Compiladores II: Introducción

¿Por qué estudiar compiladores?

- Los has utilizado mucho; sabes *(crees)* que los compiladores funcionan, pero no sabes cómo.

Premisa 1:

Es posible programar sin saber cómo funcionan los compiladores.

- Lenguajes en los que probablemente has programado:
  - **C**
  - **C++**
  - **Ada**
  - **Algún ensamblador**
  - **Lisp**
  - **Pascal?**
  - **Java?**
  - **Matlab?**
  - **Ruby?**

Compiladores II: Introducción

Hecho 1:

La premisa 1 es **falsa**; conocer el funcionamiento de los compiladores permite utilizarlos mejor.

- Comprenderás mejor la naturaleza de los lenguajes de programación y sus limitaciones.

FORTRAN IV:

```
      DIMENSION VECTORP(11)
      DIMENSION VECTORQ(5)
      ...
```

error
de compilación:
¿por qué?

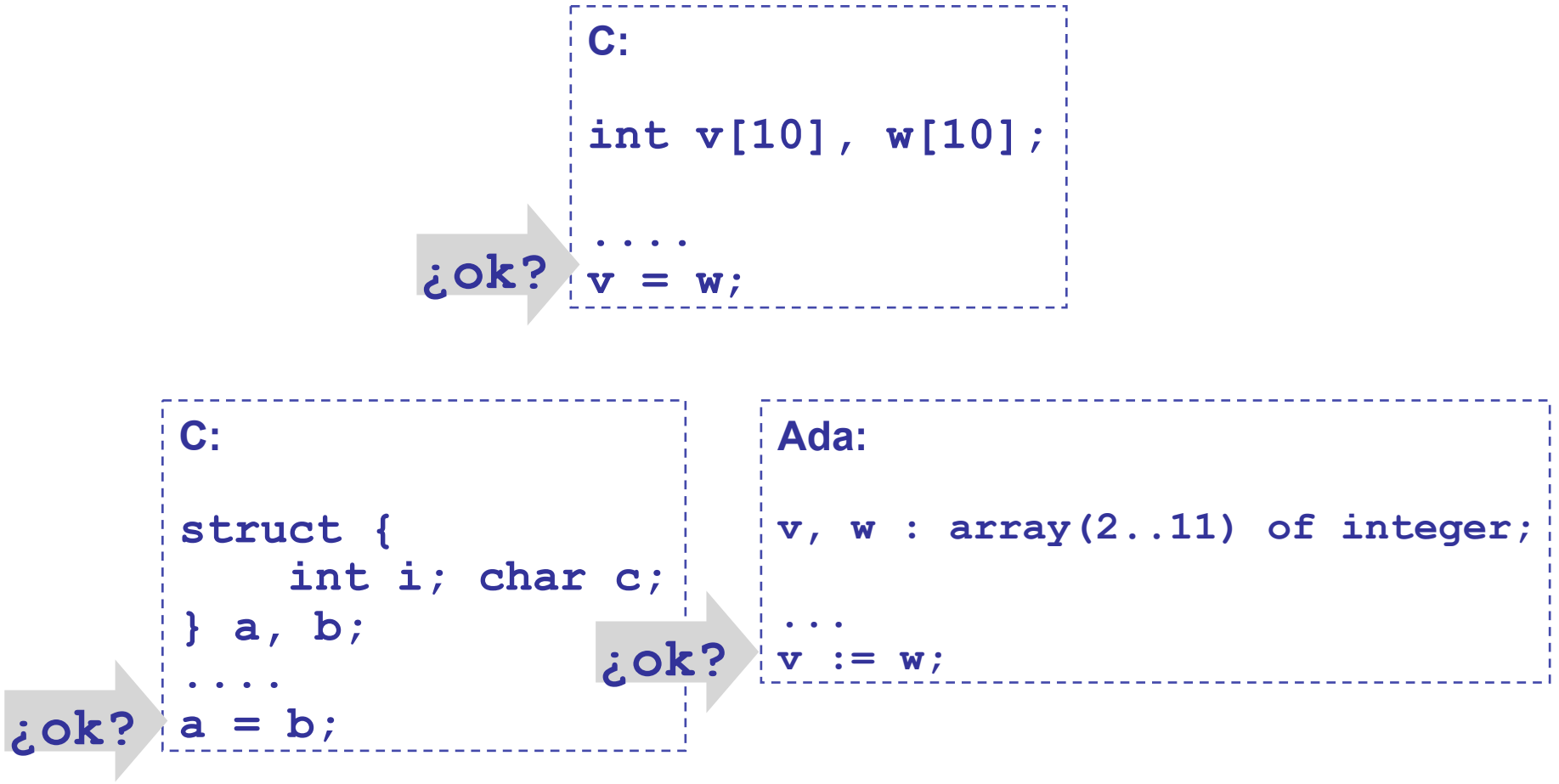
C:

```
      #include <stdio.h>

      int i;
      ...
      printf ("%d %d", i);
```

