Programación Declarativa:
Lógica y Restricciones

Introducción a la Programación Declarativa

Curso 2015-2016 (2º Semestre)

Mari Carmen Suárez de Figueroa Baonza

mcsuarez@fi.upm.es

POLITÉCNICA

Programación Convencional: 2 escenarios habituales

- Programación "**desestructurada**"
  - Punto de partida del programador:
    - una "idea", un proyecto que "tiene en la cabeza"
    - un conjunto de casos (entradas-salidas)
  - Ciclo de vida del software:
    - el código se prueba con algunos casos
    - si surgen errores, el programador parchea el código
- Programación "**estructurada**"
  - Punto de partida del programador:
    - Análisis (conceptual) del problema
    - Conjunto de especificaciones (funcionales, operativas) del software
  - Ciclo de vida del software:
    - Validación del código (casos de prueba, validación unitaria, de usuario, etc.)
    - Informes de error, modificación del código, actualización de la documentación, producción de una nueva versión del software

Programación Convencional: el problema de la corrección (I)

- Razón de existir de la ingeniería del software:
  - Abordar un proyecto de forma “desestructurada” es inviable:
    - Certificación de calidad del producto ⇨ seguimiento de una metodología
    - Mantenimiento del código en ausencia de su creador ⇨ método y documentación
    - Cumplimiento de obligaciones contractuales ⇨ cumplimiento de especificaciones
    - Responsabilidad en caso de daños ⇨ buen funcionamiento, ausencia de errores
  - Es necesario abordar los proyectos siguiendo una metodología de desarrollo:
    - Asegura el logro de estos objetivos *en alguna medida*
    - No lo asegura por completo *por falta de rigor*

Programación Convencional: el problema de la corrección (II)

- Razones para la falta de rigor/precisión:
  - La especificación se realiza en gran medida en un lenguaje natural
  - La codificación se realiza en un lenguaje de programación
  - Validar la adecuación del código a la especificación es difícil porque:
    - Es difícil formalizar el significado de especificaciones en lenguaje natural
    - Es difícil formalizar el significado del código
    - Es (aún más) difícil establecer la corrección del 2º respecto del 1º

Programación Convencional: el problema de la corrección (III)

- **Ejemplo**: un programa imperativo simple

```
#include <stdio.h>
main() {
  int Number, Square;
  Number = 0;
       while(Number <= 5)
         { Square = Number * Number;
           printf("%d\n",Square);
           Number = Number + 1; } }
```

- ¿Es correcto?
- ¿Con respecto a que?
- Se necesita un *paradigma apropiado* que:
  - Permita proporcionar especificaciones (es decir, describir problemas), y
  - Permita razonar sobre la corrección de los programas (su implementación)

Programación Convencional: el problema de la corrección (IV)

- **Ejemplo**: un programa imperativo simple

```
#include <stdio.h>
main() {
  int Number, Square;
  Number = 0;
       while(Number <= 5)
         { Square = Number * Number;
           printf("%d\n",Square);
           Number = Number + 1; } }
```

- Para calcular los cuadrados de los números naturales que son menores o iguales que 5
  - Parece ideal a primera vista, pero
  - El lenguaje natural es
    - Verboso
    - Vago
    - Ambiguo
    - Necesita contexto
- Filósofos y matemáticos ya indicaron hace tiempo los problemas del lenguaje natural

Corrección del Software

- Corrección **informal**:
  - Única opción a falta de corrección formalmente demostrada
  - Sustituto parcial: validación basada en casos de prueba
- Pasos hacia la corrección **formal** del código:
  1. *Especificación formal*: lógica, matemáticas
  2. *Codificación formal*: lenguaje de programación con un significado formal asociado al código
  3. *Demostración formal* de que el significado formal del código coincide con el significado formal de las especificaciones
- Mientras que (1) es factible, (2) y aún más (3) plantean mayores dificultades:
  - Aún así, disponer de una especificación formal es un avance nada despreciable

Especificación Formal de Programas (I)

- Es factible emplear Lógica y Matemáticas para especificar formalmente un problema y su solución:
  - Permite **representar** la información de forma precisa (teoría)
  - Permite **razonar** rigurosamente sobre las consecuencias (teoremas) de una teoría como soluciones al problema
- Impacto sobre la validación:
  - Una especificación formal expresa *rigurosamente* qué resultados deben producirse
    - Fin a los malentendidos y las ambigüedades
  - Indica además *todos* los resultados correctos que deben generarse
  - Por ambos motivos, y *aunque* no haya un significado preciso del código a validar, la especificación formal es de gran ayuda

Especificación Formal de Programas (II)

