Otras aproximaciones

Ma. Laura Cobo

Métodos Formales para Ingeniería
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Argentina

Diferentes ramas de la lógica

Para realizar aproximaciones a métodos formales, se utiliza alguna de las siguientes ramas:

1. Model Theory o Teoría de modelos: Las propiedades se expresan a través de axiomas, el modelo se deriva de conjuntos. Se relaciona la consecuencia lógica con la consecuencia semántica (conjunto de las posibles secuencias de ejecución). La semántica de las sentencias lógicas está dada por valores de verdad.

   Su simplicidad la convierte en una buena herramienta de especificación. Por ejemplo: Alloy

2. Proof Theory o Teoría de prueba: La semántica de las sentencias lógicas está asociada al conjunto de pruebas que las concluyen (en lugar de su simple valor de verdad)

Diferentes ramas de la lógica

Para realizar aproximaciones a métodos formales, se utiliza alguna de las siguientes ramas:

3. Teoría de conjuntos axiomática y teoría de tipos: considera estructuras matemáticas (conjuntos con operaciones particulares). Esta aproximación ha tenido problemas a la hora de razonar sobre los modelos en forma segura.

Su simplicidad la convierte en una buena herramienta de especificación. Por ejemplo: Z, VDM (Vienna Ddevelopment Method), B.

4. Teoría de computabilidad: se basa en el estudio de las funciones computables. Nuevamente esta aproximación ha encontrado dificultades serias a la hora de automatizar las pruebas.

Key

Los métodos formales son robustos y suficientemente poderosos para las aplicaciones pero …. necesitan volverse más accesibles.

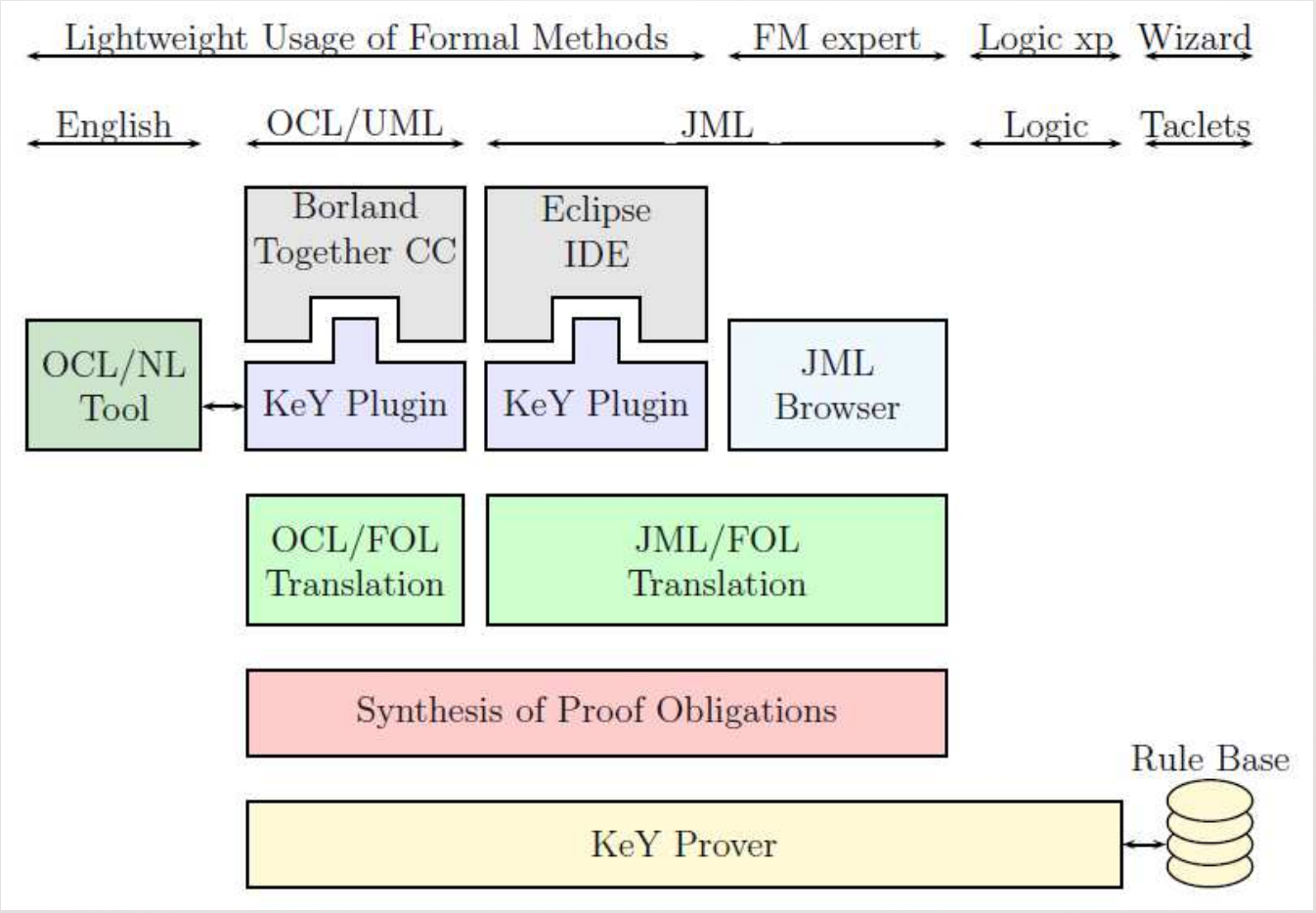
La principal idea detrás del proyecto KeY está en utilizar la lógica y la teoría de prueba.