error
¿de compilación?,
¿de ejecución?
...
¿por qué?

Compiladores II: Introducción



- La equivalencia de tipos está definida de forma diferente en C y en ADA.

Compiladores II: Introducción

C:

```
int v[10];
...
```

Ada:

```
v : array(integer range 2..11) of integer;
```

¿por qué el primer elemento siempre es v[0]?

¿por qué no existe esa limitación?

`v[i]`

es equivalente a:

`*(v + i)`

`V(i)`

es *equivalente* a:

`*(v + i - 2)`

- Lenguajes como C, apuestan por la eficiencia del compilador (aritmética de punteros), Ada por la capacidad expresiva y la corrección.

Compiladores II: Introducción

- Podrás desarrollar programas más eficientes (en tiempo y espacio).

```
M = ones(10000, 10000);

M = function recursiva(M, i)
...
  M (i, j) = 0;
  if cond then
     M = recursiva(M, i+1)
...
```

Mejor...

```
function recursiva
global M;
...
  M (i, j) = 0;
  if cond then
     recursiva (i+1)
...
```

Es muy lento...
A veces, error
de ejecución

- Aprenderás a **no** suponer que el compilador siempre hace lo más apropiado.

Compiladores II: Introducción

- Excepto en Pascal, el tipo ***conjunto*** no está predefinido:

Pascal:

```
var vocales: set of char;

vocales := ['a', 'e', 'i', 'o', 'u'];
...
if c in vocales then ...
```

- Problemas de representación interna, fundamentalmente de espacio

```
var digitos_pares: set of integer;
digitos_pares := [2, 4, 6, 8];
...
if i in digitos_pares then ...
```

Ocupa 8K bytes,
Independiente del contenido

Compiladores II: Introducción

- Te resultará más sencillo desarrollar programas correctos.

C:

```
i = 0;
while ((i <= max) && (x[i] > 0)) do
```

ok

error de ejecución: ¿por qué?

Ada:

```
i := 1;
while (i <= max) and (x[i] > 0) loop
```

- El **and** de Ada se evalúa de forma completa (no se sabe el orden de evaluación.)
- C efectúa evaluación por corto circuito (se sabe el orden de evaluación.)
- Sabrás evitar construcciones que hagan que tus programas sean poco portables.

Compiladores II: Introducción

- Para todos los lenguajes hay buenos compiladores disponibles, y a bajo coste.
  - GNU (Free Software Foundation): gpc, gcc, gcl, g77, prolog

Premisa 2:

Es poco probable que te haga falta hacer un compilador (en todo caso después de cursar Compiladores II.)

Hecho 2:

La premisa es **cierta**; sin embargo, es posible que tengas que diseñar un lenguaje, e implementar un traductor.

