



UNCUYO  
UNIVERSIDAD  
NACIONAL DE CUYO

# PARADIGMAS DE PROGRAMACIÓN

## PARADIGMA LÓGICO

Dr. Pablo Vidal

Facultad de Ingeniería  
Universidad Nacional de Cuyo

2023

- ➊ Introducción
- ➋ Paradigma Lógico
- ➌ Prolog
- ➍ Lógica 1er Orden
- ➎ Unificación

# Introducción

# Objetivo Unidad 3

Estudiar el modelo de la **programación declarativa** a través de uno de sus más importantes realizaciones la **programación lógica** y la realización estándar de esta en el **lenguaje Prolog**.

# Características de la programación convencional ...

- Fue la primera en desarrollarse.
- Responde directamente a la arquitectura establecida por von Neumann de máquinas computadoras que cargan, modifican y dejan los resultados de un proceso computacional en registros de memoria.
- Recibe la calificación de imperativa, proveniente del modo verbal imperativo con el cual los seres humanos nos comunicamos en ciertas ocasiones para expresar una orden o comando, tal y como ocurre en las instrucciones de estos lenguajes.
- El primer ejemplo significativo de lenguaje de programación imperativa fue FORTRAN.

# Programa = Lógica + Control

En la tarea de programación podemos distinguir dos aspectos fundamentales:

- **Aspectos lógicos: ¿Qué debe computarse? (especificación del problema)** Esta es la cuestión principal y la que motiva el uso de un ordenador como medio para resolver un determinado problema.
- **Aspectos de control: ¿Cómo debe computarse? (implementación del programa que lo resuelve)** Entre los que podemos distinguir:
  - Organización de la secuencia de cálculos en pequeños pasos.
  - Gestión de la memoria durante la computación.

Parece deseable que los lenguajes de programación permitan mantener las distancias entre ambos.

# Ejemplo programación imperativa

El siguiente programa en C# especifica cómo ha de llevarse a cabo el proceso de concatenar dos listas de enteros:

```
public int[] concatena(int[] l1, int[] l2)
{
    int[] l3 = new int[l1.Length + l2.Length];
    for (int i = 0; i < l1.Length; i++)
        l3[i] = l1[i];
    int sigue = l1.Length;
    for (int i = 0; i < l2.Length; i++)
        l3[sigue + i] = l2[i];
    return l3;
}
```

# Podemos resaltar:

- ❶ Presencia de instrucciones de control de flujo (if, for, while, etc.) y la gestión de memoria oscurecen el contenido lógico del programa.
- ❷ La operación de asignación cambia el estado de las variables por lo que obliga a conocer el contexto de la variable en cada momento.
- ❸ Las secuencias de instrucciones que constituyen el programa son órdenes a la máquina.
- ❹ Para entender el programa debemos ejecutarlo mentalmente estudiando cómo cambian los contenidos de las variables y otras estructuras en la memoria.
- ❺ La lógica y el control están mezclados, lo que dificulta la verificación formal del programa.



# Programación Imperativa o Procedimental

Especifica cómo ha de llevarse a cabo un proceso de computación y responde en general a la pregunta **¿cómo?**.

En el ejemplo ... ¿cómo hallar la concatenación de dos listas?

Hemos analizado algunos puntos débiles de los lenguajes convencionales, pero ellos reúnen otras ventajas que los han hecho preferidos entre los programadores. Entre estas ventajas podríamos citar:

- eficiencia en la ejecución
- modularidad
- herramientas para la compilación separadas
- herramientas para la depuración de errores

# ¿Qué es la declaratividad?

Es una modalidad lógica del **intercambio de información** entre los seres humanos. Por ejemplo, en la lengua española las siguientes proposiciones son declarativas:

- Julio es abuelo de Ana.
- El agua hierve a 100 grados centígrados de temperatura a nivel del mar.
- Un paciente tiene presión alta si su sistólica es mayor que 140 y su diastólica es mayor que 90. (Caracterización de presión alta).

Cuando afirmamos (negamos), enunciamos, definimos o describimos con proposiciones de un lenguaje, utilizamos **proposiciones declarativas** y a estas puede asociarse una **función lógica de verdad**.

# Programación Declarativa

Se especifica qué debe computarse, en vez de cómo, y a la pregunta **¿qué?** se responde declarativamente con una definición o con el enunciado o descripción de un hecho o suceso.

