Métodos Formales Livianos

Marcelo Frias

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Nazareno Aguirre

Departamento de Computación

Facultad de Ciencias Exactas, Físico-Químicas y Naturales

Universidad Nacional de Río Cuarto

Clase 1: Lógica Proposicional - Lógica de Primer Orden - Álgebra Relacional

1 Especificaciones de Sistemas

Varios enfoques tradicionales para la construcción formal y la verificación de sistemas se basan en el uso de *especificaciones*. Una especificación puede usarse para varias tareas, tales como:

- descubrir inconsistencias (por ejemplo, requerimientos contradictorios) o falta de detalles en las funcionalidades requeridas del sistema a construir,
- análisis de propiedades del sistema a construir *antes* de comenzar con su construcción,
- verificación del sistema construido con respecto a su especificación,
- construcción rigurosa de implementaciones a partir de especificaciones.

1.1 Características de los Lenguajes de Especificaciones

Las especificaciones son descripciones abstractas de sistemas. Estas descripciones se escriben en algún *lenguaje de especificaciones*. Existe una gran variedad de lenguajes de especificaciones, clasificables de acuerdo a varias características. Con frecuencia, los lenguajes de especificaciones cuentan con algunas propiedades (deseables):

- si el lenguaje de especificaciones es **formal**, éste no da lugar a ambigüedades y, por lo tanto, no requiere esfuerzos adicionales para evitar confusiones e interpretaciones erróneas de las especificaciones,

- si el lenguaje de especificaciones es **declarativo**, éste facilita la caracterización tanto del sistema a construir como de las propiedades a analizar,
- si el lenguaje de especificaciones es suficientemente **expresivo**, se podrán describir una amplia gama de características de los sistemas a construir, y de propiedades a analizar,
- si el lenguaje cuenta con **mecanismos efectivos para el análisis** de propiedades, éste demanda menos esfuerzos para el estudio de las consecuencias de las funcionalidades requeridas del sistema a construir.

2 Las Lógicas como Lenguajes de Especificaciones

Cuando la intención principal es analizar propiedades de los sistemas a construir, es crucial contar con mecanismos efectivos para el análisis.

Un enfoque usado con frecuencia es el de utilizar **lógicas**, que se ajusten a la descripción de las propiedades de interés, como lenguajes de especificaciones. Las lógicas como lenguajes de especificaciones tienen la particularidad de ser **formales**, a veces **declarativas**, y además cuentan con un mecanismo de análisis de propiedades: la *deducción*.

Con frecuencia, es necesario buscar un balance entre poder expresivo y efectividad para el análisis.

3 La Lógica Proposicional como Lenguaje de Especificaciones

La lógica proposicional es una de las lógicas más simples. Como lenguaje de especificaciones, posee las siguientes características:

- poder expresivo limitado
- importantes propiedades metalógicas y variedad de cálculos de prueba (cálculo de secuentes, deducción natural, deducción “a la Hilbert”, *tableaux*, etc) que pueden usarse para el análisis (demostración) de propiedades.

3.1 Sintaxis de la Lógica Proposicional

- variables proposicionales: $p, q, r, \ldots$
- conectivos lógicos: $\wedge, \vee, \neg, \rightarrow, \ldots$
- fórmulas: $p \wedge q, \neg r, \ldots$

Dado que los conectivos son fijos, el conjunto de fórmulas está determinado por las letras proposicionales. Se denomina *lenguaje* a un conjunto finito de letras proposicionales.

3.1.1 Especificaciones en Lógica Proposicional

- Las proposiciones son el elemento básico de especificación.
- Las proposiciones representan sentencias atómicas, que pueden ser verdaderas o falsas.
- Los conectivos lógicos permiten describir sentencias compuestas, a partir de sentencias básicas.

