Informe

Guillermo Pacheco

2 de septiembre de 2014

1. Diseño del lenguaje

1.1. Diseño general

El lenguaje presentado intenta simplificar al programador la creación y el orden de ejecución de diversas tareas concurrentemente. Las construcciones principales del lenguaje son llamadas *Tareas*. Las tareas se declaran, junto con restricciones entre las mismas, y al ejecutarse, cada una lo hace en un hilo independiente de ejecución. Cada tarea, al declararse comienza bloqueada y es ejecutada con la instrucción *Start*, por ejemplo $Start(t1)$ asumiendo que $t1$ fue declarada previamente, comienza la tarea $t1$.

Las tareas tienen tres estados posibles, *Bloqueado*, *Ejecutando* y *Finalizado*, dichos estados pueden ser usados para ordenar las tareas, los posibles predicados se detallan a continuación.

1.2. Predicados

Existen tres estados posibles de las tareas, Ejecutando, Bloqueado, Finalizado.

Se pueden expresar predicados para especificar el orden entre las mismas. Existen tres predicados básicos que indican el estado de la tarea: Ejecutando, Bloqueado, Finalizado. Por ejemplo, *Ejecutando(t)* significa que la tarea t está en estado ejecutando.

Ejemplos:

- "Bloquear la tarea t2 cuando la tarea t1 este lista."

  $\models Listo(t1) \rightarrow Bloqueado(t2)$

- "Ejecutar la tarea t2 cuando la tarea t1 finalice."

  $\models Finalizado(t1) \rightarrow Ejecutando(t2)$

- "Bloquear la tarea t3 cuando la tarea t1 finalice y la tarea t2 esté bloqueada"

  $\models Finalizado(t1) \wedge Bloqueado(t2) \rightarrow Bloqueada(t3)$

Hay axiomas sobre las tareas que se cumplen: ———————————————————
La tarea no está finalizada hasta finalicen sus hijas:
$\forall t(\exists th \in (Hijas(t)) \neg Finalizado(th) \rightarrow \neg Finalizado(t))$
Casi equivalente es decir, si finaliza la tarea padre, finalizan sus hijas:

$\forall t :: (Finalizado(t) \rightarrow \forall th : th \in Hijas(t) : Finalizado(th))$
Los estados de las tareas son exclusivos:

- $\forall t(Finalizado(t) \rightarrow \neg(Bloqueado(t) \vee Ejecutando(t)))$
- $\forall t(Bloqueado(t) \rightarrow \neg(Ejecutando(t) \vee Finalizado(t)))$
- $\forall t(Ejecutando(t) \rightarrow \neg(Bloqueado(t) \vee Finalizado(t)))$

1.3. Sintaxis, Gramática

```
ident = [_a-z][_a-zA-Z]*

params =
       | ident params

param-decl = "(" params ")"

task-decl = "Task" ident "{" task-block "}"

task-block:
           | task-elem
           | "|=" task-constraint
           | "Start" task-start
           | func-call
           | task-block

task-constraint: guard-constraint "->" action-constraint

guard-constraint:
                  | not guard-constraint
                  | bool-guard
                  | basic-constraint
                  | basic-constraint "and" guard-constraint
                  | basic-constraint "or" guard-constraint

action-constraint:
                  | basic-constraint
                  | basic-constraint and action-constraint

basic-constraint: "Blocked" ident
                | "Running" ident
                | "Finished" ident
```

```
func-decl: "Function" ident param-decl "{" func-block "}"
```

1.4. Ejemplo

```
Function hayCasa() {
  return (read(sensor_distancia) < DISTANCIA_CASA)
}

Tarea contarCasas() {
  observable cuenta = 0
  viendo = False

  While (True) {
    if not hayCasa() {
       viendo = False
    } else if not viendo {
       viendo = True
       cuenta = cuenta = 1
    }
  }
}

Task encontrarSegundaCasa {
  t: contarCasas()
  |= t.cuenta == 2 -> Finished(t)
  start(t)
}

Task avanzarRueda(numRueda, direccion) {
  while (True) {
    velocidad = VELOCIDAD_RUEDA
    if (read(sensor_color[numRueda]) < THRESHOLD) {
      velocidad = 0
    }
    setVelocidadMotor(numRueda, velocidad * direccion)
    sleep(0.5)
  }
}

Task seguirLinea {
  t1: avanzarRueda(0, -1)
```

```
  t2: avanzarRueda(1, 1)
}

Function setVelocidadMotor(numMotor, velocidad) {
  write(motor[numMotor], velocidad)
}

Task detener {
  setVelocidadMotor(0, 0)
  setVelocidadMotor(1, 0)
}

Task main {
  t1: encontrarSegundaCasa
  t2: seguirLinea
  |= Finished(t1) -> Finished(t2)
  t3: detener
  |= Finished(t1) ^ Finished(t2) -> Running(t3)
  Start(t1, t2)
}
```