En este paradigma la tarea de programación consiste en centrar la atención en la lógica dejando el control (que se asume automático) al sistema. El **componente lógico** determina el significado del programa mientras que el componente de control solamente afecta su eficiencia.

La característica fundamental de la programación declarativa es el **uso de la lógica como lenguaje de programación**.

Casi simultaneando con FORTRAN en el tiempo aparece la programación declarativa con el lenguaje LISP, un lenguaje basado en el concepto de función y de definición recursiva de funciones.

# Recursividad

- Es un poderoso instrumento de definición con un carácter declarativo muy utilizado en los lenguajes de programación declarativos para definir estructuras de datos y procesos.
- Posteriormente fue adoptado en lenguajes imperativos como ALGOL. El siguiente ejemplo de la definición recursiva de la función de potencia en ALGOL muestra su proceder declarativo:
- **$\text{potencia}(m, n) = \text{if } n = 1 \text{ then } m \text{ else potencia}(m, n-1) * m$**

# Problema: Determinar si una persona es abuelo de otra.

- Los seres humanos parten de la definición de abuelo, es decir, de una proposición declarativa, que podría ser enunciada utilizando variables de la siguiente forma:

**X es abuelo de Y si X es padre de Z y Z es padre o madre de Y.**

- La programación declarativa aspira a especificar problemas de una manera más cercana a la forma en que los especifican los seres humanos.
- La especificación de problemas es eminentemente declarativa tanto en lo que concierne a su definición como al conocimiento que se posee para resolverlo y a los datos de la instancia particular del problema que se desea resolver.

# Lenguajes declarativos (LD) vs imperativos (LI)

- ❶ Diseño del lenguaje y escritura de programas:
  - Sintaxis sencilla (LD)
  - Facilidades de soporte al proceso de análisis (LD)
  - Mecanismos de reutilización del software (LI)
  - Entornos de programación (LI)
- ❷ Verificación de programas
  - Verificación de la corrección (LD)
  - Terminación (LD)
  - Depuración de programas (LI)
- ❸ Mantenimiento (Incluye la facilidad de uso y lectura y la modularidad y compilación separada)
- ❹ Coste y eficiencia:
  - Coste de ejecución (LI)
  - Coste de desarrollo (LD)

# Aplicaciones de la Programación Declarativa

Un lenguaje de programación no es bueno para todo tipo de tareas. Cada lenguaje tiene su dominio y aplicación.

Algunos campos en los que la programación declarativa es utilizada en la actualidad:

- Procesamiento del lenguaje natural
- Representación del conocimiento
- Química y biología molecular
- Sistemas Expertos
- Metaprogramación
- Bases de datos deductivas
- Buscadores inteligentes

# Paradigma Lógico



- Tiene sus fundamentos en las teorías de la lógica proposicional.
- Cláusulas de Horn.
- Un programa lógico no tiene un algoritmo que indique los pasos que detallen la manera de llegar a un resultado, sino que es el sistema internamente el que proporciona la secuencia de control.

# Aplicaciones del Paradigma Lógico

- Inferencia de Tipos
- Demostración de teoremas
- Procesamiento natural del lenguaje
- Coincidencia de Patrones
- Generación de casos de prueba combinatoriales
- Bases de datos deductivas
- Sistemas expertos
- Inteligencia Artificial
- Lenguaje de Consultas
- Problemas de optimización
- entre otros ...

# Programación Declarativa

- La programación declarativa incluye como paradigmas más representativos la programación lógica y la funcional.
- La **programación funcional** se basa en el concepto de función (matemática), centrándose desde el punto de vista computacional, en la evaluación de expresiones (funciones) para obtener un valor.
- La **programación lógica** se basa en fragmentos de la lógica de predicados, particularmente la lógica de cláusulas definidas.
- Ambas tienen diferentes realizaciones en lenguajes como **Haskell** y **Prolog** respectivamente. En esta unidad comenzaremos con la programación lógica.

# Prolog

# Un programa en Prolog

Es un conjunto de fórmulas lógicas.

A continuación analizaremos un primer programa Prolog en el cual se da una definición recursiva de la concatenación de dos listas.

*concatena*([], *X*, *X*). *concatena*([*X*|*Rx*], *Y*, [*X*|*Z*]) :  $\neg$ *concatena*(*Rx*, *Y*, *Z*).

