Proyecto de curso

Planteo de proyecto para el curso de Lenguajes de Programación 2021 (Ing. en Informática, FIT, UCU).

Se trata de un experimento simple en *programación automática*. Es decir, la síntesis automática de un programa a partir de una especificación. En este caso, en base a una tabla de verdad dada, se desea obtener una expresión de cálculo proposicional.

Fase 0: preparación

El equipo primero debe elegir uno de los lenguajes dados en el curso para implementar el proyecto: *C*, *Elixir*, *JavaScript*, *Prolog* o *Ruby*. La totalidad del código debe hacerse en uno y solo uno de esos lenguajes, usando la distribución oficial estándar del mismo. Si el equipo piensa que necesita usar una biblioteca específica, deberá confirmarlo con los docentes.

El siguiente paso es definir una estructura de datos para representar expresiones de cálculo proposicional. Ésta debe incluir:

- variables proposicionales, e.g. (p) o (q).
- negación, e.g. $(\neg p)$.
- conjunción, e.g. $(p \wedge q)$ o $(\neg(p \wedge q))$.
- disjunción, e.g. $(p \vee q)$ o $(p \vee (\neg q \wedge r)))$.
- condicional, e.g. $(p \to q)$ o $(p \vee q \to p \wedge q)$.
- bicondicional, e.g. $(p \leftrightarrow q)$ o $(p \leftrightarrow \neg p)$.

Llámese la representación `Prop`, se deben implementar las siguientes funciones:

- `randomProp(rng, vars, maxHeight, minHeight)` calcula una `Prop` aleatoria.
  - `rng` será un generador de números aleatorios cuya semilla se debe controlar.
  - `vars` será una lista de nombres de variables.
  - `maxHeight` será la altura máxima del árbol de expresión resultante.
  - `minHeight` será la altura mínima del árbol de expresión resultante.
- `evalProp(prop, value)` evalúa una `Prop` dado valores para sus variables.
  - `prop` será una `Prop` a evaluar.
  - `value` será un mapa de nombres de variables a valores de verdad.
- `truthTable(prop, vars)` retorna la *tabla de verdad* de una `Prop`, i.e. una lista de pares *(valores, resultado)*.
  - `prop` será una `Prop` a evaluar.
  - `vars` será una lista de nombres de variables.

Fase 1: búsqueda aleatoria

La primera prueba se trata de implementar una búsqueda aleatoria. Dada una tabla de verdad, se deben generar una determinada cantidad de expresiones aleatorias, y seleccionar aquellas cuya evaluación sea más cercana.

Una búsqueda aleatoria funcionaría de la siguiente forma:

```
1: steps ← maximum amount of expressions to generate
2: step ← 0
3: bestProp ← null
4: bestFitness ← −∞
5: while bestFitness < 1 ∧ step < steps do
6:     step ← step + 1
7:     prop ← randomProp()
8:     fitness ← fitness(prop)
9:     if fitness > bestFitness then
10:        bestProp ← prop
11:        bestFitness ← fitness
12:    end if
13: end while
```

14: $result \leftarrow bestProp$

Se deben implementar las siguientes funciones:

- `randomTruthTable(rng, vars)` calcula una tabla de verdad aleatoria para poder probar la búsqueda.
  - `rng` será un generador de números aleatorios.
  - `vars` será la lista de nombres de variables.
- `fitness(prop, truthTable)` calcula la proporción de casos en los que la evaluación de la proposición dada coincide con la tabla de verdad dada.
  - `prop` será una proposición `Prop` a evaluar.
  - `truthTable` será una lista de pares *(valores, resultado)*.
- `randomSearch(rng, truthTable, count, propArgs)` ejecuta la *búsqueda aleatoria*, retornando la primera expresión encontrada con un `fitness` de 1.
  - `rng` será el generador de números aleatorios a usar.
  - `truthTable` será una lista de parse *(valores, resultado)*.
  - `count` será la cantidad de expresiones aleatorias a generar y probar.
  - `propArgs` es el conjunto de argumentos a usar en la generación aleatoria de expresiones: `vars`, `maxHeight` y `minHeight`.

De desea estimar cuantas expresiones aleatorias se debería generar en promedio para encontrar una expresión que aplique a una tabla de verdad dada, para 1 y 2 variables.

A los efectos de poder verificar la correctitud de la implementación, es conveniente hacer dos cosas. Primero implementar una buena cantidad de pruebas unitarias. Segundo que la búsqueda aleatoria imprima una bitácora de las expresiones generadas y su aptitud, para poder auditar su funcionamiento sin tener que ejecutar paso a paso.