No es posible especificar y verificar sistemas complejos de manera automática o por personas sin habilidades adquiridas en métodos formales

Lograr esta automatización es tan improbable como la programación automatizada de sistemas complejos

Página de la herramienta: www.key-project.org

Arquitectura e interface de Key



Ejemplo simple

La función de una tarjeta de pago es permitir al dueño pagar cuentas o retirar dinero de las terminales autorizadas por el proveedor de la tarjeta.
Una tarjeta contiene información acerca del balance. El balance no puede ser negativo y no puede exceder el límite dado. El límite no puede modificarse aunque cada tarjeta puede tener un límite diferente.
Cada proveedor posee una operación que actualiza el balance de acuerdo al monto involucrado en la transacción.

A pesar de la simpleza hay cuestiones sin especificar, como por ejemplo qué sucede si se excede el límite.

Lógica de primer orden

Ma. Laura Cobo

Métodos Formales para Ingeniería
Departmento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Argentina

Lógica de primer orden

La lógica utilizada aquí difiere de la presentada en los cursos sobre lógica convencionales.
La versión que se utilizará está diseñada para poder hablar de programas JAVA en forma más conveniente.

La lógica incluye un sistema de tipos con subtipado. Característica que no está presente en la mayoría de la presentaciones convencionales de la lógica

Lógica de primer orden: tipos

para poder razonar convenientemente sobre objetos del lenguajes de programación Java. Hay, por lo tanto, un vínculo entre el sistema de tipos de Java y el sistema de tipos de la lógica.

Es importante recordar que Java utiliza dos conceptos de tipo (que no deben ser confundidos)

- Cada objeto tiene un tipo dinámico (es fijado desde la creación del objeto hasta que es recolectado por el garbage collector).
- Cada expresión tiene un tipo estático: se obtiene a partir de los tipos declarados

Cada posible tipo dinámico también puede ocurrir como tipo estático. Los tipos estáticos están ordenados en una jerarquía de tipos. Luego cada tipo dinámico de un objeto es un subtipo de algún tipo estático de la expresión

Los objetos tienen un único tipo

Lógica de primer orden: tipos

Los objetos tienen un único tipo

En la lógica esta distinción se logra asignando tipos estáticos a las expresiones (”términos”) y tipos dinámicos a los valores (“elementos del dominio”)

Jerarquía de tipos: agrupa la información relevante sobre tipos y las relaciones de subtipado.

Una jerarquía de tipos de una cuádrupla, formada por:

- Un conjunto de tipos estáticos $\mathcal{T}$. Tal que $\mathcal{T} = \mathcal{T}_d \cup \mathcal{T}_a$
- Un conjunto de tipos dinámicos $\mathcal{T}_d$
- Un conjunto de tipos abstractos $\mathcal{T}_a$
- Una relación de subtipo $\sqsubseteq$ sobre $\mathcal{T}$

Los nombres en $\mathcal{T}_a$ son los nombres de las interfaces y clases abstractas de las cuales no hay instancias

Hay un tipo vacío y un tipo universal. La relación de subtipo establece un orden parcial reflexivo sobre $\mathcal{T}$

Lógica de primer orden: signature

El equivalente en lógica para un método Java son predicados y funciones

Básicamente los predicados son funciones boolenas

Signature

A first-order signature Σ consists of

- a set T_Σ of types
- a set F_Σ of function symbols
- a set P_Σ of predicate symbols
- a typing α_Σ

Intuitivamente, el tipado determina:

- Para cada función y predicado:
  - La aridad
  - Los tipos de los argumentos
- Para cada función el tipo resultado

