

Domain-Specific Languages

Mathieu Acher

Maître de Conférences

mathieu.acher@irisa.fr

Material

[https://github.com/FAMILIAR-project/
HackOurLanguages-SIF](https://github.com/FAMILIAR-project/HackOurLanguages-SIF)

Plan

- Domain-Specific Languages (DSLs)
 - Languages and abstraction gap
 - Examples and rationale
 - DSLs vs General purpose languages, taxonomy
- External DSLs
 - Grammar and parsing
 - Language workbenches, Xtext
- DSLs, DSMLs, and (meta-)modeling

Contract

- Better understanding/source of inspiration of software languages and DSLs
 - Revisit of history and existing languages
- Foundations and practice of Xtext
 - State-of-the-art language workbench (Most Innovative Eclipse Project in 2010, mature and used in a variety of industries)
- Models and Languages
 - Perhaps a more concrete way to see models, metamodels and MDE

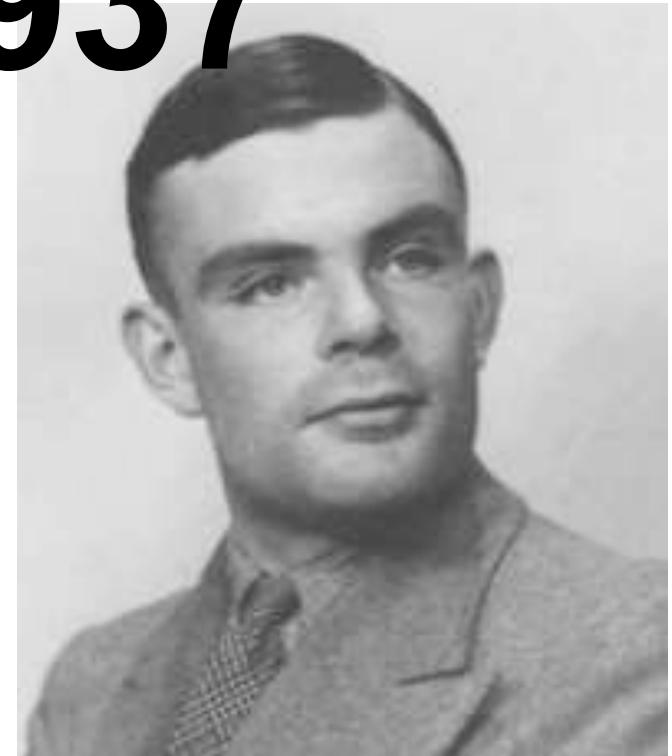
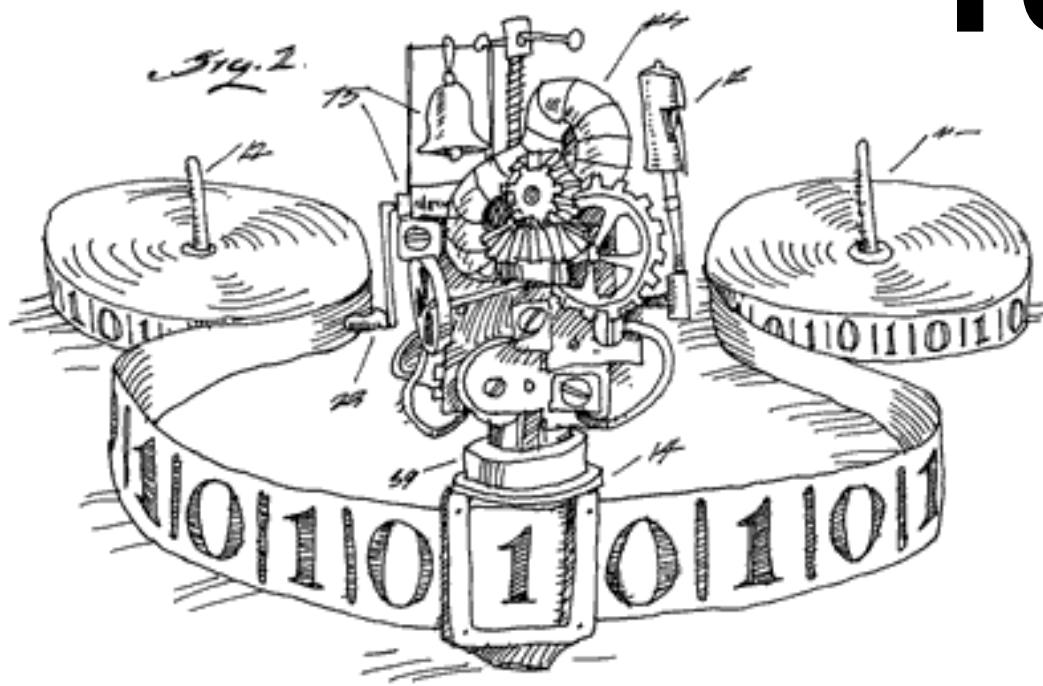
What are DSLs

Where are DSLs

Why DSLs (will) matter

The (Hi)Story of Software Engineering / Computer Science

1937

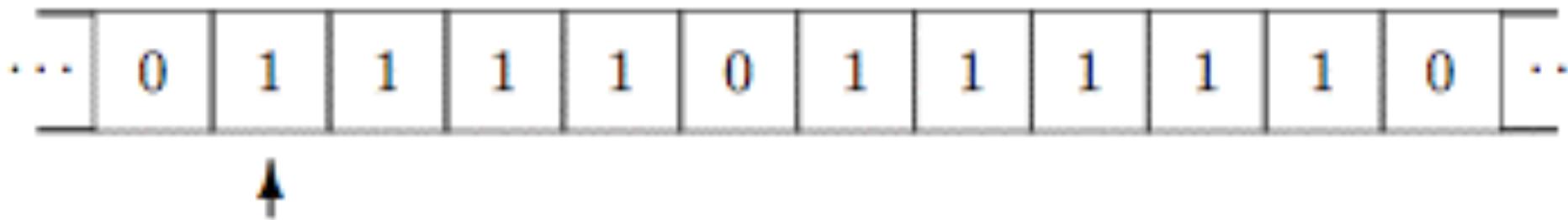


Turing Machine

- Infinite tape divided into Cells (0 or 1)
- Read-Write Head
- Transition rules

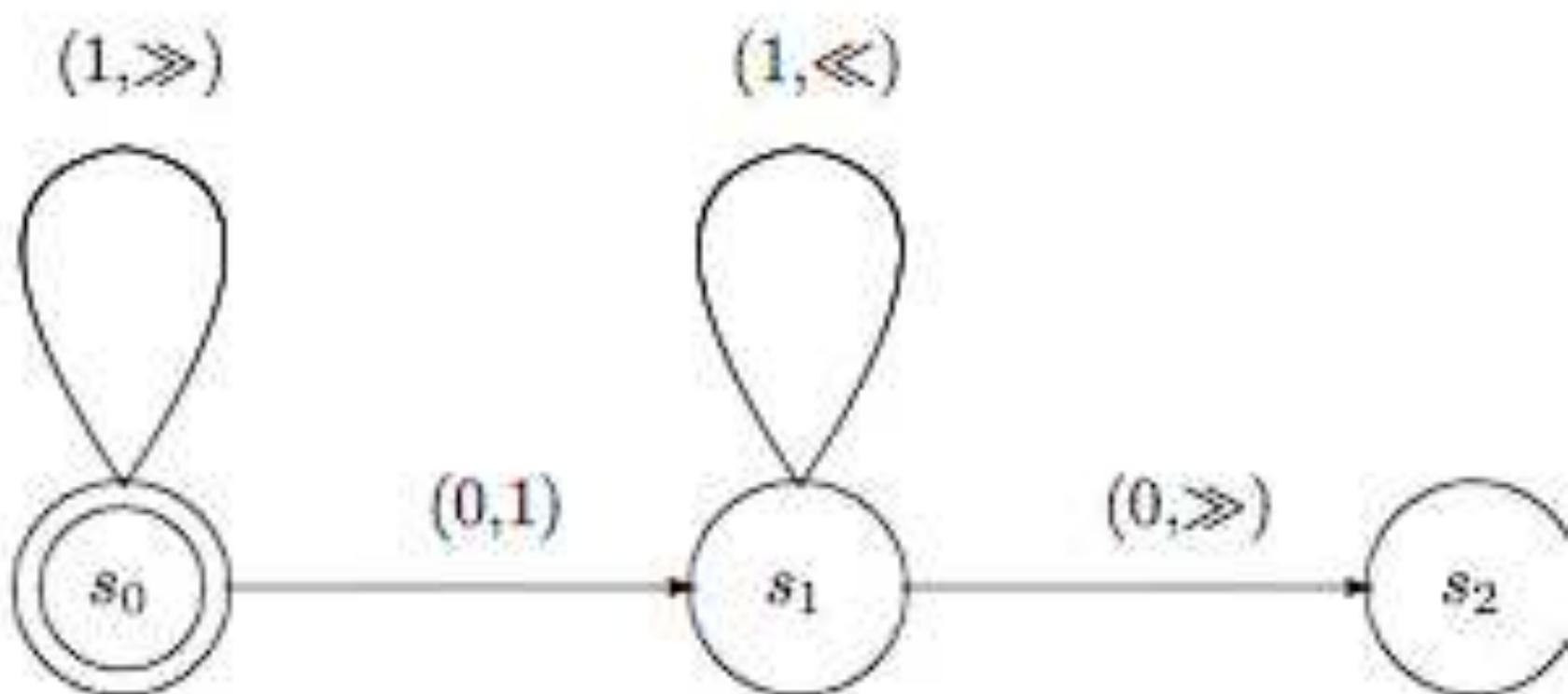
**Write a symbol
or move to left (>>) or right
(<<)**

< State_{current}, Symbol, State_{next}, Action >



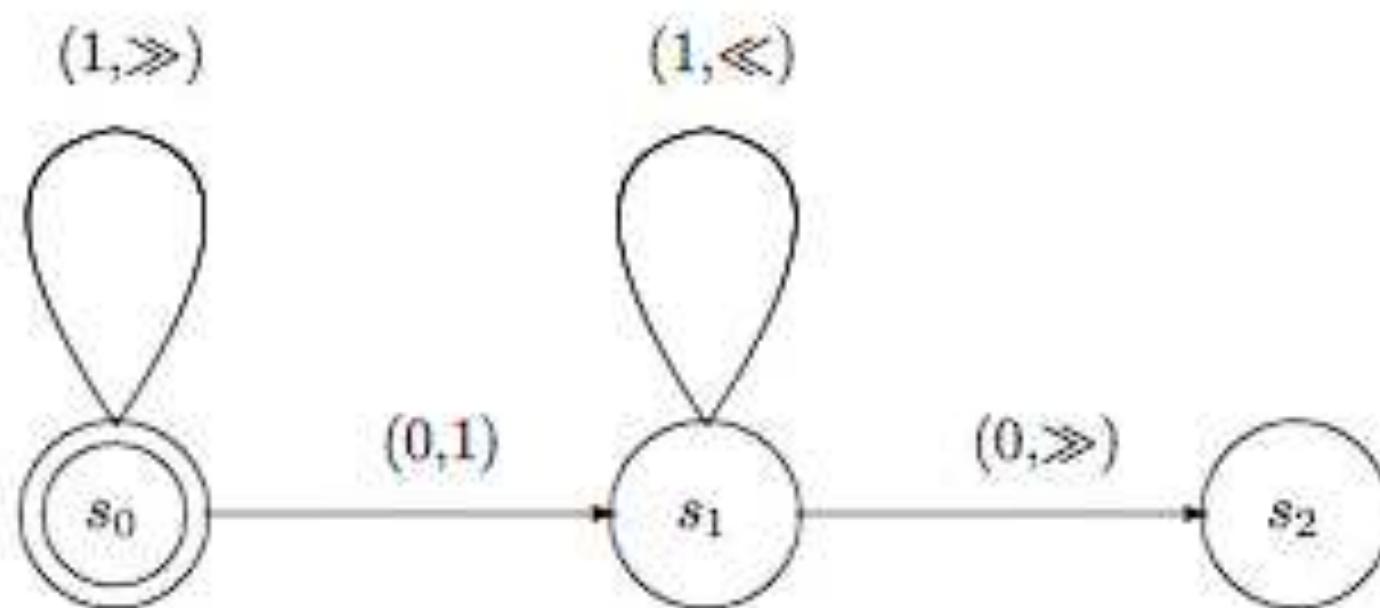
Turing Machine

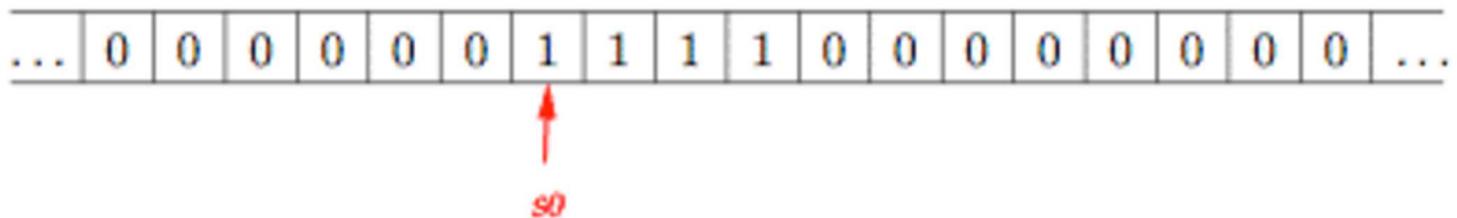
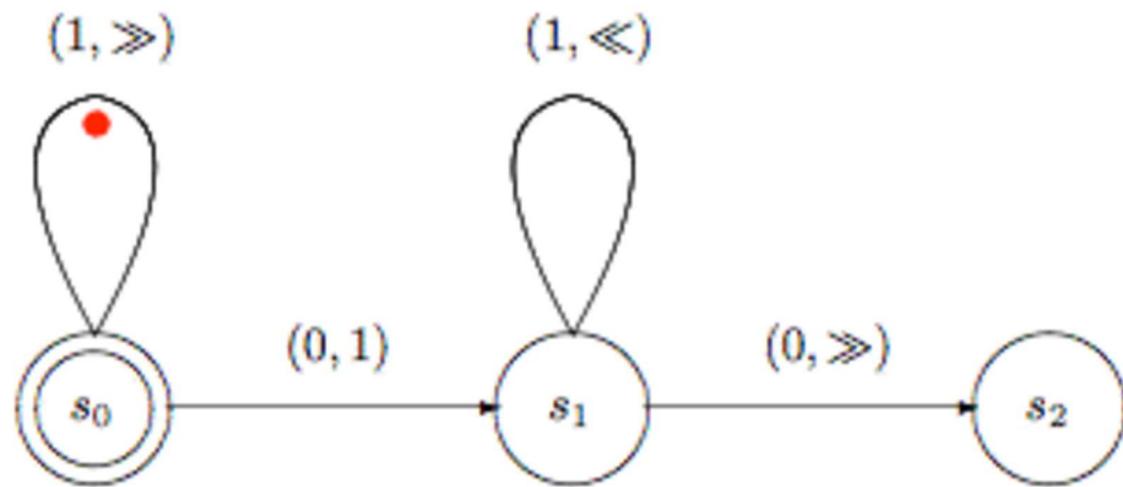
~ kind of state machine