# Intérprete de Prolog

Si el programa anterior responde a la pregunta *qué* es concatenar dos listas.  
**¿Quién se encarga entonces del control o ejecución de un programa declarativo?**

Como se estudiará en esta unidad con respecto al lenguaje Prolog, un **intérprete** hace una **lectura procedimental de un programa declarativo y lo ejecuta**. En especial el intérprete introduce el *control* necesario para la ejecución del programa.

# Intérprete de Prolog

## Notación Prolog

- las letras **mayúsculas** representan **variables**
- el símbolo **:-** representa la operación lógica  $\Rightarrow$  utilizada en su notación  
 $\langle \textit{consecuente} \rangle \Leftarrow \langle \textit{antecedente} \rangle$   
 $(\langle \textit{consecuente} \rangle \textit{ si } \langle \textit{antecedente} \rangle)$
- la **coma** representa la operación lógica de la **conjunción** ( $\wedge$ )
- el **punto y coma** representa la **disyunción** ( $\vee$ )
- el **punto** representa el **final de la fórmula**.

Ejemplo Programa Familia:

**X es abuelo de Y si X es padre de Z y Z es padre o madre de Y.**

# Listas

Desde un punto de vista lógico una lista es un tipo particular de término construido mediante el símbolo constructor binario  $.$  y el símbolo de constante  $nil$  que denota la lista vacía:

- i)  $nil$  es una lista.
- ii) Si  $s$  es un término cualquiera y  $t$  es una lista, entonces  $.(s, t)$  es una lista.

En la lista  $.(s, t)$ ,  $s$  se denomina el primer elemento de la lista y  $t$  el resto de la lista. El resto de una lista es también una lista. Ejemplos:

- $nil$
- $.(2, nil)$
- $.(2, .(5, nil))$
- $.(.(1, nil), .(1, .(3, nil)))$



# Listas en Prolog

- $[]$  denota la lista vacía.
- $[1, 2, 3]$  denota la lista  $.(1, .(2, .(3, \text{nil})))$  que tiene como únicos elementos los números 1, 2 y 3.
- $[X|Y]$  denota una lista con un primer elemento  $X$  y un resto  $Y$ . Se trata de un esquema o patrón de lista que representa cualquier lista que tiene al menos un elemento.
- $[X, Y|Z]$  es el esquema de lista que representa cualquier lista con al menos dos elementos.

# Conclusiones

- Los programas **procedimentales** responden a la pregunta **¿cómo?**, ofreciendo más que una definición un procedimiento.
- Los programas **declarativos** responden a la pregunta **¿qué?**, especifican los problemas que se quieren resolver, mediante proposiciones declarativas.
- Ambas formas de programación, declarativa y procedimental son necesarias en la programación actual, no se contraponen y hasta pueden ocurrir en un mismo programa.

La **programación declarativa** permite al programador concentrarse fundamentalmente en la representación y definición de los datos y procesos relacionados con la solución de un problema haciendo abstracción en un primer momento de detalles de implementación y en especial del control en la interpretación y ejecución de su programa.

# SWI-Prolog

- ❶ **SWI-Prolog** es una implementación versátil del lenguaje Prolog.  
(<https://www.swi-prolog.org/>)
  - Desde una terminal ejecutar comando: **swipl**
  - Comando para salir: **halt.**
  - Cargar archivos (test1.pl) : **[test1]., [test1,test2].,consult('test1').,reconsult('test2').**
  - Opciones directorio: **pwd, ls, cd**
  - Documentación online  
(<https://www.swi-prolog.org/pldoc/man?section=cmdline>)
- ❷ **SWISH** plataforma web para desarrollar y ejecutar código Prolog en un entorno colaborativo.(<https://swish.swi-prolog.org/>)

# Lógica 1er Orden

# Objetivos

## ❶ Fundamentos Teóricos

- Lógica de Primer Orden (LPO)
- Lógica de Cláusulas Definidas (C. de Horn)

Programa Prolog : Conjunto de cláusulas definidas.

## ❷ Proceso de Unificación. Conceptos y Algoritmo.

# Lógica de Primer Orden

Una *lógica de primer orden* LPO consta de un *lenguaje de primer orden* L, y de un sistema lógico asociado.