- `lex` y `yacc` son herramientas de propósito general

Lenguajes de Programación

- Pascual a Código P

```
entero i;
principio
  elementos := 79;
  i := 1;
  mq i <= elementos
   colonia[i] := (i mod 20) = 0;
   i := i + 1;
  fmq
fin
```

```
    SRF  0  3
    ASGI
    JMP L1
L1:
    SRF  1 82
    STC 79
    ASG
...
```

- Java a JVM

```
void spin() {
int i;
for (i = 0; i < 100; i++) {
    ;    // Loop body is empty
    }
}
```

```
Method void spin()
 0 iconst_0
 1 istore_1
 2 goto 8
 5 iinc 1 1
 8 iload_1
 9 bipush 100
11 if_icmplt 5
return
```

Textures

- WORD es 'wysiwyg', Latex no lo es

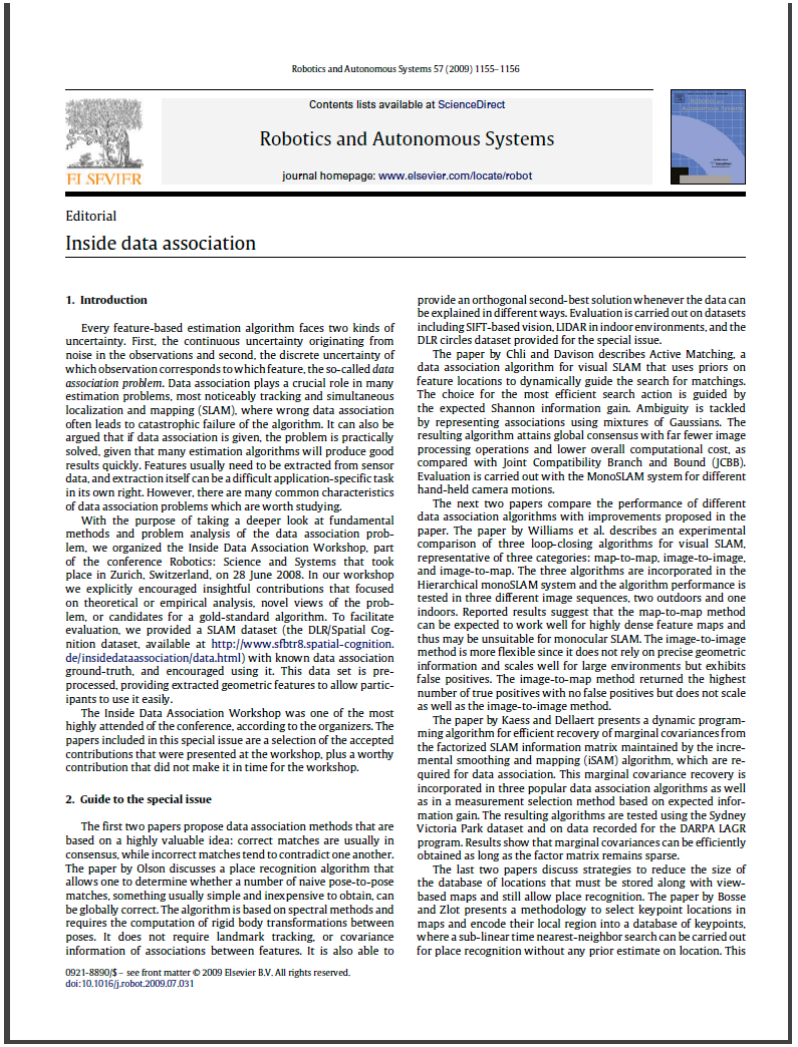
```
\documentclass[final,12pt]
{elsarticle}
\usepackage{amssymb}
\journal{Robotics and Autonomous
Systems}

\begin{document}
\begin{frontmatter}
\title{Editorial: Inside Data
Association}

\author{Udo Frese, Jos\'e Neira}
\address{}
\end{frontmatter}

\section{INTRODUCTION}
\label{section:introduction}

Every feature based estimation
algorithm faces two kinds of
uncertainty. First, the continuous
uncertainty originating from
noise in the observations and
second, the discrete uncertainty …
```



Robotics and Autonomous Systems 57 (2009) 1155–1156

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Editorial

Inside data association

1. Introduction

Every feature-based estimation algorithm faces two kinds of uncertainty. First, the continuous uncertainty originating from noise in the observations and second, the discrete uncertainty of which observation corresponds to which feature, the so-called *data association problem*. Data association plays a crucial role in many estimation problems, most noticeably tracking and simultaneous localization and mapping (SLAM), where wrong data association often leads to catastrophic failure of the algorithm. It can also be argued that if data association is given, the problem is practically solved, given that many estimation algorithms will produce good results quickly. Features usually need to be extracted from sensor data, and extraction itself can be a difficult application-specific task in its own right. However, there are many common characteristics of data association problems which are worth studying.

