arXiv:2602.08532v1 [cs.LO] 9 Feb 2026

第 1 章

程序验证中的 Craig 插值

Philipp Rümmer 德国雷
根斯堡大学，瑞典乌普萨拉大学

摘要

Craig 插值用于程序验证，以自动完成循环不变式推导和程序抽象计算等关键任务。本章介绍了过去几年在这一背景下发展起来的一些最重要的技术，重点关注两个方面：根据验证中使用的理论和数据类型推导 Craig 插值式，以及应用插值的验证算法的基本设计。

目录

1 引言 .2
1.1 非正式概述：使用 Craig 插值的不变式推断 3 1.2 本章结构.7

2 可满足性 与模理论插值.7
2.1 基本定义.7 2.1.1 一阶逻辑与理论.8 2.1.2 模理论的克雷格插值.9 2.1.3 Craig 插值的扩展形式.11 2.2 模线性算术插值.13 2.2.1 线性实数算术（LRA）.14 2.2.2 超越非严格不等式的合取.17 2.2.3 线性整数算术（LIA）.19 2.3 更多插值方法.23 2.3.1 基于证明的插值.23 2.3.2 基于图的插值.24 2.3.3 基于归约的插值.24

3 基于插值的验证算法.25
3.1 有界模型检测：有界模型检测.25 3.2 IMC：插值和基于 SAT 的模型检测.26

© 2025年 本文版权归其作者所有**草稿**最终版本将发表于 Balderten Cate、Jean Christoph Jung、Patrick Koopmann、Christoph Wernhard 和 Frank Wolter 编，*Craig* 插值理论与应用，第 XXXCHAPTER 章，第 XXXPFROM–XXXPTO 页。Ubiquity Press，2026年。XXXDOI。

2 程序验证中的 Craig 插值

3.2.1 IMC 算法.26 3.2.2 相关的基于插值的算法.28 3.3 CEGAR：反例引导的抽象精化.29 3.3.1 存在抽象.29 3.3.2 谓词抽象.30 3.3.3 反例引导的抽象精化.30 3.3.4 谓词抽象与 CEGAR 的变体.31 3.4 受约束霍恩子句：Craig 插值与受约束 Horn 子句.32 3.4.1 定义与基本性质.32 3.4.2 使用约束 Horn 子句编码递归程序.34

4 结论.35 致谢.36 参考文献.36

1 引言

在程序验证中，需要探讨如何证明（有效地，甚至自动地）给定的软件程序是否符合其规约。程序验证的研究与计算机科学一样历史悠久，多年来催生了大量不同的技术，从各个角度研究这一问题，并提出了在学术界和工业界都得到广泛应用的方法。

Craig 插值进入验证领域，源于人们发现插值可用于推导不变式 [49], ，由此开创了一种自动化验证中一项关键任务的新范式。这一发现催生了许多新的验证算法，最初主要着眼于硬件电路，的验证，为此主要采用命题逻辑中的插值；之后，这些方法又扩展到软件程序及其他系统类别的分析。Craig 插值这一新用途的发现，推动了针对多种不同逻辑（包括命题逻辑、一阶逻辑以及验证中相关的理论）开发更有效的插值算法和工具，也促使人们将 Craig 插值应用于验证以外的领域。如今，许多自动化验证工具都运用 Craig 插值来完成诸如推理控制流或数据、推导不变式及其他程序注释，或自动计算程序抽象等任务。

本章重点讨论与验证中的 Craig 插值相关的两个主要方面：针对程序验证中使用的某些理论计算 Craig 插值式的技术，以及应用插值的验证算法的基本设计。鉴于该领域的研究成果浩如烟海，本章并不试图全面综述所有成果，而是通过详细讲解并辅以示例，介绍一些主要方法，并为进一步探索基于插值的验证领域提供起点。

1.1 非正式概述：使用 Craig 插值进行不变式推断

我们首先通过一个示例说明如何使用 Craig 插值推断循环不变式。斐波那契数列 [25]由以下方程定义

$$F_0 = 0, \quad F_1 = 1, \quad F_{i+2} = F_i + F_{i+1}$$

其计算方式如图1a所示。循环执行第 i次迭代后，程序变量 `a` 存储斐波那契数 F_i，变量 `b` 存储数值 F_{i+1}。该函数的前置条件为 `n >= 0`，以 `requires` 子句的形式给出；后置条件为 `\result >= 0`，表示函数返回的结果 F_n 非负（“ensures”）。

形式化验证的目标是构造严谨的数学证明，证明程序符合其规约。为了验证 `fib`的部分正确性 [10]，必须证明：对于非负参数 n，只要函数以某个结果 a终止，就能确保满足后置条件 $a \geq 0$。部分正确性证明通常通过对程序执行的循环迭代次数进行归纳来完成。为简化这一推理过程，人们引入了归纳循环不变式（或简称循环不变式）[10],，用于描述程序执行期间不会被破坏的性质。更准确地说，循环的归纳循环不变式是一个关于循环体作用域内程序变量的公式 I，并满足以下条件：

- 初始条件：每当进入循环时，不变式 I 都成立。保持条件：每当从满足不变式 I和循环条件的状态开始执行循环体时，执行完循环体后，不变式 I 也成立。充分性：每当循环在满足循环不变式 I的状态下终止时，程序的其余部分都能无错误地执行，并且后置条件得以满足。

对于我们的示例程序，这三个条件通过图1右侧的程序指令得到了形式化表达：`assert` 和 `assume` [21]。`assert`(ϕ) 的语义是，在当前程序状态下计算条件 ϕ，如果 ϕ 不成立，则引发错误；如果 ϕ 成立，`assert`(ϕ) 则不起作用，程序执行继续。`assume`(ϕ) 的语义与此类似，但如果 ϕ 不成立，该语句不会引发错误，而是暂停程序执行，阻止程序的其余部分继续执行。尤其是，语句 `assume(false)`会停止后续程序执行，其效果与非终止循环类似。为处理 `assert` 和 `assume`，我们将部分正确性的概念加以推广：如果无论程序输入和初始状态如何，程序中的任何 部分正确 语句都不会失败，则称该程序是 `assert`。在这个例子中，充分的归纳不变式恰好是这样的公式：对于这些公式，图1b至图1d中的三个程序都是部分正确的。

我们可以观察到，这三个条件均由公式 $I = \mathtt{a} \geq 0 \wedge \mathtt{b} \geq 0$ 满足。因此，该公式 I 是一个循环不变式，并且是 `fib` 部分正确性的见证。较简单的公式 $\mathtt{a} \geq 0$ 虽然是在任意

```
/**
 * requires n >= 0;
 * ensures  \result >= 0;
 */
int fib(int n) {
  int a = 0;
  int b = 1;
  int t;

  for (int i = 0; i < n; ++i) {
    t = b;
    b = a + b;
    a = t;
  }

  return a;
}
```

(a) 计算斐波那契数列第 n 项的程序。

```
assume(n >= 0); // requires
int a = 0; int b = 1; int t;
int i = 0;
assert(I);
```

(b) 初始条件

```
assume(I);
assume(i < n);
t = b;
b = a + b;
a = t;
++i;
assert(I);
```

(c) 保持条件

```
assume(I);
assume(!(i < n));
assert(a >= 0); // ensures
```

(d) 充分性

图 1 斐波那契程序及归纳循环不变式的条件。

程序的执行也不满足保持条件，因此它不是正确的归纳不变式。更准确地说，在图 1c 中将 $\mathtt{a} \geq 0$ 替换为 I 后，我们得到的程序并非部分正确；当程序从状态 $\mathtt{a} = 0, \mathtt{b} = -1, \mathtt{i} = 0, \mathtt{n} = 1$ 开始运行时，会发生断言失败。

提出归纳循环不变式（例如公式 $\mathtt{a} \geq 0 \wedge \mathtt{b} \geq 0$）是验证中的一项已知难题，原因与在证明中找到归纳公式并不容易相同。Craig 插值是一种实用且应用广泛的技术，可用于寻找不变式候选。为了明确 Craig 插值与循环不变式之间的联系，请考虑图2a中斐波那契程序的展开：它通过列出执行路径上执行两次循环体的语句而得到。添加到程序中的假设描述了循环条件的状态：在两次循环迭代期间为真，之后为假，并确保图2a中程序的执行与图1a中原始程序执行两次循环迭代时完全对应。通过循环展开来获得无循环程序，是验证中的常见技术。

那么，考虑可能的中间断言 K_0，K_1，K_2 它们使我们能够在标记位置将图 2a 中的程序分解为多个部分。若程序 $P;Q$ 部分正确，且程序 $P;\mathtt{assert}(A)$，则称公式 A 是该程序的中间断言

```
assume(n >= 0);
int a = 0;
int b = 1;
int t;
int i = 0;

// Intermediate assertion K0

assume(i < n); // 1st iteration
t = b; b = a + b; a = t; ++i;

// Intermediate assertion K1

assume(i < n); // 2nd iteration
t = b; b = a + b; a = t; ++i;

// Intermediate assertion K2

assume(!(i < n)); // Loop exit
assert(a >= 0);
```

(a) 循环展开。

```
assume(n >= 0); // requires
int a = 0; int b = 1; int t;
int i = 0;
assert(K0);
```

(b) 子程序 1

```
assume(K0);
assume(i < n);
t = b; b = a + b; a = t; ++i;
assert(K1);
```

(c) 子程序 2

```
assume(K1);
assume(i < n);
t = b; b = a + b; a = t; ++i;
assert(K2);
```

(d) 子程序 3

```
assume(K2);
assume(!(i < n));
assert(a >= 0); // ensures
```

(e) 子程序 4

图 2 展开斐波那契程序中循环的两次迭代。

和 `assume(`A`);`Q 也都是部分正确的；换言之，如果 A 是由第一个子程序 P确保的后置条件，同时也是确保第二个子程序 Q无错误执行的前置条件。图 2a 中的三个中间断言同样将程序拆分为四个子程序，如图 2 右侧所示；每个子程序都必须部分正确，中间断言才算合适。

使用 Craig 插值式寻找归纳循环不变式，基于两个关键观察：

(i) 中间断言是归纳循环不变式的一种松弛形式；**(ii)** 中间断言是 Craig 插值式的一种情形。

为说明第 (i) 点，我们可以比较图 1 中的条件与图 2 中给出的条件。条件 1b 和 2b 彼此对应，条件 1d 和 2e 也是如此，而后继条件 1c 仅由条件 2c 和 2d 近似表示。因此，将任何满足图 1 中条件的公式 I 代入 K_0, K_1, K_2 后，它也将满足图 2 中的条件，并产生程序的正确分解。

```
1 assume(n >= 0);2  int a = 0,
  b
4
5
67
89
10
11
12
```

(a) 计算单个中间断言。

$$
\left.\begin{array}{l} n_0 \geq 0 \wedge \\ a_0 = 0 \wedge b_0 = 1 \wedge i_0 = 0 \wedge \\ i_0 < n_0 \wedge \\ t_1 = b_0 \wedge b_1 = a_0 + b_0 \wedge a_1 = t_1 \wedge \\ i_1 = i_0 + 1 \end{array}\right\} A
$$

$$
\wedge
$$

$$
\left.\begin{array}{l} i_1 < n_0 \wedge \\ t_2 = b_1 \wedge b_2 = a_1 + b_1 \wedge a_2 = t_2 \wedge \\ i_2 = i_1 + 1 \wedge \\ i_2 \not< n_0 \end{array}\right\} C
$$

$$
\rightarrow a_2 \geq 0
$$

(b) 部分正确性的验证条件。

图 3 中间断言。

反过来说，并非所有可能的中间断言 K_0，K_1，K_2 都是正确的循环不变式，因为中间断言通常不满足保持条件 1c。不过，若有一种方法可以探索可能的中间断言 K_0，K_1，K_2，我们就能生成归纳循环不变式的候选项 。我们将在第 3 节中介绍一些已提出的算法，这些算法以这种方式生成和改进不变式候选。

为说明要点 (ii)，我们聚焦于这样一种情况：对于图 3a 所示的展开斐波那契程序，只需计算一个中间断言。通过将程序（包括 `assert` 和 `assume` 语句）转换为 验证条件 [10], ，即可检查程序的部分正确性；本例中的验证条件是一个线性整数算术公式。验证条件如图 3b 所示，其构造方式确保该条件有效 当且仅当程序部分正确。 为此，该条件列出程序中所作的所有假设，以方程形式包含对变量的赋值，并将断言作为最后一个不等式 $a_2 \geq 0$。由于程序变量在程序执行过程中可能改变其值，因此一个程序变量可能对应于验证条件中的多个符号：例如，程序变量 `a` 会变成逻辑常数 a_0，a_1，a_2。

通过将图 3b 中的验证条件与我们对中间断言的定义进行比较，我们可以看到，正确的中间断言 K 对应于满足蕴含关系 $A \rightarrow \phi$ 和 $\phi \rightarrow C$ 的公式 ϕ，其中 A，C，如图 3b 所示。事实上， K 必须是一个公式，它由程序中相应位置之前的程序语句所蕴含，这由蕴含关系 $A \rightarrow \phi$ 表示；同时，它还必须足够强，以确保程序的其余部分能够正确执行，因此有 $\phi \rightarrow C$。这已经表明，中间断言对应于*Craig* 插值式

对于蕴含 $A \to C$，因为插值式必须满足相同的蕴含条件。

为了进一步弥合中间断言与 Craig 插值式之间的差距，我们必须考虑哪些符号可以出现在中间断言 K中。公式 K 应当是由程序变量 `n`、 `a`、 `b`、 `t`、 `i`构成的表达式。在验证条件中，这些变量已被替换为不同的带索引版本；在应当对 K 求值的位置（图 3a 第 7 行），最新版本分别记为 n_0、a_1、b_1、t_1、i_1。自然地，这些恰好就是同时出现在 A 和C中的符号，因为变量所有更早的旧版本只会出现在 A中，而未来版本则只会出现在 C中。因此，蕴含 $A \to C$ 的 Craig 插值式恰好只涉及正确的一组符号 n_0、a_1、b_1、t_1、i_1 。要为我们的程序推导中间断言，我们可以先为 $A \to C$计算 Craig 插值式 ϕ，然后重命名这些符号，从而得到中间断言 $K = \phi[n_0 \mapsto \texttt{n}, a_1 \mapsto \texttt{a}, b_1 \mapsto \texttt{b}, t_1 \mapsto \texttt{t}, i_1 \mapsto \texttt{i}]$。

本章稍后将回到斐波那契示例，届时会讨论如何针对程序相关的理论计算 Craig 插值式。届时我们推导出的一个可能的插值式为公式 $\phi = (b_1 \geq 1)$，它可以转换为中间断言 $K =$(`b` ≥ 1)。

1.2 本章结构

本章其余内容分为两个主要部分。第 2 节将介绍逻辑方面所需的背景知识，并正式定义 Craig 插值的若干概念。随后，我们将介绍验证领域中为计算 Craig 插值式而发展出的一些技术，重点讨论对验证至关重要的算术公式的插值式。第 3 节作为第二部分，将通过探讨若干基于插值的验证算法，展示插值过程的应用。

2 理论下的可满足性与插值

2.1 基本定义

验证中的自动化方法通常采用一阶逻辑中的无量词理论 设置，其中理论的选择旨在反映编程语言中使用的数据类型，而对无量词公式的限制则是为了确保其属于可判定语言。为了对模理论的无量词公式进行推理，研究者提出了模理论的可满足性 （SMT）范式 [5]，将 SAT 求解器与理论求解器相结合，从而获得高效的判定过程。同样，验证领域中关于 Craig 插值的大量研究都聚焦于计算模理论的无量词 Craig 插值式的方法。