Un *lenguaje de primer orden* está constituido por un conjunto de **expresiones** con las cuales se denotan las **entidades** de un dominio y se enuncian las **propiedades** y las **relaciones** entre las entidades del dominio.

**Sintaxis** : Determina las reglas de formación o gramática de las expresiones aceptadas en un lenguaje.

**Semántica**: Determina las reglas de interpretación en el dominio de tales expresiones.

# LPO

Proviene de la lógica de predicados de primer orden y se dispone de:

- Un conjunto de elementos simples llamados átomos.
- Los átomos están representados por caracteres minúsculas (Ej.: a, b, julieta, 23, etc.).
- Un vocabulario V de variables (X, Y, Z).
- Las variables están representadas-en mayúsculas (X, Y, Z).
- Un vocabulario F de símbolos funcionales.
- Los símbolos funcionales se representan en minúsculas.

# Representación en forma de implicación

Dado que todas las variables que ocurren en una cláusula están cuantificadas universalmente, podemos omitir la representación de los cuantificadores:

Ejemplos:

- $abuelo(x, y) \Leftarrow padre(x, z) \wedge padre(z, y)$
- $concatena([x|v], y, [x|z]) \Leftarrow concatena(v, y, z)$



# Lógica de Cláusulas Definidas

Una cláusula se denomina *cláusula definida* si la misma contiene a lo sumo un literal positivo. Es decir, una cláusula definida es de la forma  $A \Leftarrow B_1 \wedge \dots \wedge B_m$

- Denominaremos **regla** a tal cláusula definida, siendo A la **cabeza** de la cláusula y  $B_1 \wedge \dots \wedge B_m$  el **cuerpo** de la cláusula.
- Si la cláusula tiene sólo cabeza se denomina **hecho** y si tiene sólo cuerpo se denomina **objetivo**.
- Para diferenciar entre hecho y objetivo, los objetivos comienzan con el símbolo  $\Leftarrow$ .

# Cláusulas Definidas

Ejemplos:

- $abuelo(x, y) \Leftarrow padre(x, z) \wedge padre(z, y).$
- $padre(a, b).$
- $\Leftarrow abuelo(a, x).$
- $concatena([], [2], [2]).$
- $\Leftarrow concatena([1], [2], x).$

# Cláusulas Definidas

Las cláusulas definidas son de uso generalizado en la programación lógica y en particular en Prolog: Un programa Prolog es un **conjunto de cláusulas definidas**.

Los hechos y las reglas de un programa que comienzan con el mismo nombre de predicado o relación constituyen la definición de dicho predicado en el programa.

# Predicado Concatena

$concatena([], X, X). \text{ concatena}([X|Rx], Y, [X|Z]) : \neg concatena(Rx, Y, Z).$

Utilizando la notación vista en la primera clase y el programa concatena se pueden formar los siguientes objetivos:

$: \neg concatena([1, 2, 3], [4, 5], X).$

Existe la lista X tal que X es la concatenación de [1,2,3] y [4,5]

$: \neg concatena(X, [1, 2, 3], [1, 2, 3]).$

Existe la lista X, tal que al concatenarla con [1,2,3] se obtiene la lista [1,2,3]

# Observación

Observe que:

- un **programa** es propiamente una teoría de primer orden en la lógica de cláusulas definidas constituyendo los hechos y las reglas del programa los axiomas de la teoría.
- un **objetivo** es un teorema que queremos establecer como consecuencia lógica de un programa.

# Observaciones

Un intérprete de Prolog:

- Será desde un punto de vista lógico un **demostrador por refutación** de teoremas.
- Simultáneamente con la refutación del objetivo debe **suministrar valores** para las variables.

# Unificación

# Unificación

En la PL los valores con los que se puede **instanciar** una **variable** son términos cualesquiera del lenguaje de un programa.

La instanciación se basa en la operación lógica de **sustitución** (de una variable por un término) que constituye la operación fundamental de un proceso denominado **unificación**.



# Sustitución

Una **sustitución**  $\sigma$  es un conjunto finito de sustituciones  $t_1/v_1, \dots, t_n/v_n$  donde  $v_i$  es una variable y  $t_i$  es un término, tales que  $t_i \neq v_i$  y  $v_i \neq v_j$  para  $i \neq j$ .

