001, 36(3): 14–26. [doi: 10.1145/373243.375719]

[3] O'Hearn P, Reynolds J, Yang H. Local reasoning about programs that alter data structures. In: Proc. of the 15th Int'l Workshop on Computer Science Logic. Paris: Springer, 2001. 1–19. [doi: 10.1007/3-540-44802-0_1]

[4] Yang H, Lee O, Berdine J, Calcagno C, Cook B, Distefano D, O'Hearn P. Scalable shape analysis for systems code. In: Proc. of the 20th Int'l Conf. on Computer Aided Verification. Princeton: Springer, 2008. 385–398. [doi: 10.1007/978-3-540-70545-1_36]

[5] Berdine J, Calcagno C, O'Hearn PW. A decidable fragment of separation logic. In: Proc. of the 24th Int'l Conf. on Foundations of Software Technology and Theoretical Computer Science. Chennai: Springer, 2004. 97–109. [doi: 10.1007/978-3-540-30538-5_9]

[6] Haase C, Ishtiaq S, Ouaknine J, Parkinson MJ. SeLoger: A tool for graph-based reasoning in separation logic. In: Proc. of the 25th Int'l Conf. on Computer Aided Verification. Saint Petersburg: Springer, 2013. 790–795. [doi: 10.1007/978-3-642-39799-8_55]

[7] Brotherston J, Fuhs C, Pérez JAN, Gorogiannis N. A decision procedure for satisfiability in separation logic with inductive predicates. In: Proc. of the Joint Meeting of the 23rd EACSL Annual Conf. on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symp. on Logic in Computer Science. Vienna: ACM, 2014. 25. [doi: 10.1145/2603088.2603091]

[8] Iosif R, Rogalewicz A, Vojnar T. Deciding entailments in inductive separation logic with tree automata. In: Proc. of the 12th Int'l Symp. on Automated Technology for Verification and Analysis. Sydney: Springer, 2014. 201–218. [doi: 10.1007/978-3-319-11936-6_1510.1007/978-3-319-11936-6_15]

[9] Pérez JAN, Rybalchenko A. Separation logic modulo theories. In: Proc. of the 11th Asian Symp. on Programming Languages and Systems. Melbourne: Springer, 2013. 90–106. [doi: 10.1007/978-3-319-03542-0_7]

[10] Khoo SC, Chin WN, Ta QT, Le TC, Aishwarya S, Nguyen TT. The Songbird prover: An inductive theorem prover for separation logic entailments. https://songbird-prover.github.io

[11] Enea C, Lengál O, Sighireanu M, Vojnar T. Compositional entailment checking for a fragment of separation logic. In: Proc. of the 12th Asian Symp. on Programming Languages and Systems. Singapore: Springer, 2014. 314–333. [doi: 10.1007/978-3-319-12736-1_17]

[12] Le QL. S2S: An efficient solver for separation logic. https://loc.bitbucket.io/s2s/index.html

[13] Gu XC, Chen TL, Wu ZL. A complete decision procedure for linearly compositional separation logic with data constraints. In: Proc. of the 8th Int'l Joint Conf. on Automated Reasoning. Coimbra: Springer, 2016. 532–549. [doi: 10.1007/978-3-319-40229-1_36]

[14] Gao C, Chen TL, Wu ZL. Separation logic with linearly compositional inductive predicates and set data constraints. In: Proc. of the 45th Int'l Conf. on Current Trends in Theory and Practice of Informatics. Nový Smokovec: Springer, 2019. 206–220. [doi: 10.1007/978-3-030-10801-4_17]

[15] Gu XC. A complete decision procedure for linearly compositional separation logic with data constraints: Theory and tool [MS. Thesis]. Beijing: University of Chinese Academy of Sciences, 2017 (in Chinese with English abstract).

[16] Gao C. Separation logic with linearly compositional inductive predicates and set data constraints [MS. Thesis]. Beijing: University of Chinese Academy of Sciences, 2019 (in Chinese with English abstract).

[17] MONA. https://www.brics.dk/mona/

[18] SMT-LIB 2. http://smtlib.cs.uiowa.edu/

[19] SL-COMP 2019, 2019. https://sl-comp.github.io

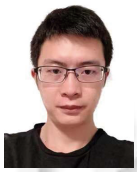
[20] Xu ZW, Chen TL, Wu ZL. Satisfiability of compositional separation logic with tree predicates and data constraints. In: Proc. of the 26th Int'l Conf. on Automated Deduction. Gothenburg: Springer, 2017. 509–527. [doi: 10.1007/978-3-319-63046-5_31]

附中文参考文献:

[15] 古新才. 带线性可组合归纳谓词与数据约束的分离逻辑的判定程序: 理论研究与工具实现 [硕士学位论文]. 北京: 中国科学院大学, 2017.

[16] 高冲. 带线性可组合归纳谓词和集合数据约束的分离逻辑公式求解 [硕士学位论文]. 北京: 中国科学院大学, 2019.

苏婉昀(1997－), 女, 硕士, 主要研究领域为分离逻辑约束求解.



古新才(1991－), 男, 硕士, 主要研究领域为分布式计算.

高冲(1994－), 男, 硕士, 主要研究领域为编程语言.

吴志林(1980－), 男, 博士, 研究员, 博士生导师, CCF 专业会员, 主要研究领域为自动推理, 程序验证, 自动机理论.