Fase 2: estrategia evolutiva

La segunda prueba se trata de implementar un algoritmo evolutivo por mutación llamado *estrategia evolutiva*. Un algoritmo evolutivo mantiene un conjunto de *individuos* llamado *población*. Cada individuo tiene una expresión `Prop` y un valor de aptitud o *fitness* (nulo inicialmente).

Una estrategia evolutiva funcionaría de la siguiente forma:

1: $count \leftarrow$ amount of individuals in the population
2: $steps \leftarrow$ maximum amount of steps to take
3: $step \leftarrow 0$
4: $population \leftarrow initialPopulation()$
5: $assessPopulation(population)$
6: $best \leftarrow$ individual in population with higher fitness
7: **while** $bestFitness < 1 \land step < steps$ **do**
8: $\quad step \leftarrow step + 1$
9: $\quad population \leftarrow$ count mutated individuals selected from the previous population
10: $\quad assessPopulation(population)$
11: $\quad best \leftarrow$ individual in population with higher fitness
12: **end while**
13: $result \leftarrow best$

Se deben implementar las siguientes funciones:

- `initialPopulation(rng, vars, count)` calcula los individuos con los cuales inicia algoritmo evolutivo.
  - `rng` será el generador de números aleatorios a usar.
  - `vars` será la lista de nombres de variables.
  - `count` será la cantidad de expresiones a generar.
- `assessPopulation(population, truthTable)` agrega la aptitud a cada individuo en la población.
  - `population` será la población del algoritmo evolutivo.
  - `truthTable` será la tabla de verdad contra la cual calcular la aptitud de la expresión de cada individuo.

- `selection(rng, population, count)` selecciona individuos de la población aleatoriamente, con una probabilidad proporcional a la aptitud del individuo.
  - `rng` será el generador de números aleatorios a usar.
  - `population` será la población del algoritmo evolutivo.
  - `count` será la cantidad de individuos a seleccionar.
- `mutation(rng, prop, propArgs)` calcula una nueva expresión como modificación por una expresión dada. Se toma un nodo del árbol de expresión al azar y se lo reemplaza por un subárbol generado al azar.
  - `rng` será el generador de números aleatorios a usar.
  - `prop` será una expresión `Prop` a mutar.
  - `propArgs` es el conjunto de argumentos a usar en la generación aleatoria de expresiones: `vars`, `maxHeight` y `minHeight`. La mutación debe respetar las restricciones de altura.
- `evolutionStrategy(rng, truthTable, steps, count, propArgs)` ejecuta la evolución por mutación, retornando el mejor individuo.
  - `rng` será el generador de números aleatorios a usar.
  - `truthTable` será la tabla de verdad de referencia.
  - `steps` será la cantidad máxima de pasos a ejecutar.
  - `count` será la cantidad de individuos de la población.
  - `propArgs` es el conjunto de argumentos a usar en la generación aleatoria de expresiones: `vars`, `maxHeight` y `minHeight`.

Al igual que con la búsqueda aleatoria, se desea estimar cuantos pasos se deberían dar en promedio para encontrar una expresión que aplique a una tabla de verdad dada, para 1 y 2 variables.

A los efectos de poder verificar la correctitud de la implementación, es conveniente hacer dos cosas. Primero implementar una buena cantidad de pruebas unitarias. Segundo que la estrategia evolutiva imprima estadísticas sobre la población en cada paso. Por esto se entiende mostrar el número de paso actual, las aptitudes máxima, promedio y mínima de toda la población, y el mejor individuo actual.

Extras

Quedan como tareas opcionales de este trabajo:

1. Realizar los experimentos con 3 variables, además de 1 y 2.
2. Realizar los experimentos con tablas de verdad que correspondan con una expresión generada al azar, en lugar de tablas de verdad completamente aleatorias.

Entrega

La entrega se realizará el viernes 26 de noviembre de 2021 a las 23:55 horas, vía Webasignatura. Luego cada miembro del equipo tomará una defensa escrita (también vía Webasignatura), el viernes 3 de diciembre a las 20:00 horas.

La entrega será un único archivo comprimido (formatos Zip, 7z, GZip o BZ2) con una carpeta con todos los fuentes del proyecto. No se debe incluir código generado, dependencias o archivos del versionado. Se espera además un muy breve reporte de los experimentos realizados, describiendo los datos obtenidos y las conclusiones a las que el equipo llega con éstos.

Si el equipo lo desea puede indicar un repositorio en línea donde ver la evolución del trabajo.

Fin