- Definición precisa del significado de un programa:
  - La mayoría de lenguajes de programación son imperativos
  - Hay una profunda diferencia entre el significado *declarativo* de la lógica y las matemáticas y el significado *imperativo* del código

Paradigma Imperativo vs. Declarativo

Paradigma Imperativo	Paradigma Declarativo
Se describe **cómo** resolver el problema	Se describe **qué** es el problema
Programa = **secuencia** de instrucciones	Programa = **conjunto** de declaraciones (o sentencias)
Instrucción = **modificación** del estado de la máquina en la que se ejecuta	Declaración = **afirmación** que se considera verdadera
El significado **efectivo** de un programa son sus efectos sobre la máquina en la que se ejecuta: registros, memoria, etc.	El significado **efectivo** de un programa es el conjunto de afirmaciones que se deducen de él
El significado **pretendido** es el conjunto (posiblemente infinito) de entradas-salidas correctas	El significado **pretendido** es el conjunto de afirmaciones que son consecuencia lógica de la teoría
La tarea del programador es codificar una cadena de instrucciones (el cómo) que resuelva el problema	La tarea del programador es codificar una teoría (el qué) que recoja las condiciones del problema

La Lógica (Recordatorio)

- Es un medio para **clarificar/formalizar** el proceso del pensamiento humano
- Por ejemplo, (la lógica clásica) nos puede contar que:
  - A Aristóteles le gustan las espinacas, y
  - Platón es amigo de cualquiera al que le gusten las espinacas
  - Eso implica que: Platón es amigo de Aristóteles
- Este mismo razonamiento en lógica simbólica:
  - a1: gustar (aristoteles, espinacas)
  - a2: $\forall$X gustar (X, espinacas) $\rightarrow$ amigo (platon, X)
  - t1: amigo (platon, aristoteles)
  - T [a1, a2] $\vdash$ t1

Ejemplo de Especificación (I): Generación de Cuadrados

- Usamos la representación de Peano para los números (por simplicidad):
  - 0 → 0 1 → s(0) 2 → s(s(0)) 3 → s(s(s(0))) etc.
- **Definición de los números naturales**:
  - nat(0) ∧ nat(s(0)) ∧ nat(s(s(0))) ∧ …
  - Una solución mejor:
    - nat(0) ∧ ∀X (nat(X) → nat(s(X)))
- **Orden de los números naturales**:
  - ∀X (menor(0,X) ∧
  - ∀X∀Y (menor(X,Y) → menor(s(X), s(Y)))
- **Suma de los números naturales**:
  - ∀X (nat(X) → suma(0, X, X)) ∧
  - ∀X∀Y∀Z (suma(X,Y,Z) → suma(s(X),Y,s(Z)))

Ejemplo de Especificación (II): Generación de Cuadrados

- **Producto de los números naturales**:
  - $\forall X$ (nat(X) $\rightarrow$ producto(0, X, 0)) $\wedge$
  - $\forall X \forall Y \forall Z \forall W$ (producto(X,Y,W) $\wedge$ suma(W,Y,Z) $\rightarrow$ producto(s(X),Y,Z))
- **Cuadrado de los números naturales**:
  - $\forall X \forall Y$ (nat(X) $\wedge$ nat(Y) $\wedge$ producto(X,X,Y) $\rightarrow$ cuadrado(X,Y))
- Ahora podemos escribir la **especificación del programa** (imperativo), es decir, las condiciones que queremos que el programa cumpla:
  - Precondición:
    - Vacía
  - Postcondición:
    - $\forall X$ (output(X) $\leftarrow$ ($\exists Y$ nat(Y) $\wedge$ menor(Y,s(s(s(s(s(0)))))) $\wedge$ cuadrado(Y,X)))

Más allá de la Especificación Formal

- La Lógica se emplea en la especificación rigurosa de programas para:
  - Representación formal del problema
  - Razonamiento sobre los resultados pretendidos
- La Lógica también puede emplearse **como lenguaje de programación**:
  - Si tenemos un procedimiento *correcto* de deducción automática,
    - Corrección lógica: toda fórmula deducida es consecuencia lógica
  - Entonces, basta con
    - definir una teoría lógica que represente el problema y
    - dejar al demostrador automático la extracción de soluciones
- Y como resultado:
  - No es necesario validar el código, pues sólo se deducirán resultados correctos