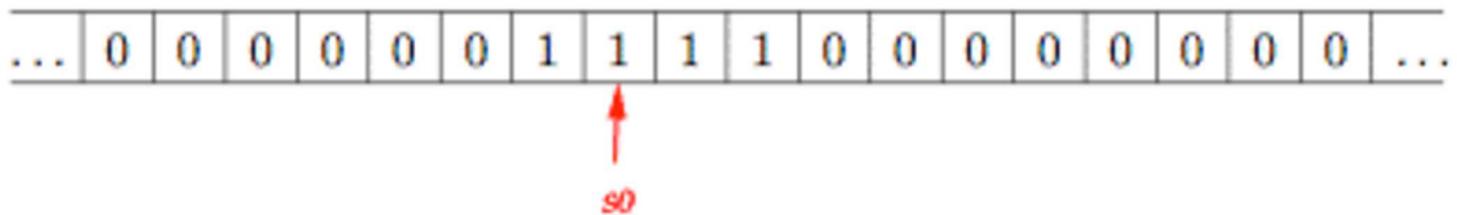
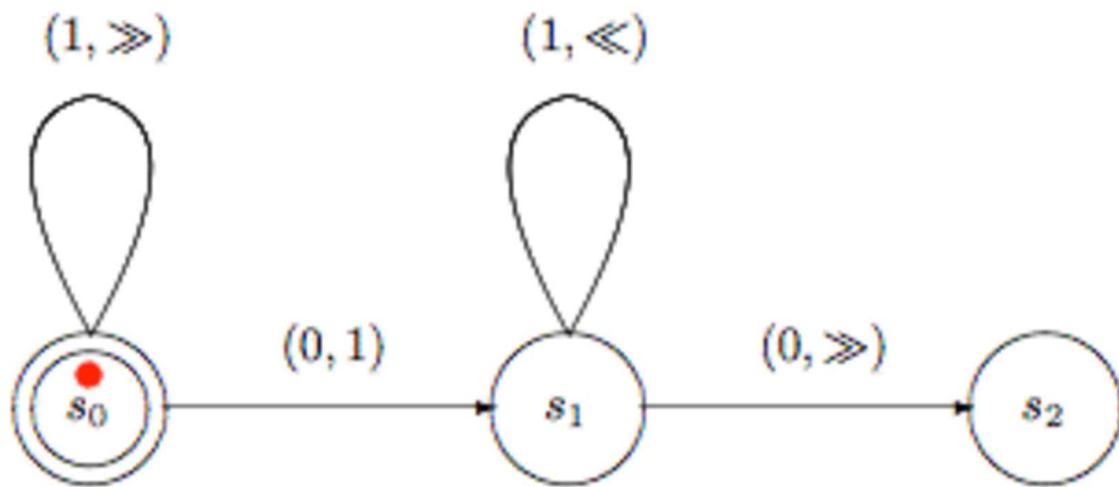


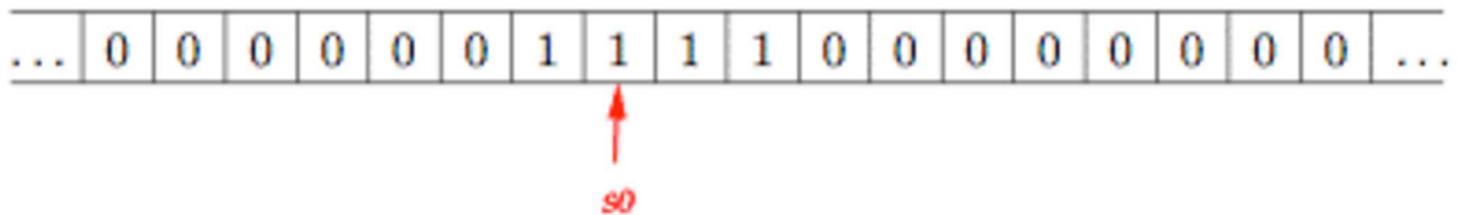
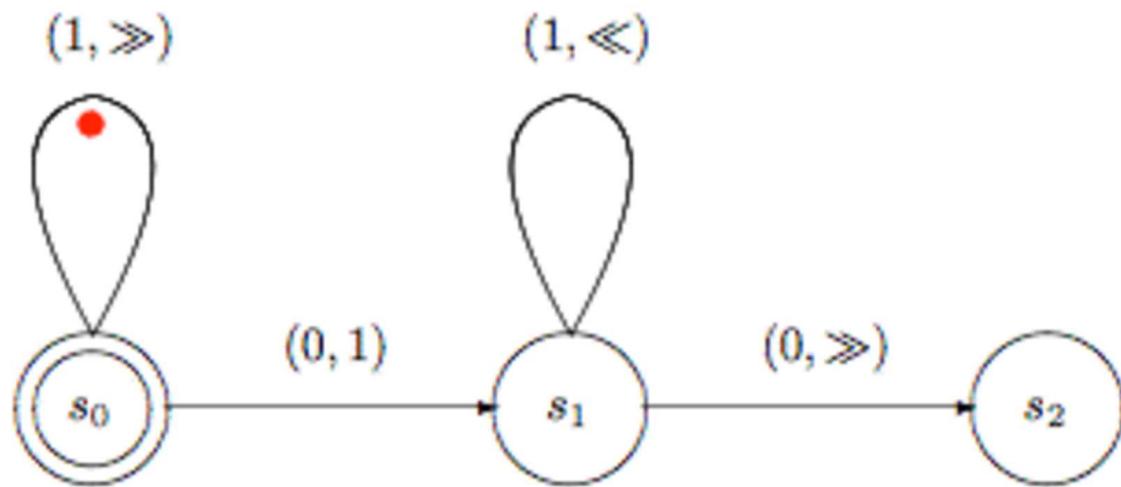
Quizz Time

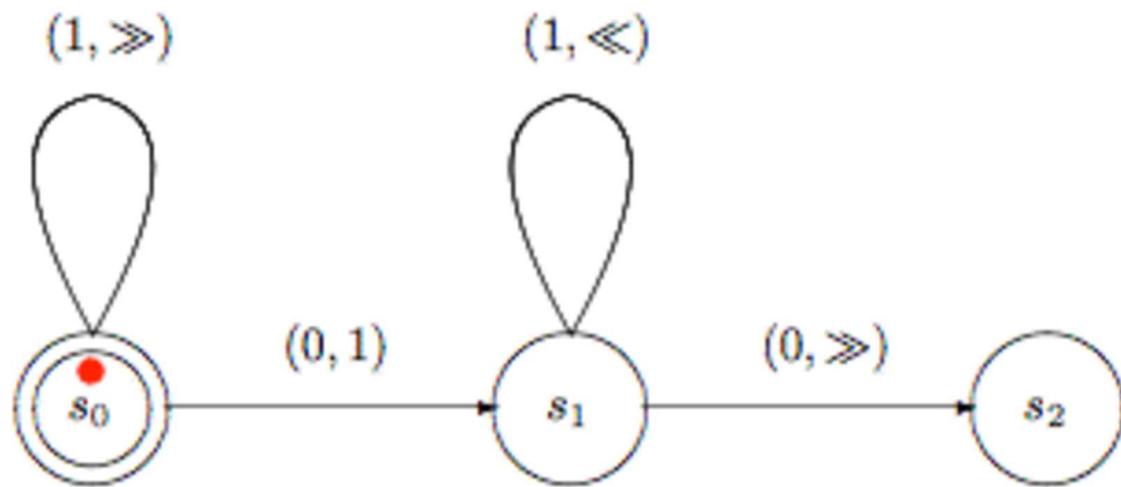
Question: what does it compute?

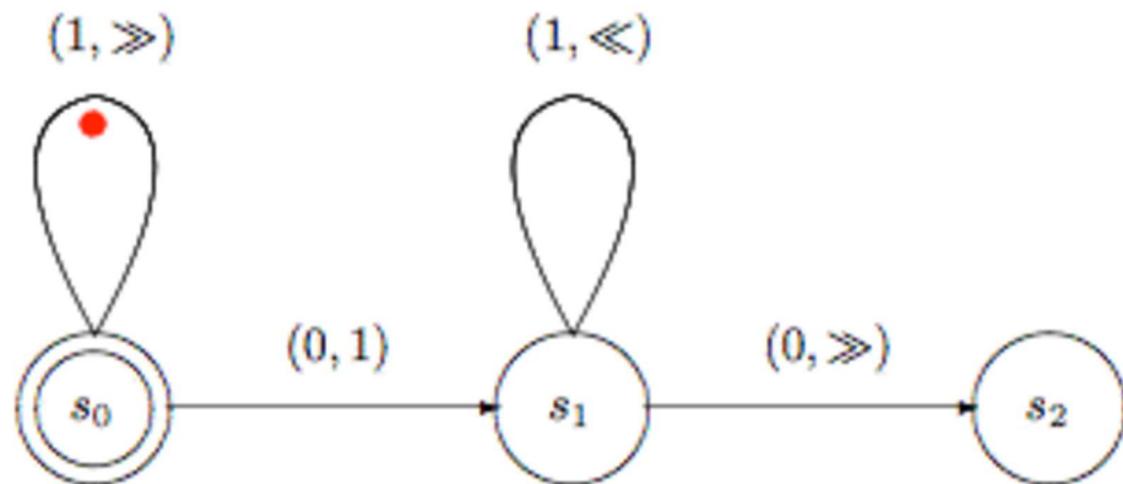


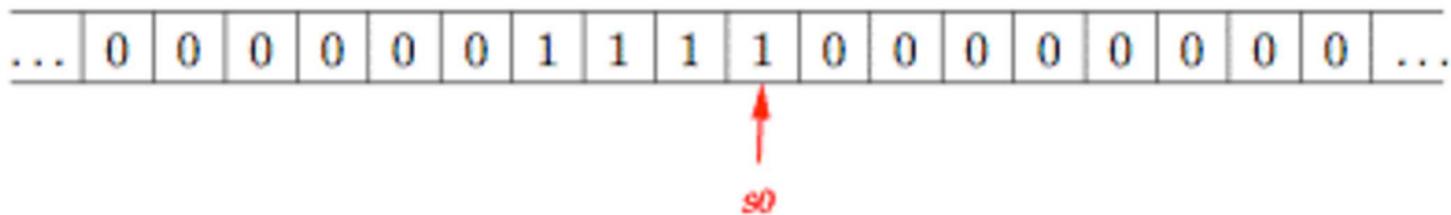
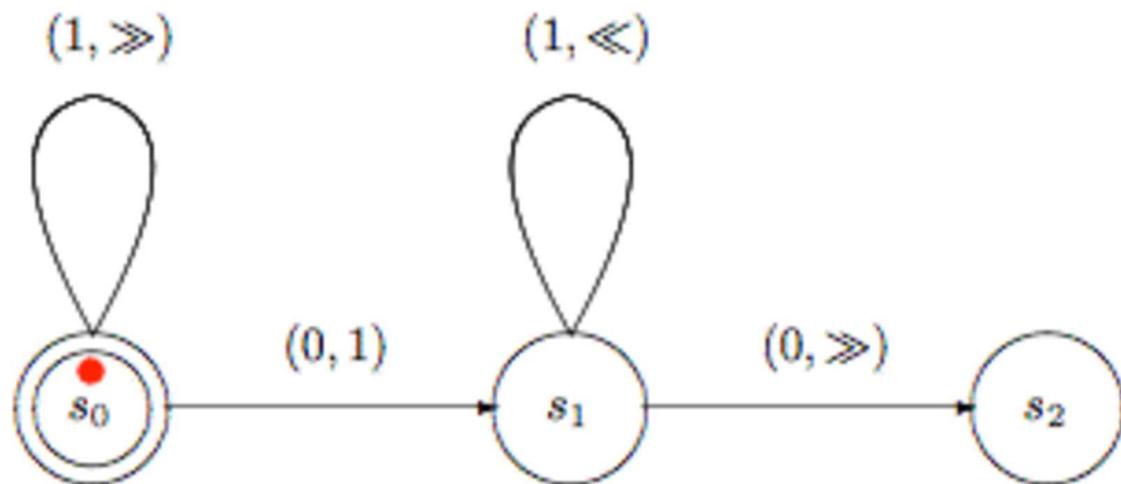