2.1.1 一阶逻辑与理论

我们假设采用多排序（经典）一阶逻辑的框架，例如参见 [32]。在第 2 章引入的记法基础上稍作扩展，现在将签名 Σ定义为由关系符号、常量符号、函数符号以及排序组成的集合，其中每个谓词 R 都具有唯一确定的秩 $\alpha(R) = (\sigma_1, \ldots, \sigma_n)$，其内容为参数排序；每个常量符号和函数符号 f也具有一个秩 $\alpha(f) = (\sigma_1, \ldots, \sigma_n, \sigma_0)$，由参数排序 $\sigma_1, \ldots, \sigma_n$和结果排序 σ_0组成。公式 ϕ和项 t由以下语法定义：

$$\phi ::= R(t, \ldots, t) \mid t = t \mid \top \mid \phi \wedge \phi \mid \neg\phi \mid \exists x : \sigma. \phi \ , \qquad t ::= x \mid f(t, \ldots, t) \ .$$

在语法中， R 表示关系符号， f 表示函数符号， x 表示变量， σ 表示排序。常数被视为零元函数。我们通常假设项和公式都是类型正确的，这意味着关系符号和函数符号只能作用于排序正确的项，而且 $=$ 只能用于比较同一排序的项。我们还使用简写记号 $\bot$、 $\phi \vee \phi$、 $\phi \to \phi$和 $\forall x :\ \sigma. \phi$，它们的定义均为通常定义。给定公式 ϕ 或项 t，我们分别用 sig(ϕ) $\subseteq \Sigma$ 和 sig(t) $\subseteq \Sigma$表示其中出现的关系符号集和函数符号集。

一个闭公式 或句子 是所有变量都受约束的公式。一个基公式 是不包含任何变量（因此也不包含量词）的公式。请注意，基公式仍然可以包含常量符号。一个无量词 公式是不包含量词但可能包含变量的公式。

公式的语义通常是根据 Σ-结构 S 来定义的：它将 Σ 中的每个排序 σ 映射到某个非空论域 D_σ，将 Σ 中的每个关系符号映射到定义在参数论域上的集合论关系，并将 Σ 中的每个函数符号映射到从参数论域到结果论域的集合论函数。为了定义公式关于理论的可满足性，我们需要签名和结构的扩展这一概念。我们称签名 Ω 是 Σ的扩展，记作 $\Sigma \subseteq \Omega$，若 Ω 比 Σ 包含更多的排序、关系符号或函数符号，同时不改变 Σ中已有符号的秩 α（R）或 α（f）。如果 $\Sigma \subseteq \Omega$，那么当一个 Ω-结构 S_Ω以相同方式解释 Σ中已有的排序和符号，同时为 Ω 中的新排序和符号添加解释时， Σ-结构 S_Σ 就被 S_Ω 一个 Ω-结构所扩展。

▶**定义 1**（理论）。 一个理论 T 由签名 Σ_T 和一组 S_T 上的结构定义。若签名 Σ_T上的句子 ϕ在 S_T中的某个结构上取值为真，则称其为 T-可满足（或 T-sat）。若扩展签名 $\Omega \supseteq \Sigma_T$上的句子 ϕ在 S_T 中某个结构扩展到 Ω 后取值为真，则称其为 T-可满足。

对于某个比 Σ_T 更大的签名 Ω 中的句子，这意味着额外的排序以及关系或函数符号都可以任意解释。作为一种特殊情况，若句子 ϕ 在空理论 $T_\emptyset$中是 $T_\emptyset$-可满足的，我们就称它为简单地可满足。不是 T-可满足的句子称为 T-不可满足（或 T-不可满足），而其否定是

T-不可满足的句子称为 T-有效。如果存在一个共同结构满足集合 M 中的所有句子，我们也称该句子集合是 T-可满足的。

我们将 $\Gamma \models_T \psi$ 记作：句子集合 Γ 在模 T 的意义下蕴涵句子 ψ；也就是说，只要 $\mathcal{S}_T$ 中某个结构的扩展满足 Γ 中的所有句子，它也满足 ψ。作为一种特殊情况，我们将 $\phi \models_T \psi$ 记作：句子 ϕ 在模 T 的意义下蕴涵句子 ψ；这等价于 $\{\phi\} \models_T \psi$，也等价于合取式 $\phi \wedge \neg\psi$ 是 T-不可满足的。我们将 $\phi \equiv_T \psi$ 用于表示两个句子是 T-等价的，其定义为 $\phi \models_T \psi$ 和 $\psi \models_T \phi$；上下文清楚时，我们省略 T。

2.1.2 模理论 Craig 插值

由于验证中的算法（与自动推理中的算法一样）通常以可满足或不可满足公式来表述，而不是以有效或反可满足公式来表述，因此该领域通常也会以“反向”方式定义 Craig 插值。标准 Craig 插值式与分离式之间的关系已在第 1 章中讨论，说明至少在经典逻辑中，这两个概念之间的差异大多只是表面上的。为避免与文献中的定义混淆，本章大部分内容都采用这种反向插值版本，同时也出于完备性考虑介绍标准版本。

▶ **定义 2**（理论模 Craig 插值式）。假设 T 是一个理论，$A \to C$ 是一个句子。如果句子 I 满足以下条件，则称其为关于 $A \to C$ 的 理论模 Craig 插值式 T：

- $A \models_T I$ 且 $I \models_T C$，并且sig(I)
- $\subseteq$ (sig(A) $\cap$ sig(C)) $\cup \Sigma_T$。

与第 2 章中一阶逻辑的 Craig 插值式定义相比，我们现在考虑关于理论 T 的蕴含 $\models_T$，并允许插值 I 包含理论签名 Σ_T 中任意的解释符号。另一个更为技术性的区别是，我们在定义中只考虑闭公式（句子）；这一限制很容易解除，但通常并无问题，因为自由变量可以替换为常量符号。除此之外，第 2 章中的定义在空理论 $T_\emptyset$ 这一特殊情况下与我们的定义相对应。

▶ **例 3**。考虑蕴含 $A \to C$，其中 A，C 在图 3b 中根据线性整数算术（LIA）理论定义。关于 LIA 的正确 Craig 插值式包括$I_1 = b_1 \geq 1$ 和 $I_2 = a_1 \geq 0 \wedge b_1 \geq 0$。⌟

如第 1 章所述，一个插值问题 $A \to C$ 可以等价地表示为一个合取，从而引出另一种反向的插值概念：

▶ **定义 4** (分隔符，反向克雷格插值式)。假设 T 是一个理论，$A \wedge B$ 是一个句子。若句子 I 满足以下条件，则称其为 T-分隔符 或 模 T 的反向克雷格插值式$A \wedge B$：

- $A \models_T I$ 并且 $I \wedge B \models_T \bot$，且
- $\text{sig}(I) \subseteq (\text{sig}(A) \cap \text{sig}(B)) \cup \Sigma_T$。

▶ **定理 5** （存在 T-分离式 [44]）。设 T 是一个理论，$A \wedge B$ 是一个 T-不可满足的句子。那么存在一个 T-分离式 $A \wedge B$；更确切地说，存在T-分离式 I_1、I_2，使得：

- $A \models_T I_1$ 且 $I_1 \wedge B \models \bot$，$A \models I_2$ 且
- $I_2 \wedge B \models_T \bot$，$\text{sig}(I_1) \subseteq (\text{sig}(A) \cup \Sigma_T)$
- $\cap\, \text{sig}(B)$，且$\text{sig}(I_2) \subseteq \text{sig}(A) \cap (\text{sig}(B)$
- $\cup\, \Sigma_T)$。

该定理关于分离式存在性的结果比定义 4 所要求的更强。定理中的句子 I_1、I_2 根据定义 4 都是 T-分离式，但我们甚至可以假设，两个蕴含条件中的一个是一般 蕴含 $\models$，而另一个蕴含则是模理论 T下的蕴含。类似地，该定理对 I_1、I_2中所用签名提供了更强的保证，因为公式只需涉及 Σ_T 中的解释符号，且这些符号也出现在两个公式 A、B之一中。

尽管定理 5 为反向插值式的存在性提供了一般性保证，但它通常不足以满足验证应用的需求，因为该定理并未对 I_1 和 I_2中是否出现量词 作出任何说明。一般而言，即使公式 A, B 是无量词的，也可能只有包含量词的 T-分离式，而这通常不太适用于实际应用。

▶ **示例 6.** McCarthy 提出了一种函数数组理论，如今广泛用于程序验证 [48]。该理论提供了函数$select(M, a)$，用于读取数组 M 在索引 a处的值；还提供了函数$store(M, a, x)$，用于将数组 M 在索引 a 处更新为新值 x。在数组理论中，合取

$$\underbrace{M' = store(M, a, x)}_{A} \wedge \underbrace{b \neq c \wedge select(M, b) \neq select(M', b) \wedge select(M, c) \neq select(M', c)}_{B}$$

不可满足，但所有反向插值式都包含量词 [51]。事实上，插值式只能引用符号 M，M，这些符号同时出现在 A和 B中，以及数组函数$select$和$store$。一个带量词的插值式为 $\exists a, x.\, M' = store(M, a, x)$。」

▶**定义 7 (**一般无量词插值 [14])。 理论 T 具有一般无量词插值性质，当且仅当对于每个 T-不可满足合取式 $A \wedge B$，都存在一个无量词T-分隔符。

如示例所示，标准数组理论不具有一般无量词插值性质，而许多其他理论（例如线性实数算术（LRA），第 2.2.1 节）具有这一性质。不过，可以扩展数组理论，从而使其具备无量词插值性质 [13, 62, 36] （另见第 2.3.1 节）。

考虑一种稍弱的无量词插值判据通常很有用：

▶ **定义 8** (普通无量词插值 [14])。 如果对于每个 T-不可满足的合取式 $A \wedge B$ （其中仅包含来自 Σ_T 的符号以及常量符号），都存在一个无量词 T-分隔符，则称理论 T 满足普通无量词插值。

“一般”与“普通”的区别在于，前者要求即使合取中含有不属于 T 的关系或函数符号 $A \wedge B$，也要给出无量词插值式；而后者只要求额外包含常量符号即可。

▶ **示例 9.** 线性整数算术（LIA）理论能够得到普通无量词插值式，前提是该理论包含针对每个 $k \in \mathbb{N}$ 的整除语句 $k \mid \cdot$；公式 $k \mid t$ 读作“k 整除 t”，并断言 t 是 k 的倍数。普通无量词插值之所以成立，是因为我们注意到，可以通过量词消去 [32], 来计算 LIA 中的最弱和最强插值，其方式与第 1 章中命题逻辑所示的完全相同。LIA 是否具有一般无量词插值，取决于所假定理论的确切签名。为说明这一点，请考虑以下不可满足的合取

$$\underbrace{x = 2 \cdot y \wedge p(y)}_{A} \wedge \underbrace{x = 2 \cdot z \wedge \neg p(z)}_{B}$$

其中 p 是一个关系符号，不属于线性整数算术的签名。合取式 $A \wedge B$ 的一个分离式是公式 $I = p(x \div 2)$，其中 $\div$ 表示带余除法。如果函数 $\div$ 不包含在线性整数算术的签名中，则不存在无量词分离式 [11],，且该理论不再具有一般无量词插值性质。」

2.1.3 Craig 插值的扩展形式

上一节定义的分离式（或反向 Craig 插值式）有时称为二元插值，因为其中只涉及两个公式 A，B。还可以定义相应的 n元版本，以分离任意数量的公式，这在验证中往往很有优势。如今使用的两种主要扩展插值概念，且通常受 SMT 求解器支持，分别是序列插值和树插值。

▶ **定义 10** （序列插值式 [34, 52]）。设 T为一个理论，$A_1 \wedge \cdots \wedge A_n$ 为一个句子。对于 $A_1 \wedge \cdots \wedge A_n$，一个模 T 序列插值或模 T 归纳插值式序列是一个句子序列 I_0、I_1、.....、I_n，满足：

- $I_0 = \top, I_n = \bot$，对于所有 $i \in \{1, \ldots, n\}$： $I_{i-1} \wedge A_i \models_T I_i$；对于所有 $i \in \{0, \ldots, n\}$： $\mathrm{sig}(I_i) \subseteq (\mathrm{sig}(A_1 \wedge \cdots \wedge A_i) \cap \mathrm{sig}(A_{i+1} \wedge \cdots \wedge A_n)) \cup \Sigma_T$。

容易看出，I 的分隔符 $A_1 \wedge A_2$ 对应于序列插值 $\top$，I，$\bot$，因此序列插值严格扩展了二元插值。需要

同时指出，序列插值中的各个公式彼此相关，因为每个公式 I_{i-1} 都必须足够强，能够与 A_i 一起推出下一个公式 I_i。序列插值可以通过反复计算二元插值得出：我们设定$I_0 = \top$，然后通过为合取（$I_{i-1} \wedge A_i$）$\wedge$（$A_{i+1} \wedge \cdots \wedge A_n$）计算分隔符，来计算每个公式 I_i，其中 $i = \{1, \ldots, n\}$。不过，出于效率考虑，这种迭代计算序列插值的方式很少使用；相反，SMT 求解器会实现能够一次性计算整个序列插值的插值过程。

▶ **示例 11.** 正如我们在第 1.1 节和图 3b 中观察到的，通过将程序路径上的语句转换为蕴含式 $A \to C$，可以利用二元插值式推导出程序的单个中间断言。我们可以使用序列插值来推导程序中多个连续的断言。为此，再次考虑图 2 所示的斐波那契示例版本，其中循环展开了两次，目的是找到中间断言 K_0，K_1，K_2。程序中中间断言的位置表明，现在应将程序语句分组，从而得到四个合取项；对断言 `a >= 0` 取反，以表明我们关注的是违反该断言的运行：

$$\underbrace{\begin{array}{l} n_0 \geq 0 \wedge a_0 = 0 \wedge \\ b_0 = 1 \wedge i_0 = 0 \end{array}}_{A_1} \wedge \underbrace{\begin{array}{l} i_0 < n_0 \wedge t_1 = b_0 \wedge \\ b_1 = a_0 + b_0 \wedge \\ a_1 = t_1 \wedge i_1 = i_0 + 1 \end{array}}_{A_2} \wedge \underbrace{\begin{array}{l} i_1 < n_0 \wedge t_2 = b_1 \wedge \\ b_2 = a_1 + b_1 \wedge \\ a_2 = t_2 \wedge i_2 = i_1 + 1 \end{array}}_{A_3} \wedge \underbrace{i_2 \not< n_0 \wedge a_2 \not\geq 0}_{A_4}$$

合取式 $A_1 \wedge A_2 \wedge A_3 \wedge A_4$ 不可满足，序列插值由以下公式给出

$$\top,\ a_0 + b_0 \geq 1,\ b_1 \geq 1,\ a_2 \geq 1,\ \bot\ .$$

序列插值中的公式可以转换为程序的中间断言：$K_0 =$ `a + b >= 1`，$K_1 =$ `b >= 1`，$K_2 =$ `a >= 1`。」

树插值式推广了归纳插值序列，并针对过程间程序验证而设计。在此语境中，插值问题的树结构对应于程序的调用图。树插值式不仅能够推导程序中的中间断言，还能够推导关联多个程序状态的注解，例如后置条件 [33, 30]（另见第 3.4.2 节）。

▶ **定义 12**（树插值式 [33, 59, 31]）。设 T 是一个理论，且 $G =$（V，E）为一棵有限有根树，并记 E（v，w）表示节点 w 是 v的直接子节点。此外，设树的每个节点 $n \in V$ 都标有一个句子 A（n）。若函数 I 为每个节点 $n \in V$ 标注一个句子 I（n），则当下列性质成立时，称其为树插值式模 T（对于 G 与 A）：

(i) 对于根节点 $v_0 \in V$，情况是 $I(v_0) = \bot$，

(ii) 对于每个节点 $v \in V$，以下蕴含关系成立：

$$\bigwedge_{\substack{w \in V \\ (v,w) \in E}} I(w) \wedge A(v) \models_T I(v) ,$$