Preguntas y Respuestas en el Ejemplo de Especificación

Query	Answer
$nat(s(0))$ **?**	(yes)
$\exists X\ add(s(0), s(s(0)), X)$ **?**	$X = s(s(s(0)))$
$\exists X\ add(s(0), X, s(s(s(0))))$ **?**	$X = s(s(0))$
$\exists X\ nat(X)$ **?**	$X = 0 \vee X = s(0) \vee X = s(s(0)) \vee \ldots$
$\exists X \exists Y\ add(X, Y, s(0))$ **?**	$(X = 0 \wedge Y = s(0)) \vee (X = s(0) \wedge Y = 0)$
$\exists X\ nat\_square(s(s(0)), X)$ **?**	$X = s(s(s(s(0))))$
$\exists X\ nat\_square(X, s(s(s(s(0)))))$ **?**	$X = s(s(0))$
$\exists X \exists Y\ nat\_square(X, Y)$ **?**	$(X = 0 \wedge Y = 0) \vee (X = s(0) \wedge Y = s(0)) \vee (X = s(s(0)) \wedge Y = s(s(s(s(0))))) \vee \ldots$
$\exists X output(X)$ **?**	$X = 0 \ \vee\ X = s(0) \ \vee\ X = s(s(s(s(0)))) \ \vee\ X = s^{9}(0) \ \vee\ X = s^{16}(0) \ \vee\ X = s^{25}(0)$

¿Qué Lógica emplear? (I)

- Hemos razonado sobre la conveniencia de representar un problema usando la lógica, pero
- **¿Qué Lógica emplear?**
  - Lógica proposicional
  - Cálculo de predicados (lógica de primer orden)
  - Lógicas de orden superior
  - Lógicas modales
  - Cálculo λ, etc.
- **¿Qué proceso de razonamiento?**
  - Deducción natural, métodos clásicos
  - Resolución
  - Prawitz/Bibel, Tableaux
  - Re-escritura, etc.

¿Qué Lógica emplear? (II)

- Se intenta maximizar el poder expresivo
- Pero, uno de los principales requisitos es disponer de un proceso de razonamiento **efectivo**
- Es importante entender las propiedades subyacentes y los límites teóricos de cada tipo de lógica

¿Qué Lógica emplear? (III)

- **Ejemplo**: Proposiciones vs. Fórmulas de Primer Orden
  - Lógica Proposicional
    - "Pluto es un perro" p
    - "Los perros tienen cola" q

    pero, ¿cómo podemos concluir que Pluto tiene cola?
  - La Lógica de Predicados extiende el poder expresivo de la Lógica Proposicional:
    - perro(pluto)
    - $\forall X$ perro(X) $\rightarrow$ tieneCola(X)

    ahora, usando deducción podemos concluir
    - tieneCola(pluto)

¿Qué Lógica emplear? Hacia la Programación Lógica (I)

- La lógica proposicional carece de la potencia expresiva/deductiva suficiente para abordar cualquier problema computable
- La lógica de 1^{er} orden posee suficiente potencia expresiva, pero en general, no es posible automatizar la respuesta a:
  - ¿T $\models$ A? ¿es A consecuencia lógica de la teoría T?
  - ¿T $\vdash$ A? ¿es A deducible (teorema) de la teoría T?

¿Qué Lógica emplear? Hacia la Programación Lógica (II)

- Entre ambos extremos, la que se puede denominar lógica de cláusulas definidas:
  - Permite expresar cualquier problema computable
  - Existe un procedimiento correcto y mecanizable de deducción
  - Con él, puede decidirse mecánicamente si $T \models A$ (o $T \vdash A$) siempre que $T \cup \{ \neg A \}$ sea insatisfacible
  - Sin embargo, no puede decidirse en general si $T \models A$ o $T \not\models A$
    - Si se pudiera, no existiría el problema de la parada: todo problema computable sería resoluble en tiempo finito mediante un algoritmo

Cláusulas (Recordatorio)

- Todas las fórmulas pueden transformarse en un conjunto de cláusulas
- Una **cláusulas** tiene la forma

  "conclusiones ← condiciones"

  - Conclusiones: literales unidos por disyunción ('*or*')
  - Condiciones: literales unidos por conjunción ('*and*')
  - Todas las variables están (implícitamente) cuantificadas universalmente (∀)
  - Notación compacta: las conectivas se pueden sustituir por comas
- Una **cláusula de Horn** tiene la forma

  "conclusion ← cond1, …, condn"

¿Qué Lógica emplear? Hacia la Programación Lógica (III)

- Consideremos la siguiente teoría T:
  - *Todo hombre cuyo padre sea rico, es él también rico.*
    - $\forall x(R(f(x)) \rightarrow R(x))$ `rico(X) ← rico(padre(X)).`
  - *El abuelo de Juan era rico.*
    - $R(f(f(a)))$ `rico(padre(padre(juan))).`
- ¿ T |– R(f(a)) ? `?- rico(padre(juan)).`
- ¿ T |– R(a) ? `?- rico(juan).`
- ¿ T |– $\exists x R(x)$? `?- rico(X).`
- ¿ T |– R(b) ? `?- rico(pedro).`