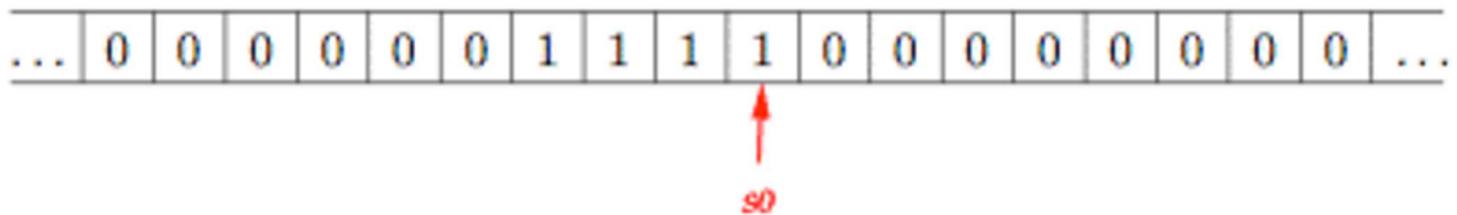
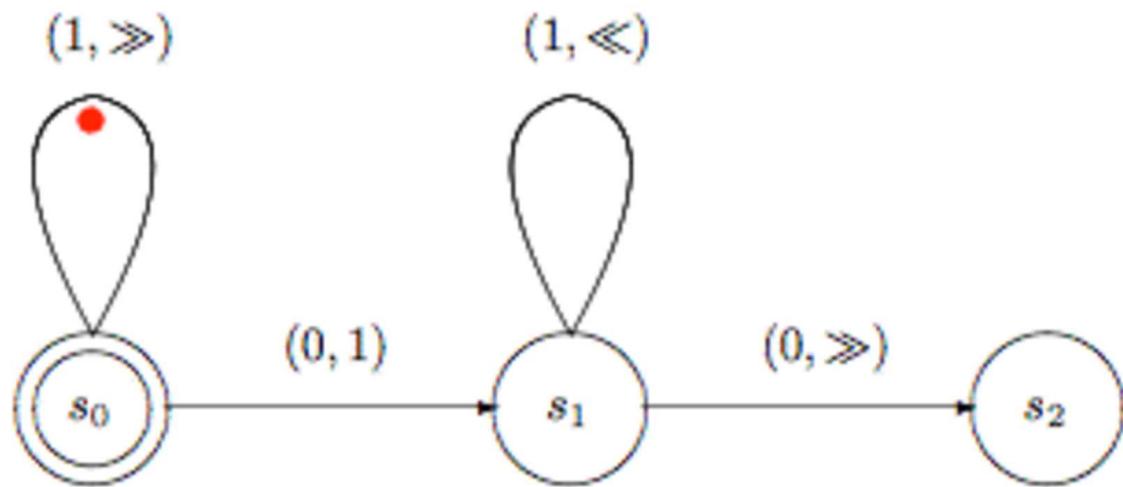


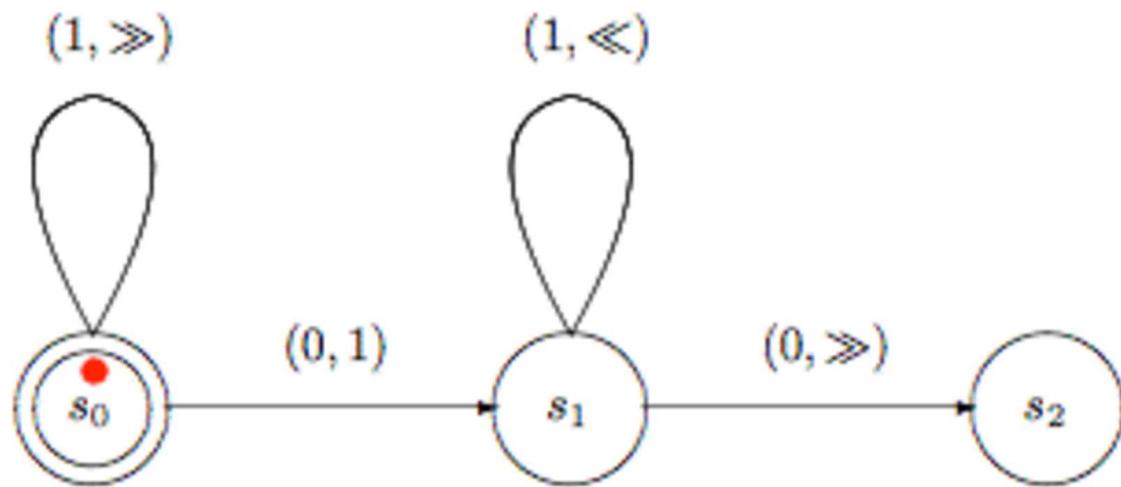


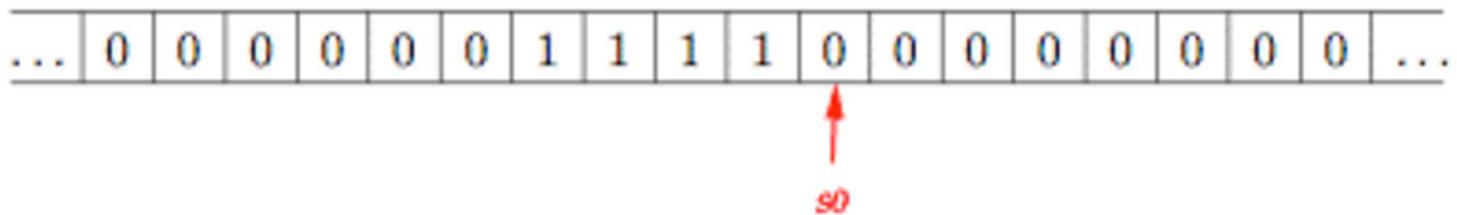
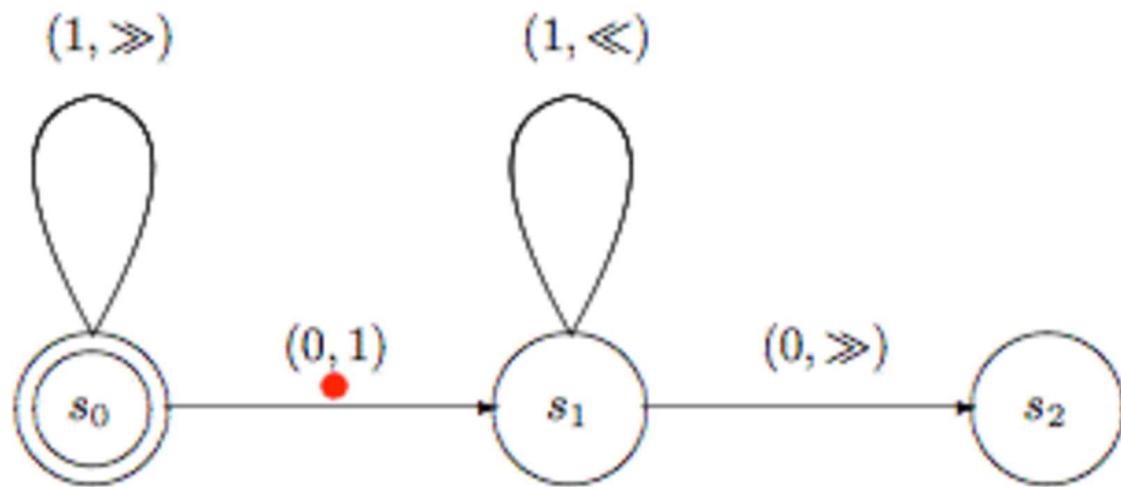


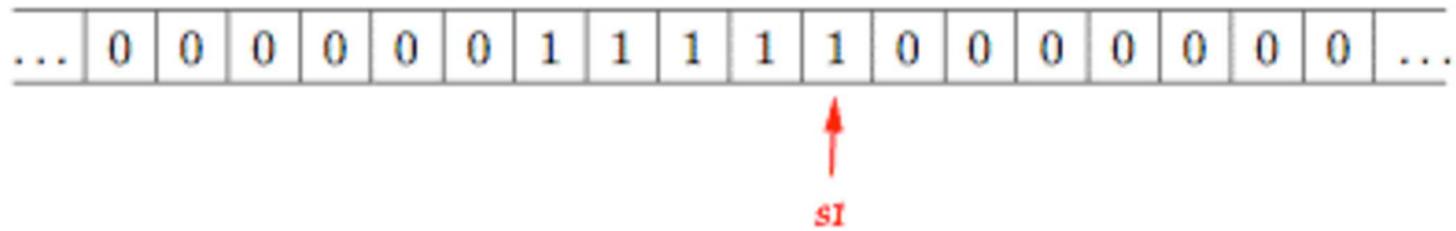
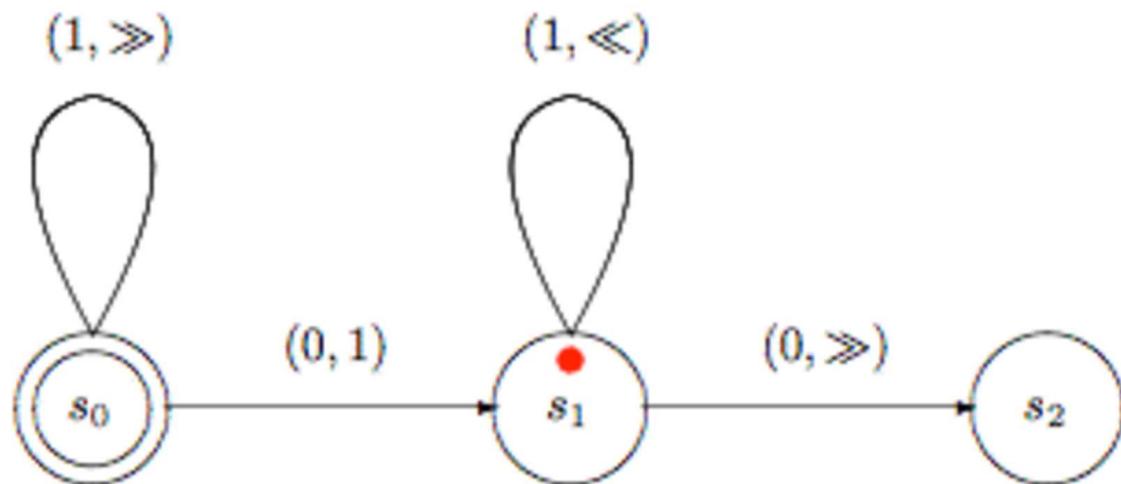


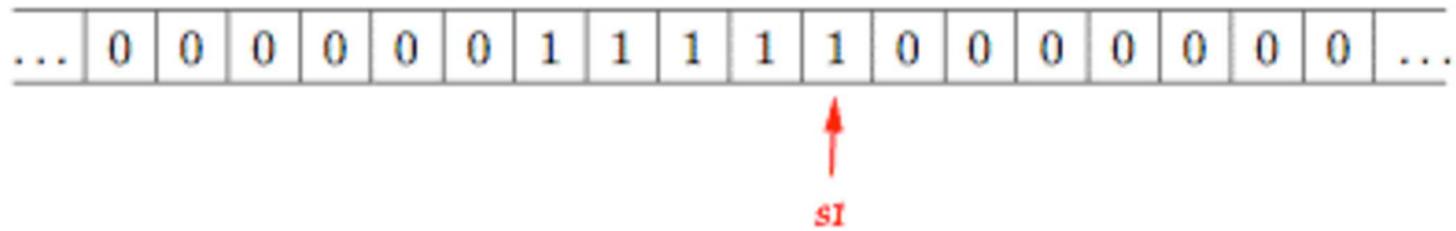
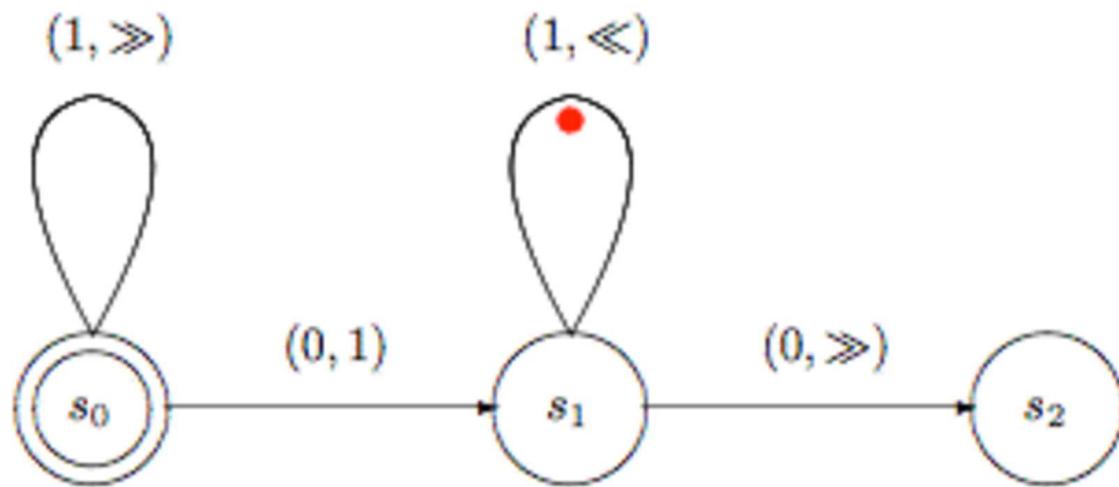