Observaciones:

- $t_i/v_i$  se lee: "sustitución de todas las ocurrencias de la variable  $v_i$  por el término  $t_i$ ".
- Denotaremos por  $\epsilon$  la sustitución idéntica.

Ejemplos:  $\sigma_1 = \{2/X, W/Y\}$ ,  $\sigma_2 = \{g(Z)/X, a/Y\}$ ,  $\sigma_3 = \{c/X, a/Y\}$

# Sustitución

Si  $C$  es la cláusula  $R(X, f(Y), b)$  entonces la aplicación de la sustitución  $\sigma = \{2/X, W/Y\}$  a  $C$  da como resultado la cláusula  $R(2, f(W), b)$  que denotaremos por  $C\sigma$ .

# Aplicando sustitución a un conjunto de expresiones simples

Denominaremos **expresión simple** a todo literal o término.

Sea  $E$  un conjunto finito de expresiones simples y  $\sigma$  una sustitución, entonces  $E\sigma$  es el conjunto de las expresiones que se obtienen al aplicar la sustitución  $\sigma$  a cada expresión simple de  $E$ .

Ejemplo: Dados  $E = \{p(X, Y), p(f(a), Z), p(f(Z), Y)\}$  y  $\sigma = \{f(a)/X, a/Y, c/Z\}$  se tiene

$$E\sigma = \{p(f(a), a), p(f(a), c), p(f(c), a)\}$$

# Unificador

Sea  $E$  un conjunto finito de expresiones simples y  $\sigma$  una sustitución,  $\sigma$  es un **unificador** de  $E$  si y sólo si  $E\sigma$  es un conjunto unitario.

Un conjunto finito de expresiones simples  $E$  es **unificable** si existe un unificador de  $E$ .

Ejemplo: Para  $E = \{p(X, Y), p(f(a), Z), p(f(Z), Y)\}$  se tiene que  $\sigma = \{f(a)/X, a/Y, a/Z\}$  es un unificador de  $E$ , dado que  $E\sigma = \{p(f(a), a)\}$ .

# Unificador (UMG)

Def.: La sustitución  $\sigma$  es un unificador de máxima generalidad (UMG) de los términos  $t_1$  y  $t_2$  si

- $\sigma$  es un unificador de  $t_1$  y  $t_2$ .
- $\sigma$  es más general que cualquier unificador de  $t_1$  y  $t_2$ .

Ejemplos:

- 1  $[x/g(z), y/z]$  es un UMG de  $f(x, g(z))$  y  $f(g(y), x)$ .
- 2  $[x/g(y), z/y]$  es un UMG de  $f(x, g(z))$  y  $f(g(y), x)$ .
- 3  $[x/g(a), y/a]$  no es un UMG de  $f(x, g(z))$  y  $f(g(y), x)$ .

Nota: Las anterior definición se extienden a conjuntos de términos y de literales

# Unificador (UMG)

Numerosos algoritmos han sido propuestos para encontrar el umg de dos términos, siendo el más conocido el **Algoritmo de Unificación de Robinson(AUR)** (1965).



- Filósofo, matemático y científico de la computación.
- Preparó el terreno para el paradigma de la programación lógica, en particular para el lenguaje Prolog.
- Premio Herbrand de 1996 por sus distinguidas contribuciones al razonamiento automatizado.

# Elementos a considerar en AUR

**La comprobacion de apariciones:** Con la comprobación de apariciones (*occur check*) simplemente queremos detectar cuándo una variable aparece dentro de un término. Si bien desde un punto de vista formal esto no ofrece ninguna complicación, sí la ofrece cuando se intenta sistematizar este procedimiento de tener que buscar por cada argumento y dentro del mismo la aparición de una variable. En cualquier caso, sólo se trata de eso.

Ejemplo: La variable  $X$  se encuentra dentro de los términos  $p(a, f(b, X))$  y  $q(X, g(X, Y))$  pero no de los términos  $p(a, Z, g(a, b))$  y  $q(a, b)$ .



