Algorítmos y Estructuras de Datos II
Trabajo Práctico 1 (Especificación)
Grupo 6

Bayardo, Julián
julian@bayardo.com.ar
850/13

Cuneo, Christian
chriscuneo93@gmail.com
755/13

Gambaccini, Ezequiel
ezequiel.gambaccini@gmail.com
715/13

Lebrero Rial, Ignacio Manuel
ignaciolebrero@gmail.com
751/13

9 de Septiembre del 2014

1. Aclaraciones

Asumimos que las estaciones, junto con las sendas y las restricciones entre ellas no son mutables: una instancia del TAD Ciudad comienza con un mapa definido y no puede ser cambiado. Suponemos también que las estaciones pueden tener sendas que las conecten a sí mismas, así como una cantidad contable de sendas que la conecten a otra estación definida, siempre y cuando sean consideradas distintas. Aparte, los robots pueden tomar cualquier senda que los una con otra estación, incluso si esa senda le agrega una infracción, y sin importar que haya otras sendas que lo unan directamente y no le agreguen infracciones (es decir, nunca se asume que el robot vaya a tomar un camino que minimice las multas). Aparte, consideramos que pueden haber estaciones no conexas, y en cuyo caso no son consideradas bloqueantes (aunque los robots en ellas no podrían moverse hacia ningún lado).

Consideramos que dos robots son equivalentes cuando su RUR (que es asignado automáticamente por el modelo) lo es. Los RUR se asignan desde el 1 de forma incremental, y nunca se repiten: si un robot es borrado por medio de una inspección, ningún otro robot va a tener el mismo RUR asignado. No puede existir un robot que no tenga ninguna característica.

Aparte, consideramos que dos sendas entre dos estaciones iguales son equivalentes cuando las restricciones impuestas sobre ellas lo son. Dos restricciones son equivalentes cuando su tabla de verdad es la misma para todo posible conjunto de características. Cabe destacar que consideramos que dos ciudades son equivalentes cuando las estaciones, sus conexiones, y el conjunto de robots en circulación, junto con su pasado, son equivalentes observacionalmente.

Cabe destacar que no modelamos el paso del tiempo, por lo que las inspecciones no son automáticas, sino que deben ser causadas deliberadamente por quien maneje el sistema.

