Las especificaciones formales existen
No son los padres

Nicolás Kicillof

DC – FCEyN

UBA

Especificaciones funcionales

- Cuál tiene que ser el resultado de cada operación del programa
- En un lenguaje formal
  - Interpretable por una computadora
  - Puede servir de base a herrramientas de software
- QUÉ vs. CÓMO

Ejemplos de especificación

- Sort(Seq⟨int⟩ xs)
  requires NonRep(xs);
  modifies xs;
  ensures IsPermutation(xs, old(xs)) and Ordered(xs);
- bool NonRep(Seq⟨int⟩ xs)
  ensures result ⇔ (Size(xs) = Size({i | i in xs}));
- bool Ordered(Seq⟨int⟩ xs)
  ensures result ⇔ (forall i,j in Indices(xs): i < j ⇒ xs[i] < xs[j]);
- bool IsPermutation(Seq⟨int⟩ xs, ys)
  requires NonRep(xs) and NonRep(ys);
  ensures {i | i in xs} = {i | i in ys};

Otras versiones

- bool NonRep(Seq⟨int⟩ xs)

  ensures result ⇔ forall x in xs: x notin xs - x;
- bool IsPermutation(Seq⟨int⟩ xs, ys)

  ensures ys in Permutations(xs);
- Set⟨Seq⟨int⟩⟩ Permutations(Seq⟨int⟩ xs)

  ensures xs = [] ⇒ result = {[]};

  ensures xs ≠ [] ⇒ result = {x.p | x in xs, p in Permutations(xs - x)};

Más versiones

- Sort(Seq⟨int⟩ xs)

  modifies xs;

  repeat until fixedpoint SwapPair(xs);
- SwapPair(Seq⟨int⟩ xs)

  modifies xs;

  choose i,j ← Indices(xs) | i < j, old(xs)[i] > old(xs)[j];

  ensures xs = Swap(i,j,xs);

Conclusiones

- La línea entre QUÉ y CÓMO es delgada
- El código fuente (lenguaje de programación) es también una especificación
- Entonces, ¿para qué sirven las especificaciones?

(Mi) Respuesta

- Redundancia
- La especificación y el código son descripciones formales del resultado
  - Tal vez en diferentes niveles de abstracción
  - Tal vez con distinto detalle
  - Tal vez con otro nivel de declaratividad
  - Las dos pueden tener errores
- ¿Qué gané?

(Mi) Respuesta

- Tengo dos descripciones de lo mismo
- Puedo fijarme si coinciden o no
- Las dos son formales
  - Puede compararlas una computadora
  - En general, es un problema indecidible
  - Verificación
  - Testing
- Si no tengo especificación, no hay doble control

Especificaciones formales en la industria de desarrollo de software

- Durante años las miraron con desconfianza
  - Complicadas
    - Se necesita personal muy capacitado
    - Tan propensas a errores como el código
    - Poca ganancia directa
    - Falta de herramientas
  - De juguete
    - No escalan bien
    - Condiciones poco realistas
      - Aliasing
      - Heap
      - Entrada / salida
      - Efectos colaterales
      - Concurrencia
      - Etc.

¿Qué cambió?

- Design By Contract
  - Eiffel, Bertrand Meyer, 1992
- Especificaciones (contratos) como expresiones del lenguaje de programación
  - No hace falta gente (tan) capacitada
  - Las herramientas ya existen (compiladores)
  - Validación en ejecución
- Especificaciones parciales
  - No describo completamente el resultado
  - Solamente garantizo que no se van a producir ciertas fallas

Oportunidad + Peligro

- Buenísimo, la industria nos da bola
- Pero
  - Tenemos que mostrar herramientas
  - Tenemos que acercarnos a la realidad
    - Ese “último paso” puede ser un salto mayor que todos los anteriores juntos
- ¿Es cuestión de ellos?
  - Nosotros tuvimos las ideas
  - Nosotros entendemos bien la semántica
  - Nosotros tenemos la experiencia
  - El riesgo es que se cansen y desechen los métodos

Una herramienta

- Spec#
  - Grupo Foundations of Software Engineering, Microsoft Research Redmond
- Lenguaje de programación
  - Extensión conservativa de C#
- Compilador
  - Emite validaciones de las especificaciones en ejecución
- Verificador estático
  - (Boogie) Pruebas modulares de que las especificaciones valen
- Entorno
  - Contratos para las clases básicas de .NET
  - Extensión del entorno de desarrollo (VisualStudio)
- Contempla todas las características de .NET

Objetivos de Spec#

- Facilitar la expresión de decisiones de diseño detallado

⇒ Proveer herramientas para hacer cumplir estas decisiones

⇒ Ayudar a impedir y detectar bugs

⇒ Disminuir el costo del desarrollo de software

Demo

Squiggles

¿Alcanza?

- Estamos describiendo resultados
  - Cómo cambia el estado al invocar un método
- La historia también importa
  - Podemos codificarla en el estado, pero se pierde claridad
- FSE habla de “decisiones de diseño detallado”
- ¿Y si no tenemos el diseño detallado?
  - No sabemos qué operaciones vamos a tener
  - No sabemos cómo se van a llamar las variables
  - No conocemos los resultados exactos
  - No sabemos definir un “estado”
- En ingeniería de software nos preocupa también el diseño
  - Cómo se divide el sistema en partes
  - Al relevar requerimientos es común que no tengamos esa información

Modelo

- Descripción de posibles trazas del sistema
- Puede refinarse para obtener diseños más detallados
- Herramientas
  - Model checking (el modelo es correcto con respecto a ciertas propiedades deseables)
  - Testing basado en modelos
    - Criterio de cubrimiento (todas las trazas del modelo)
    - Adecuación (el sistema se comporta como el modelo)

Escenarios

- Descripciones parciales de trazas
  - Excluyen algunos eventos
  - No describen todas las trazas
- Historias de cómo se comporta el sistema en determinados casos
- Para
  - Expresar requerimientos parciales
  - Describir situaciones “interesantes” para testing
    - Se inicializó la aplicación, se abrió un archivo, se escribieron dos páginas: ahora empecemos a probar

Combinación de estrategias

- Modelos y escenarios dan información temporal
  - Evoluciones posibles del sistema
  - Útiles en etapas tempranas del diseño
  - Comprobables en un sistema terminado
- Pueden enriquecerse con especificaciones basadas en el estado
  - Guardas de operaciones
  - Descripción de los resultados
  - Filtrado de operaciones inviables
  - Etc.

Una herramienta

- Cord
- Lenguaje para describir y componer comportamientos
  - Modelos
  - Escenarios
  - Aplicaciones
- Puede usarse desde distintas herramientas
- Soporta todo el framework .NET
- Permite ejecución simbólica

Composiciones

substitución

negación (falla)

traducción

disyunción

intercalación

repetición

secuencia sin orden

Demo

Bingo y Cord