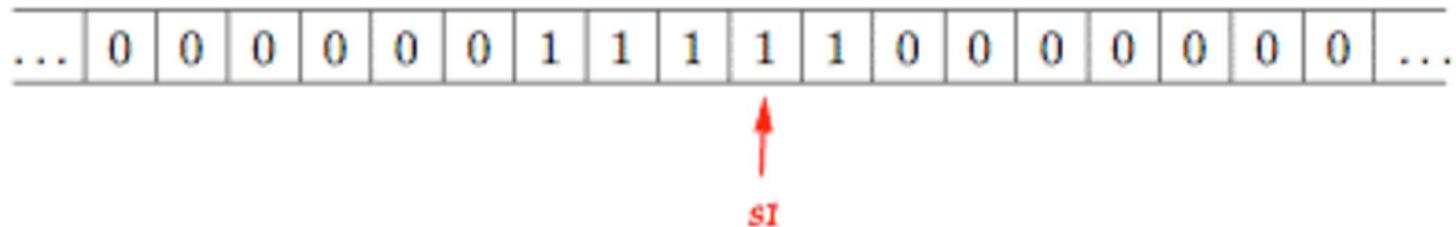
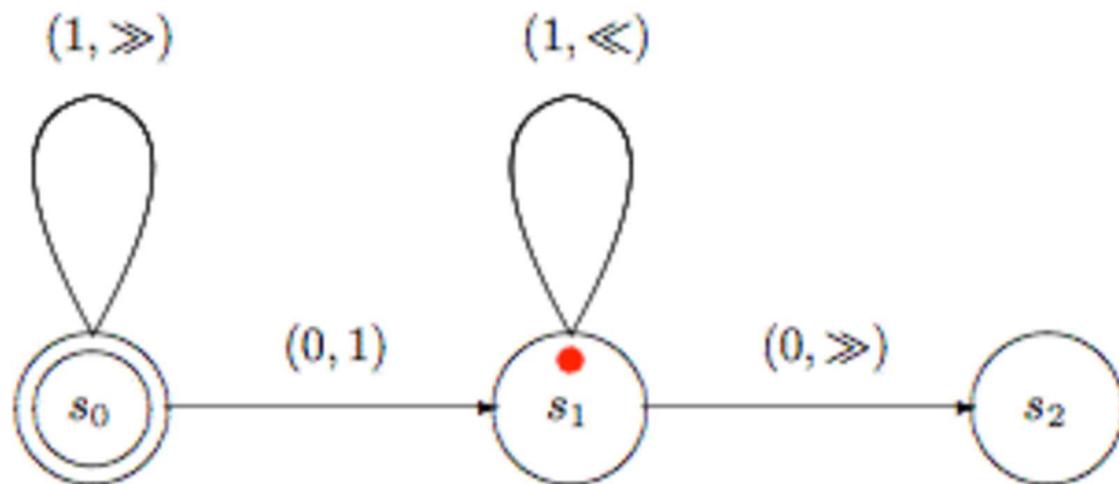


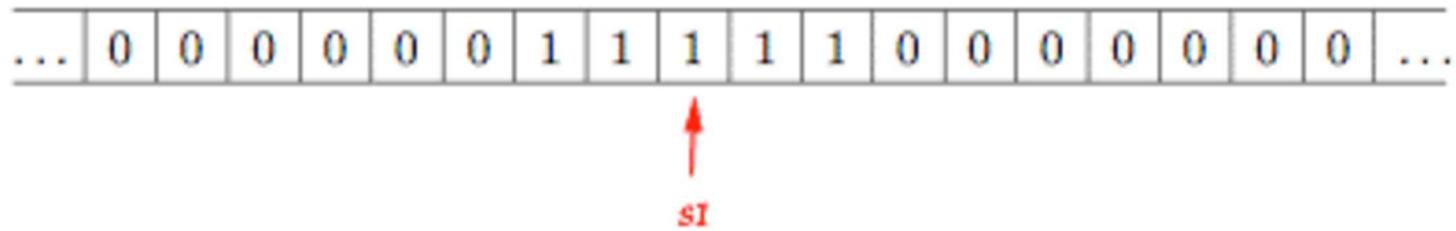
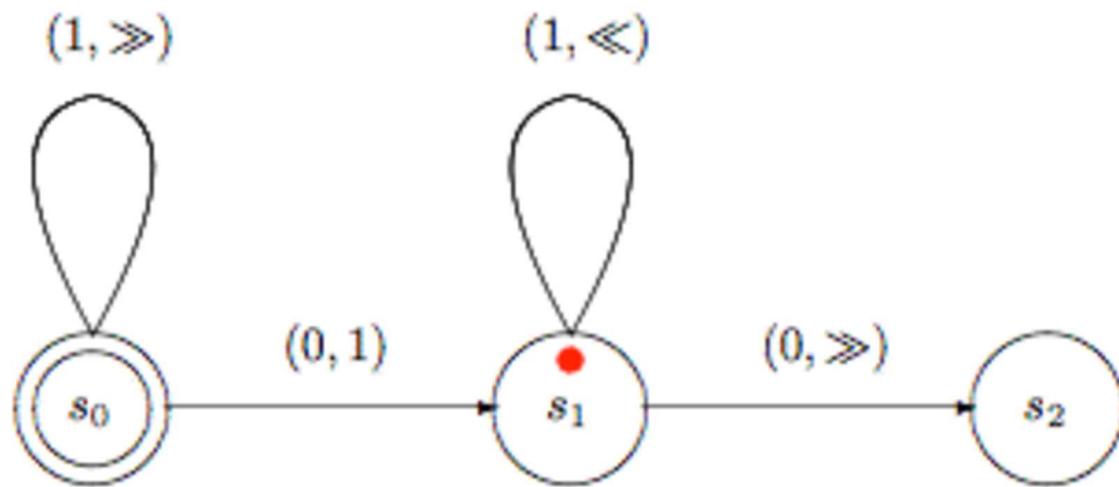


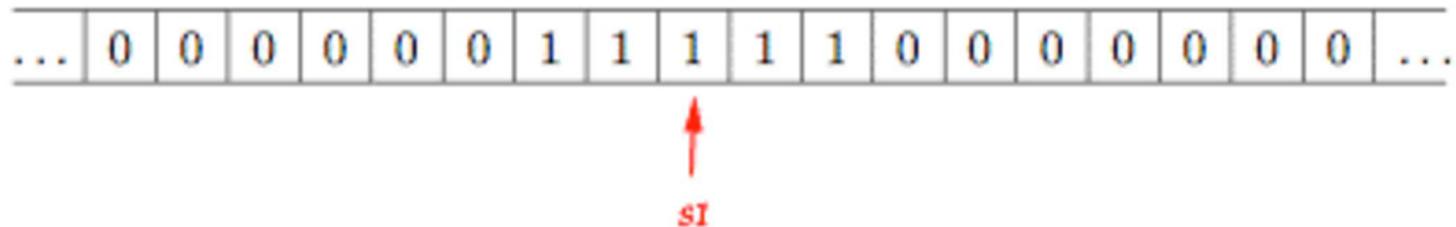
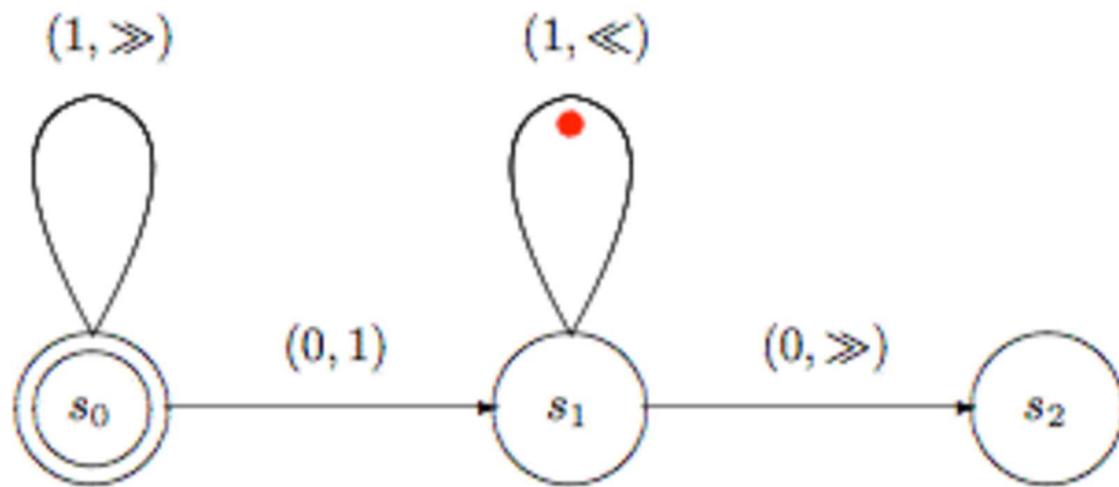


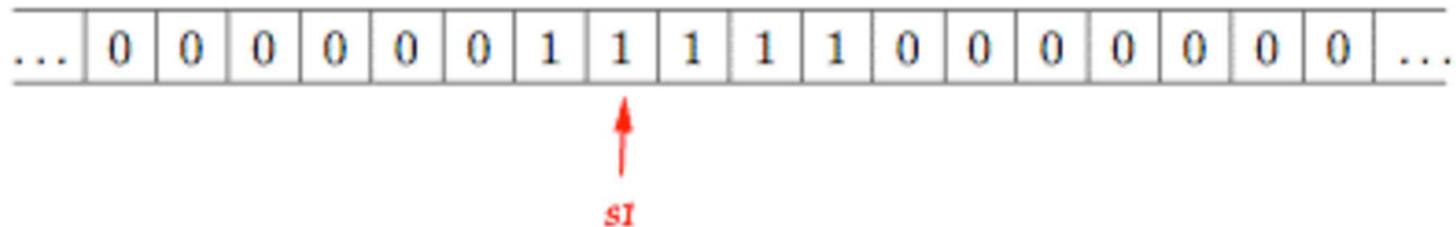
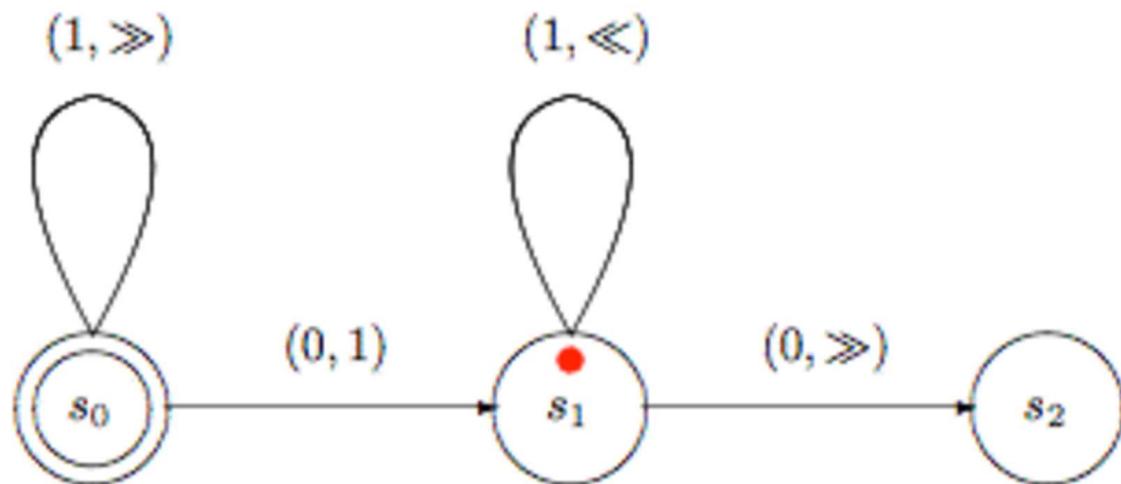


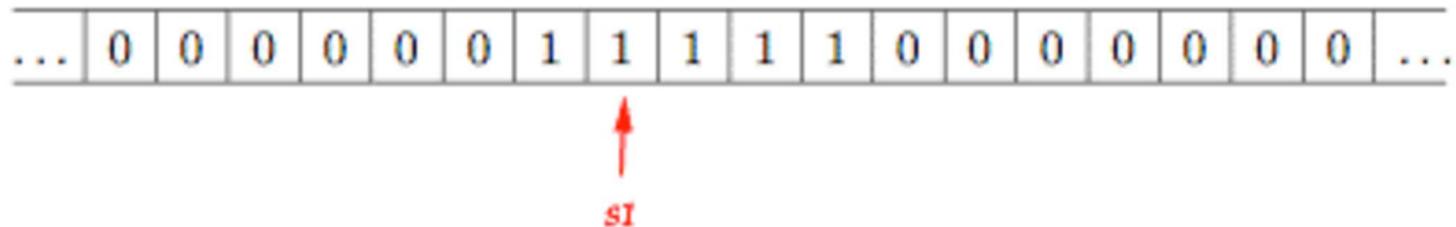
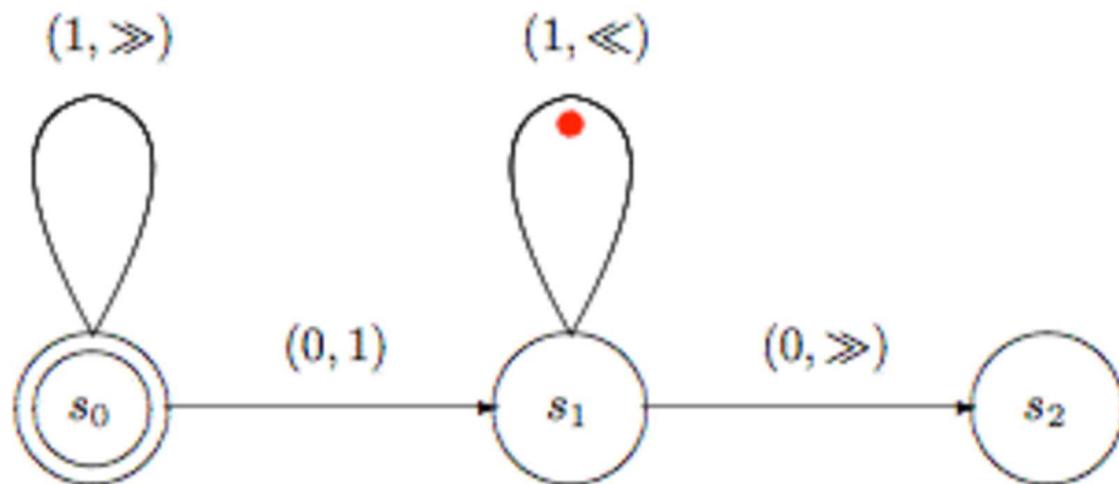