(iii) 对于每个节点 $v \in V$，以下条件成立：

$$\mathrm{sig}(I(v)) \subseteq \Big(\bigcup_{\substack{w \in V \\ E^*(v,w)}} \mathrm{sig}(A(w)) \cap \bigcup_{\substack{w \in V \\ \neg E^*(v,w)}} \mathrm{sig}(A(w)) \Big) \cup \Sigma_T$$

其中 E^* 是 E 的自反传递闭包。

与序列插值式的情况类似，树插值式也可以通过反复计算二元插值 [59]，得出，不过 SMT 求解器通常也提供对树插值式的原生支持。

树插值概括了迄今为止我们遇到的所有更专门的插值形式。要了解树插值如何涵盖序列插值，请注意，n 元合取 $A_1 \wedge \cdots \wedge A_n$ 可以转换为一棵树 $G_1 = (\{1, \ldots, n\}, \{(i, i-1) \mid 2 \leq i \leq n\})$，其中每个节点 i 都是节点 $i+1$ 的子节点，并标记为公式 A_i。树的根节点为 n。类似地，我们可以将第 1 章介绍的并行插值 [43, 54]，转换为树插值：将蕴含 $\varphi_1 \wedge \cdots \wedge \varphi_n \rightarrow \psi$ 排列成一棵树 $G_2 = (\{0, 1, \ldots, n\}, \{(0, i) \mid 1 \leq i \leq n\})$，其中根节点 0 标记为 $\neg\psi$，每个子节点 i 则标记为公式 φ_i。并行插值与对称插值式 [39], 的概念相关，也可以用类似方法将其转换为树插值。

树插值式还可以进一步推广为析取插值式 [58], ，但后者尚未在验证领域得到更广泛的应用。人们还注意到，不同的插值变体对应于求解受约束Horn子句 [59], 的特定无递归片段，这一联系将在第3.4节中回到。

2.2 插值模线性算术

现在，我们概述一些为理论 T 构造无量词插值式的方法，重点关注验证中无处不在的线性算术情形。对于算术公式，通常会在某种合适的证明系统中从证明中提取插值式，这与前面章节讨论的方法相似。对于验证而言，从证明中提取插值式很有吸引力，因为证明往往聚焦于验证条件中的相关信息。使用基于证明的插值计算中间断言（如第 1.1 节所述）时，只会提及对 SMT 求解器证明程序正确性至关重要的程序变量。因此，从证明中得到的插值式是计算程序抽象的实用工具。

2.2.1 线性实数算术（线性实数算术）

支持插值的 SMT 求解器所支持的最常见理论之一，是关于实数或整数 [61, 20] 的线性算术。在线性实数算术（Linear Real Arithmetic，LRA）理论中，我们考虑包含线性不等式 $0 \leq \alpha_1 x_1 + \cdots + \alpha_k x_k + \alpha_0$ 的公式，其中$\alpha_0, \alpha_1, \ldots, \alpha_k \in \mathbb{Q}$ 是有理数，$x_1, \ldots, x_k$ 是 Real 类型的常量符号。乘以某个有理数 αx、加法 $+$ 以及关系符号 $\leq$ 的语义均符合预期。起初，我们只考虑由正不等式 $0 \leq \alpha_1 x_1 + \cdots + \alpha_k x_k + \alpha_0$ 合取而成的公式。实值项之间的方程 $s = t$ 可以转换为两个不等式的合取 $0 \leq s - t \wedge 0 \leq t - s$；而严格不等式 $0 < s$、析取和否定方程的处理将在第 2.2.2 节讨论。

为对线性不等式进行插值，我们依赖线性规划中的一个经典结果：

▶ **引理 13** (Farkas 引理 [61])。设 $A \in \mathbb{R}^{m \times n}$ 是一个实值矩阵，$b \in \mathbb{R}^m$ 是一个列向量。当且仅当对于每个满足 $y \geq 0$ 且 $yA = 0$ 的行向量 $y \in \mathbb{R}^m$，都有 $yb \geq 0$ 时，存在一个列向量 $x \in \mathbb{R}^n$，使得 $Ax \leq b$。

该引理给出了实数域上线性不等式不可满足性见证的一种简单、统一的表示：每当合取 $\bigwedge_i 0 \leq t_i$ 个线性不等式不可满足时，我们就能对这些不等式作非负线性组合 $0 \leq \sum_i \alpha_i t_i$，使右端 $\sum_i \alpha_i t_i$ 化简为一个负的常数值 $\beta < 0$。如果引理中的矩阵 A 和向量 b 只包含有理数，那么向量 y 也可以取为有理数。

▶ **引理 14**（线性不等式的插值式 [55, 51]）. 假设

$$\underbrace{\bigwedge_i 0 \leq t_i}_{A} \;\wedge\; \underbrace{\bigwedge_j 0 \leq s_j}_{B}$$

是线性不等式的合取，且线性组合 $\sum_i \alpha_i t_i + \sum_j \beta_j s_j$（其中 α_i、$\beta_j \geq 0$）为常数且为负。则 $I = \left(0 \leq \sum_i \alpha_i t_i\right)$ 在化简并去除系数为 0 的项之后，是 $A \wedge B$ 的 *LRA* 分隔式。

▶**例 15**。 我们首先用一个例子来说明这一结果。考虑合取

$$\underbrace{0 \leq y - 1 \wedge 0 \leq z - x - 2y - 2}_{A} \;\wedge\; \underbrace{0 \leq x \wedge 0 \leq -z + 2}_{B} .$$

该句子不可满足，因为我们可以构造这些不等式的一个非负矛盾线性组合：

$$0 \leq 2 \cdot (y - 1) + (z - x - 2y - 2) + x + (-z + 2) \;\equiv\; 0 \leq -2 .$$

因此，我们可以通过考虑 A 中不等式对应的线性组合，构造一个 LRA 分隔式 I。请注意，化简之前，该线性组合

仍然包含符号 y，而该符号并非 A 和 B 的共有符号。化简之后，这个符号消失了：

$$0 \leq 2 \cdot (y-1) + (z - x - 2y - 2) \equiv \underbrace{0 \leq z - x - 4}_{I} .$$

⌟

引理 14 的证明。根据构造，$A \models_{\mathrm{LRA}} I$ 和 $I \wedge B \models_{\mathrm{LRA}} \bot$ 成立。为说明 sig(I) $\subseteq$ (sig(A) $\cap$ sig(B)) $\cup \Sigma_{\mathrm{LRA}}$，注意到 I 只能包含出现在 A中的常量符号。现在假设某个常量符号 x 以非零系数出现在 $\sum_i \alpha_i t_i$ 中。由于 $\sum_i \alpha_i t_i + \sum_j \beta_j s_j$ 是常量，符号 x 也必然出现在 $\sum_j \beta_j s_j$中，因此 x 属于 sig(A) $\cap$ sig(B)。（译注：原文此处缺少句末内容。）

通常，我们用证明规则来定义这种计算 LRA 分离式的方法。为此，我们可以先考虑表 1a 中所示的证明规则，这些规则构成了一个用于推导一组不等式 [51]不可满足性的演算。这些证明规则以序列 $\Gamma \vdash \phi$的形式定义，其中 Γ 是一个有限的句子集合（在我们的例子中是不等式集合），而ϕ 是一个单独的句子。序列的语义是，Γ中的公式的合取蕴含 ϕ。证明规则表示，横线下方的序列（即结论）可以由横线上方的序列（即前提）推导出来。

Hyp 和 Comb 规则可以推导给定不等式的正线性组合 Γ。Con 规则用于检测不可满足的不等式，即常数项为负的常量不等式。Simp 规则借助重写关系 $\rightsquigarrow$简化不等式；实现方法可以是对不等式中的项排序，并删除系数为 0 [57]的所有项。图 4 给出了一个证明示例。

根据引理 13，表 1a 中的演算对于证明 LRA 不可满足性是可靠且完备的：当且仅当 Γ 是一个有限的 LRA 不可满足不等式集合时，才能推导出序列 $\Gamma \vdash \bot$ 。

为推导插值式，我们用注解 扩充证明规则；这些注解在证明树的根节点处化为 LRA 分隔式。由此得到的插值证明规则，在理念上类似于第 1 章介绍的命题逻辑插值证明规则；与命题逻辑的规则一样，插值式从证明的叶节点开始自底向上 构造。我们沿用 [51, 27], 中的插值证明规则风格，但对其表述进行了简化，因为目前不考虑与其他理论的组合。LRA 插值证明规则见表 1b。对于表 1a 中的每条证明规则，都有一条或多条插值规则，它们现在针对两类序列进行推理。较简单的一类序列形式为 (A, B) $\blacktriangleright I$，包含两个有限的句子集合 A, B，以及一个句子 I。当且仅当 I根据定义 4 是 $\bigwedge A \wedge \bigwedge B$的 LRA 分隔式时，序列 ($A$, B) $\blacktriangleright I$ 才有效。(A, B) $\blacktriangleright I$有效意味着 $A \cup B$ 是 LRA 不可满足的。

$$\textsc{Hyp}\ \frac{}{\Gamma \vdash 0 \le t}\ (0 \le t \in \Gamma) \qquad \textsc{Comb}\ \frac{\Gamma \vdash 0 \le s \qquad \Gamma \vdash 0 \le t}{\Gamma \vdash 0 \le \alpha s + \beta t}\ (\alpha,\ \beta > 0)$$

$$\textsc{Con}\ \frac{\Gamma \vdash 0 \le \alpha}{\Gamma \vdash \bot}\ (\alpha < 0) \qquad \textsc{Simp}\ \frac{\Gamma \vdash 0 \le s}{\Gamma \vdash 0 \le t}\ (s \rightsquigarrow t)$$

(a) 证明 ru　线性实数算术规则。箭头 $\rightsquigarrow$ 表示对表达式进行化简。

$$\text{Hyp-A}^I\ \frac{}{(A,\ B) \vdash 0 \le t\,[t]}\ (0 \le t \in A) \qquad \text{Hyp-B}^I\ \frac{}{(A,\ B) \vdash 0 \le t\,[0]}\ (0 \le t \in B)$$

$$\text{Comb}^I\ \frac{(A,B) \vdash 0 \le s\,[s'] \qquad (A,B) \vdash 0 \le t\,[t']}{(A,B) \vdash 0 \le \alpha s + \beta t\,[\alpha s' + \beta t']}\ (\alpha, \beta > 0)$$

$$\text{Con}^I\ \frac{(A,B) \vdash 0 \le \alpha\,[t']}{(A,\ B) \blacktriangleright 0 \le t'}\ (\alpha < 0) \qquad \text{Simp}^I\ \frac{(A,\ B) \vdash 0 \le s\,[s']}{(A,B) \vdash 0 \le t\,[t']}\ (s \rightsquigarrow t,\ s' \rightsquigarrow t')$$

(b) 线性实数算术的插值证明规则。

$$\text{Disj-A}^I\ \frac{(A \cup \{K_1\}\ B) \blacktriangleright I \qquad (A \cup \{K_2\}\ B) \blacktriangleright J}{(A \cup \{K_1 \vee K_2\}\ B) \blacktriangleright I \vee J}$$

$$\text{析取-B}^I\ \frac{(A, B \cup \{K_1\}) \blacktriangleright I \qquad (A, B \cup \{K_2\}) \blacktriangleright J}{(A, B \cup \{K_1 \vee K_2\}) \blacktriangleright I \wedge J}$$

(c) 析取的插值证明规则。

$$\textsc{BnB}\ \frac{\Gamma \cup \{0 \le n - x\} \vdash \bot \qquad \Gamma \cup \{0 \le x - n - 1\} \vdash \bot}{\Gamma \vdash \bot}\ (x \in \text{sig}(\Gamma)\ n \in \mathbb{Z})$$

$$\text{强化}\ \frac{\Gamma \cup \{t = 0\} \vdash \bot \qquad \Gamma \cup \{0 \le t - 1\} \vdash \bot}{\Gamma \cup \{0 \le t\} \vdash \bot}$$

$$\textsc{Div}\ \frac{\Gamma \vdash 0 \le \alpha_1 x_1 + \cdots + \alpha_k x_k + \alpha_0}{\Gamma \vdash 0 \le (\alpha_1/\beta)x_1 + \cdots + (\alpha_k/\beta)x_k + \lfloor \alpha_0/\beta \rfloor}\ (\beta = \gcd(\alpha_1, \ldots, \alpha_k)\ \beta > 0)$$

(d) 线性整数算术的证明规则。

$$\text{BnB-A}^I\ \frac{(A \cup \{0 \le n - x\}\ B) \blacktriangleright I \qquad (A \cup \{0 \le x - n - 1\}\ B) \blacktriangleright J}{(A, B) \blacktriangleright I \vee J}\ (x \in \text{sig}(A)\ n \in \mathbb{Z})$$

$$\text{BnB-B}^I\ \frac{(A,\ B \cup \{0 \le n - x\}) \blacktriangleright I \qquad (A, B \cup \{0 \le x - n - 1\}) \blacktriangleright J}{(A,\ B) \blacktriangleright I \wedge J}\ (x \in \text{sig}(B)\ n \in \mathbb{Z})$$

$$\text{Div}^I\ \frac{(A, B) \vdash 0 \le \alpha_1 x_1 + \cdots + \alpha_k x_k + \alpha_0\,[t']}{(A, B) \vdash 0 \le (\alpha_1/\beta)x_1 + \cdots + (\alpha_k/\beta)x_k + \lfloor \alpha_0/\beta \rfloor\,[t' \div \beta]}\ (\beta = \gcd(\alpha_1, \ldots, \alpha_k)\ \beta > 0)$$

（**e**）线性整数算术的插值证明规则。

表1 线性实数算术（LRA）和线性整数算术（LIA）的证明规则。

$$\textsc{Comb}\dfrac{\textsc{Simp}\dfrac{\textsc{Comb}\dfrac{\textsc{Simp}\dfrac{\textsc{Comb}\dfrac{\textsc{Hyp}\dfrac{}{\Gamma \vdash 0 \le z-x-2y-2} \quad \textsc{Hyp}\dfrac{}{\Gamma \vdash 0 \le x}}{\Gamma \vdash 0 \le (z-x-2y-2)+x}}{\Gamma \vdash 0 \le z-2y-2} \quad \textsc{Hyp}\dfrac{}{\Gamma \vdash 0 \le y-1}}{\Gamma \vdash 0 \le (z-2y-2)+2(y-1)}}{\Gamma \vdash 0 \le z-4}}{\mathcal{P}}$$

$$\textsc{Comb}\dfrac{\mathcal{P} \quad \textsc{Hyp}\dfrac{}{\Gamma \vdash 0 \le -z+2}}{\textsc{Simp}\dfrac{\Gamma \vdash 0 \le (z-4)+(-z+2)}{\textsc{Con}\dfrac{\Gamma \vdash 0 \le -2}{\Gamma \vdash \bot}}}$$

图 4 示例 15 中 $\Gamma = \{0 \le y-1,\ 0 \le z-x-2y-2,\ 0 \le x,\ 0 \le -z+2\}$ 的不可满足性证明。

第二种序列（也称为 不等式插值 [51]）的形式为（A, B） $\vdash 0 \le t\,[t']$，同样包含两组 A, B 句子、一条不等式 $0 \le t$和一个项 t'，这些共同构成注释。其直观含义是，这种序列对应于一个非插值序列 $A \cup B \vdash 0 \le t$，但除此之外，它还记录了从 A对$0 \le t$ 的贡献。如果满足以下条件，则序列（A, B） $\vdash 0 \le t\,[t']$ 是有效的：

- $A \models_{\mathrm{LRA}} 0 \le t'$,
- $B \models_{\mathrm{LRA}} 0 \le t - t'$，并且
- $\mathsf{sig}(t') \subseteq \mathsf{sig}(A) \cup \Sigma_{\mathrm{LRA}}$ 且 $t - t' \rightsquigarrow s$，其中 $\mathsf{sig}(s) \subseteq \mathsf{sig}(B) \cup \Sigma_{\mathrm{LRA}}$。