2. Sobre el formato

Inicialmente escribimos la especificación utilizando UTF-8 en lugar de LaTeX, pensando que podríamos fácilmente incluir el archivo en nuestro informe, mas esto resultó ser problemático por varias cuestiones. Finalmente, optamos por incluirlo por separado como un archivo en otro pdf. Debido a problemas con esta solución, tuvimos que ajustar el texto para que no pase de 90 columnas; adoptamos una convención que es que si las precondiciones de una función superan tal tamaño, la guarda se pasa a otra linea y se indenta al mismo nivel que el tipo de la función.

```
TAD rur ES nat
TAD caracteristica ES string

TAD Robot
  generos robot
  exporta id, cars, instanciar
  igualdad observacional
    (∀r₁, r₂ : robot) (r₁ =obs r₂ ↔ id(r₁) =obs id(r₂))
  observadores basicos
    id: robot → rur
    cars: robot → conj(caracteristica)
  generadores
    instanciar: rur i × conj(caracteristica) cs → robot {i ≥ 1 ∧ cs ≠obs ϕ}
  axiomas
    id(instanciar(r, cs)) ≡ r
    cars(instanciar(r, cs)) ≡ cs
Fin TAD

TAD Restriccion
  generos restriccion
  exporta AND, OR, NOT, VAR, cumple
  igualdad observacional
    (∀ r₁, r₂ : restriccion) (r₁ =obs r₂ ↔
      (∀ c : conj(caracteristica)) (cumple(c, r₁) =obs cumple(c, r₂)))
  observadores basicos
    cumple: conj(caracteristica) × restriccion → bool
  generadores
    AND: restriccion × restriccion → restriccion
    OR: restriccion × restriccion → restriccion
    NOT: restriccion → restriccion
    VAR: caracteristica → restriccion
  otras operaciones
    TRUE: → restriccion
    FALSE: → restriccion
  axiomas
    cumple(cs, VAR(c)) ≡ c ∈ cs
    cumple(cs, NOT(c)) ≡ ¬(cumple(cs, c))
    cumple(cs, AND(c1, c2)) ≡ cumple(cs, c1) ∧ cumple(cs, c2)
    cumple(cs, OR(c1, c2)) ≡ cumple(cs, c1) ∨ cumple(cs, c2)

    TRUE ≡ OR(VAR("dummy"), NOT(VAR("dummy")))

    FALSE ≡ AND(VAR("dummy"), NOT(VAR("dummy")))
Fin TAD

TAD estacion ES string
TAD conexion ES ⟨estacion, restriccion⟩
TAD conexiones ES conj(conexion)

TAD Mapa
  generos mapa
  exporta estaciones, conexiones, nuevo, crearEst, conectar,
          esBloqueante, conectadas, caminos
  igualdad observacional
    (∀ m₁, m₂ : mapa) (m₁ =obs m₂ ↔
      estaciones(m₁) =obs estaciones(m₂) ∧ₗ
      (∀ e ∈ estaciones(m₁)) (conexiones(m₁, e) =obs conexiones(m₂, e)))
  observadores basicos
    estaciones : mapa → conj(estacion)
    conexiones : mapa m × estacion e → conj(conexion) {e ∈ estaciones(m)}
  generadores
    nuevo    : → mapa
    crearEst : mapa m × estacion a → mapa {a ∉ estaciones(m)}
    conectar : mapa m × estacion a × estacion b × restriccion → mapa {a, b ∈ estaciones(m)}
  otras operaciones
    esBloqueante  : mapa m × conj(caracteristica) × estacion e → bool {e ∈ estaciones(m)}
    esBloqueante' : conj(conexion) × conj(caracteristica) → bool

    conectadas    : mapa m × estacion a × estacion b → bool {a, b ∈ estaciones(m)}
    conectadas'   : estacion × conj(conexion) → bool

    caminos       : mapa m × estacion a × estacion b → conj(restriccion)
                   {a, b ∈ estaciones(m) ∧ conectadas(m, a, b)}
```

```
    caminos'      : estacion × conj(conexion) → conj(restriccion)
  axiomas
    conexiones(crearEst(m, e), k) ≡ ϕ
    conexiones(conectar(m, a, b, r), e) ≡
      (if e ≡ a then
        ⟨b, r⟩
      else
        if e ≡ b then
          ⟨a, r⟩
        else
          ϕ
        fi
      fi) ∪ conexiones(m, e)

    estaciones(nuevo) ≡ ϕ
    estaciones(crearEst(m, e)) ≡ { e } ∪ estaciones(m)
    estaciones(conectar(m, a, b)) ≡ estaciones(m)

    esBloqueante(m, r, e) ≡.
      if ϕ?(conexiones(m, e)) then
        False
      else
        esBloqueante'(conexiones(m, e), r)
      fi

    esBloqueante'(c, r) ≡
      if ϕ?(c) then
        True
      else
        ¬(cumple(r, π₂(dameUno(c)))) ∧ₗ esBloqueante'(sinUno(c), r)
      fi

    conectadas(m, a, b) ≡ conectadas'(b, conexiones(m, a))

    conectadas'(a, c) ≡
      if ϕ?(c) then
        False
      else
        π₁(dameUno(c)) ≡ a) vₗ conectadas'(a, sinUno(c))
      fi

    caminos(m, a, b) ≡ caminos'(b, conexiones(m, a))

    caminos'(a, c) ≡
      if ϕ?(c) then
        ϕ
      else
        (if π₁(dameUno(c)) ≡ a then
          π₂(dameUno(c))
        else
          ϕ
        fi) ∪ caminos'(a, sinUno(c))
      fi