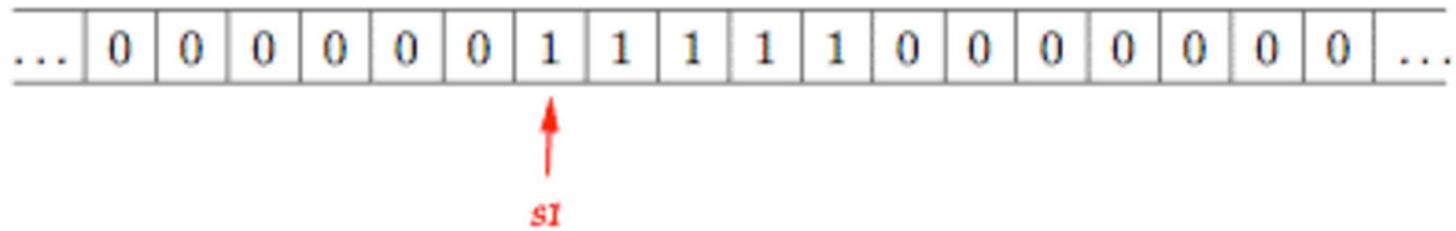
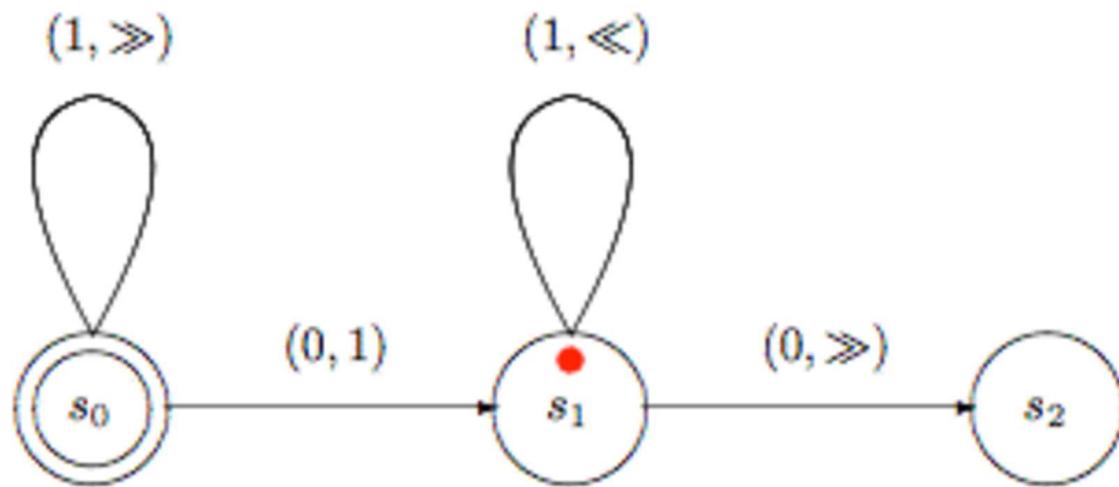


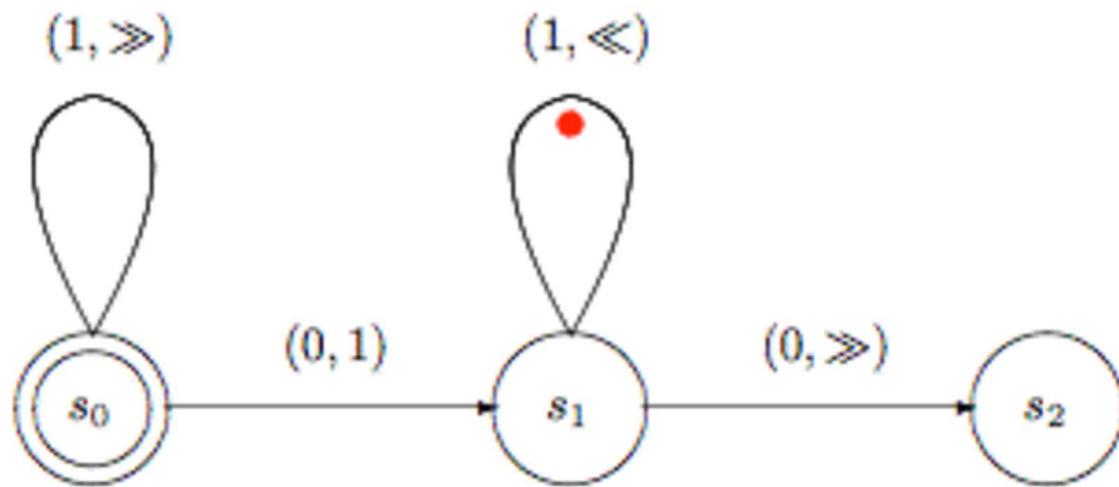






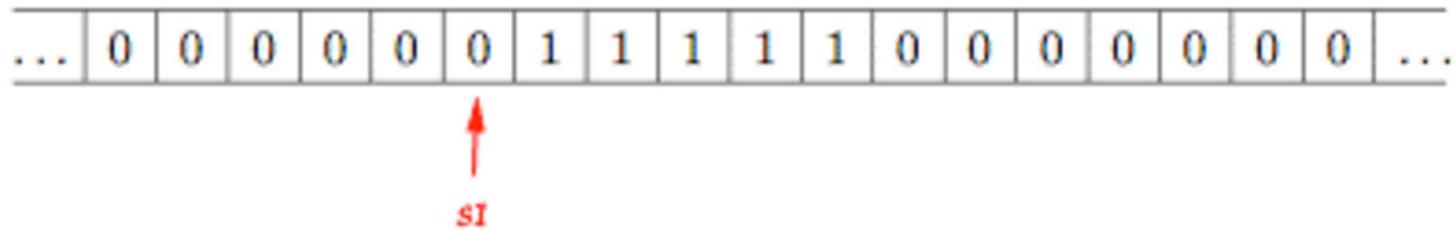
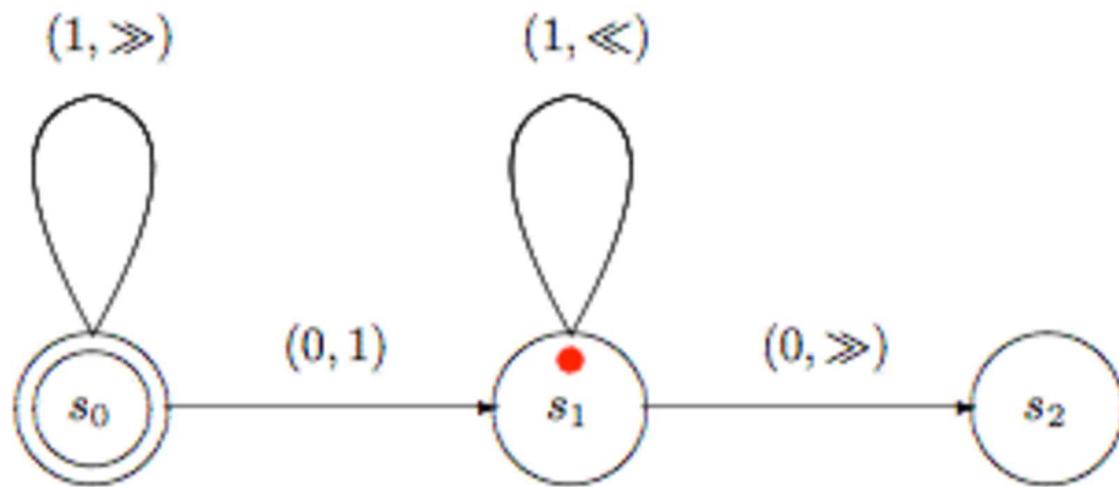


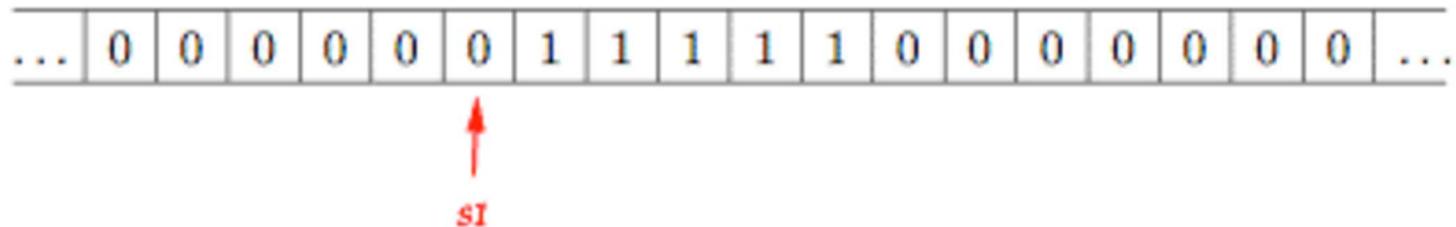
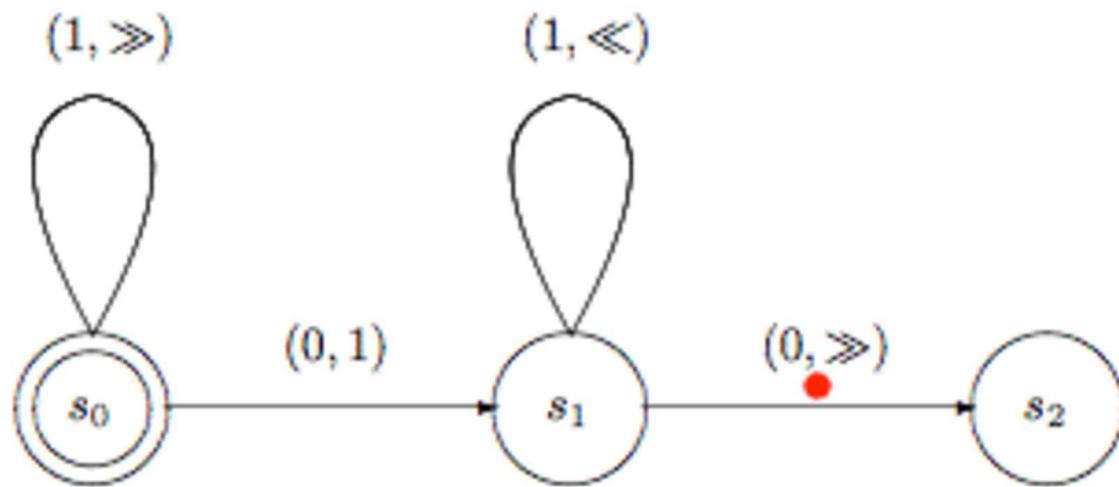


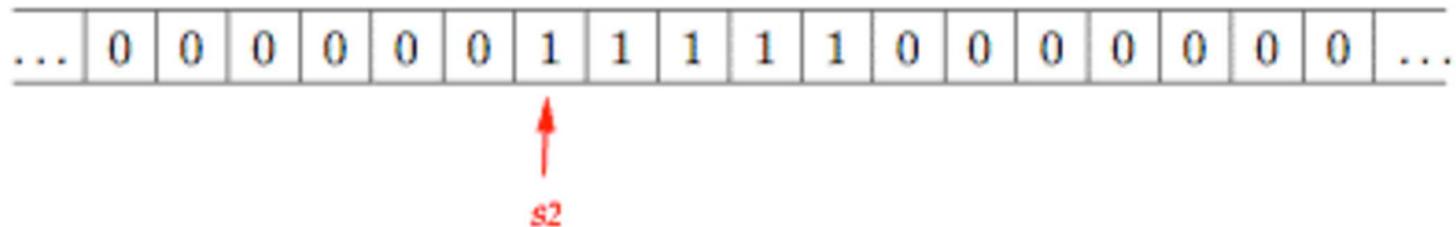
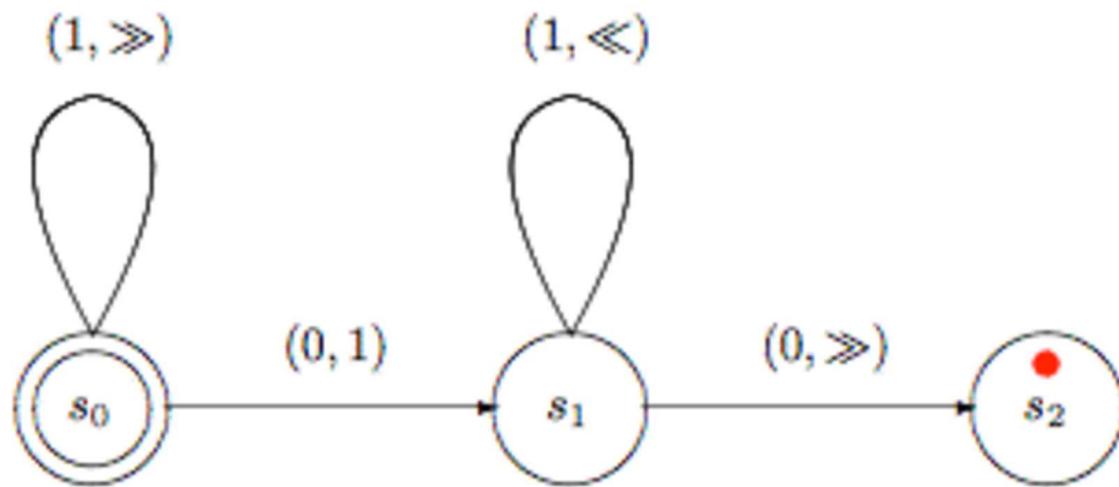


... 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 ...