Generación de Cuadrados en un Sistema de Programación Lógica (I)

```
:- module(_,_,['bf/af']).

nat(0) <- .
nat(s(X)) <- nat(X).

le(0,_X) <- .
le(s(X),s(Y)) <- le(X,Y).

add(0,Y,Y) <- nat(Y).
add(s(X),Y,s(Z)) <- add(X,Y,Z).

mult(0,Y,0) <- nat(Y).
mult(s(X),Y,Z) <- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) <- nat(X), nat(Y), mult(X,X,Y).

output(X) <- nat(Y), le(Y,s(s(s(s(s(0))))))), nat_square(Y,X).
```

Generación de Cuadrados en un Sistema de Programación Lógica (II)

Query	Answer
?- nat(s(0)).	yes
?- add(s(0),s(s(0)),X).	X = s(s(s(0)))
?- add(s(0),X,s(s(s(0)))).	X = s(s(0))
?- nat(X).	X = 0 ; X = s(0) ; X = s(s(0)) ; ...
?- add(X,Y,s(0)).	(X = 0 , Y=s(0)) ; (X = s(0) , Y = 0)
?- nat_square(s(s(0)), X).	X = s(s(s(s(0))))
?- nat_square(X,s(s(s(s(0))))).	X = s(s(0))
?- nat_square(X,Y).	(X = 0 , Y=0) ; (X = s(0) , Y=s(0)) ; (X = s(s(0)) , Y=s(s(s(s(0))))) ; ...
?- output(X).	X = 0 ; X = s(0) ; X = s(s(s(s(0)))) ; ...

Ejemplo I: Ordenar 3 números

- **Problema**: ordenar 3 números distintos
- **Entrada**:
  - Conjunto X = {X1, X2, X3}
- **Solución**:
  - ∃A, ∃B, ∃C tales que:
    - A, B, y C ∈ X ∧
    - A <> B <> C ∧
    - A < B ∧
    - B < C
  - No es una solución imperativa:
    - No es un algoritmo de ordenación
  - Es una solución declarativa:
    - Especifica las condiciones que caracterizan a la solución

Ejemplo II: Descomponer un número

- **Problema**: descomponer un número en la suma de dos pares
- **Entrada**:
  - Un natural N
- **Solución**:
  - $\exists A, \exists B$ tales que:
    - $A, B \in$ Naturales $\wedge$
    - A mod 2 = 0 $\wedge$
    - B mod 2 = 0 $\wedge$
    - N = A + B

Ejemplo III: El cruce de rebaños

- **Problema**: Dos pastores se encuentran en un cruce y se produce la siguiente conversación:
  - Pastor 1: “Dame una de tus ovejas para que tengamos las mismas”
  - Pastor 2: “No, mejor dame tú una a mí para que yo tenga el doble que tú”

  ¿Cuántas ovejas tiene cada pastor?

- **Solución**:
  - $\exists A, \exists B$ tales que:
    - $A, B \in$ Naturales $\wedge$
    - $B - 1 = A + 1 \wedge$
    - $2 * (A - 1) = B + 1$

Ejemplo III: El cruce de rebaños

- Al igual que puede haber varios algoritmos para un mismo problema, es posible que haya varias especificaciones de la solución
- **Solución (alternativa)**:
  - ∃A, ∃B tales que:
    - A, B ∈ Naturales ∧
    - B = A + 2 ∧
    - 2 * (A - 1) = B + 1
- **Programa:**
  - rebaño2(A, B) :-

    B is A+2,

    2*(A-1) =:= B+1.

Resumen

- Vamos a usar la Lógica como sistema de especificación y como lenguaje de programación
- Hay que pensar de forma declarativa
- La idea esencial de la Programación Lógica es
  - Programa= lógica + control
    - 2 componentes independientes
    - **Lógica** (programador): hechos y reglas para representar conocimiento
    - **Control** (interprete): deducción lógica para dar respuestas (soluciones)
      - Se puede proporcionar de manera efectiva a través del ordenamiento de los literales en las cláusulas
      - Normalmente no hay que preocuparse del control gracias a la resolución

Programación Declarativa:
Lógica y Restricciones

Introducción a la Programación Declarativa

Curso 2015-2016 (2º Semestre)

Mari Carmen Suárez de Figueroa Baonza

mcsuarez@fi.upm.es

POLITÉCNICA