# Instrucciones de AUR:

Sean  $E$  y  $F$  dos términos que queremos unificar. Consideramos inicialmente  $\sigma_0 = \{\}$  una sustitución vacía, es decir, que no cambia ninguna variable. Dado que vamos a realizar un proceso iterativo, consideramos inicialmente  $E_0 = \sigma_0(E)$  y  $F_0 = \sigma_0(F)$ . En cada iteración  $k$  del algoritmo se realizan los siguientes pasos:

**Paso 1** Si  $E_k = F_k$  entonces las cláusulas  $E$  y  $F$  son unificables y un unificador de máxima generalidad es  $\sigma = \sigma_k \dots \sigma_0$ . Además, el término  $E_k$  es el término unificado. En este caso el proceso termina aquí.

**Paso 2** Si  $E_k \neq F_k$  entonces se busca el primer par de discordancia entre  $E_k$  y  $F_k$ . Sea este  $D_k$ .

# Instrucciones de AUR:

**Paso 3** Si  $D_k$  contiene una variable y un término (pueden ser dos variables y una de ellas hace de término) pasamos al siguiente paso. En otro caso los términos no son unificables y terminamos el proceso.

**Paso 4** Si la variable aparece en el término (*occur check*)  $E$  y  $F$  no unifican y terminamos. Si esto no ocurre pasamos al siguiente paso.

**Paso 5** Construimos una nueva sustitución que vincule la variable con el término de  $D_k$ . Sea esta sustitución  $\sigma_k + 1$ . Construimos ahora dos nuevos términos  $E_k + 1 = \sigma_k + 1(E_k)$  y  $F_k + 1 = \sigma_k + 1(F_k)$  y volvemos al paso 1.

# Ejemplo:

Sean los términos  $p(a, X)$  y  $p(X, Y)$  aplicando el algoritmo paso a paso tenemos:  
 $E_0 = p(a, X)$  ,  $F_0 = p(X, Y)$  y  $\sigma_0 = \{\}$

**Iteración 1** Desarrollamos para  $k = 0$

**Paso 1** Como  $E_0 \neq F_0$  vamos al paso 2.

**Paso 2**  $D_0 = \{a, X\}$ . Vamos al paso 3.

**Paso 3** En  $D_0$  uno es un término ( $a$ ) y el otro una variable ( $X$ ). Vamos al paso 4.

**Paso 4** La variable  $X$  no aparece en el término  $a$ . Vamos al paso 5.

**Paso 5** Sea  $\sigma_1 = \{a/X\}$ . Sean también  $E_1 = \sigma_1(E_0) = p(a, a)$  y  
 $F_1 = \sigma_1(F_0) = p(a, Y)$ . Volvemos al paso 1.

# Ejemplo:

**Iteración 2** Desarrollamos para  $k = 1$

**Paso 1** Como  $E_1 \neq F_1$  vamos al paso 2.

**Paso 2**  $D_1 = \{a, Y\}$ . Vamos al paso 3.

**Paso 3** En  $D_1$  uno es un término ( $a$ ) y el otro una variable ( $X$ ). Vamos al paso 4.

**Paso 4** La variable  $Y$  no aparece en el término  $a$ . Vamos al paso 5.

**Paso 5** Sea  $\sigma_2 = \{a/Y\}$ . Sean también  $E_2 = \sigma_2(E_1) = p(a, a)$  y  $F_2 = \sigma_2(F_1) = p(a, a)$ . Vamos al paso 1.

**Iteración 3** Desarrollamos para  $k = 2$



**Paso 1** Como  $E_2 = F_2$  el algoritmo termina con  $\sigma = \sigma_2\sigma_1\sigma_0 = \{a/X, a/Y\}$  como unificador de máxima generalidad.

# Algoritmo de unificación en Prolog

El algoritmo de unificación que utiliza Prolog es básicamente igual al aquí descrito, excepto que no dispone del Paso 4 (*occur check*) por cuestión de eficiencia, ya que la comprobación de *occur check* consume demasiado tiempo, y hay que realizarla muchas veces.

Por otro lado, no es habitual encontrar situaciones que deriven en programas donde se produzcan *occur checks*. En cualquier caso, si se presenta, dependiendo de la implementación de Prolog, se comporta de una manera u otra. Hay sistemas Prolog que lo detectan y avisan del hecho, y otros que simplemente dejan de funcionar.

# Referencias

-  MAX BRAMER. 2014. **Logic Programming with Prolog (2nd. ed.)**. Springer Publishing Company, Incorporated.
-  CLOCKSIN, W. F., MELLISH, C. S. 2003. *Programming in Prolog*. Berlin: Springer. ISBN: 978-3-540-00678-7