最后一个条件表示，t' 是一个仅包含来自 A 的符号的项，而差值 $t - t'$ 可以化简为一个仅包含来自 B 的符号的项（忽略 Σ_{LRA} 中的符号）。换言之，该注释将不等式 $0 \le t$ 拆分为一个 A 部分和一个 B 部分。表 1b 中的证明规则以这样的方式表述：每个可推导的序列都是有效的；为此，Hyp-A^I 和 Hyp-B^I 记录某个不等式是来自 A 还是来自 B，Comb^I 传播这些信息，而 Simp^I 则将注释 t' 与推导出的公式 $0 \le t$一并化简。规则 Con^I 得出一个 LRA 分隔式：如果已经推导出一个不可满足的不等式 $0 \le t$，则注释 t'会产生一个分隔符 $0 \le t'$。图 4 中证明的插值版本见图 5，并产生了与示例 15 中推理所得相同的分隔符 $0 \le z - x - 4$。

2.2.2 超越非严格不等式的合取

我们目前的讨论仅限于非严格不等式的合取 $0 \le t$ 和方程 $s = t$。然而，以插值证明规则来表述，可构建一个灵活的框架，并可扩展到包含更多运算符：

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma^I \vdash\ 0 \le z-x-2y-2\,[z-x-2y-2]}\,\textsc{Hyp-A}^I \quad \dfrac{}{\Gamma^I \vdash\ 0 \le x\,[0]}\,\textsc{Hyp-B}^I}{\Gamma^I \vdash\ 0 \le (z-x-2y-2)+x\,[(z-x-2y-2)+0]}\,\textsc{Comb}^I}{\Gamma^I \vdash\ 0 \le z-2y-2\,[z-x-2y-2]}\,\textsc{Simp}^I}{\mathcal{P}_1}$$

$$\dfrac{\dfrac{\mathcal{P}_1 \quad \dfrac{}{\Gamma^I \vdash\ 0 \le y-1\,[y-1]}\,\textsc{Hyp-A}^I}{\Gamma^I \vdash\ 0 \le (z-2y-2)+2(y-1)\,[(z-x-2y-2)+2(y-1)]}\,\textsc{Comb}^I}{\Gamma^I \vdash\ 0 \le z-4\,[z-x-4]}\,\textsc{Simp}^I}{\mathcal{P}_2}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{P}_2 \quad \dfrac{}{\Gamma^I \vdash\ 0 \le -z+2\,[0]}\,\textsc{Hyp-B}^I}{\Gamma^I \vdash\ 0 \le (z-4)+(-z+2)\,[(z-x-4)+0]}\,\textsc{Comb}^I}{\Gamma^I \vdash\ 0 \le -2\,[z-x-4]}\,\textsc{Simp}^I}{\Gamma^I \blacktriangleright\ 0 \le z-x-4}\,\textsc{Con}^I$$

图 5 图 4 中证明的插值版本。我们使用缩写 $\Gamma^I = (A,\ B) = (\{0 \le y-1,\ 0 \le z-x-2y-2\},\ \{0 \le x,\ 0 \le -z+2\})$。

- 布尔结构 可以按照第 5 章提出的原则，通过引入更多证明规则来处理公式。为确保完备性，我们在表 1c 中展示了用于拆分 A 集和 B 集中析取的插值证明规则。根据考察的是来自 A 还是 B 的公式，这些规则会在插值中引入析取或合取。也可以为合取和否定定义类似的规则。

  通常也可以将某个理论 T 的插值过程与第 1 章中的命题方法结合起来，从而得到能够处理任意无量词公式 [51] 的插值过程。插值 SMT 求解器采用的正是这种方法：它们将用于高效命题推理的 SAT 求解器与理论求解器结合起来，而理论求解器只处理理论文字的合取 [5]。

- 严格不等式 $0 < t$ 可以通过用 Motzkin 转置定理 [61], 替代 Farkas 引理来处理；该定理为严格不等式与非严格不等式的合取不可满足提供了完备判据。基于转置定理，可以提出一个类似于引理 14 的引理，用于处理同时包含严格不等式和非严格不等式的情况 [60]。另一种方法是为方程和未解释函数 [51], 添加单独的证明规则，然后利用这些规则，通过将严格不等式 $0 < t$ 改写为合取 $0 \le t \wedge 0 \ne t$来进行推理。

- 否定方程 $s \ne t$ 可以替换为析取 $0 < s-t \vee 0 < t-s$，然后便可使用析取和严格不等式的规则进行处理。

2.2.3 线性整数算术（LIA）

从实数转向整数时，计算 Craig 插值式会变得稍微复杂一些。好消息是，上一节介绍的证明规则对整数算术同样是可靠的；因此，使用表 1b 中的规则推导出的任何 LRA 分隔式也都是正确的 LIA 分隔式 [51]。因此，早期基于 Craig 插值的软件验证工具即使面对处理整数或有界机器整数的程序，也采用纯 LRA 推理。当然，LRA 证明规则并不能完备地处理整数推理，因为有些公式在实数范围内可满足，在整数范围内却不可满足。为了证明或插值这类公式，SMT 求解器会在 LRA 规则的基础上加入针对 LIA 的专用规则，但这些规则更难转换为插值规则。

对于线性整数算术，LIA，我们考虑可包含线性不等式 $0 \leq \alpha_1 x_1 + \cdots + \alpha_k x_k + \alpha_0$ 的公式，其中 $\alpha_0, \alpha_1, \ldots, \alpha_k \in \mathbb{Z}$ 现在表示整数，而 $x_1, \ldots, x_k$是类型为*Int*的常量符号；此外，还包括对每个 $k \in \mathbb{N}$ 成立的整除语句 $k \mid \cdot$（见示例 9）。这类公式的语义符合预期。

▶ **示例 16.** 考虑以下公式族（其中 $n \in \mathbb{N}$ $n > 1$）[45]:

$$A_n \;=\; 0 \leq y + 2nx + n - 1 \wedge 0 \leq -y - 2nx, \qquad B_n \;=\; 0 \leq y + 2nz - 1 \wedge 0 \leq -y - 2nz + n\,.$$

采用更直观的记法，这些公式可以写成 $A_n \equiv -n < y + 2nx \leq 0$ 和$B_n \equiv 0 < y + 2nz \leq n$，很容易看出，$A_n \wedge B_n$ 在实数范围内可满足，但在整数范围内不可满足（对任意 $n > 1$）。在等价意义下，$A_n \wedge B_n$ 的唯一 LIA 分隔式是以下句子:

$$I_n \;=\; (2n \mid y) \vee (2n \mid y + 1) \vee (2n \mid y + 2) \vee \cdots \vee (2n \mid y + n - 1)\,. \tag{1}$$

给定一个仅包含不等式和整除语句的线性整数算术签名，分隔式的大小与 n 成线性关系，因此相对于 A_n，B_n 的大小呈指数级增长。」

除了上一节中已经介绍过的线性实数算术规则外，求解器通常还会实现三种不同的线性整数算术证明规则:

- 分支定界 *(BnB)* [61]: 求解器首先检查公式 ϕ 在实数域上的可满足性，而不是在整数域上检查，这种方法称为 松弛。 如果发现 ϕ 在线性实数算术下不可满足，那么它必然也在线性整数算术下不可满足；另一方面，如果找到实数域上的一个解，求解器就会检查是否有符号 x 被赋予了非整数值 $\alpha \in \mathbb{R} \backslash \mathbb{Z}$。当存在这样的 x 时，搜索会进行分支，并分别针对两种情况 $x \leq \lfloor \alpha \rfloor$ 和 $x \geq \lceil \alpha \rceil$ 继续进行。表 1d 展示了表示分支定界策略的 BnB 证明规则。

  虽然分支定界是求解器所使用的一条重要证明规则，并且在许多公式上往往表现良好，但它并不能产生针对线性整数算术的完备演算。这一点在考虑示例 16 中的公式时可以看出，这些公式具有无界的线性实数算术解

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash 0 \le y + 2nx + n - 1}\,\textsc{Hyp} \qquad \dfrac{}{\Gamma \vdash 0 \le -y - 2nz + n}\,\textsc{Hyp}}{\Gamma \vdash 0 \le (y + 2nx + n - 1) + (-y - 2nz + n)}\,\textsc{Comb}}{\Gamma \vdash 0 \le 2nx - 2nz + 2n - 1}\,\textsc{Simp}}{\Gamma \vdash 0 \le x - z + 0}\,\textsc{Div}}{\Gamma \vdash 0 \le x - z}\,\textsc{Simp}}{\mathcal{P}_1}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash 0 \le -y - 2nx}\,\textsc{Hyp} \qquad \dfrac{}{\Gamma \vdash 0 \le y + 2nz - 1}\,\textsc{Hyp}}{\Gamma \vdash 0 \le (-y - 2nx) + (y + 2nz - 1)}\,\textsc{Comb}}{\Gamma \vdash 0 \le 2nz - 2nx - 1}\,\textsc{Simp}}{\Gamma \vdash 0 \le z - x - 1}\,\textsc{Div}}{\mathcal{P}_2}
$$

$$
\dfrac{\dfrac{\dfrac{\mathcal{P}_1 \qquad \mathcal{P}_2}{\Gamma \vdash 0 \le (x - z) + (z - x - 1)}\,\textsc{Comb}}{\Gamma \vdash 0 \le -1}\,\textsc{Simp}}{\Gamma \vdash \bot}\,\textsc{Con}
$$

图 6 $\Gamma = 0 \le y + 2nx + n - 1$ 示例中公式 $\{, 0 \le -y - 2nx, 0 \le y + 2nz - 1, 0 \le -y - 2nz + n\}$ 不可满足性的割平面证明示意图。

大小，但没有线性整数算术解。无论BnB 规则被应用多少次，仍然会留下一些证明分支，其中 Γ 中的公式是线性实数算术可满足但线性整数算术不可满足的。强化 [56, 57]：与分支定界类似，不等式 $0 \le t$ 在求解时可用于将搜索划分为情形 $t = 0$ 和 $1 \le t$，表达了整数项 t 不能取严格介于 0 和 1 之间的值这一事实。强化是 Omega 测试所使用的核心规则 [56], ，其中边界情形 $t = 0$ 也被称为“碎片”。通过适当地限制所引入碎片的数量，Omega 测试利用强化进行线性整数算术中的量词消去。

相应的强化规则如表 1d 所示。强化可用于构造线性整数算术的完备演算，但还需要一组单独的规则来求解由碎片 [56, 57] 导出的方程；此处未展示这类方程规则。在示例 16 中，可对 A_n 中的第一个不等式应用强化规则 n 次，从而得到一组在线性实数算术下不可满足的不等式；这组不等式可用表 1a 中的规则证明。此外，还有 n 个碎片可通过等式推理处理。

切割 [61, 20] 允许求解器对不等式中的常数项进行取整，同时确保不会丢失任何整数解。切割可以看作是强化规则的一种有针对性的应用，将其应用范围限制在所引入的方程 $t = 0$立即不可满足的情形。切割有多种可用策略，包括

Gomory 割 [61, 20] 和基于证明的割方法 [17], 都常用于求解器中。实现切割的一般规则 Div 在表 1d 中作为第三条规则给出。当不等式中非常数项的系数具有某个公共因子 $\beta > 0$ 时，规则 Div 适用。在这种情况下，可以将不等式中的所有系数除以 β，同时对常数项向下取整 α_0/β，从而强化该不等式。切割为线性整数算术提供了完备演算，其优点是不需要进行分支；包含切割的证明也称为割平面证明。为了在实践中获得更好的性能，许多求解器将切割与分支定界法结合使用。

我们在图 6 中展示了示例 16 公式的割平面证明。该证明适用于任意正值 n，需要应用两次 Div 规则，之后即可使用 Comb 推导出矛盾。

这三类证明规则都可以构造成插值版本。我们在表 1e 中展示了分支定界和切割的插值版本，其序列记法与表 1b 中线性实数算术规则的插值版本相同。强化规则的插值版本稍复杂一些，因为还需要引入线性方程的插值规则 [38], ，不过也可以推导出来 [12, 27]。

最容易构造插值版本的是分支定界，因为 BnB 执行的分支操作可以看作对析取 $0 \le n - x \vee 0 \le x - n - 1$ （这是线性整数算术中的一个有效公式）进行拆分。与 BnB 对应的两个插值规则是 BnB-A^I 和 BnB-B^I：根据常数 x 是来自 A 还是来自 B，必须使用其中一个规则。规则 BnB-A^I 和 BnB-B^I 对应于用于拆分析取的规则 Disj-A^{I}和 Disj-B^I ；与析取规则一样，它们通过对A，B的子插值 I，J进行组合（使用析取或合取），来推导总体插值。如果 x 同时出现在 A 和 B中，则可以使用任一规则，但通常这两条规则会导出不同的插值式。

插值割平面证明会更加复杂，因为使用规则 Div 的证明中经收紧的不等式可能包含一些只出现在 A 中的符号，以及一些只出现在 B 中的符号，因此无法将它们唯一地归属于任一方。不过，只要允许序列 $(A,\ B) \vdash 0 \le t\ [t']$ 中的注释 t' 包含带余数的整数除法算子 $\div$，且分母为整数常数 [27, 55]，就可以将用于 LRA 插值的注释方法扩展到 LIA。我们采用带余数除法的欧几里得定义 [46], ，其中，对于 $a,\ b \in \mathbb{Z},\ a \div b$ 定义为满足 $a = b \cdot q + r$ 且 $0 \le r < |b|$ 的唯一整数 $q \in \mathbb{Z}$。为了处理整数除法，还需要推广 Simp^I 规则，增加一个将项移出 $\div$作用域的情形，即只要 $\beta \mid \alpha$，就有 $(\alpha s + t) \div \beta \rightsquigarrow (\alpha/\beta)s + (t \div \beta)$。这个重写是正确的，因为对于欧几里得除法，只要 $\beta \neq 0$，等式 $(\beta x + y) \div \beta = x + (y \div \beta)$ 就成立。

表 1e 中的规则 Div^I 看似简单，实际上它使用 $\div$ 将序列中的注释 t' 除以与主不等式相同的数 β。Div^I 的效果最好通过一个例子来说明。图 7 展示了图 6 中证明的插值版本，该证明证明了

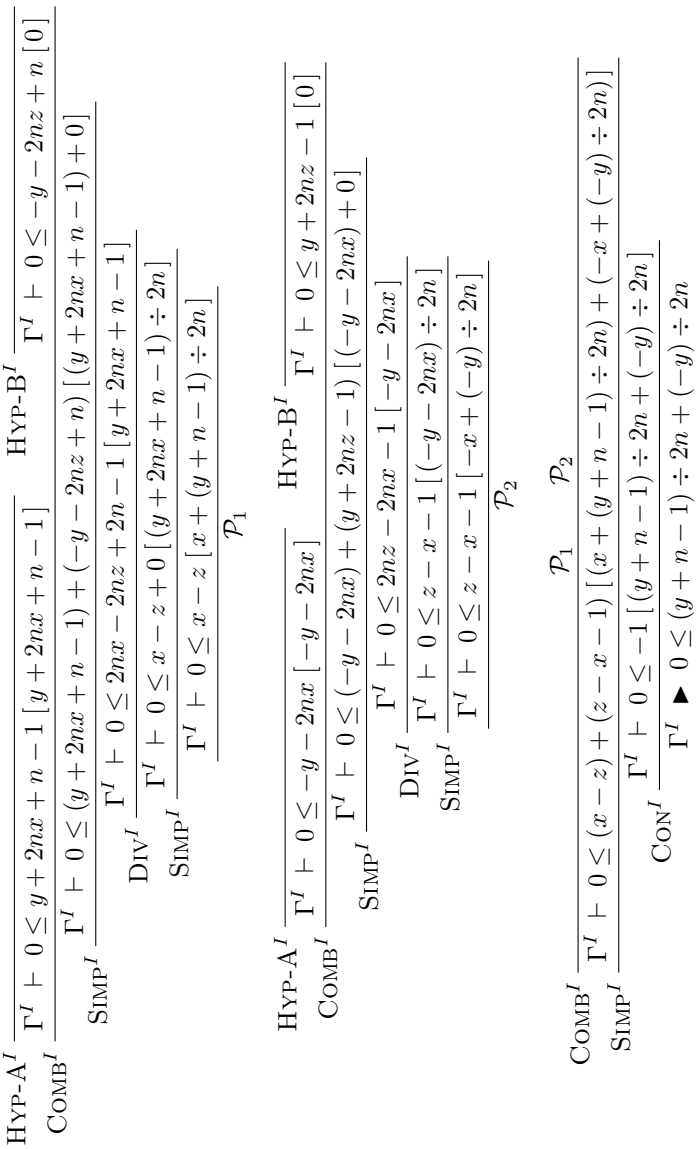


图7 示例 16 中公式的插值证明： $\Gamma^I = (A,\ B) = (\{0 \le y+2nx+n-1,\ 0 \le -y-2nx\ ,\ 0 \le y+2nz-1\ 0 \le -y-2nz+n \}\ ,\ \})$。