3.2 Semántica de la Lógica Proposicional

Las variables proposicionales se interpretan como *proposiciones*, es decir, como enunciados que pueden ser verdaderos o falsos. Los conectivos lógicos tienen una *interpretación fija*: $\wedge$ se interpreta como “y”, $\vee$ se interpreta como “o”, etc.

Formalmente, dado un lenguaje $\mathcal{L}$, una *interpretación* es una función:

$$I : \mathcal{L} \rightarrow \{\mathsf{T}, \mathsf{F}\}$$

El valor de verdad de una fórmula bajo una interpretación está recursivamente definido por las siguientes reglas:

- $I \models p$ ssi $I(p) = \mathsf{T}$, donde $p \in \mathcal{L}$,
- $I \models \neg\alpha$ ssi no es cierto que $I \models \alpha$,
- $I \models \alpha \wedge \beta$ ssi $I \models \alpha$ y $I \models \beta$.

Semántica de la Lógica Proposicional (cont.)

El símbolo $\models$ es con frecuencia sobrecargado, para denotar una relación entre conjuntos de fórmulas. Para dos conjuntos de fórmulas Γ y Δ :

$\Gamma \models \Delta$ ssi para toda interpretación I, si $I \models \Gamma$ entonces $I \models \Delta$

($I \models \Delta$ ssi $I \models \delta, \forall \delta \in \Delta$)

3.3 Algunas Propiedades Metalógicas

Lema de la Deducción (a nivel semántico):

$$\Gamma, \alpha \models \beta \quad \Longleftrightarrow \quad \Gamma \models \alpha \rightarrow \beta$$

Decidibilidad de Satisfacción:

Dada una fórmula α, existe un **algoritmo** que permite decidir si existe algún modelo M tal que

$$M \models \alpha$$

o no

4 SAT Solving en Lógica Proposicional

Usando las propiedades metalógicas anteriores, se puede usar una técnica llamada *SAT Solving* para realizar deducciones:

Supongamos que queremos saber si una fórmula α es consecuencia de ciertas premisas Γ. Es decir, queremos saber si es cierto que:

$$\Gamma \models \alpha$$

Gracias al Lema de la Deducción, podemos decir que esto es cierto si y sólo si no existe M tal que $M \models \neg(\Gamma \rightarrow \alpha)$. Podemos usar el algoritmo de decidibilidad para decidir esto último.

5 Ventajas y Desventajas de SAT Solving

La desventaja de reducir deducción a un problema de satisfacción como se explicó anteriormente es la **complejidad temporal** de este enfoque. La dificultad principal es que el algoritmo de decidibilidad de satisfacción en lógica proposicional es $O(n \times 2^n)$, donde n representa el número de proposiciones en la fórmula a la cual se aplica el algoritmo.

La ventaja fundamental de reducir la deducción a un problema de satisfacción es que, como resultado, obtenemos una técnica absolutamente **automática** para decidir si un enunciado es consecuencia de un conjunto de premisas.

6 Actividades

1. Demuestre que $\models$, como relación binaria entre conjuntos de fórmulas de la lógica proposicional, es *monótona*, es decir, que:

$$\Gamma \models \Delta \text{ implica } \Gamma \cup \Gamma' \models \Delta$$

   cualesquiera sean los conjuntos de fórmulas Γ, Γ' y Δ (sobre un mismo lenguaje).

2. Demuestre o refute lo siguiente:

   Si una fórmula es satisfacible, su negación necesariamente no lo es.

7 Lógicas de Primer Orden

Las limitaciones en poder expresivo de la lógica proposicional hacen que ésta sea de poca utilidad como lenguaje de especificaciones. Por supuesto, pueden considerarse como alternativas más expresivas algunas lógicas de primer orden.

El problema de satisfacción de fórmulas **no es decidible** en lógica de primer orden. Esto impide, en principio, usar como algoritmo de prueba la reducción de deducción a un problema de satisfacción.

Existen, sin embargo, **cálculos de prueba completos** para lógica de primer orden.