With the purpose of taking a deeper look at fundamental methods and problem analysis of the data association problem, we organized the Inside Data Association Workshop, part of the conference Robotics: Science and Systems that took place in Zurich, Switzerland, on 28 June 2008. In our workshop we explicitly encouraged insightful contributions that focused on theoretical or empirical analysis, novel views of the problem, or candidates for a gold-standard algorithm. To facilitate evaluation, we provided a SLAM dataset (the DLR/Spatial Cognition dataset, available at http://www.sfbtr8.spatial-cognition.de/insidedataassociation/data.html) with known data association ground-truth, and encouraged using it. This data set is pre-processed, providing extracted geometric features to allow participants to use it easily.

The Inside Data Association Workshop was one of the most highly attended of the conference, according to the organizers. The papers included in this special issue are a selection of the accepted contributions that were presented at the workshop, plus a worthy contribution that did not make it in time for the workshop.

2. Guide to the special issue

The first two papers propose data association methods that are based on a highly valuable idea: correct matches are usually in consensus, while incorrect matches tend to contradict one another. The paper by Olson discusses a place recognition algorithm that allows one to determine whether a number of naive pose-to-pose matches, something usually simple and inexpensive to obtain, can be globally correct. The algorithm is based on spectral methods and requires the computation of rigid body transformations between poses. It does not require landmark tracking, or covariance information of associations between features. It is also able to provide an orthogonal second-best solution whenever the data can be explained in different ways. Evaluation is carried out on datasets including SIFT-based vision, LIDAR in indoor environments, and the DLR circles dataset provided for the special issue.

The paper by Chli and Davison describes Active Matching, a data association algorithm for visual SLAM that uses priors on feature locations to dynamically guide the search for matchings. The choice for the most efficient search action is guided by the expected Shannon information gain. Ambiguity is tackled by representing associations using mixtures of Gaussians. The resulting algorithm attains global consensus with far fewer image processing operations and lower overall computational cost, as compared with Joint Compatibility Branch and Bound (JCBB). Evaluation is carried out with the MonoSLAM system for different hand-held camera motions.

The next two papers compare the performance of different data association algorithms with improvements proposed in the paper. The paper by Williams et al. describes an experimental comparison of three loop-closing algorithms for visual SLAM, representative of three categories: map-to-map, image-to-image, and image-to-map. The three algorithms are incorporated in the Hierarchical monoSLAM system and the algorithm performance is tested in three different image sequences, two outdoors and one indoors. Reported results suggest that the map-to-map method can be expected to work well for highly dense feature maps and thus may be unsuitable for monocular SLAM. The image-to-image method is more flexible since it does not rely on precise geometric information and scales well for large environments but exhibits false positives. The image-to-map method returned the highest number of true positives with no false positives but does not scale as well as the image-to-image method.

The paper by Kaess and Dellaert presents a dynamic programming algorithm for efficient recovery of marginal covariances from the factorized SLAM information matrix maintained by the incremental smoothing and mapping (iSAM) algorithm, which are required for data association. This marginal covariance recovery is incorporated in three popular data association algorithms as well as in a measurement selection method based on expected information gain. The resulting algorithms are tested using the Sydney Victoria Park dataset and on data recorded for the DARPA LAGR program. Results show that marginal covariances can be efficiently obtained as long as the factor matrix remains sparse.