示例 16 中的公式对于任意正数 n 都不可满足。Div^I 的两次应用在注释项中引入了除法 $\div 2n$。随后通过应用规则 Simp^I 来简化注解，将项 $2nx$ 和 $-2nx$ 分别化简为 x 和 $-x$，从而将符号 x 移出 $\div$ 的作用域。由于 x 和 $-x$ 相互抵消，最终的 LIA 分隔式是公式 $0 \le (y + n - 1) \div 2n + (-y) \div 2n$，其中只含有 y 这个 A 与 B 共有的符号。读者可以验证，该分隔式与示例 16 中的 I_n 等价，但由于使用了 $\div$，表达简洁得多，长度呈指数级缩短。

当然，有趣的问题在于，为什么规则 Div^I 总能生成正确的插值式。针对 LIA 的情况，插值序列 $(A,\ B) \vdash 0 \le t\,[t']$ 需要满足以下条件：

- $A \models_{\mathrm{LIA}} 0 \le t'$、$B \models_{\mathrm{LIA}} 0 \le t - t'$，以及 $\mathsf{sig}(t') \subseteq \mathsf{sig}(A) \cup \Sigma_{\mathrm{LIA}}$
- 和 $t - t' \rightsquigarrow s$ 其中 $\mathsf{sig}(s) \subseteq \mathsf{sig}(B) \cup \Sigma_{\mathrm{LIA}}$。

我们可以通过纯算术推理检查得出，使用表 1b 中的规则和规则 Div^{I}推导出的任意序列（$A,\ B$）$\vdash 0 \le t\,[t']$ 都满足前两个条件。要了解最后几个条件为何在 Div^{I}下保持不变，请注意，将 $t - t'$ 改写为一个满足 $\mathsf{sig}(s) \subseteq \mathsf{sig}(B) \cup \Sigma_{\mathrm{LIA}}$ 的项 s的能力意味着：每当项 αx 出现在 t中时，对于某个未出现在 B中的 x，t' 中就必定有一个对应项 αx与之抵消。应用 Div^{I}时，α 必定是分母 β的倍数。因此，应用规则 Simp^I 即可将注释（$\cdots + \alpha x + \cdots$）$\div \beta$化简为（$\alpha/\beta$）$x +$（$\cdots$）$\div \beta$。

更一般地，应用 Simp^I 后，注释 t' 中的 $\div$ 运算符的作用范围内绝不会包含 B 中未出现的符号。尽管注释中使用了 $\div$，插值证明规则仍能以与 LRA 大致相同的方式，从注释中消除出现在 A 但未出现在 B 中的符号，并且由此导出的插值式是正确的。

2.3 其他插值方法

2.3.1 基于证明的插值

不仅可以从 LRA 和 LIA 的证明中提取（无量词）T-分隔式，也可以从许多其他理论的证明中提取。LRA 的插值证明规则可以扩展，以处理等式与未解释函数理论（EUF）[51]。LRA 演算也可以扩展到非线性算术（非线性实数算术，NRA），用平方和（SOS）见证 [16, 65] 替代 Farkas 引理所产生的见证。也可以借助模型构造可满足性演算 MCSAT [40] 来计算 NRA 插值式。

针对数组理论 [13, 62, 36]，已提出若干插值演算。如示例 6 所示，数组方面有待解决的一个困难在于：标准数组版本不具有（普通或一般的）无量词插值。这一问题

可以通过用满足以下公理的函数符号 $diff$ 扩展该理论来解决:

$$\forall M, M'.\ \big(M \neq M' \rightarrow select(M, diff(M, M')) \neq select(M', diff(M, M'))\big)\ .$$

该公理蕴含数组外延性，即两个数组相等，当且仅当它们具有相同的内容。因此，在示例 6 中，我们可以构造无量词反向插值式

$$M' = store\big(M, diff(M, M'), select(M', diff(M, M'))\big)\ .$$

2.3.2 基于图的插值

在 SMT 的语境中，空理论 $T_\emptyset$中公式的可满足性这一特殊情形通常称为等式与未解释函数模可满足性（*EUF*）。在 SMT 求解器中，EUF 通常借助同余闭包算法 [2], 处理，该算法用于判定给定方程 $s = t$ 是否可由有限个 E 基项等式推出，即$E \models_{T_\emptyset} s = t$。为此，同余闭包会构造一个等式图（等式图），其中 E 中的项和子项是节点，方程 E 及推导出的等式是边，另有一组边用于表示子项关系。同余闭包可以通过推导出某个方程 $s = t$可由 ϕ中的方程推出，从而证明公式 ϕ 不可满足；而 ϕ 还包含取反等式 $s \neq t$。等式图表示了用于解释公式不可满足性的证明。

对于不可满足的方程合取式 $A \wedge B$，可以从等式图 [22]中提取分离式。这是通过概括图中的路径来完成的；如果一条路径仅由公式 A中的方程构成，就可以用一个由 A蕴含的单一方程来表示；对于 B中的方程也同样如此。根据同余关系进行推理（$s = t$蕴含 $f(s) = f(t)$），会在计算得到的分离式中产生蕴含关系。

2.3.3 通过归约进行插值

计算 Craig 插值式的另一种范式是归约 [41]。只要在某个理论 T中直接对合取式 $A \wedge B$进行插值有困难，但能够将 $A \wedge B$转换为理论 T'中的合取式 $A' \wedge B'$，且在该理论中可以计算插值式，就可以应用归约。随后必须将 T'-分离式 I'转换回来，以获得原始合取式 $A \wedge B$的 T-分离式 I；在特殊情况下，若 T是 T'的扩展，则无需进行反向翻译，因为 I'已经是 $A \wedge B$的正确 T-分离式。

归约方法可应用于广泛的理论。数组理论以及带有有限基数约束的集合理论可以归约为等式与未解释函数，而多重集理论可以归约为等式与未解释函数和线性整数算术的组合理论 [41]。然而，由于归约可能引入量词，因此通常需要使用量词消去过程对插值式进行后处理，以实现无量词插值。在后续研究中，研究者证明，带有大小约束的代数数据类型理论可以

归约为等式与未解释函数和线性整数算术的组合理论 [37], ，并且位向量理论可以归约为线性整数算术 [3]。后一种归约以惰性方式进行，借助量身定制的证明规则，逐步将位向量公式转换为线性整数算术公式。

3 基于插值的验证算法

我们现在介绍一些主要的基于 Craig 插值的验证算法，首先从 McMillan 在 2003 [49] 中提出的基于 SAT 的模型检测算法开始。我们在理论 T 中的无量词插值这一设定下给出这些算法，尽管更早的论文侧重于命题插值。

3.1 有界模型检测：有界模型检测

基于插值的验证算法受到有界模型检测（有界模型检测），的启发；在该方法中，系统在最初 k 个执行步骤中的行为借助 SAT（布尔可满足性问题）或 SMT 求解器 [7] 加以分析。催生基于插值的算法的关键洞见是：可以为有界模型检测中所考虑的公式提取插值式，从而把有界模型检测变为无界 模型检测。

为了介绍有界模型检测，我们固定一个理论 T，并假设待验证系统的状态由一个常数向量 $\bar{x} = (x_1, \ldots, x_n)$ 来描述；在最初基于 SAT（布尔可满足性问题）的表述中，$\bar{x}$ 是一个命题变量向量。我们使用记号 $\bar{x}' = (x'_1, \ldots, x'_n)$ 表示这些常数的一个新的加撇副本，并类似地用 $\bar{x}_i$ 表示以 i 为下标的副本。有界模型检测所处理的是用公式表示的迁移系统，这些公式例如用于对时序硬件电路进行建模。一个迁移系统（B, TR）由两个句子 $B[\bar{x}]$ 与 $TR[\bar{x}, \bar{x}']$ 给出，二者分别定义初始状态 与迁移关系，。记号 $B[\bar{x}]$表示常数 $\bar{x}$ 可以出现在 B中，而 $B[\bar{y}]$ 表示常数 $\bar{x}$ 被另一个由项构成的向量 $\bar{y}$所替换。直观而言， $\bar{x}$ 的任一满足 $B[\bar{x}]$的赋值都被视为该系统的一个初始状态，而 $\bar{x}, \bar{x}'$ 中满足 $TR[\bar{x}, \bar{x}']$ 的赋值则表示从 $\bar{x}$ 到 $\bar{x}'$的一次转移。

引入另一个句子 $E[\bar{x}]$ 来表示错误状态。验证的目标是确定是否可以从初始状态到达任何错误状态。部分正确性性质的验证，例如第 1.1 节中讨论的性质，可以归约为这种形式：使用公式 E对违反后置条件或断言的状态进行建模。在有界模型检测中，通过将迁移关系展开有限次来刻画错误状态的可达性：当且仅当存在一个自然数 $k \in \mathbb{N}$ 使得该句子

$$B[\bar{x}_0] \ \wedge \ TR[\bar{x}_0, \bar{x}_1] \wedge TR[\bar{x}_1, \bar{x}_2] \wedge \cdots \wedge TR[\bar{x}_{k-1}, \bar{x}_k] \ \wedge \ \big(E[\bar{x}_0] \vee E[\bar{x}_1] \vee \cdots \vee E[\bar{x}_k]\big) \quad (2)$$

是 T-可满足的。事实证明，得益于这类求解器的性能，将 SAT 或 SMT 求解器应用于 (2)，是发现可能发生错误的系统的一种非常有效的方法。相比之下，如果公式 (2) 是 T-不可满足的，就无法（至少在一般情况下无法）判断

系统究竟确实不会遇到任何错误，还是所选的 k 只是不够大，因而未能找到错误。

3.2 IMC：插值和基于 SAT 的模型检测

3.2.1 IMC 算法

第一个基于插值的模型检测算法在 BMC 的基础上，通过 Craig 插值计算归纳不变式，从而能够确切判定对于 $k\in$ 的所有取值，(2) 都是 T 不可满足的。我们沿用 [6] 中给出的名称，将该算法称为 IMC 算法。要证明对于每个 $k\in\mathbb{N}$，公式 (2) 都不可满足，只需找到一个句子 $R[\bar{x}]$，即归纳不变式，并满足第 1.1 节中讨论的性质：

- 初始条件：$B[\bar{x}]\models_T R[\bar{x}]$；保持条件：
- $R[\bar{x}]\wedge TR[\bar{x},\bar{x}']\models_T R[\bar{x}']$；充分性：
- $R[\bar{x}]\wedge E[\bar{x}]\models_T\bot$。

事实上，假设公式 (2) 成立，由于 初始条件 和 保持条件，可知R对所有状态 $\bar{x}_0,\ldots,\bar{x}_i$均成立；并且根据 充分性，可得出与 E-公式的析取矛盾。

为便于说明，我们作一个简化假设：关系 TR是 全定义的，即 $\forall\bar{y}.\exists\bar{z}.\ TR[\bar{y},\bar{z}]$ 成立。为得到归纳不变式 $R[\bar{x}]$的候选，IMC 算法构造一个句子序列 $R_0,R_1,R_2,\ldots$，使得每个 R_i 都是至多经过 i 步从初始状态可达的状态的过近似。自然，我们可以设定 $R_0=B$。为了构造其他句子，我们固定满足 $j\leq k$的常数 $j,k\in\mathbb{N}$，它们是 IMC 的参数。随后，通过考察以下合取，从 R_i 推导出每个公式 R_{i+1}：

$$\underbrace{R_i[\bar{x}_0]\wedge TR[\bar{x}_0,\bar{x}_1]}_{A[\bar{x}_0,\bar{x}_1]}\ \wedge\ \underbrace{\bigwedge_{l=1}^{k-1}TR[\bar{x}_l,\bar{x}_{l+1}]\wedge\bigvee_{l=j}^{k}E[\bar{x}_l]}_{B[\bar{x}_1,\ldots,\bar{x}_k]}\ . \tag{3}$$

公式 (3) 对应有界模型检测查询 (2)，不同之处在于，初始状态 B已替换为第 i 次近似 R_i，并且 $E[\bar{x}_l]$ 的析取范围通过 j 参数化。可能有三种情况：

- **情况 (i)：** 公式 (3) 可满足，且 $i=0$。这意味着在迁移系统中，$R_i=R_0=B$ 和错误状态是可达的。因此，对于相同的 k，公式 (2) 也可满足。
- **情况 (ii)：** 公式 (3) 可满足，且 $i>0$。这可能是因为错误状态可达（如情况 (i)），也可能是因为近似 R_i 不够精确。在这种情况下无法得出任何结论，算法必须中止。

- **情况 (iii)：**公式 (3) 不可满足。于是，我们可以从合取 $A[\bar{x}_0, \bar{x}_1] \wedge B[\bar{x}_1, \ldots, \bar{x}_k]$中提取一个 T-分隔符 $I_i[\bar{x}_1]$，因为 $\bar{x}_1$ 是同时出现在 $A[\bar{x}_0, \bar{x}_1]$ 和 $B[\bar{x}_1, \ldots, \bar{x}_k]$中的常数。将 $\bar{x}_1$ 重命名为 $\bar{x}$后，我们得到句子 $I_i[\bar{x}]$，它是从 R_i出发一步可达状态的过近似。接着将下一个近似设为 $R_{i+1}[\bar{x}] = R_i[\bar{x}] \vee I_i[\bar{x}]$，通过弱化 R_i 来进一步考虑迁移系统的一步转移。

结果表明，序列 R_0、R_1、R_2、.....中往往包含一些 R_i，当 i取较小值时，它们是归纳不变式。直观地说，这是因为句子 R_i 的构造旨在近似迁移系统的可达状态。Craig 插值的使用能够加速不动点构造，将重点放在那些足以与式（3）的 B部分保持不一致的状态性质上（即在 k步内不会到达任何错误状态），而忽略公式 B、TR E中可能包含但与排除错误无关的其他细节。通过从证明中获取插值式即可实现这种加速效果；这是利用了这样一个事实：证明很可能只涉及证明式（3）不可满足所必需的事实和约束，而这些恰好就是避免错误状态所需的性质。

根据构造，句子 R_0、R_1、R_2、.....都满足归纳不变式的初始条件。要检查句子 R_i 是否满足保持条件，可以显式验证该条件（$R_i[\bar{x}] \wedge TR[\bar{x}, \bar{x}'] \models_T R_i[\bar{x}']$）；或者，作为充分条件，也可以检查 $R_{i+1} \equiv_T R_i$是否始终成立；根据 T-分离式的定义，这可以推出保持条件。句子 R_i 的充分性也可以通过显式验证（$R_i[\bar{x}] \wedge E[\bar{x}] \models_T \bot$）来检查。也可以对 IMC 进行设置，使所有构造出的公式 R_i 都满足充分性：为此，必须验证初始状态 $R_0 = B$ 均非错误状态，且 j 的值必须为 1 [49]。（此时还需要假设 T 是全定义的。）