7.1 Sintaxis y Semántica

Sintaxis:

- variables: $x, y, z, \ldots$
- símbolos de función: $f(x, y), g(x, y, z), c, \ldots$
- predicados: $p, q(x), r(s, t), \ldots$
- conectivos y cuantificadores: $\forall, \wedge, \vee, \neg, \rightarrow, \ldots$
- fórmulas: $\forall x : p(x), p(c) \wedge q(c, f(c)), \ldots$

Semántica:

- las interpretaciones requieren un dominio
- los símbolos de función se interpretan como funciones y los predicados como relaciones sobre el dominio de la interpretación
- las interpretaciones incluyen una asignación, para interpretar las variables
- los conectivos lógicos tienen una *interpretación fija*: $\wedge$ se interpreta como “y”, $\vee$ se interpreta como “o”, etc
- $\forall$ tiene una interpretación fija: requiere que la fórmula en cuestión sea verdadera para todas las posibles asignaciones a la variable ligada al cuantificador

8 SAT Solving en Lógica de Primer Orden

Si bien el problema de satisfacción de fórmulas **no es decidible** en lógica de primer orden, es útil aplicar una técnica similar a la aplicada para lógica proposicional.

El número de interpretaciones posibles para fórmulas en lógica de primer orden no puede, en general, reducirse a un conjunto finito, pues las variables pueden tomar valores arbitrarios de sus dominios, y los dominios pueden ser infinitos.

Sin embargo, si ponemos una **cota** k en el **número máximo de elementos en el dominio de una interpretación**, podemos examinar exhaustivamente **todas las interpretaciones posibles de "tamaño" a lo sumo** k. Si empleando este procedimiento **encontramos un modelo** de la fórmula en cuestión, entonces dicha fórmula **es satisfacible**. **Sino**, sólo podemos garantizar que **no existen modelos de la fórmula de "tamaño" menor o igual que** k

8.1 Un Ejemplo

Consideremos la fórmula siguiente:

$$\forall x : p(x, c) \wedge q(f(x, x))$$

Tenemos una función binaria (f), una constante (c), y predicados de aridades uno y dos (q y p, respectivamente). Cuántas interpretaciones posibles de “tamaño” 5 existen?

9 Poder Expresivo de la Lógica de Primer Orden

La lógica de primer orden es un formalismo muy expresivo, que permite especificar gran cantidad de propiedades (en el contexto de especificaciones sistemas de software).

Algunas propiedades, como por ejemplo aquellas relacionadas con minimidad, no pueden expresarse en lógica de primer orden.

10 Alloy: Un Lenguaje de Primer Orden basado en Relaciones

Alloy es un lenguaje de especificaciones de primer orden basado en el uso de relaciones. Alloy está orientado a la especificación de propiedades estructurales de sistemas.

Alloy intenta resolver la complejidad en la expresión de algunas propiedades estructurales comunes en lógica de primer orden utilizando operadores relacionales, definidos en la *lógica relacional*, el formalismo subyacente a Alloy.

11 Sintaxis de Alloy

11.1 Signaturas

Las signaturas son la base del lenguaje Alloy. Éstas permiten, entre otras cosas, denotar dominios.

Consideremos, como un ejemplo inicial, la especificación de un sistema de memorias. En principio, parece necesario contar con los dominios de las direcciones (de memoria) y de datos (a almacenarse en direcciones de memoria). Estos dominios se pueden especificar en Alloy de la siguiente manera:

```
sig Addr { }        sig Data { }
```

11.2 Signaturas Complejas

Las signaturas anteriores son básicas, ya que no tienen estructura interna. Se pueden definir estructuras más complejas usando signaturas. Por ejemplo:

```
sig Memory {
        addrs: set Addr
    map: addrs ->! Data
  }
```

Esta signatura indica, intuitivamente, que una memoria consta de dos “campos”: un conjunto `addrs` de direcciones (subconjunto de `Addr`) y una función total de `addrs` en `Data`.