Lógica de primer orden: signature

Signature

A first-order signature Σ consists of

- a set T_Σ of types
- a set F_Σ of function symbols
- a set P_Σ of predicate symbols
- a typing α_Σ

$$T_{\Sigma_1} = \{\mathtt{int}\},$$
$$F_{\Sigma_1} = \{+,-\} \cup \{\ldots,-2,-1,0,1,2,\ldots\},$$
$$P_{\Sigma_1} = \{<\}$$

$$\alpha_{\Sigma_1}(<) = (\mathtt{int},\mathtt{int})$$
$$\alpha_{\Sigma_1}(+) = \alpha_{\Sigma_1}(-) = (\mathtt{int},\mathtt{int},\mathtt{int})$$
$$\alpha_{\Sigma_1}(0) = \alpha_{\Sigma_1}(1) = \alpha_{\Sigma_1}(-1) = \ldots = (\mathtt{int})$$

Lógica de primer orden: signature

Se utilizará una notación más breve para las signaturas.

Type declaration of signature symbols

- Write $\tau\ x$; to declare variable x of type τ
- Write $p(\tau_1, \ldots, \tau_r)$; for $\alpha(p) = (\tau_1, \ldots, \tau_r)$
- Write $\tau\ f(\tau_1, \ldots, \tau_r)$; for $\alpha(f) = (\tau_1, \ldots, \tau_r, \tau)$

$r = 0$ is allowed, then write f instead of $f()$, etc.

Ejemplo:

- Variables: `integarArray a; int i;`
- Predicados: `estaVacia(Lista); alertaEncendida;`
- Funciones: `int buscarArreglo(int); object o;`

La aridad de o es cero

Lógica de primer orden: signature

$$\alpha_{\Sigma_1}(<) = (\texttt{int},\texttt{int})$$
$$\alpha_{\Sigma_1}(+) = \alpha_{\Sigma_1}(-) = (\texttt{int},\texttt{int},\texttt{int})$$
$$\alpha_{\Sigma_1}(0) = \alpha_{\Sigma_1}(1) = \alpha_{\Sigma_1}(-1) = \ldots = (\texttt{int})$$

Ahora:

- menor: `<(int,int);`
- mas: `int +(int,int)`
- numéricas: `int 0; int 1; …….`

Lógica de primer orden: términos

Se definen en forma recursiva:

Terms

A first-order term of type $\tau \in T_\Sigma$

- is either a variable of type τ, or
- has the form $f(t_1, \ldots, t_n)$, where $f \in F_\Sigma$ has result type τ, and each t_i is term of the correct type, following the typing α_Σ of f.

Asumiendo:

un conjunto de variables V $(V \cap (F_\Sigma \cup P_\Sigma) = \emptyset)$.

cada $\mathrm{v} \in \mathrm{V}$ tiene un único tipo $\alpha_\Sigma(v) \in T_\Sigma$.

Lógica de primer orden: términos

ejemplos:

- `-7`
- `null`
- `new(13,null)`
- `elem(new(13,null))`
- `next(next(o))`

Para el modelado de funciones de primer orden se permite la notación posfija de punto

- `new(13,null).elem`
- `o.next.next`

Fórmulas atómicas

Logical Atoms

Given a signature Σ.

A logical atom has either of the forms

- *true*
- *false*
- $t_1 = t_2$ (*"equality"*),
  where t_1 and t_2 have the same type.
- $p(t_1, \ldots, t_n)$ (*"predicate"*),
  where $p \in P_\Sigma$, and each t_i is term of the correct type,
  following the typing α_Σ of p.

Fórmulas generales

Formulas

- each atomic formula is a formula
- with ϕ and ψ formulas, x a variable, and τ a type, the following are also formulas:
  - $\neg\phi$ ("not ϕ")
  - $\phi \wedge \psi$ ("ϕ and ψ")
  - $\phi \vee \psi$ ("ϕ or ψ")
  - $\phi \rightarrow \psi$ ("ϕ implies ψ")
  - $\phi \leftrightarrow \psi$ ("ϕ is equivalent to ψ")
  - $\forall\, \tau\ x;\ \phi$ ("for all x of type τ holds ϕ")
  - $\exists\, \tau\ x;\ \phi$ ("there exists an x of type τ such that ϕ")

La variable x debe estar ligada, es decir no puede ser una variable libre

Las fórmulas sin variables libres están cerradas.

Semántica de las fórmulas

Por ahora el significado queda acotado a lo que indica la fórmula, lo cual está asociado a la asignación de las variables.

Para poder trabajar con el significado de una fórmula sobre la evolución de estados será necesario apelar la lógica dinámica.

Para establecer la semántica se requiere establecer el dominio $\mathcal{D}$ y la interpretación $\mathcal{I}$.