为使该方法有效，通常要求公式 B、TR E为无量词公式，并要求所考虑的理论 T 支持普通无量词插值；这确保句子 R_0、R_1、R_2、.....也都是无量词的，且所需的检查仍然是可判定的。原始 IMC 算法 [49] 使用从命题归结证明中提取的 Craig 插值式（参见第 1 章）。

参数 j、k 可用于控制算法的行为。较大的 k 值和较小的 j 值会带来更精确的过近似，因为 (3) 的 B 部分 ensures 子句保证，从分隔符 $I_i[\bar{x}_1]$ 出发，在至少 $j-1$ 步且至多 $k-1$ 步内无法到达任何错误状态。增大 k 尤其会使计算得到的公式 $I_i[\bar{x}_1]$ 更强，从而降低后续迭代中落入情况 (ii) 的概率。在某些假设 [49] 下，可以证明 IMC 甚至能够完整地计算归纳不变式：

- 该迁移系统具有有界反向深度，这意味着存在一个数值 $d \in \mathbb{N}$，使得每当 E 可从某个状态到达时，也存在一条从

该状态到 E 的路径，其长度至多为 d。

- 计算得到的不变式候选序列 R_0，R_1，.....保证最终会达到稳定状态，也就是说，存在某个 $i \in \mathbb{N}$，使得 $R_i \equiv_T R_{i'}$ 对每个 $i' \geq i$都成立。

如果第一个假设成立，只需选择足够大的 k 并将 j 设为 1，就能避免算法因情形 (ii) 而失败。此外，如果第二个假设也成立，则存在某个 R_i 使得 $R_i \equiv_T R_{i+1}$，从而保证通过保持条件，并得出一个归纳不变式。

命题逻辑中的迁移系统和不变式都满足这两个假设。对于第二个条件，注意到每个 R_i 都蕴含下一个 R_{i+1}，因此这些候选 R_0，R_1，.....构成一个递增链。由于命题公式 R_i（按等价关系计）只有有限多个，因此每条链 R_0，R_1，.....最终都会达到稳定状态。相比之下，对于状态空间无限的系统，这类系统使用允许无限论域的理论 T 建模，通常不满足这些条件。根据经验，即使对于这类系统，IMC 算法也往往表现良好 [6]。

3.2.2 相关的基于插值的算法

人们提出了许多不同版本和改进版的基于 SAT 的模型检测算法 IMC。最初，IMC 是针对命题逻辑中的迁移系统和插值定义的 [49]。后来，IMC 扩展到了理论和软件验证领域 [6]，本章也采用了这种设定。

基于插值序列的模型检测算法 *ISB* 采用了与 IMC 类似的定点循环，但使用序列插值（定义 10），而非二元插值 [63]。IMC 和 ISB 的主要区别在于插值查询的定义方式：IMC 在用于计算下一个 R_{i+1}的查询 (3) 中纳入了前一个近似 R_i，而 ISB 所用的查询仅涉及定义迁移系统和错误状态的句子 B，TR E；不过，ISB 会在每次迭代中增加所考虑的转移次数（TR的副本数）。

软件程序的验证算法通常会考虑程序的控制结构。此时，我们不会采用整体迁移系统（B，TR），而是从一个程序出发，该程序由控制流自动机组织的一组迁移关系描述。这种设定可以缩小插值查询的规模，因为每次只需考虑一条程序路径。*Impact* 算法采用了这一策略，利用序列插值为特定程序路径生成不变式候选 [52]。

IC3 算法更加局部，它分析单个系统状态，而非整个迁移关系，以获得不变式候选 [9]。虽然 IC3 与 IMC 和 ISB 算法有相似之处，但 IC3 不再依赖 Craig 插值，而是直接推导出作为归纳不变式的子句。*Spacer* 算法是 IC3 的扩展，它将插值作为可选的抽象步骤 [42]。

3.3 CEGAR：反例引导的抽象细化

谓词抽象 [24, 4] 和 反例引导的抽象精化 *(CE- GAR)* [15] 为利用 Craig 插值验证系统提供了另一种视角。这两个概念都早于 IMC 算法及插值在验证中的应用，但后来证明它们非常适合与插值结合使用 [34, 50]。

3.3.1 存在抽象

我们再次固定一个理论 T。为便于阐述，我们假设该理论对应一个唯一确定的结构 S_T，也就是说，定义 1 中的集合 $\mathcal{S}_T = \{S_T\}$ 是单元素集合。由于系统的状态由一个常数向量 $\bar{x} = (x_1, \ldots, x_n)$ 描述，其排序为 $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n)$，因此系统的状态空间是笛卡尔积 $D = D_{\sigma_1} \times \cdots \times D_{\sigma_n}$。由于该理论只有一个结构 S_T，稍微滥用一下记号，我们可以写作 $B[s]$，表示状态 $s \in D$ 满足句子 B，也就是说， s 是一个初始状态；类似地，可以用 $TR[s, s']$ 表示从 s 到 s' 存在一次转移。

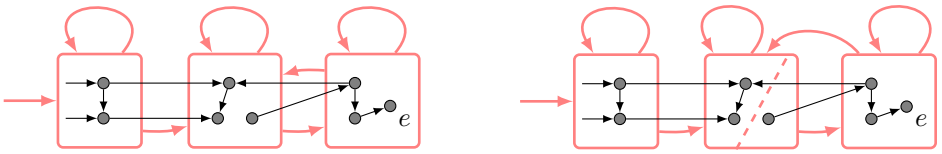
为了验证迁移系统（B, TR）的性质，例如错误状态 E的可达性，考虑状态数更少的迁移系统$\hat{}$的一个抽象版本通常很有用。我们可以通过选取非空集合 $\hat{D}$作为抽象状态空间，并给出一个抽象函数 h： $D \to \hat{D}$ 来定义抽象；该函数以抽象初始状态的集合 B 以及抽象$\hat{}$迁移关系 TR诱导$\hat{}$出抽象迁移系统:

$$\hat{B} = \{\hat{s} \in \hat{D} \mid \text{there is } s \in D \text{ with } h(s) = \hat{s} \text{ and } B[s]\} ,$$
$$\hat{TR} = \{(\hat{s}, \hat{s}') \in \hat{D}^2 \mid \text{there are } s, s' \in D \text{ with } h(s) = \hat{s}, h(s') = \hat{s}' \text{ and } TR[s, s']\} .$$

这种构造称为存在抽象 [28]。它之所以有用，是因为存在抽象中状态的不可达性也蕴含具体系统中的不可达性：只要 $M \subseteq D$ 并且抽象状态 $h(M)$在$(\hat{B}, \hat{TR})$中从初始状态不可达，具体状态 M 在(B, TR)中同样不能从初始状态到达。更一般地，存在抽象可用于证明具体系统 [28]的ACTL* 性质。

存在抽象如图 8a 所示。黑色状态和边表示具体迁移系统（B， TR），初始状态以额外的入边标出。函数 h 将具体状态归并为红色抽象状态。只要所包含的任一具体状态之间存在转移，抽象状态之间就会建立转移，从而对行为进行过近似。例如，图 8a 中的具体迁移系统不包含任何状态到自身的转移，而抽象迁移系统中则存在此类转移。

因此，$(\hat{B}, \hat{TR})$ 中状态的可达性并不意味着 (B, TR) 中对应状态的可达性，因为抽象迁移系统可能包含伪路径，而这些路径在 (B, TR) 中没有等价路径。例如，在图 8a 中，存在一条从最左侧初始状态到最右侧状态的抽象



(a) 初始抽象。 (b) 精化后的抽象。

图 8 一个迁移系统（黑色）和两个存在抽象（红色）。

路径，但不存在从初始状态到最右侧状态 e 的具体路径。

3.3.2 谓词抽象

定义存在抽象的一种便捷方法是**谓词抽象** [24]。为此，我们假设有一个有限的句子集合 $\mathcal{P} = \{P_1[\bar{x}], \ldots, P_k[\bar{x}]\}$，其中的句子可以引用表示系统状态的常数 $\bar{x}$。使用**布尔抽象** [4], 定义的迁移系统是多种不同版本的谓词抽象之一，它将无法通过$\mathcal{P}$中的谓词加以区分的状态归为一组。抽象状态可以表示为位向量 $\hat{D} = \{0,1\}^k$，为$\mathcal{P}$ 中的每个谓词记录该谓词在某个具体状态 $s \in D$ 中是否成立：

$$h(s) = (p_1(s), \ldots, p_k(s)), \qquad p_i(s) = \begin{cases} 1 & \text{if } P_i[s] \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } i \in \{1, \ldots, k\} .$$

对于具有 k 个谓词的抽象，这将得到一个具有2^k 个抽象状态的迁移系统$(\hat{B}, \hat{TR})$，该系统也可以根据定义具体迁移系统的句子 B, TR显式构造。为此，我们将 $p^{-1}(\hat{s})[\bar{x}]$ 定义为与抽象状态$\hat{s} = (b_1, \ldots, b_k) \in \hat{D}$相对应的（正或负）谓词的合取：

$$p^{-1}(\hat{s})[\bar{x}] \;=\; \bigwedge_{i=1}^{k} (P_i[\bar{x}] \leftrightarrow b_i) \; .$$

抽象迁移系统随后由下式给出：

$$\begin{aligned} \hat{B} &= \{\hat{s} \in \hat{D} \mid B[\bar{x}] \wedge p^{-1}(\hat{s})[\bar{x}] \text{ is } T\text{-satisfiable}\} \; , \\ \hat{TR} &= \{(\hat{s}, \hat{s}') \in \hat{D}^2 \mid p^{-1}(\hat{s})[\bar{x}] \wedge TR[\bar{x}, \bar{x}'] \wedge p^{-1}(\hat{s}')[\bar{x}'] \text{ is } T\text{-satisfiable}\} \; . \end{aligned}$$

3.3.3 反例引导的抽象精化

CEGAR 算法的核心思想是，当抽象过于粗糙、无法证明具体迁移系统的某些期望性质时，便细化这些抽象；这样，验证

某些系统可以从一个非常小且不精确的抽象迁移系统开始（例如，由空谓词集$\mathcal{P}_0 = \emptyset$ 得到的系统），并逐步细化抽象，直到能够证明性质 [15]。精化可以看作是对抽象迁移系统的状态进行细分，从而消除伪路径。例如，在图 8b 中，中间的抽象状态被拆分为两个状态，从而移除抽象路径，并使包含 e 的抽象状态从初始状态不可达。一般而言，必须拆分多个抽象状态，才能移除抽象迁移系统中的路径。抽象细化可以借助 Craig 插值 [34]来实现。

为此，假设 $E[\bar{x}]$ 是表示错误状态的句子，而 $\hat{}E = \{\hat{s} \in \hat{}D \mid$ 是满足 $s \in D$ with $h(s) = \hat{s}$and $E[s]\}$ 的对应抽象集合。假设使用谓词 $\mathcal{P}_l$ 进行的抽象分析已识别出一条状态路径 $\hat{s}_1, \hat{} s_2, \ldots, \hat{s}_m$，使得对于所有 $i \in \{1, \ldots, m-1\}$，都有 $\hat{s}_1 \in B$, $(\hat{s}_i, \hat{s}_{i+1}) \in \hat{}TR$，且 $\hat{s}_m \in \hat{}E$；然而，在具体迁移系统中不存在对应的路径。后一个假设意味着公式

$$\underbrace{B[\bar{x}_1]}_{A_1} \wedge \underbrace{TR[\bar{x}_1, \bar{x}_2]}_{A_2} \wedge \underbrace{TR[\bar{x}_2, \bar{x}_3]}_{A_3} \wedge \cdots \wedge \underbrace{TR[\bar{x}_{m-1}, \bar{x}_m]}_{A_m} \wedge \underbrace{E[\bar{x}_m]}_{A_{m+1}}$$

是 T-不可满足的。路径 $\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_m$ 也称为“虚假反例”，可通过添加更多谓词来消除，从而提高谓词抽象的精度。令 $I_0, I_1, \ldots, I_{m+1}$ 为合取 $A_1 \wedge \cdots \wedge A_{m+1}$ 的序列插值。根据定义，对于 $i \in \{1, \ldots, m\}$ 中的每个 I_i，它只涉及符号 $\bar{x}_i$，因此，通过重命名可得到 m 个新句子 $I_1[\bar{x}], \ldots, I_m[\bar{x}]$；这些句子产生细化后的抽象迁移系统，其定义为 $\mathcal{P}_{l+1} = \mathcal{P}_l \cup \{I_1[\bar{x}], \ldots, I_m[\bar{x}]\}$。

在精化后的抽象系统中，虚假反例不再存在，也就是说，不存在一条长度为 m、到达错误状态的抽象路径 $\hat{} s'_1$，$\hat{} s'_2$，……，$\hat{s}'_m$。这是由于引入了新的谓词：在每个初始状态 $\hat{} s \in B$ 中，对应于新谓词 I_1 的位必须为 1；因此，从初始状态一步可达的每个状态中，I_2 对应的位都必须为 1，依此类推。最后，在经过 $m-1$ 步可达的每个状态中都存在的谓词 I_m，与定义错误状态的句子 E 不一致，这意味着不存在长度为 m、到达错误状态的路径。当然，也可能存在长度不同的错误状态路径，这些路径可以通过进一步的精化步骤消除。

3.3.4 谓词抽象和 CEGAR 的变体

显式构造抽象迁移系统往往十分耗时，因此在较新的 CEGAR 实现中，通常会采用惰性抽象范式 [35]来避免这一步。惰性抽象不显式处理抽象状态，而是构造一个可达性图，其中每个节点都标记有$\mathcal{P}$中的谓词的布尔组合，用以概括一整组抽象状态。从节点 n 到 n' 的一条边意味着，分别标记 n, n'的公式 ϕ, ϕ'满足 $\phi[\bar{x}] \wedge TR[\bar{x}, \bar{x}'] \models_T \phi'[\bar{x}']$。可达性图在探索过程中从初始状态开始即时构建，并在遇到错误状态时进行细化

。Impact 算法甚至不会存储和重用谓词，而是在每次考虑一条新路径时生成新的谓词（使用 Craig 插值） [52]。

CEGAR 的大多数论文中采用的另一种优化，是再次考虑软件程序的控制结构（例如，[35, 52, 33]）。由于每次只考虑一条路径，这不仅能缩小插值查询的规模；对控制流的敏感性还使验证工具能够为每个控制位置维护一组独立的谓词。下一节将介绍更通用的受约束 Horn 子句框架，它能够处理许多不同类型的控制结构，包括过程调用和并发。

3.4 CHC：Craig 插值与受约束 Horn 子句

受约束 Horn 子句（CHCs）框架将程序验证重新表述为在扩展理论 T 上进行模型构造，并加入额外的谓词 [26, 8]。CHCs 可视为一种中间语言，使验证前端能够以标准化方式与验证后端通信。CHCs 还涵盖了第 2.1.3 节中定义的 Craig 插值扩展形式（并行插值、序列插值、树插值式），并使我们能够陈述一个相应且更一般的结果。

3.4.1 定义和基本性质

▶ **定义 17**（受约束 Horn 子句 [59]）。设 T 是一个理论，R 是一组与 T 的签名 Σ_T 不相交的关系符号。受约束 *Horn* 子句（*CHC*）是一个句子 $\forall \bar{x}.\, C \wedge B_1 \wedge \cdots \wedge B_n \to H$，其中：

- C 是一个关于 Σ_T 的公式，可以引用变量 $\bar{x}$；每个 B_i 都是关系符号 R 对 Σ_T 上的项的应用 $p(t_1, \ldots, t_k)$；H 要么是 $\bot$，要么是关系符号 R 对 Σ_T 上的项的应用 $p(t_1, \ldots, t_k)$。