The last two papers discuss strategies to reduce the size of the database of locations that must be stored along with view-based maps and still allow place recognition. The paper by Bosse and Zlot presents a methodology to select keypoint locations in maps and encode their local region into a database of keypoints, where a sub-linear time nearest-neighbor search can be carried out for place recognition without any prior estimate on location. This

doi:10.1016/j.robot.2009.07.031

Postscript (1985)

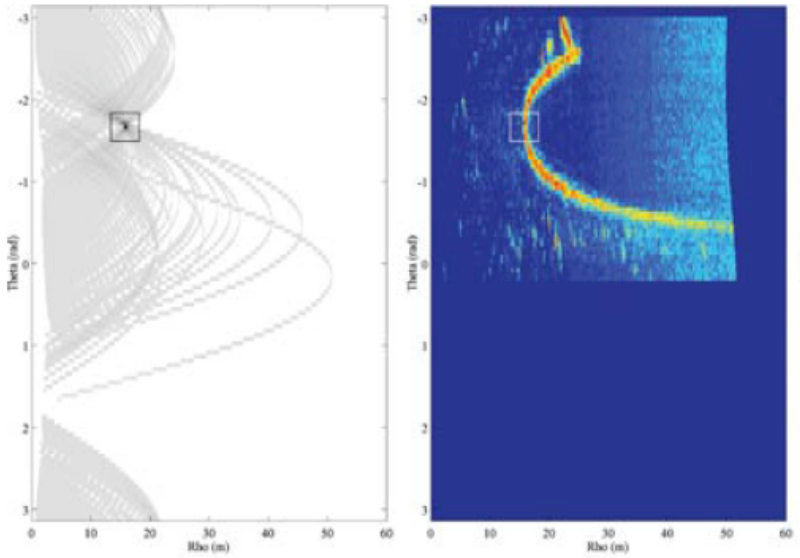
- Postscript es un lenguaje de descripción de páginas, ***independiente*** del dispositivo .

```
%!PS-Adobe-2.0 EPSF-1.2
%%Creator: conversor_postscript v1.0
%%Pages: 1
%%BoundingBox: -70 40 440 530
%%Este fichero ha sido generado para un A4,
%%resolucion de 72 dpi
%%EndComments

%%Page: 1 1

% trazado de los segmentos de rect
gsave
0 setlinecap
 1.0 setlinewidth
60.0 390.0 translate
0 rotate
 0.4  0.4 scale

60.0 -501.5 moveto
-52.0 -503.5 lineto
....
```



(a) (b)

HTML

- Lenguaje de publicación de hipertexto en la WWW

```
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=iso-8859-1">
   <META NAME="GENERATOR" CONTENT="Mozilla/4.04 [en] (X11; I; SunOS
5.5 sun4m) [
Netscape]">
   <TITLE>CPS Home Page</TITLE>
</HEAD>
<BODY TEXT="#000000" BACKGROUND="paper.jp
 
<CENTER><TABLE BORDER=5 CELLSPACING=6 CEI
<TR>
<TD><IMG SRC="cps3D.gif" BORDER=0 HEIGHT=
</TR>
</TABLE></CENTER>

<CENTER> </CENTER>

<CENTER>
<H1>
Centro Polit&eacute;cnico Superior</H1></
```

Compiladores II: Introducción

Hecho 3:

La Teoría de Compiladores es una de las más sólidas de la Informática.

- Desde las máquinas de Turing, ha contribuido con algunos de los resultados teóricos más importantes.

Old St.	Read Sym.		Wr. Sym.	Mv.	New St.
s1	1	->	0	R	s2
s2	1	->	1	R	s2
s2	0	->	0	R	s3
s3	0	->	1	L	s4
s3	1	->	1	R	s3
s4	1	->	1	L	s4
s4	0	->	0	L	s5
s5	1	->	1	L	s5
s5	0	->	1	R	s1

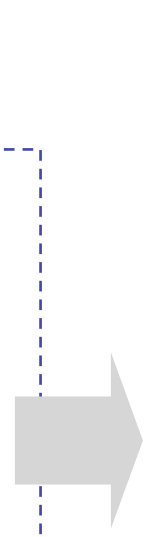
Step	State	Tape	Step	State	Tape
1	s1	**11**	**9**	**s2**	10**01**
2	s2	0**1**	10	s3	100**1**
3	s2	01**0**	**11**	**s3**	1001**0**
4	s3	010**0**	12	s4	100**11**
5	s4	01**01**	13	s4	10**011**
6	s5	0**101**	14	s5	1**0011**
7	s5	**0101**	15	s1	11**011**
8	s1	1**101**	-- halt --		

- Los ordenadores actuales son conceptualmente máquinas de Turing.