El dominio $\mathcal{D}$ es el conjunto de elementos que le da significado a los términos y variables

Una interpretación $\mathcal{I}$ asigna el significado a las funciones y predicados

Una valuación a partir de una interpretación y un dominio evalúa las fórmulas cerradas a verdadero o falso

Semántica de las fórmulas

El dominio $\mathcal{D}$ es el conjunto de elementos que le da significado a los términos y variables

Una interpretación $\mathcal{I}$ asigna el significado a las funciones y predicados

Un valuación a partir de una interpretación y un dominio evalúa las fórmulas cerradas a verdadero o falso

Las fórmulas evaluadas a verdadero en todos los dominios e interpretaciones son válidas

En el contexto de métodos formales el par dominio, interpretación es llamado estado $(\mathcal{D}, \mathcal{I})$.

Sintaxis concreta

	Text book	KeY
Negation	$\neg$	!
Conjunction	$\wedge$	&
Disjunction	$\vee$	|
Implication	$\rightarrow, \supset$	->
Equivalence	$\leftrightarrow$	<->
Universal Quantifier	$\forall x; \phi$	\forall τ x; ϕ
Existential Quantifier	$\exists x; \phi$	\exists τ x; ϕ
Value equality	$\doteq$	=

Cálculo

El cálculo elegido para trabajar es el sequent calculus (cálculo de secuentes).

Los datos básicos manipulados por la reglas de este cálculo son **secuentes**, es decir fórmulas de la forma:

$$\Phi_1, \ldots, \Phi_n \Rightarrow \Psi_1, \ldots, \Psi_m$$

La fórmula previa, de sequent calculus, resulta válida cuando la fórmula

$$\Phi_1 \wedge \ldots \wedge \Phi_n \rightarrow \Psi_1 \vee \ldots \vee \Psi_m$$

lo es

Reglas proposicionales del cálculo

main	left side (antecedent)	right side (succedent)
not	$\dfrac{\Gamma \Rightarrow \phi, \Delta}{\Gamma, \neg\phi \Rightarrow \Delta}$	$\dfrac{\Gamma, \phi \Rightarrow \Delta}{\Gamma \Rightarrow \neg\phi, \Delta}$
and	$\dfrac{\Gamma, \phi, \psi \Rightarrow \Delta}{\Gamma, \phi \wedge \psi \Rightarrow \Delta}$	$\dfrac{\Gamma \Rightarrow \phi, \Delta \qquad \Gamma \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \phi \wedge \psi, \Delta}$
or	$\dfrac{\Gamma, \phi \Rightarrow \Delta \qquad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \phi \vee \psi \Rightarrow \Delta}$	$\dfrac{\Gamma \Rightarrow \phi, \psi, \Delta}{\Gamma \Rightarrow \phi \vee \psi, \Delta}$
imp	$\dfrac{\Gamma \Rightarrow \phi, \Delta \qquad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \phi \rightarrow \psi \Rightarrow \Delta}$	$\dfrac{\Gamma, \phi \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \phi \rightarrow \psi, \Delta}$

close $\dfrac{}{\Gamma, \phi \Rightarrow \phi, \Delta}$ true $\dfrac{}{\Gamma \Rightarrow \text{true}, \Delta}$ false $\dfrac{}{\Gamma, \text{false} \Rightarrow \Delta}$

Departamento de Ciencias e Ingeniería de la Computación – Universidad Nacional del Sur, Argentina

FOL a simple vista

	left side, antecedent	right side, succedent
$\forall$	$\dfrac{\Gamma, \forall \tau\, x;\ \phi,\ [x/t']\,\phi \Longrightarrow \Delta}{\Gamma, \forall \tau\, x;\ \phi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow [x/c]\,\phi, \Delta}{\Gamma \Longrightarrow \forall \tau\, x;\ \phi, \Delta}$
$\exists$	$\dfrac{\Gamma, [x/c]\,\phi \Longrightarrow \Delta}{\Gamma, \exists \tau\, x;\ \phi \Longrightarrow \Delta}$	$\dfrac{\Gamma \Longrightarrow [x/t']\,\phi,\ \exists \tau\, x;\ \phi, \Delta}{\Gamma \Longrightarrow \exists \tau\, x;\ \phi, \Delta}$
$\doteq$	$\dfrac{\Gamma, t \doteq t' \Longrightarrow [t/t']\,\phi, \Delta}{\Gamma, t \doteq t' \Longrightarrow \phi, \Delta}$ (+ application rule on left side)	$\dfrac{}{\Gamma \Longrightarrow t \doteq t, \Delta}$

- $[t/t']\,\phi$ is result of replacing each occurrence of t in ϕ with t'
- t,t' variable-free terms of type τ
- c **new** constant of type τ (occurs not on current proof branch)