我们称 C 为子句的约束，H 为头部，并称 $C \wedge B_1 \wedge \cdots \wedge B_n$ 为子句的体。约束 $C = \top$ 通常会省略。在对程序的部分正确性进行建模时，关系符号$\mathcal{R}$ 用于表示归纳不变式或契约等验证制品，而子句则对这些制品必须满足的条件进行建模。验证通过构造子句的模型来完成，该模型包含证明程序正确性所需的全部制品。我们在第 3.2 节和第 3.3 节中遇到的算法可以调整后用于处理 CHC 集合。

我们通过子句可解性的两种概念，将这一过程形式化。为便于阐述，我们再次聚焦于带有唯一确定结构 S_T 的理论 T，也就是说，定义 1 中的集合 $\mathcal{S}_T = \{S_T\}$ 是单元素集合。若一组约束 Horn 子句可以将结构 S_T 扩展为签名 $\Sigma_T \cup \mathcal{R}$ 上的一个结构，且该结构满足所有子句，也就是说，若这组约束 Horn 子句是 T-可满足的，则称其为语义可解 的；其定义见

定义 1。[1] 若通过对签名 Σ_T 上的公式进行统一替换，取代$\mathcal{R}$ [59]中的关系符号后，所有子句都能变为T-有效，则称这组约束 Horn 子句句法可解 。句法可解性与逻辑 [64]中的经典求解问题有关。

▶ **例 18**。 我们使用约束 Horn 子句对图1a中的斐波那契示例建模。为此，我们引入一个关系符号$loop$ ，表示函数 `fib` 的循环不变式，并针对初始条件、保持条件，和充分性 条件构造子句。子句 (4) 表明，不变式在进入循环时应成立，即对于任意满足 $n \geq 0$ 且$a = 0$、$b = 1$、 $i = 0$的值 n ，不变式都应成立。子句 (5) 确保循环体保持不变式：只要$loop$对值 n、a、 b、 i成立，且满足循环条件 $i < n$ ，那么$loop$对执行一次循环迭代后的状态也应成立。子句 (6) 确保，只要$loop$ 对值 n、 a、 b、 i成立，并且循环因 $i \geq n$而终止，后置条件 $a \geq 0$就不会被违反。

$$\forall n.\ n \geq 0 \to loop(n, 0, 1, 0) \tag{4}$$

$$\forall n, a, b, i.\ i < n \wedge loop(n, a, b, i) \to loop(n, b, a + b, i + 1) \tag{5}$$

$$\forall n, a, b, i.\ i \geq n \wedge a \not\geq 0 \wedge \text{循环}(n, a, b, i) \to \bot \tag{6}$$

子句的每个语法解都对应一个归纳循环不变式。现有的约束 Horn 子句求解器通常采用基于插值的方法，可以轻松计算出此类解；一种可能的解是赋值 $\{loop(n,\ a,\ b,\ i) \mapsto a \geq 0 \wedge b \geq 0\}$，这证明了该程序是部分正确的。请注意， $a \geq 0 \wedge b \geq 0$ 与第 1.1 节中发现的归纳不变式相同。」

如果一组受约束霍恩子句在句法上可解，那么它在语义上也可解。反之则不一定成立，因为一组受约束霍恩子句可能只有无法在理论 T [59] 中显式定义的模型。不过，得益于克雷格插值定理，对于无递归的 受约束霍恩子句片段，语义可解与句法可解这两个概念是一致的。为说明受约束霍恩子句与 T-分隔式之间的关系，考虑公式 $A[\bar{x}, \bar{y}], B[\bar{x}, \bar{z}]$: 它们共有的变量和符号只有 $\bar{x}$ 和 Σ_T，而 $\bar{y}, \bar{z}$ 是局部变量，分别只出现在 $A[\bar{x}, \bar{y}]$ 或 $B[\bar{x}, \bar{z}]$ 中。接着考虑受约束霍恩子句 $\forall \bar{x}, \bar{y}.\ A[\bar{x}, \bar{y}] \to I(\bar{x})$ 和 $\forall \bar{x}, \bar{z}.\ B[\bar{x}, \bar{z}] \wedge I(\bar{x}) \to \bot$。根据定义，这两个子句的任何句法解都是合取式 $A[\bar{x}, \bar{y}] \wedge B[\bar{x}, \bar{z}]$ 的 T-分隔符[2]，反之亦然；这意味着，这两个受约束霍恩子句句法可解，当且仅当 $A[\bar{x}, \bar{y}] \wedge B[\bar{x}, \bar{z}]$ 是 T-不可满足的。类似地，序列插值和树插值式也可以用受约束霍恩子句集来表示。

---

[1] 对于包含多个结构的理论 T 的一般情形，通常使用两种语义可解性的定义。在宽松语义 [59, 8], 下，如果每个 T-结构都可以扩展为满足所有子句的结构，则认为一组约束 Horn 子句在语义上可解。现有的约束 Horn 子句求解器更常采用 SMT 求解器也使用的严格语义，它要求是否存在某个 T-结构，可以扩展为满足所有子句的结构。约束 Horn 子句的表述方式往往使这两种语义得出相同的结果。[2] 此处，我们将自由变量 $\bar{x}$, $\bar{y}$， $\bar{z}$ 视为常量符号。

为更一般地陈述这一结果，对于关系符号 $\mathcal{R}$ 上的受约束霍恩子句集$\mathcal{C}$ ，我们引入依赖关系 $\rightarrow_{\mathcal{C}} \subseteq \mathcal{R}^2$, ：若$\mathcal{C}$ 中存在一个受约束霍恩子句，其头部包含 p ，且其体包含 q ，则定义 $p \rightarrow_{\mathcal{C}} q$ 。若 $\rightarrow_{\mathcal{C}}$ 是无环的，则称集合$\mathcal{C}$ 为无递归的 ；否则称其为递归的 。

▶ **定理 19** (受约束 Horn 子句的可解性 [59])。 设 C 是理论 T 上有限且无递归的约束 *Horn* 子句集。那么，C 在语义上可解[3] ，当且仅当它在语法上可解。

为证明该结果中的“⇒”方向，可以展开无递归约束 Horn 子句，得到析取插值问题 [58], ，再将其转换为树插值问题；然后可根据树插值式模 T 提取语法解。然而，无递归约束 Horn 子句比树插值问题简洁得多，呈指数级。判定理论 LIA 上的无递归约束 Horn 子句集是否有解，是 co-NEXPTIME 完全问题；而检查一个公式是否为线性整数算术不可满足（以及是否存在插值式），则只是 co-NP 完全问题。

在无量词设定下，语义可解性与语法可解性之间的关系同样成立，前提是理论 T 支持无量词插值：

▶**引理 20** （受约束 Horn 子句的无量词可解性）。 假设 T 是一个允许普通无量词插值的理论。设 C 是一组有限且无递归的、定义在 T 上的受约束 *Horn* 子句，其中的约束不含量词。那么，C 语义可解[4] 当且仅当它在语法上可解，且存在一个不含量词的解。

3.4.2 使用受约束 Horn 子句对递归程序进行编码

受约束 Horn 子句还可用于归纳不变式之外的推理任务，从而能够自动验证递归程序或并发程序 [26]。考虑图 9 中的程序，它是第 1.1 节中斐波那契函数的递归实现。递归实现的性质可以通过应用契约式设计 原则 [53], 来验证：使用前置条件和后置条件描述函数的行为，它们的作用类似于归纳循环不变式：每次对函数的递归调用

```
1 int fibR(int n) {
2   if (n == 0)
3     return 0;
4   else if (n == 1)
5     return 1;
6   else
7     return fibR(n-1) + fibR(n-2);
8 }
```

图 9 计算斐波那契数列第 n 项的递归程序。

必须满足前置条件，然后就可以依赖函数在返回时建立后置条件。

[3] 对于具有多个模型的理论，采用脚注 1 中讨论的宽松语义。[4] 如定理 19 所示，对于具有多个模型的理论，采用脚注 1 中讨论的宽松语义。

我们将说明如何运用这种推理方式来证明关于斐波那契数列实现的一项性质：只要 $n \geq 3$，函数的结果就是一个 `fibR`$(n) \geq 2$。为这个递归函数 [8] 编写约束 Horn 子句有多种方法。选择一种接近经典契约式设计方法的编码方式，我们可以通过引入两个关系符号 前置条件 和后置条件，分别表示 `fibR` 的前置条件和后置条件，从而对函数进行推理。前置条件 前置条件(n) 是关于函数参数 n 的公式，而后置条件 后置条件$(n,\ r)$ 则是参数 n 与结果 r 之间的关系。接下来，我们编写约束 Horn 子句，说明这些符号必须满足的性质：

$$\forall n.\ n \geq 3 \to pre(n) \tag{7}$$
$$\forall n.\ n = 0 \wedge pre(n) \to post(n, 0) \tag{8}$$
$$\forall n.\ n \neq 0 \wedge n = 1 \wedge pre(n) \to post(n, 1) \tag{9}$$
$$\forall n.\ n \neq 0 \wedge n \neq 1 \wedge pre(n) \to pre(n-1) \tag{10}$$
$$\forall n.\ n \neq 0 \wedge n \neq 1 \wedge pre(n) \to pre(n-2) \tag{11}$$
$$\forall n, r, s.\ n \neq 0 \wedge n \neq 1 \wedge pre(n) \wedge post(n-1, r) \wedge post(n-2, s) \to post(n, r+s) \tag{12}$$
$$\forall n, r.\ n \geq 3 \wedge r \not\geq 2 \wedge post(n, r) \to \bot \tag{13}$$

子句 (7) 表示，待推导的契约应包含输入 $n \geq 3$；而 (13) 表示，对于输入 $n \geq 3$，结果 r 不应小于 2。其余子句则对递归函数本身进行编码。子句 (8) 和 (9) 对应第 3 行和第 5 行中的 `return` 语句，分别指出结果为 0 和 1 时必须满足后置条件。第 7 行中的第三个 `return` 语句执行两次递归调用，因此产生了稍复杂的子句：子句 (10) 和 (11) 确保递归调用满足该函数的前置条件；而 (12) 指出，假设递归调用产生的值 r、s 满足后置条件，那么计算所得的和 $r + s$ 也必须满足后置条件。子句的一组解由以下赋值给出：{pre(n) $\mapsto$ $n \geq 0$, post(n, r) $\mapsto n \geq 0 \wedge r \geq 0 \wedge (r < 2 \to n = 0 \vee (n = 1 \wedge r = 1) \vee (n = 2 \wedge r = 1))$}。

子句 (12) 被称为非线性子句，因为其体包含三个原子$pre(n)$，$post(n-1,\ r)$，$post(n-2,\ s)$，而所有其他子句都是线性的，且至多有一个体原子。将递归程序或并发程序转换为约束 Horn 子句时，自然会出现非线性子句，约束 Horn 子句求解器借助树插值来处理它们。

4 结论

本章探讨了 Craig 插值在程序验证中的应用，以及为在 SMT 求解器中有效计算 Craig 插值式而发展的一些技术。如今，Craig 插值已成为验证工具中采用的标准方法，还可以讨论许多其他方向，但这些内容超出了本章的范围，包括：

- 推导组合理论中的 Craig 插值式的技术，例如数组与线性实数算术的（签名不相交）组合。作为组合理论中进行无量词插值的一个充分标准，理论具有等式插值这一概念已被提出 [66]。后来证明，等式插值等价于强子 *amalgamation*这一模型论概念 [14]。

- 在程序验证中使用一致插值式，在此语境下也称为覆盖 [29, 23]。一致插值可用于定义一种模型检测算法，其思想类似于 IMC 算法（第 3.2 节），但该算法计算出的不变式候选 R_i 是精确的。

- 用于控制 Craig 插值结果的方法。在验证等应用中，并非所有插值式都同样有效；IMC 等算法计算归纳不变式的能力，很大程度上取决于所采用的插值过程能否找到合适的插值式。为此，人们开发了用于控制插值式的逻辑强度 [19, 31],大小 [1], 或词汇表 [18, 47] 的方法。

致谢

作者谨向 Jean Christoph Jung、Sibylle Möhle-Rotondi 和 Christoph Wernhard 致谢，感谢他们在准备本章期间认真校对并开展了富有帮助的讨论。本章的工作得到瑞典研究理事会（VR）项目 2021-06327、Wallenberg 项目 UPDATE、以太坊基金会项目 FY25-2124 以及 EuroProofNet COST 行动的支持。

参考文献

1 Leonardo Alt、Grigory Fedyukovich、Antti E. J. Hyvärinen 和 Natasha Sharygina。基于证明的紧凑命题插值式方法。载于 Arie Gurfinkel 和 Sanjit A. Seshia 编，验证软件：理论、工具与实验（*VSTTE*），计算机科学讲义第 9593 卷，计算机科学讲义，第 1–18 页。施普林格，2015 年。doi:10.1007/978-3-319-29613-5_1。2 Leo Bachmair 、 Ashish Tiwari 和 Laurent Vigneron。抽象同余闭包。 自动推理杂志，31(2)：129–168，2003 年。doi:10.1023/B:JARS.0000009518.26415.49。3 Peter Backeman、Philipp Rümmer 和 Aleksandar Zeljic。使用未解释谓词和 Presburger 算术对位向量公式进行插值。 形式化方法与系统设计，57(2)：121–156，2021 年。 doi:10.1007/S10703-021-00372-6。4 Thomas Ball、Andreas Podelski 和 Sriram K. Rajamani。用于模型检测 C 程序的布尔抽象和笛卡尔抽象。载于 Tiziana Margaria 和 Wang Yi 编，系统构造与分析工具和算法（*TACAS*），计算机科学讲义第 2031 卷，计算机科学讲义，第 268–283 页。施普林格，2001 年。doi:10.1007/3-540-45319-9_19。5 Clark W. Barrett、Roberto Sebastiani、Sanjit A. Seshia 和 Cesare Tinelli。模理论的可满足性。载于 Armin Biere、Marijn Heule、Hans van Maaren 和 Toby Walsh 编，可满足性手册 —— 第二版，人工智能与应用前沿第 336 卷，人工智能与应用前沿，第 1267–1329 页。IOS 出版社，2021 年。 doi:10.3233/FAIA201017。