11.3 Extensión de Signaturas

Pueden definirse tipos más específicos de signaturas extendiendo otras signaturas existentes. Por ejemplo:

```
sig MainMemory extends Memory { }

    sig Cache extends Memory {
      dirty: set addrs
    }
```

Aquí, se define `MainMemory` como un tipo particular de `Memory`, sin estructura adicional; `Cache`, por otra parte, se define como un tipo particular de `Memory` en el cual un subconjunto de su espacio de direcciones se reconoce como “sucio”.

Para completar el ejemplo, podemos considerar una signatura que defina sistemas:

```
sig System {
        cache: Cache
  main: MainMemory
}
```

La intención de esta signatura es describir un sistema (de memoria) como una estructura compuesta por una memoria principal (campo de tipo `MainMemory`) y una cache (campo de tipo `Cache`).

11.4 Operaciones en un Modelo Alloy

Las signaturas permiten describir dominios y sus estructuras. Siguiendo el estilo de Z, se pueden especificar operaciones mediante *expresiones* que relacionan el estado previo a la aplicación de una operación y el estado posterior correspondiente:

```
fun Write(m,m': Memory, d: Data, a: Addr) {
        m'.map = m.map ++ (a -> d)
}
```

La función `Write` transforma una memoria, sobreescribiendo el valor asociado a una dirección `a` con `d`.

Esta función, como los esquemas para operaciones en Z, no está directamente asociada a ninguna signatura. Las variables “primadas” corresponden a estados posteriores a la ejecución de la operación (es una convención).

11.5 Hechos (*facts*)

Por supuesto, las especificaciones acerca de la estructura de un sistema que uno puede lograr mediante el uso exclusivo de signaturas son limitadas. Generalmente, se necesita complementar las signaturas con restricciones estructurales adicionales.

Por ejemplo, podríamos complementar la especificación anterior con un *hecho* que las memorias principales no son caches y viceversa:

```
fact {no (MainMemory & Cache)}
```

11.6 Aserciones

Todos los elementos de Alloy descriptos hasta el momento forman parte de la descripción de modelos. Las *aserciones*, en cambio, son aquellos enunciados que queremos comprobar si son o no propiedades del sistema especificado. Por ejemplo, podríamos querer saber si el sistema de memorias tiene la propiedad siguiente:

```
assert {
   all s : System |
          no s.cache.dirty => s.cache.map in s.main.map
  }
```

12 Características de Alloy

- el lenguaje es *mínimo*, no incluye ni siquiera tipos básicos,
- el lenguaje es *relacional*, con una sintaxis simple y elegante basada en operadores relacionales tales como composición, unión, complemento, etc.
- el lenguaje es *expresivo*, permitiendo cuantificación similar a la cuantificación sobre individuos, pero con una semántica relacional, y clausura transitiva,
- el lenguaje es *fácil de usar*, con construcciones que recuerdan elementos de orientación a objetos, tales como extensión de signaturas y la notación de punto para acceso a servicios (pero el lenguaje **no es** orientado a objetos),
- el lenguaje hace que las especificaciones sean *analizables* automáticamente, usando técnicas de SAT solving (para validación, pues el lenguaje es de primer orden)

13 Actividades

1. Averigüe acerca de la semántica de los distintos operadores de la lógica relacional y la semántica de las construcciones de Alloy.
2. Cuál es la semántica del operador ‘·’?
3. Instale el Alloy Analyzer.
4. Estudie la documentación provista en “Micromodels of Software: Lightweight Modelling and Analysis with Alloy” (D. Jackson 2002).
5. Complete el modelo de memorias con cache y compruebe, usando el Alloy Analyzer, si la aserción vista en estas notas es consecuencia del modelo obtenido.
6. Modele grafos dirigidos y algunas de sus operaciones en Alloy. Modele con una aserción la aciclicidad de grafos.