

1 简介

最近不得不做一些和 SMT dialect 有关的形式化验证，而第一步肯定是去看 Lean 的 `bv_normalize` 和 `bv_decide` 这两个别人已经做好的东西。

2 标准化

`bv_normalize` 会将较复杂的语法降低到 `bv_decide` 所能接受的十几个 AST 节点上。为了保证改写正确，它会在返回新表达式的同时，返回它与原表达式等价的证明。

考虑到 `bv_decide` 是依靠 SAT 求解器返回不可行性证明的，它们都需要将命题改写为从假设推出 False。所以，如果目标不是 False，就将其取反后作为新的假设引入。这一切都是用各种神秘的反射/元编程方式完成的，细节是真的看不懂一点。

不管怎么说，既然目标已经固定，我们现在只需要看内部假设的简化方式了。在收集完假设之后，`bv_normalize` 会对它们执行类型论中的几个标准规约。

β -规约是函数的应用， ζ -规约是 `let` 的展开。不过，Lean 提供了两种 ζ -规约，普通的 ζ 只会展开目标中的 `let`，而 $\zeta\text{-}\delta$ 规约则会将假设中的 `let` 也一并展开。

接下来它会展开 `struct` 和 `enum`，不过是否展开、展开哪些似乎是可以选项控制的。

`bv_normalize` 内部好像有一套复杂的规则确定哪些是值得展开的。总之我没看懂，怎么展开的也没看懂。似乎是 `struct` 会变为新的假设，而 `enum` 会变为 `bitvector`。

当然，固定宽度的整数自然也可以用 `bitvector` 表示。

转换完成之后，下面就是简化了。它会填充 `pass pipeline` 之后迭代到不动点。具体的简化似乎主要是依靠一大堆标了 `@[bv_normalize]` 的引理完成的。

3 证明

在 `bv_normalize` 结束后，剩下的假设会进入 `bv_decide`。它会利用反射将命题翻译为 AST，然后将它转化为与-反相图 (AIG)，最后生成 CNF 子句，喂给 SAT 求解器。SAT 会返回一个不可满足性核，Lean 也需要检查，不过我们这里不管。

3.1 AST 的结构

到这里之后，所有的命题应当都是判断位向量表达式之间的关系，不过命题之间也可以自由组合。因此，AST 一共有三个部分。最上面的是命题逻辑，有真假常量、布尔函数之类的，它之下是谓词（例如向量相等、小于、某一位为 1），然后就是位向量的 AST 了。

这个 AST 也是可以解释执行的。

要构造这样的 AST，只需要将 Lean 的语法一一映射即可。无法处理的部分会认为是一个不透明的原子变量。如果这样处理之后命题不可满足，那么原命题自然不可满足；然而反过来并不成立。所以，`bv_decide` 有的时候会提示 “a spurious counterexample”。

3.2 AIG

AIG 是 AST 进一步降低之后的结果，只剩下逻辑非和逻辑与，当然还有外部输入。这是电路优化里常见的格式，不过其实并不是最优的，因为我已经试着做过一个可以产生更少子句的 `bit-blaster` 了。

不过，AIG 的具体转换值得研究一下。我们希望获得最少的与门，这是因为非门在 CNF 中不会添加任何子句，因为每个变量本来就会变成一对文字 x 和 $\sim x$ 。

3.2.1 位运算

与、或显然是一个门。异或是 $x \oplus y = (x \vee y) \wedge \sim (x \wedge y)$ ，所以需要三个。

对于移位，常数距离的移位是显然的，而变量距离的则是分级移位器。

3.2.2 四则运算

这是普通的行波进位加法器。

$$s = x \oplus y \oplus c, \quad c' = (x \oplus y) \wedge c \vee (x \wedge y)$$

考虑到 hash-consing, $x \oplus y$ 只有一个节点。那么一个全加器一共是 9 个门。还有一个只算进位的变体, 用来判断溢出的。

乘法是移位相加, 所以需要一个额外的 multiplexer。这真的没啥好说的。

除法是长除法。

3.3 比较

相等比较自然是每一位异或之后取反, 最后 $\wedge$ 起来。

对于无符号比较, $a < b$ 可以看作是 $a - b$ 借位了, 所以它等价于 $a + \neg b + 1$ 的进位为 0。

有符号比较会提前改写为无符号比较, 再加上符号位的比较即可。