6 Dirk Beyer、Nian-Ze Lee 和 Philipp Wendler。重新审视插值和基于 SAT 的模型检测：面向软件验证的应用。自动推理杂志，69(1)：5，2025年。 doi:10.1007/S10817-024-09702-9。7 Armin Biere、Alessandro Cimatti、Edmund M. Clarke 和 Yunshan Zhu。无需二元决策图的符号模型检测。载于 W. Rance Cleaveland 编，系统构造与分析工具和算法（TACAS），计算机科学讲义第 1579 卷，第 193–207 页。施普林格，1999 年。 doi:10.1007/3-540-49059-0_14。8 Nikolaj S. Bjørner、Arie Gurfinkel、Kenneth L. McMillan 和 Andrey Rybalchenko。用于程序验证的霍恩子句求解器。载于 Lev D. Beklemishev、Andreas Blass、Nachum Dershowitz、Bernd Finkbeiner 和 Wolfram Schulte 编，逻辑与计算领域 II，计算机科学讲义第 9300 卷，第 24–51 页。施普林格，2015 年。 doi:10.1007/978-3-319-23534-9_2。9 Aaron R. Bradley。无需展开的基于 SAT 的模型检测。载于 Ranjit Jhala 和 David Schmidt 编，验证、模型检测与抽象解释（VMCAI），计算机科学讲义第 6538 卷，第 70–87 页。施普林格，2011 年。 doi:10.1007/978-3-642-18275-4_7。10 Aaron R. Bradley 和 Zohar Manna。计算演算——用于验证的判定过程及其应用。施普林格，2007 年。 doi:10.1007/978-3-540-74113-8。11 Angelo Brillout、Daniel Kroening、Philipp Rümmer 和 Thomas Wahl。Presburger 算术扩展中的无量词插值之外。载于 Ranjit Jhala 和 David A. Schmidt 编，验证、模型检测与抽象解释（VMCAI），计算机科学讲义第 6538 卷，第 88–102 页。施普林格，2011 年。 doi:10.1007/978-3-642-18275-4_8。12 Angelo Brillout、Daniel Kroening、Philipp Rümmer 和 Thomas Wahl。无量词 Presburger 算术的插值序列演算。自动推理杂志，47(4)：341–367，2011 年。 doi:10.1007/S10817-011-9237-Y。13 Roberto Bruttomesso、Silvio Ghilardi 和 Silvio Ranise。基于重写的数组理论无量词插值。载于 Manfred Schmidt-Schauß 编，重写技术与应用（RTA），LIPIcs 第 10 卷，第 171–186 页。莱布尼茨国际计算机科学研究中心 Schloss Dagstuhl，2011 年。 doi:10.4230/LIPICS.RTA.2011.171。14 Roberto Bruttomesso、Silvio Ghilardi 和 Silvio Ranise。等式插值理论组合中的无量词插值。ACM 计算逻辑汇刊，15(1)：5:1–5:34，2014 年。 doi:10.1145/2490253。15 Edmund M. Clarke、Orna Grumberg、Somesh Jha、Yuan Lu 和 Helmut Veith。用于符号模型检测的反例引导抽象细化。ACM 杂志，50(5)：752–794，2003 年。doi:10.1145/876638.876643。16 Liyun Dai、Bican Xia 和 Naijun Zhan。通过半定规划生成非线性插值式。载于 Natasha Sharygina 和 Helmut Veith 编，计算机辅助验证（CAV），计算机科学讲义第 8044 卷，第 364–380 页。施普林格，2013 年。 doi:10.1007/978-3-642-39799-8 25。_17 Isil Dillig、Thomas Dillig 和 Alex Aiken。证明导出的割：一种用于求解整数上线性不等式的完备实用技术。形式化方法与系统设计，39(3)：246–260，2011 年。 doi:10.1007/S10703-011-0127-Z。18 Vijay D’Silva。命题插值与抽象解释。载于 Andrew D. Gordon 编，欧洲程序设计研讨会（ESOP），计算机科学讲义第 6012 卷，第 185–204 页。施普林格，2010 年。 doi:10.1007/978-3-642-11957-6_11。

19 Vijay Victor D'Silva、Daniel Kroening、Mitra Purandare 和 Georg Weissenbacher。插值式的强度。载于 Gilles Barthe 和 Manuel V. Hermenegildo（编），《验证、模型检测与抽象解释》（VMCAI），计算机科学讲义第 5944 卷，第 129–145 页。施普林格，2010年。doi:10.1007/978-3-642-11319-2_12。 20 Bruno Dutertre 和 Leonardo Mendonça de Moura。用于 DPLL(T) 的快速线性算术求解器。载于 Thomas Ball 和 Robert B. Jones（编），《计算机辅助验证》（CAV），计算机科学讲义第 4144 卷，第 81–94 页。施普林格，2006年。doi:10.1007/11817963_11。 21 Cormac Flanagan 和 James B. Saxe。避免指数级膨胀：生成紧凑的验证条件。载于 Chris Hankin 和 Dave Schmidt（编），《编程语言原理》（POPL），第 193–205 页。ACM，2001年。 doi:10.1145/360204.360220。 22 Alexander Fuchs、Amit Goel、Jim Grundy、Sava Krstic 和 Cesare Tinelli。等式理论的基插值。《逻辑方法与计算机科学》，8(1)，2012年。 doi:10.2168/LMCS-8(1:6)2012。 23 Alessandro Gianola。面向数据感知业务流程自动验证的一致插值。载于 Luca Geatti、Guido Sciavicco 和 Alessandro Umbrico（编），《人工智能与形式验证、逻辑、自动机和综合研讨会》，CEUR Workshop Proceedings 第 3311 卷，第 73–79 页。CEUR-WS.org，2022年。 24 Susanne Graf 和 Hassen Saïdi。使用 PVS 构造抽象状态图。载于 Orna Grumberg （编），《计算机辅助验证》（ CAV ），计算机科学讲义第 1254 卷，第 72–83 页。施普林格，1997年。 doi:10.1007/3-540-63166-6_10。 25 Ronald L. Graham、Donald E. Knuth 和 Oren Patashnik。《具体数学：计算机科学基础》，第 2 版。Addison-Wesley，1994年。 26 Sergey Grebenshchikov 、 Nuno P. Lopes 、 Corneliu Popeea 和 Andrey Rybalchenko。从证明规则合成软件验证器。载于 Jan Vitek、Haibo Lin 和 Frank Tip（编），《程序设计语言设计与实现》（PLDI），第 405–416 页。ACM，2012年。doi:10.1145/2254064.2254112。 27 Alberto Griggio、Thi Thieu Hoa Le 和 Roberto Sebastiani。线性整数算术可满足性模理论中的高效插值生成。《逻辑方法与计算机科学》，8(3)，2010年。doi:10.2168/LMCS-8(3:3)2012。 28 Orna Grumberg。模型检测中的抽象与精化。载于 Frank S. de Boer、Marcello M. Bonsangue、Susanne Graf 和 Willem P. de Roever（编），《组件与对象形式化方法》（FMCO），计算机科学讲义第 4111 卷，第 219–242 页。施普林格，2005年。doi:10.1007/11804192_11。 29 Sumit Gulwani 和 Madan Musuvathi。覆盖算法及其组合。载于 Sophia Drossopoulou（编），《欧洲程序设计研讨会》（ESOP），计算机科学讲义第 4960 卷，第 193–207 页。施普林格，2008年。doi:10.1007/978-3-540-78739-6_16。 30 Ashutosh Gupta、Corneliu Popeea 和 Andrey Rybalchenko。求解 LI+UIF 上的无递归 Horn 子句。载于 Hongseok Yang（编），《编程语言与系统》（APLAS），计算机科学讲义第 7078 卷，第 188–203 页。施普林格，2011年。doi:10.1007/978-3-642-25318-8_16。 31 Arie Gurfinkel、Simone Fulvio Rollini 和 Natasha Sharygina。插值性质与基于 SAT 的模型检测。载于 Dang Van Hung 和 Mizuhito Ogawa（编），《验证与分析自动化技术》（ATVA），计算机科学讲义第 8172 卷，第 255–271 页。施普林格，2013年。 doi:10.1007/978-3-319-02444-8_19。

32 John Harrison。《实用逻辑与自动推理手册》。剑桥大学出版社，2009年。 33 Matthias Heizmann、Jochen Hoenicke 和 Andreas Podelski。嵌套插值式。载于 Manuel V. Hermenegildo 和 Jens Palsberg（编），《编程语言原理》（POPL），第 471–482 页。ACM，2010年。 doi:10.1145/1706299.1706353。 34 Thomas A. Henzinger、Ranjit Jhala、Rupak Majumdar 和 Kenneth L. McMillan。从证明中抽象。载于 Neil D. Jones 和 Xavier Leroy（编），《编程语言原理》（POPL），第 232–244 页。ACM，2004年。 doi:10.1145/964001.964021。 35 Thomas A. Henzinger、Ranjit Jhala、Rupak Majumdar 和 Grégoire Sutre。惰性抽象。载于 John Launchbury 和 John C. Mitchell（编），《编程语言原理》（POPL），第 58–70 页。ACM，2002年。 doi:10.1145/503272.503279。 36 Jochen Hoenicke 和 Tanja Schindler。数组理论的高效插值。载于 Didier Galmiche、Stephan Schulz 和 Roberto Sebastiani（编），《自动推理》（IJCAR），计算机科学讲义第 10900 卷，第 549–565 页。施普林格，2018年。 doi:10.1007/978-3-319-94205-6_36。 37 Hossein Hojjat 和 Philipp Rümmer。通过归约判定并插值代数数据类型。载于 Tudor Jebelean、Viorel Negru、Dana Petcu、Daniela Zaharie、Tetsuo Ida 和 Stephen M. Watt（编），《科学计算的符号与数值算法》（SYNASC），第 145–152 页。IEEE计算机学会，2017年。 doi:10.1109/SYNASC.2017.00033。 38 Himanshu Jain、Edmund M. Clarke 和 Orna Grumberg。线性丢番图（不）等式与线性模方程的高效 Craig 插值。《形式化方法与系统设计》，35(1)：6–39，2009年。 doi:10.1007/S10703-009-0069-X。 39 Ranjit Jhala 和 Kenneth L. McMillan。基于插值项的转移关系近似。《逻辑方法与计算机科学》，3(4)，2007年。 doi:10.2168/LMCS-3(4:1)2007。 40 Dejan Jovanovic 和 Bruno Dutertre。非线性算术的插值与模型检测。载于 Alexandra Silva 和 K. Rustan M. Leino（编），《计算机辅助验证》（CAV），计算机科学讲义第 12760 卷，第 266–288 页。施普林格，2021年。URL: https://doi.org/10.1007/978-3-030-81688-9_13, doi:10.1007/978-3-030-81688-9_13。 41 Deepak Kapur、Rupak Majumdar 和 Calogero G. Zarba。数据结构的插值。载于 Michal Young 和 Premkumar T. Devanbu（编），《软件工程基础》（FSE），第 105–116 页。ACM，2006年。 doi:10.1145/1181775.1181789。 42 Anvesh Komuravelli、Arie Gurfinkel 和 Sagar Chaki。递归程序的基于SMT的模型检查。《形式化方法与系统设计》，48(3)：175–205，2016年。 doi:10.1007/S10703-016-0249-4。 43 George Kourousias 和 David Makinson。信念变化中的并行插值、分解与相关性。《符号逻辑杂志》，72(3)：994–1002，2007年。 doi:10.2178/jsl/1191333851。 44 Laura Kovács 和 Andrei Voronkov。插值与符号消除。载于 Renate A. Schmidt（编），《自动演绎》（CADE），计算机科学讲义第 5663 卷，第 199–213 页。施普林格，2009年。 doi:10.1007/978-3-642-02959-2_17。 45 Daniel Kroening、Jérôme Leroux 和 Philipp Rümmer。无量词 Presburger 算术的插值。载于 Christian G. Fermüller 和 Andrei Voronkov（编），《程序设计、人工智能与推理逻辑》（LPAR），计算机科学讲义第 6397 卷，第 489–503 页。施普林格，2010年。 doi:10.1007/978-3-642-16242-8_35。 46 Serge Lang。《代数》（第 3 版）。Addison-Wesley，1993年。

40 程序验证中的 Craig 插值

47 Jérôme Leroux、Philipp Rümmer 和 Pavle Subotic。利用领域特定抽象引导 Craig 插值。Acta Informatica，53(4):387–424，2016年。doi:10.1007/S00236-015-0236-Z。48 John McCarthy。迈向计算的数学科学。载于 C. M. Popplewell（编），《1962 年信息处理大会：IFIP 第 62 届大会论文集》，第 21–28 页，阿姆斯特丹，1963 年。北荷兰。49 Kenneth L. McMillan。插值和基于 SAT（布尔可满足性问题）的模型检测。载于 Warren A. Hunt Jr. 和 Fabio Somenzi（编），《计算机辅助验证》（CAV），计算机科学讲义第 2725 卷，第 1–13 页。施普林格，2003年。doi:10.1007/978-3-540-45069-6_1。50 Kenneth L. McMillan。Craig 插值式在模型检测中的应用。载于 Nicolas Halbwachs 和 Lenore D. Zuck（编），《系统构造与分析工具和算法》（TACAS），计算机科学讲义第 3440 卷，第 1–12 页。施普林格，2005年。 doi:10.1007/978-3-540-31980-1_1。51 Kenneth L. McMillan。一个插值定理证明器。理论计算机科学，345(1):101–121，2005年。 doi:10.1016/J.TCS.2005.07.003。52 Kenneth L. McMillan。利用插值式进行惰性抽象。载于 Thomas Ball 和 Robert B. Jones（编），《计算机辅助验证》（CAV），计算机科学讲义第 4144 卷，第 123–136 页。施普林格，2006年。 doi:10.1007/11817963_14。53 Bertrand Meyer。应用“契约式设计”。IEEE Computer，25(10):40–51，1992年。doi:10.1109/2.161279。54 Rohit Parikh。Beth 可定义性、插值与语言分裂。Synth.，179(2):211–221，2011年。 doi:10.1007/S11229-010-9778-3。55 Pavel Pudlák。归结证明、割平面证明和单调计算的下界。J. Symb. Log.，62(3):981–998，1997年。 doi:10.2307/2275583。56 William Pugh。Omega 测试：一种用于依赖分析的快速实用整数规划算法。载于 Joanne L. Martin（编），《超级计算》（SC），第 4–13 页，美国纽约州纽约市，1991年。ACM。57 Philipp Rümmer。一阶逻辑与线性整数算术的约束序列演算。载于 Iliano Cervesato、Helmut Veith 和 Andrei Voronkov（编），《程序设计、人工智能与推理逻辑》（LPAR），计算机科学讲义第 5330 卷，第 274–289 页。施普林格，2008年。 doi:10.1007/978-3-540-89439-1_20。58 Philipp Rümmer、Hossein Hojjat 和 Viktor Kunčak。用于 Horn 子句验证的析取插值式。载于《计算机辅助验证》（CAV），计算机科学讲义第 8044 卷，第 347–363 页。施普林格，2013年。doi:10.1007/978-3-642-39799-8_24。59 Philipp Rümmer、Hossein Hojjat 和 Viktor Kunčak。关于无递归 Horn 子句与 Craig 插值。形式化方法与系统设计，47(1):1–25，2015年。doi:10.1007/S10703-014-0219-7。60 Andrey Rybalchenko 和 Viorica Sofronie-Stokkermans。用于插值的约束求解。J. Symb. Comput.，45(11):1212–1233，2010年。doi:10.1016/J.JSC.2010.06.005。61 Alexander Schrijver。线性规划与整数规划理论。约翰·威立父子公司，1998年。62 Nishant Totla 和 Thomas Wies。完备的基于实例化的插值。载于 Roberto Giacobazzi 和 Radhia Cousot（编），《编程语言原理》（POPL），第 537–548 页。ACM，2013年。 doi:10.1145/2429069.2429132。

63 Yakir Vizel 和 Orna Grumberg。基于插值序列的模型检测。载于 Armin Biere 和 Carl Pixley（编），《计算机辅助设计中的形式化方法》（*FMCAD*），第 1–8 页。IEEE，2009年。 doi:10.1109/FMCAD.2009.5351148。64 Christoph Wernhard。从谓词逻辑视角看布尔解问题。载于 Clare Dixon 和 Marcelo Finger（编），《组合系统前沿》（*FroCoS*），计算机科学讲义第 10483 卷，计算机科学讲义，第 333–350 页。施普林格，2017年。 doi:10.1007/978-3-319-66167-4_19。65 Hao Wu、Jie Wang、Bican Xia、Xiakun Li、Naijun Zhan 和 Ting Gan。通过分离半代数集，在无界域上生成非线性 Craig 插值。载于 André Platzer、Kristin Yvonne Rozier、Matteo Pradella 和 Matteo Rossi（编），《形式化方法》（*FM*），计算机科学讲义第 14933 卷，计算机科学讲义，第 92–110 页。施普林格，2024年。 doi:10.1007/978-3-031-71162-6_5。66 Greta Yorsh 和 Madanlal Musuvathi。一种生成插值式的组合方法。载于 Robert Nieuwenhuis（编），《自动演绎》（*CADE*），计算机科学讲义第 3632 卷，计算机科学讲义，第 353–368 页。施普林格，2005年。 doi:10.1007/11532231_26。