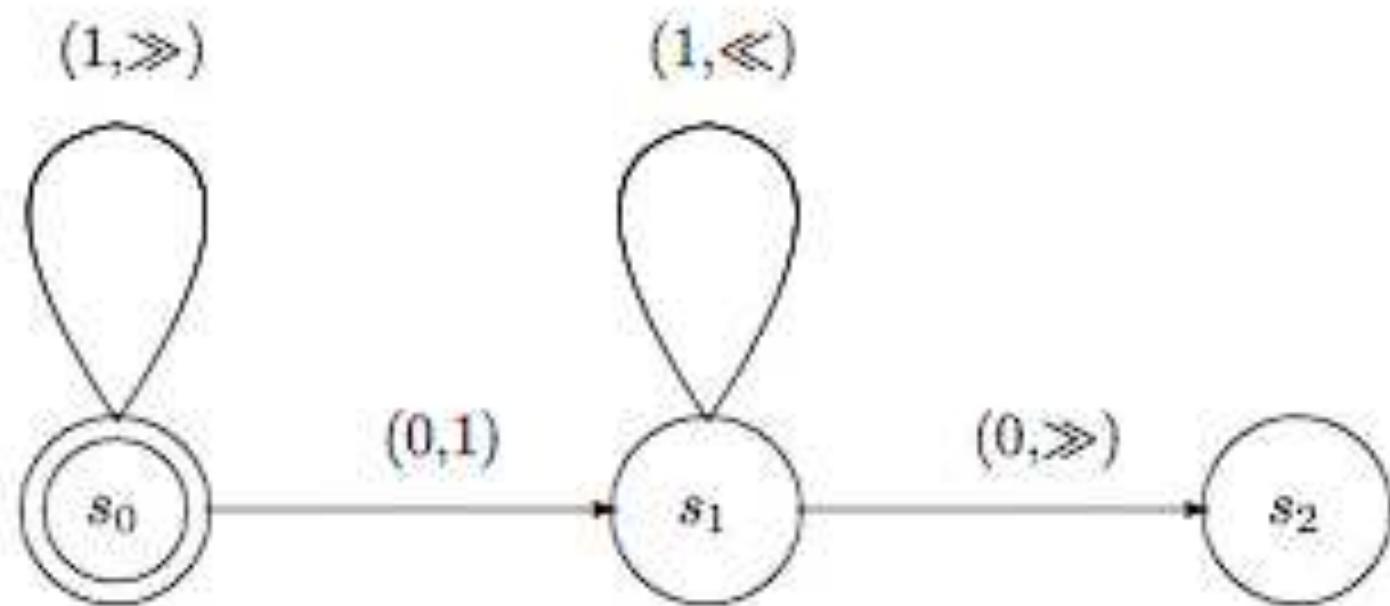
s1

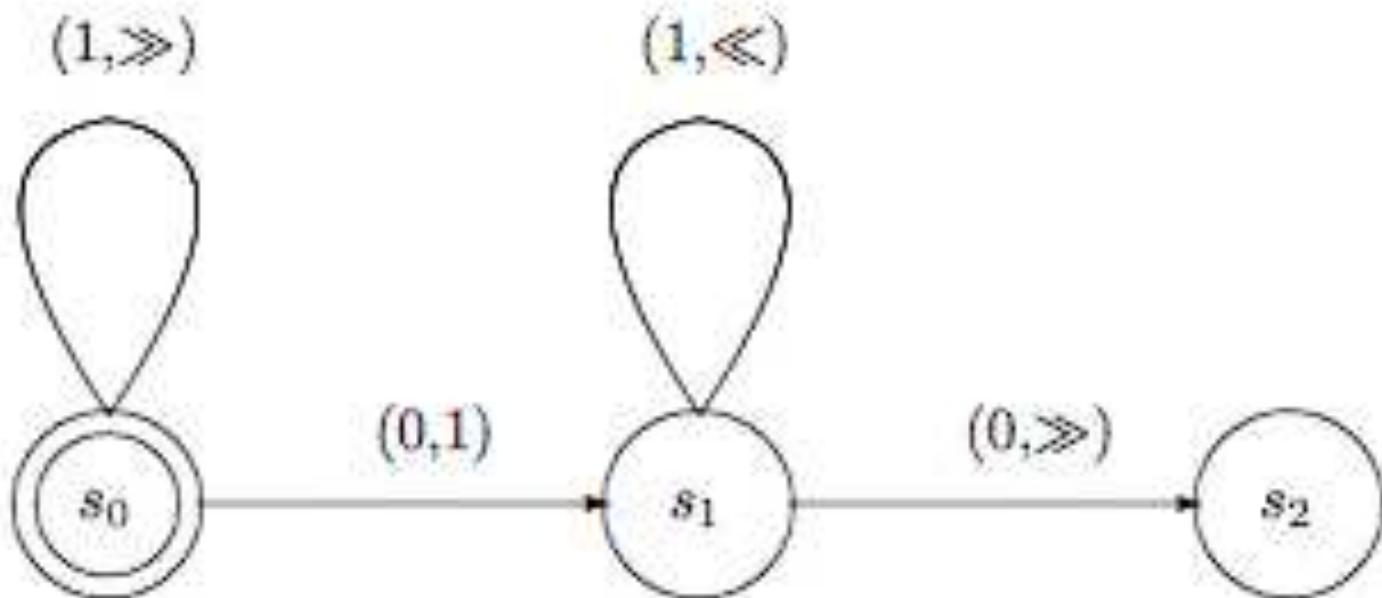






Question: what does it compute?



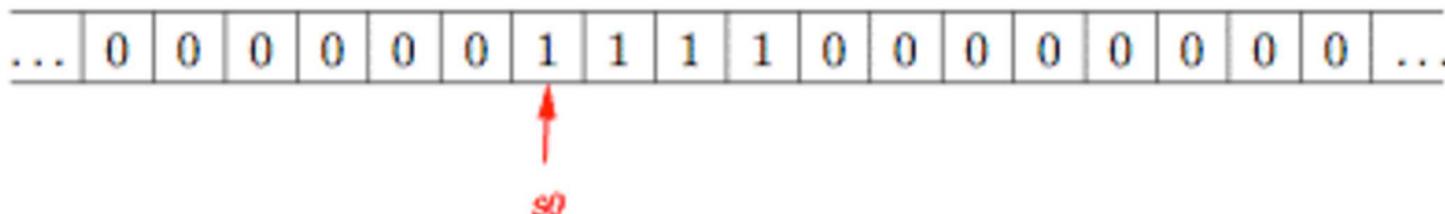
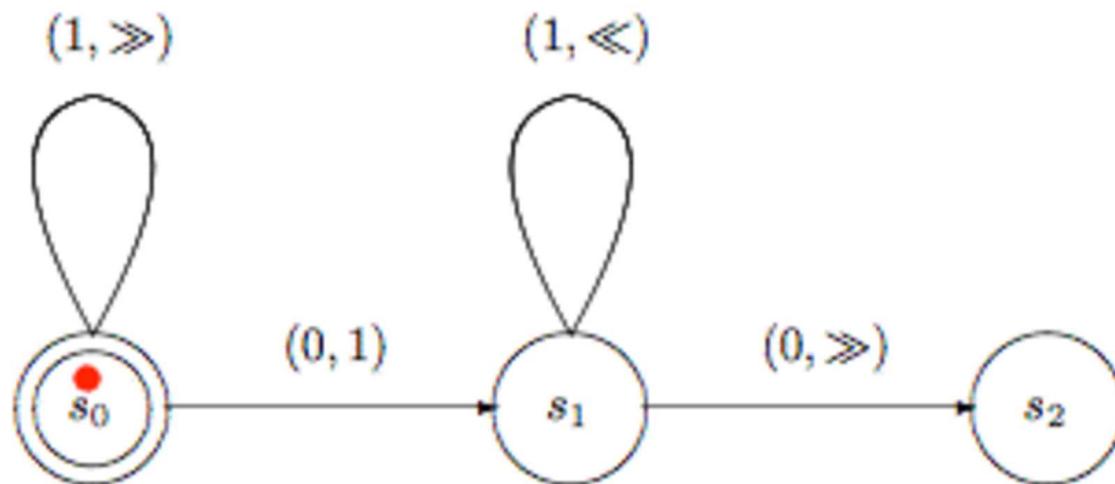


```
function succ (n) {  
    return n + 1;  
}
```

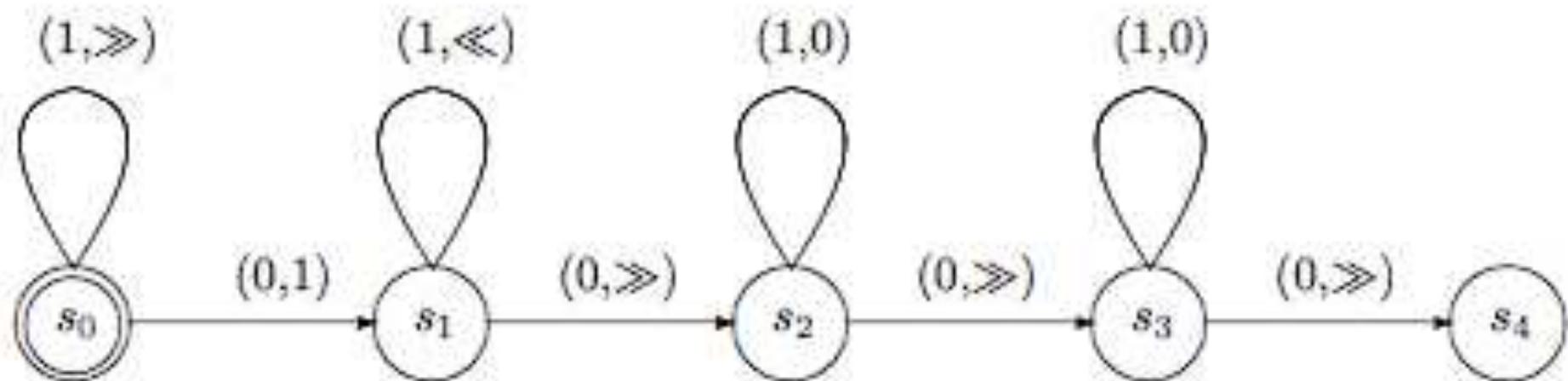
```
(lambda (x) (+ x 1))
```

Successor (add-one) function

assuming that number n as a block of $n+1$ copies of the symbol '1' on the tape (here, $n=3$)



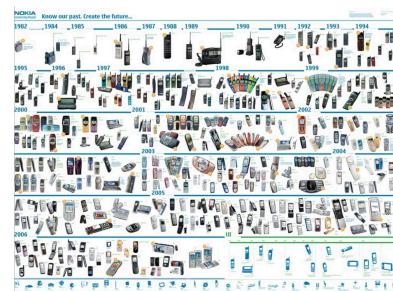
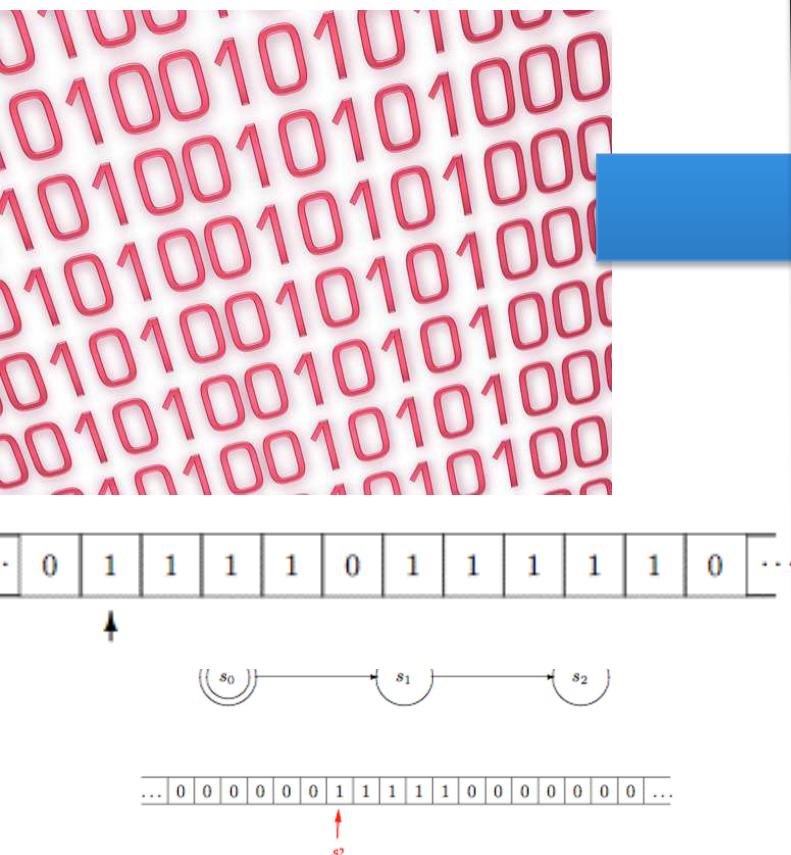
Addition of n+m



<http://graphics.stanford.edu/~seander/bithacks.html>

Maybe you prefer to use bit operations?

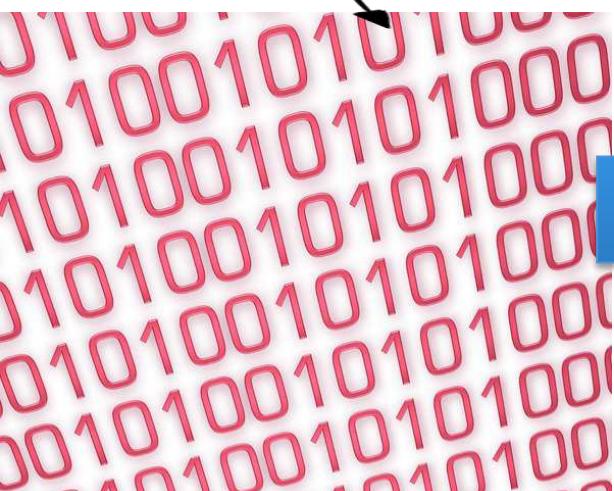
The (Hi)Story of Software Engineering Computer Science



Google
twitter



Software Languages



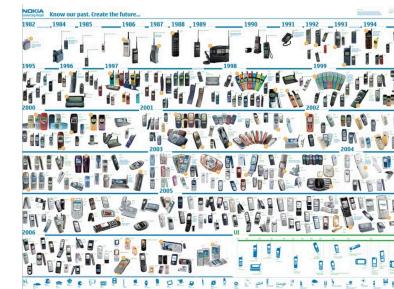
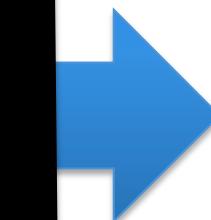
0100
101001010100
101001010100
01010010101000
01010010101000
001010010101000
001010010101000
001010010101000
001010010101000

0 1 1 1 1 0 1 1 1 1 1 0 ...



... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...

s2



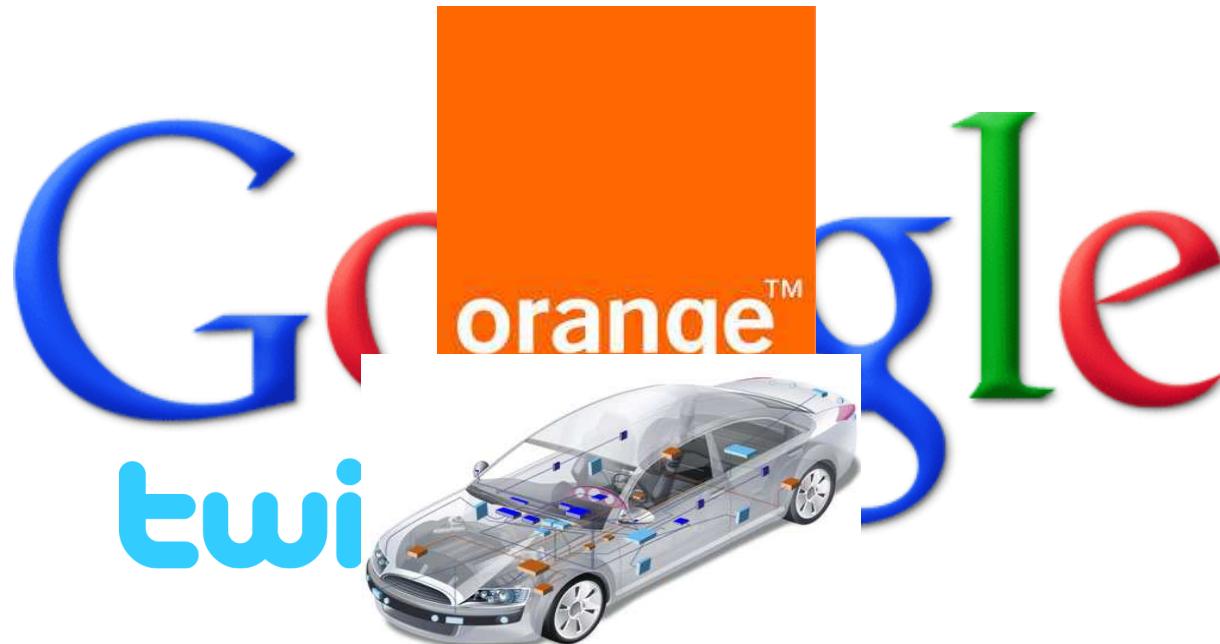
Google
twitter



Programming the Turing Machine

Why aren't we using tapes, states and transitions after all ?

Complex Systems



Distributed systems

Thousands of
engineers/expertise

Web dev.

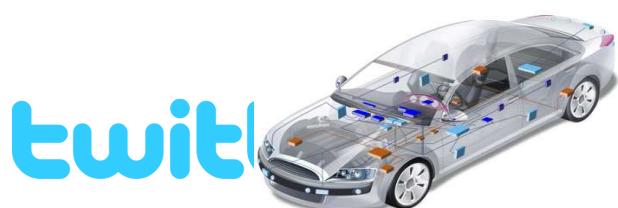
Large-scale systems

Critical Systems

Programming the Turing Machine

Why aren't we using tapes, states and transitions after all ?

You cannot be serious



[SUBMIT A LINK](#)[FEATURES](#)[REVIEWS](#)[PODCASTS](#)[VIDEO](#)[FORUMS](#)[MORE ▾](#)

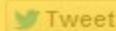
3CD LIMITED EDITION BOX SET • 2CD • 2LP • DOWN
AVAILABLE DECEMBER 10, 2013

Implementing a Turing machine in Excel

Cory Doctorow at 2:20 pm Fri, Sep 20, 2013



142



24



The screenshot shows a Microsoft Excel spreadsheet titled "Turing Machine SuccessTable - Microsoft Excel". The spreadsheet contains a large grid of binary digits (0s and 1s) representing a state transition table for a Turing machine. The columns and rows are labeled with binary strings, likely representing states and symbols. The formula bar at the top shows a complex formula involving VLOOKUP, IFERROR, and other functions, demonstrating the implementation of a Turing machine's logic within Excel's formula system.

Formulas are Turing complete

DOING DOING GIFT GUIDE 2013 2013 GIFT GUIDE

AN ASTRONAUT'S GUIDE TO LIFE ON EARTH

The image shows the cover of a gift guide for 2013. The title "DOING DOING GIFT GUIDE 2013" is displayed in a stylized, 3D font. Below it, the subtitle "2013 GIFT GUIDE" is shown. The main image on the cover is a photograph of an astronaut in a spacesuit floating in the void of space, with stars visible in the background.

Formulas are Turing complete

Turing Machine Successor																	
File		Home		Insert		Review		Add-Ins		Data		Page Layout		Formulas		Expector	
Clipboard		Font		Font		Font		Font		Font		Font		Font		Font	
A1	B1	C1	D1	E1	F1	G1	H1	I1	J1	K1	L1	M1	N1	O1	P1	Q1	
1																	
2					14	4 <---											
3																	
4	4 S1	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
5	5 S1	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
6	6 S1	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
7	7 S1	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
8	8 S2	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
9	9 S2	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
10	10 S2	-	-	-	-	1	1	1	-	-	-	-	-	-	-	-	-
11	9 S3	-	-	-	-	1	1	1	1	-	-	-	-	-	-	-	-
12	8 S3	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
13	7 S3	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
14	6 S3	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
15	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
16	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
17	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
18	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
19	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
20	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
21	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
22	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
23	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
24	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
25	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
26	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
27	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
28	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
29	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-
30	7 S4	-	-	-	-	1	1	1	1	1	-	-	-	-	-	-	-

Youtube video <https://t.co/RTfJAxXYaX>

<http://fr.slideshare.net/Felienne/spreadsheets-are-code-online>

Esoteric programming languages

- Designed to test the boundaries of computer programming language design, as a proof of concept, as software art, or as a joke.
 - extreme paradigms and design decisions
 - Eg <https://esolangs.org/wiki/Brainfuck>
- Usually, an esolang's creators do not intend the language to be used for mainstream programming.

(brainfuck)

What does it compute?

```
++++++[>++++++>++++++>+++<<<-]>++.>+.++++++  
..+++.>++.<<+++++++.>.+++.-----.-----.>+.
```

Quizz Time

- Why assembly language is not the mainstream language?
- Why spreadsheets are not used for building Google?
- Why esoteric languages are not used for mainstream programming?

Programming the Turing Machine

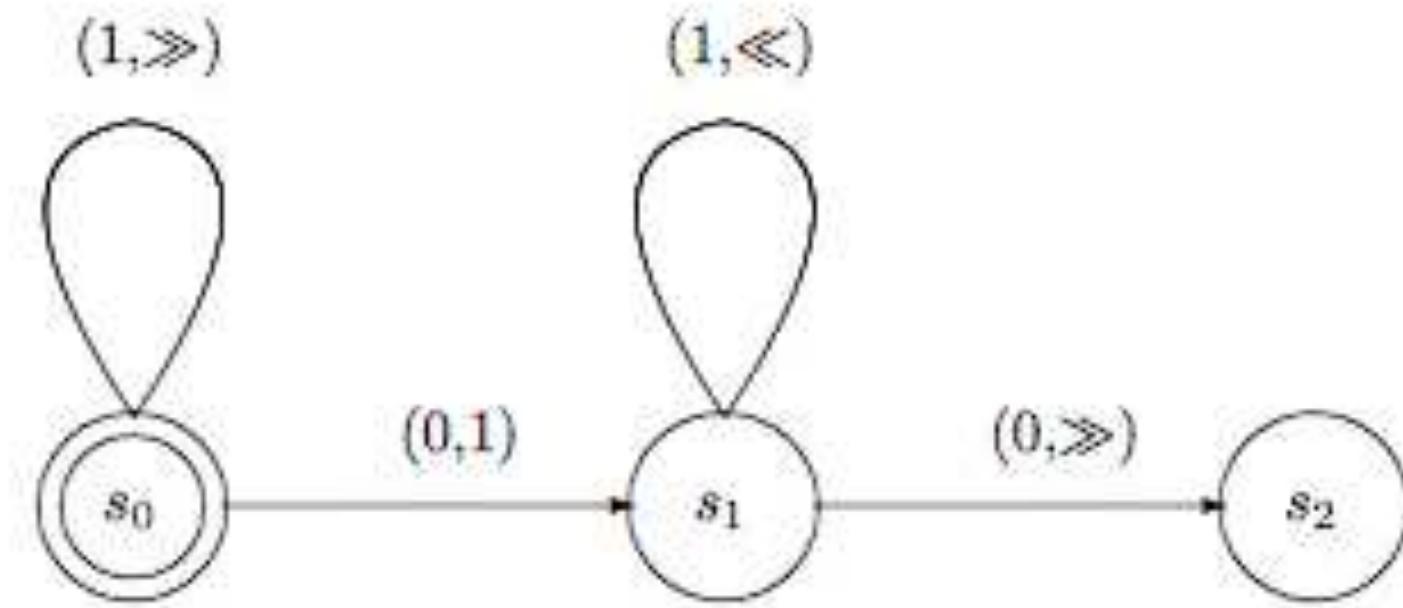
Why aren't we using tapes, states and transitions after all ?

Software Languages



Hard to write and understand.
No abstractions.
Hard to debug and test.
Poor language constructs. Poor tooling support.
Performance.
Usability, productivity,
reusability, safety,
expressiveness, learnability.

Question: what does it compute?

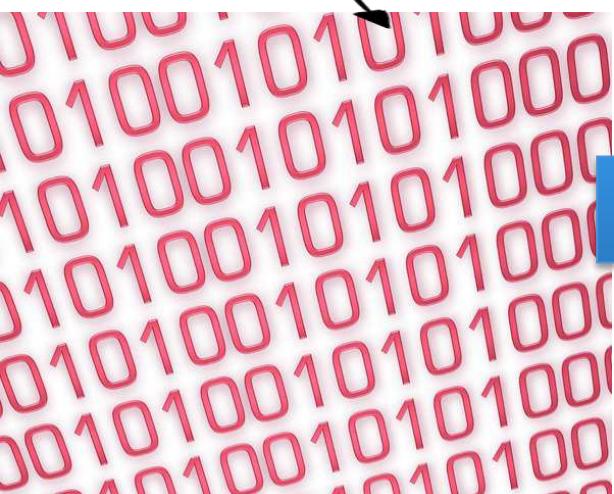


**Performance, usability,
productivity, reusability, safety,
expressiveness, learnability.**

Qualities and challenges

- Cognitive dimensions (see references after)
- Abstractions
 - Eg Kramer “Abstraction and Modelling - A Complementary Partnership” MODELS’08
- Separation of concerns/modularity
 - Eg Tarr et al., ICSE’99
- Scalability
 - Growing a language (like Scala)
- Performance
- ...

Languages



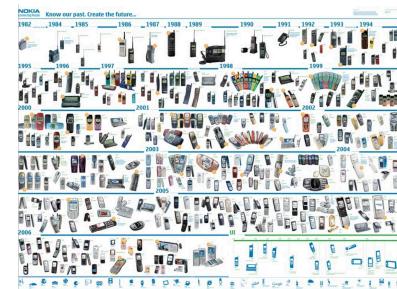
0100
101001010100
1010010010101000
0101001010101000
0101001010101000
00101001010101000
00101001010101000
00101001010101000

0 1 1 1 1 0 1 1 1 1 1 0 ...



... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...

Complex Systems



orange™



Google

twitter



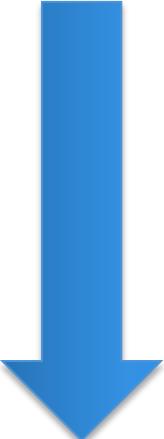
Instagram



We need languages

1. At a high level of abstraction
 1. Still general-purpose
 2. Generation of other artefacts written in other languages
 3. Transformation, refinement
2. Multiplicity of languages
 1. Divide and conquer
 2. Specific to a problem or “domain”
 3. Induce a way to “compose” languages

(Combemale et al. “On the Globalization of Domain-Specific Languages”)