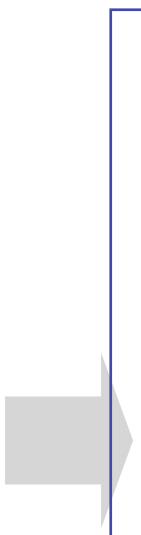
Compiladores II: Introducción

- La teoría de los lenguajes formales ha impulsado el desarrollo de la Informática
- Hace posible expresar las instrucciones en términos de mucho mayor nivel que las máquinas

```
int expr(int n)
{
    int d, l;
    l = 5 * n;
    d = 4 * n * n
      * (n + 1) * (n + 1);
    return d;
}
```



```
    lda $30,-32($30)
    stq $26,0($30)
    stq $15,8($30)
    bis $30,$30,$15
    bis $16,$16,$1
    stl $1,16($15)
    lds $f1,16($15)
    sts $f1,24($15)
    ldl $5,24($15)
    bis $5,$5,$2
    s4addq $2,0,$3
    ldl $4,16($15)
    mull $4,$3,$2
    ldl $3,16($15)
    addq $3,1,$4
    mull $2,$4,$2
    ldl $3,16($15)
    addq $3,1,$4
    mull $2,$4,$2
    stl $2,20($15)
    ldl $0,20($15)
    br $31,$33
$33:
    bis $15,$15,$30
    ldq $26,0($30)
    ldq $15,8($30)
    addq $30,32,$30
    ret $31,($26),1
```



```
011101010
000011010
110101010
101101110
110101010
101010101
011111000
010101010
101010101
001110101
010101010
101010100
101010111
111111111
111101010
101010110
100101011
001110101
110110100
011010101
010101101
010110110
```

Compiladores II: Introducción

- Un compilador ***mejora*** el programa
- Tiene un efecto práctico claro

```
int expr(int n)
{
    int d, l;
    l = 5 * n;
    d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```

```
    lda $30,-32($30)
    stq $26,0($30)
    stq $15,8($30)
    bis $30,$30,$15
    bis $16,$16,$1
    stl $1,16($15)
    lds $f1,16($15)
    sts $f1,24($15)
    ldl $5,24($15)
    bis $5,$5,$2
    s4addq $2,0,$3
    ldl $4,16($15)
    mull $4,$3,$2
    ldl $3,16($15)
    addq $3,1,$4
```

```
    mull $2,$4,$2
    ldl $3,16($15)
    addq $3,1,$4
    mull $2,$4,$2
    stl $2,20($15)
    ldl $0,20($15)
    br $31,$33
$33:
    bis $15,$15,$30
    ldq $26,0($30)
    ldq $15,8($30)
    addq $30,32,$30
    ret $31,($26),1
```

```
s4addq $16,0,$0
mull $16,$0,$0
addq $16,1,$16
mull $0,$16,$0
mull $0,$16,$0
ret $31,($26),1
```

Compiladores II: Introducción

- Ha hecho aportaciones a otras áreas del conocimiento (y también ha recibido aportaciones de tales áreas).

- Jerarquía de gramáticas N. Chomsky:

  - Tipo 0: recursivamente enumerables, producciones sin restricciones
  - Tipo 1: sensible al contexto $\alpha A \beta \rightarrow \alpha \gamma \beta$ — A. semántico
  - Tipo 2: indep. del contexto $A \rightarrow \gamma$ — A. sintáctico
  - Tipo 3: regulares $A \rightarrow aB$ $A \rightarrow a$ — A. léxico