Assignment 5

April 17, 2019

1 Interpolants

This question concerns the logical notion of *Craig interpolants*. You do not require prior knowledge about interpolation to answer this question.

Given two formulas A and B in first-order logic, such that $A \wedge B$ is unsatisfiable, there exists a formula I, called an interpolant, such that

1. $A \Rightarrow I$ is valid (recall that a formula ϕ is valid iff all models satisfy it)
2. $I \wedge B$ is unsatisfiable;
3. $vars(I) \subseteq vars(A) \cap vars(B)$, where $vars(\phi)$ is the set of all variables that appear in ϕ.

As an example, consider the following formulas in propositional logic (i.e., all variables are Boolean):

$$A \triangleq a \wedge b$$
$$B \triangleq \neg b \wedge c$$

We know that $A \wedge B$ is unsatisfiable. An interpolant I here is b. Observe that $A \Rightarrow I$ is valid, $I \wedge B$ is unsatisfiable, and I only contains variables that appear in A and B.

1.1

An alternative definition of an interpolant is as follows:

Suppose we have two formulas A and C such that $A \Rightarrow C$ is valid, then there exists a formula I such that

1. $A \Rightarrow I$ is valid;
2. $I \Rightarrow C$ is valid;
3. $vars(I) \subseteq vars(A) \cap vars(C)$.

Prove that the two definitions of an interpolant are equivalent.

1.2

Give two formulas A and B such that $A \wedge B$ is unsatisfiable, does there always exist a unique interpolant (up to logical equivalence)? If not, provide an example of two formulas A and B and two interpolants I_1 and I_2, such that $I_1 \neq I_2$.

1.3

Suppose you are working with formulas in quantifier-free linear integer arithmetic: meaning, formulas that are Boolean combinations (conjunctions, disjunctions, negations) of linear inequalities over integers of the form: $a_1x_1 + \ldots a_nx_n \leq c$, where a_i, c are integer constants, and x_i are integer variables.

Consider the following two formulas in quantifier-free linear integer arithmetic:

$$A \triangleq x = 2y$$
$$B \triangleq x = 2z - 1$$

Is $A \wedge B$ satisfiable? If not, does there exist an interpolant for A and B that is also in quantifier-free linear integer arithmetic? If no such interpolant exists, explain why that is the case.

2 Extending Static Analysis to a Probabilistic Setting

Let $\mathcal{L}$ be a very simple programming language where a program $P \in \mathcal{L}$ is comprised of assignment, conditional, and while-loop statements. Assume also that all variables are either Boolean or real-valued, and that all programs of interest are well-typed and exhibit no runtime errors (e.g., divisions by zero).

For a program P, we assume it has a set of input variables V_i and a set of output variables V_r. For the rest of this question, imagine that we have at our disposal an almighty static analyzer $\mathcal{A}$ that, given $P \in \mathcal{L}$, returns a set S of all states reachable by executing P from any possible input. A state $s \in S$ is considered to be a map from every variable in P to a value. Given variable x, we use $s[x]$ to denote the value of x in s.

Example 1 For illustration, consider the following example:

```
def p(x)
  y = x + 1
  return y
```

Given the above program, the static analyzer returns the set

$$\{s \mid s[x] = c, s[y] = c + 1, c \in \mathbb{R}\}$$

In other words, it returns the set of all states s where $y = x + 1$.

Example 2 We also assume that our language $\mathcal{L}$ allows non-deterministic choice, which is denoted by `if (*) ... else ...`, where the program can non-deterministically execute either branch of the conditional statement. Consider, for example, the following simple program:

```
def p()
  if (*)
    y = 1
  else
    y = 2
  return y
```

Given the above program, the static analyzer returns the set

$$\{s \mid s[y] \in \{1, 2\}\}$$

2.1 Using $\mathcal{A}$ for probabilistic reasoning

Suppose that we decide to extend our language $\mathcal{L}$ into a new language $\mathcal{L}^{\sim}$ with random assignments of the form

```
  x ~ bern(c)
```

where Boolean variable `x` is assigned true with probability `c` and false with probability `1-c`, where `c` is a constant in $[0, 1]$. (`bern` stands for a Bernoulli distribution.)

Given a program $P \in \mathcal{L}^{\sim}$, we are typically interested in the probability of the program returning a specific set of values. For example, consider the following probabilistic program:

```
def p()
  x ~ bern(0.5)
  y ~ bern(0.5)
  z =  x && y
  return z
```

The probability that `p` returns true is 0.25, as both `x` and `y` have to be true.

Your task in this question is as follows: Given a program $P \in \mathcal{L}^{\sim}$ and the static analyzer $\mathcal{A}$, you are to use $\mathcal{A}$ to compute the probability that P returns a value in some set X. Note that $\mathcal{A}$ only accepts programs in $\mathcal{L}$, so you have to somehow transform $P \in \mathcal{L}^{\sim}$ into a new program in $\mathcal{L}$ in order to be able to use $\mathcal{A}$. Once you have the output set S of $\mathcal{A}$, you may apply any mathematical operation on S to extract your desired result.

You should assume that P is always terminating, has no non-deterministic conditionals, has no input variables (like the above example), and only manipulates Boolean variables. **Hint:** your transformation of P may introduce non-determinism, real-valued variables, and lists.

2.2 Discovering maximizing inputs

In the first part of the question, we assumed that P has no input variables. In this part, we will assume that P has input variables. Our goal is to find a value for the input variables that maximizes the probability of returning some output value. Consider the following example:

```
def p(x)
  if (x)
    y ~ bern(0.5)
  else
    y ~ bern(0.9)
  return y
```

Suppose we want to maximize the probability that `p` returns the value true. To do so, we have to set the input variable `x` to false in order to force the program down the else branch of `p`, which has a higher probability of setting `y` to true.

Describe how you would extend your technique from Part 2.1 to this setting.

3 Galois connections

Consider these two definitions of Galois connections:

Definition 1: (L, α, γ, M) is a Galois connection between the complete lattices $(L, \sqsubseteq)$ and $(M, \sqsubseteq)$ if and only if $\alpha : L \to M$ and $\gamma : M \to L$ are monotone functions that satisfy the following conditions:

$$\text{forall } l \in L.\, \gamma(\alpha(l)) \sqsupseteq l$$

$$\text{forall } m \in M.\, \alpha(\gamma(m)) \sqsubseteq m$$

where $\circ$ is function composition—that is, $\alpha \circ \gamma$ is the function that applies α *to the result of* γ.

Definition 2: (L, α, γ, M) is a Galois connection between the complete lattices $(L, \sqsubseteq)$ and $(M, \sqsubseteq)$ if and only if $\alpha : L \to M$ and $\gamma : M \to L$ are total functions such that for all $l \in L, m \in M$,

$$\alpha(l) \sqsubseteq m \Leftrightarrow l \sqsubseteq \gamma(m)$$

Prove that the two definitions are equivalent.