Fin TAD

TAD Ciudad
  generos ciudad
  exporta nueva, agregar, mover, borrar,
          mapeo, robots, historial, buscar, ultimoId, inspeccion
  igualdad observacional
    (∀ c₁, c₂ : ciudad) (c₁ =obs c₂ ↔
      (mapeo(c₁) =obs mapeo(c₂) ∧ robots(c₁) =obs robots(c₂)) ∧ₗ
      (∀ r ∈ robots(c₁)) (historial(c₁, r) =obs historial(c₂, r)) ∧ₗ
      ((robots(c₁) ≠obs ϕ) ⟹  ultimoId(c₁) =obs ultimoId(c₂)))
  observadores basicos
    mapeo     : ciudad → mapa
    robots    : ciudad → conj(robot)
    historial : ciudad c × robot r → pila(conexion) {r ∈ robots(c)}
  generadores
    nueva   : mapa → ciudad
    agregar : ciudad c × conj(caracteristica) cs × estacion e → ciudad
              {cs ≠obs ϕ ∧ₗ e ∈ estaciones(mapeo(c)) ∧ₗ ¬esBloqueante(mapeo(c), cs, e)}
    mover   : ciudad c × robot r × conexion x → ciudad {esConexionValida(c, r, x)}
    borrar  : ciudad c × robot r → ciudad {r ∈ robots(c)}
```

```
otras operaciones
  esConexionValida: ciudad × robot × conexion → bool

  buscar         : ciudad c × rur i → robot {(∃r ∈ robots(c)) (id(r) =obs i)}
  buscar'        : conj(robot) c × rur i → robot {(∃r ∈ c) (id(r) =obs i)}

  ultimoId       : ciudad c → rur {robots(c) ≠obs ϕ}
  ultimoId'      : ciudad c × rur i × conj(robot) k → rur
                   {k ⊆ robots(c) ∧ (∃r ∈ robots(c)) (id(r) =obs i)}

  inspeccion     : ciudad c × estacion e → ciudad {e ∈ estaciones(mapeo(c))}
  inspeccion'    : ciudad c × estacion e × ⟨robot, nat⟩ r × conj(⟨robot, nat⟩) k → ciudad
                   {e∈estaciones(mapeo(c)) ∧ι k⊆obtenerRobots(c, e) ∧ι r∈obtenerRobots(c, e)}

  infracciones   : ciudad c × robot r → nat {r ∈ robots(c)}
  infracciones'  : conj(caracteristica) × pila(conexion) → nat

  obtenerRobots  : ciudad c × estacion e → conj(⟨robot, nat⟩) {e ∈ estaciones(mapeo(c))}
  obtenerRobots' : ciudad c × conj(robot) k × estacion e → conj(⟨robot, nat⟩)
                   {e ∈ estaciones(mapeo(c)) ∧ k ⊆ robots(c)}
axiomas
  mapeo(nueva(m)) ≡ m
  mapeo(agregar(c, r, e)) ≡ mapeo(c)
  mapeo(mover(c, r, e)) ≡ mapeo(c)
  mapeo(borrar(c, r)) ≡ mapeo(c)

  robots(nueva(m)) ≡ ϕ
  robots(agregar(c, cs, e)) ≡
    { instanciar(if ϕ?(robots(c)) then 1 else ultimoId(c) + 1 fi, cs) } ∪ robots(c)
  robots(mover(c, r, e)) ≡ robots(c)
  robots(borrar(c, r)) ≡ robots(c) - { r }

  historial(agregar(c, cs, e), r) ≡
    if id(r) ≡ ultimoId(c) + 1 then
      apilar(⟨e, cs⟩, vacia)
    else
      historial(c, r)
    fi

  historial(mover(c, r', x), r) ≡
    if r ≡ r' then
      apilar(x, historial(c, r))
    else
      historial(c, r)
    fi

  esConexionValida(c, r, x) ≡
    r ∈ robots(c) ∧ι
    π1(x) ∈ estaciones(mapeo(c)) ∧ι
    conectadas(mapeo(c), tope(historial(c, r)), π1(x)) ∧ι
    π2(x) ∈ caminos(mapeo(c), tope(historial(c, r)), π1(x))

  buscar(c, i) ≡ buscar'(robots(c), i)

  buscar'(cs, i) ≡
    if id(dameUno(cs)) ≡ i then
      dameUno(cs)
    else
      buscar'(sinUno(cs), i)
    fi

  ultimoId(c) ≡ ultimoId'(id(dameUno(robots(c))), sinUno(robots(c)))

  ultimoId'(m, c) ≡
    if ϕ?(c) then
      m
    else
      ultimoId'(if id(dameUno(c)) > m then id(dameUno(c)) else m fi, sinUno(c))
    fi

  inspeccion(c, e) ≡
    if ϕ?(obtenerRobots(c, e)) then
      c
    else
```

```
222         inspeccion'(c, e, dameUno(obtenerRobots(c, e)), sinUno(obtenerRobots(c, e)))
223       fi
224
225     inspeccion'(c, e, r, cs) ≡
226       if ϕ?(cs) then
227         if π₂(r) > 0 then
228           borrar(c, r)
229         else
230           c
231         fi
232       else
233         inspeccion'(c, e, if π₂(dameUno(cs)) > π₂(r) then dameUno(cs) else r fi, sinUno(cs))
234       fi
235
236     obtenerRobots(c, e) ≡ obtenerRobots'(c, robots(c), e)
237
238     obtenerRobots'(c, cs, e) ≡
239       if ϕ?(cs) then
240         ϕ
241       else
242         (if tope(historial(c, dameUno(cs))) ≡ e then
243           ⟨dameUno(cs), infracciones(c, dameUno(cs))⟩
244         else
245           ϕ
246         fi) ∪ obtenerRobots'(c, sinUno(cs), e)
247       fi
248
249     infracciones(c, r) ≡ infracciones'(cars(r), historial(c, r))
250
251     infracciones'(c, p) ≡
252       if tamaño(p) ≤ 1 then
253         0
254       else
255         β(¬cumple(c, π₂(tope(p)))) + infracciones'(c, desapilar(p))
256       fi
257 Fin TAD
```