How Language Shapes Thought

The languages we speak affect our perceptions of the world

By Lera Boroditsky

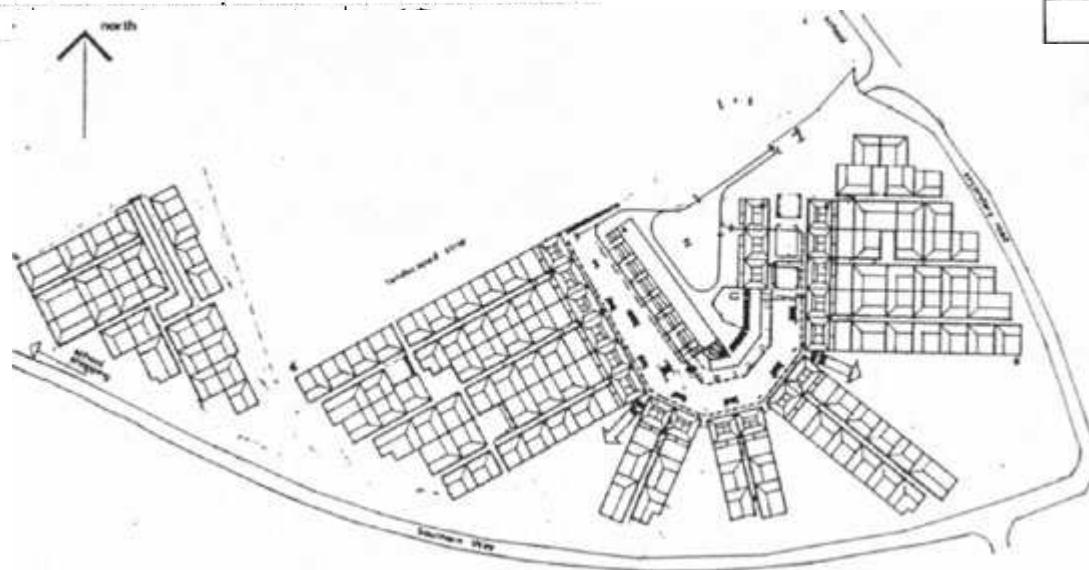
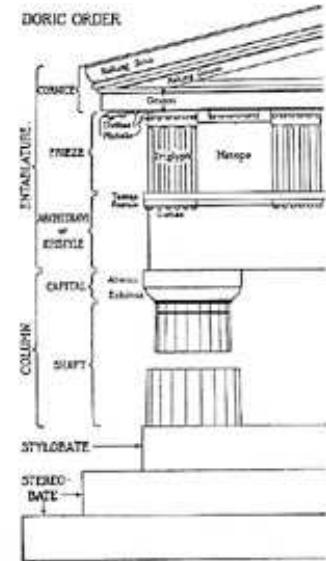
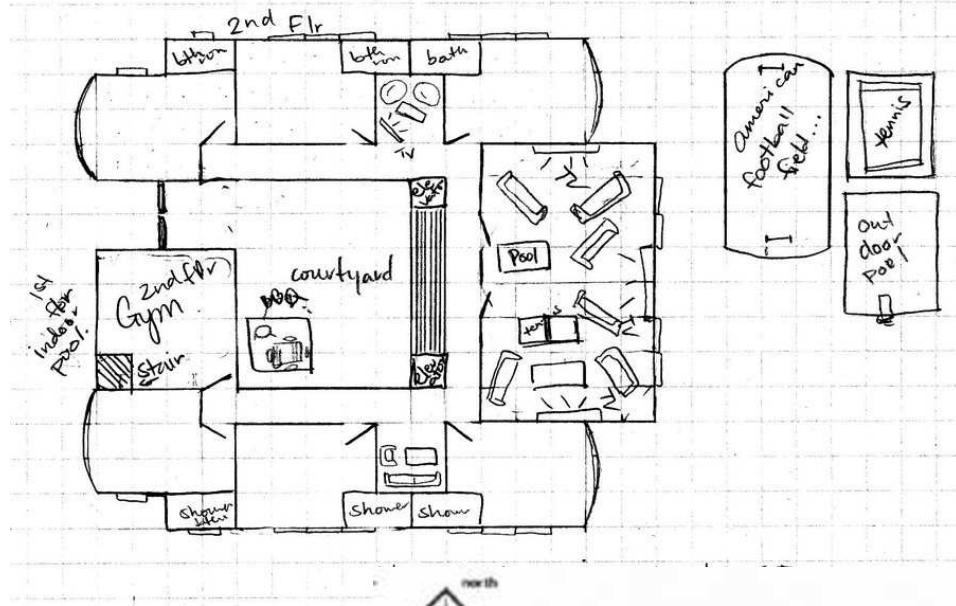
“Even variations in grammar can profoundly affect how we see the world.”

She's talking about real languages; **what about synthetic, programming languages?**

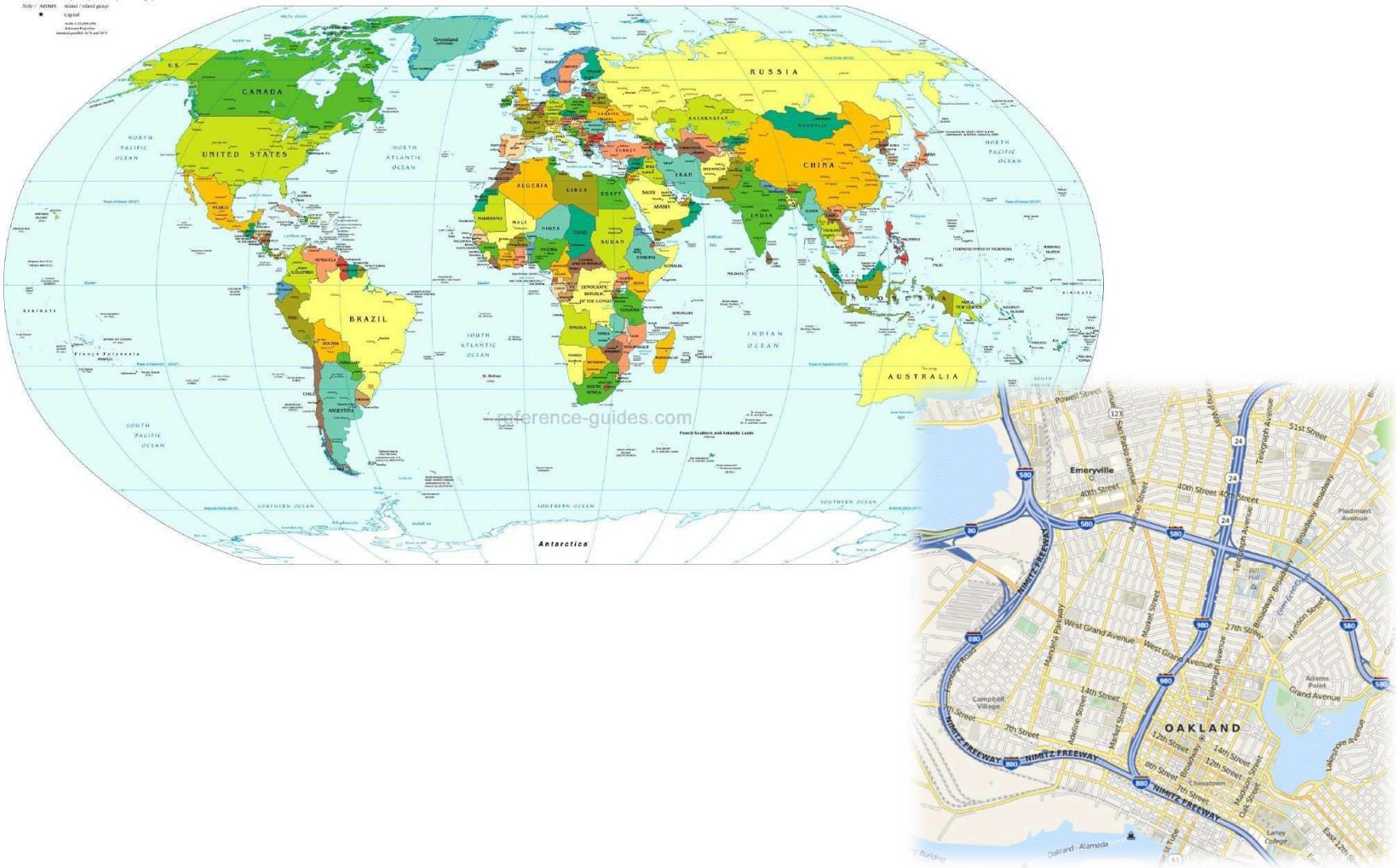
What is a language?

- « A system of signs, symbols, gestures, or rules used in **communicating** »
- « The **special** vocabulary and usages of a scientific, professional, or other group »
- « A system of symbols and rules used for communication with or between computers. »

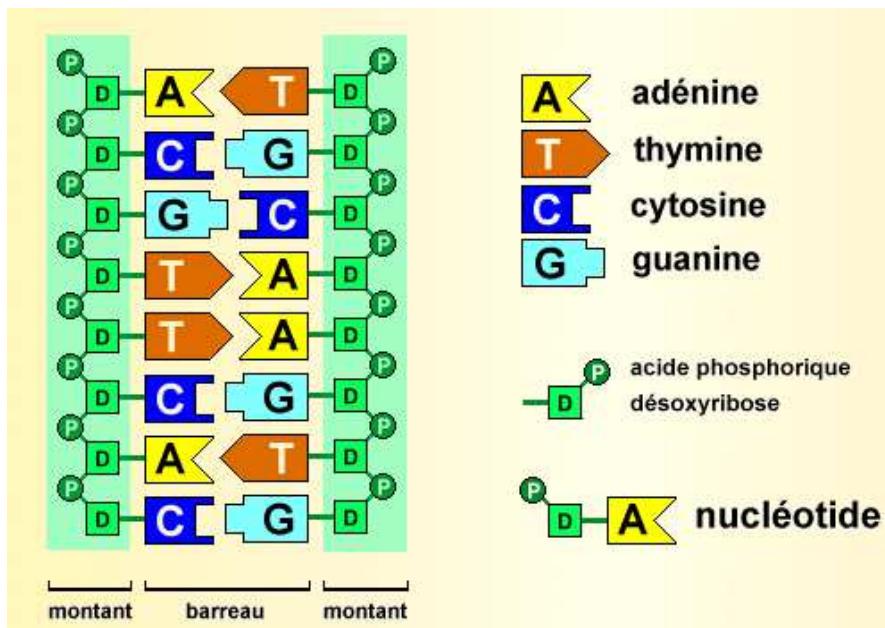
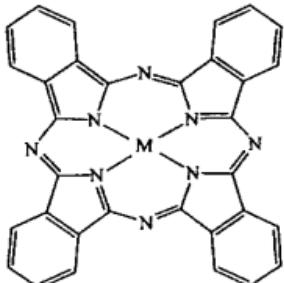
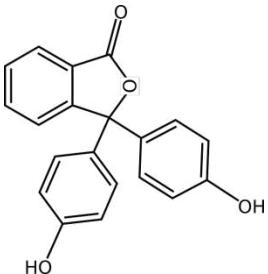
Architecture



Cartography



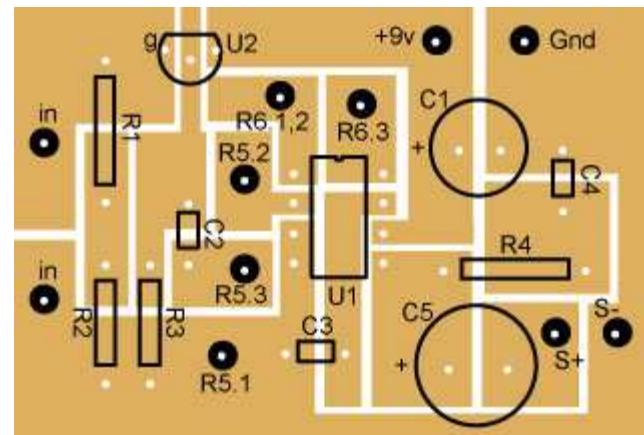
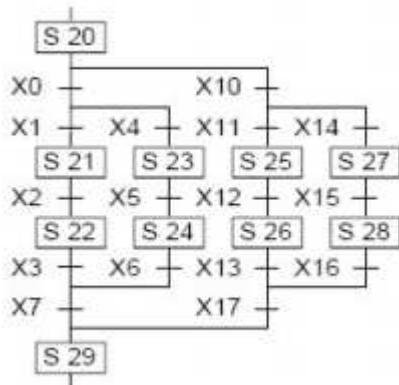
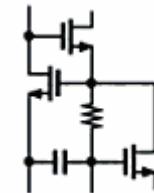
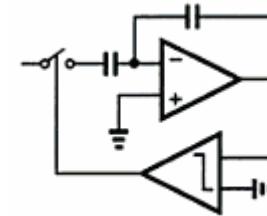
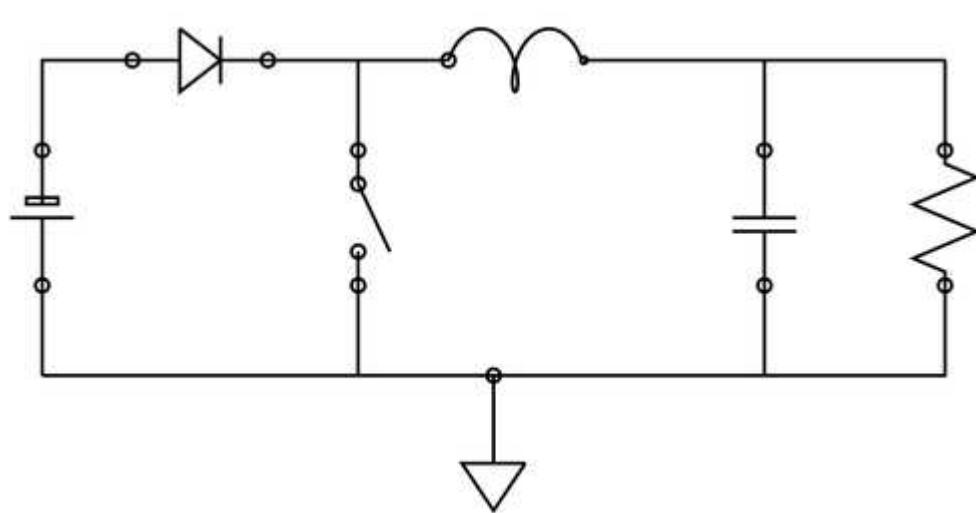
Biology



60	70	80	90	100
AGACCCCCAG	CAACCCCCGG	GGCCGTGCG	CGTCGGTCGT	GTCGTGTGAT
160	170	180	190	200
AGACCCCAGC	TACGAATGCC	GGTCCACCAA	CAACCCGTGG	GCTTCGCAAGC
260	270	280	290	300
CTGCCGGGCA	TGTACAGTC	TTGTCGGCAG	TTCTTCCACA	AGGAAGACAT
360	370	380	390	400
GGCTTGCTGG	GGCCCCCGCC	ACCAGCACTA	CAGACCTCCA	GTACGTCGTG
460	470	480	490	500
GGCCTATCCC	ACGCTCGCCG	CCAGCCACAG	AGTTATGCTT	GCCGAGTACA
560	570	580	590	600
GAAAGGGTGG	CGCCGATGAA	GAGACTATT	AAGCTGGAA	ACAAGGTGGT
660	670	680	690	700
ATAGTGGTTA	ACTTCACCTC	CAGACTCTTC	GCTGATGAAC	TGGCCGCCCT
760	770	780	790	800
AAAATATACA	GGCATTTGGC	CTGGGGTGCG	TATGCTCACG	TGAGACATCT
860	870	880	890	900
CCTGGAGGAG	GTTCGCCCCG	ACAGCCTGCG	CCTAACGCGG	ATGGATCCCT
960	970	980	990	1000
AGCAACACCC	AGCTTACAGT	GCTACCCCCA	TTTTTTAGCC	GAAGAGGATAC
1060	1070	Pvu II site	1090	1100
TGCCGCAGCA	ACTGGGGCAC	GCTATTCTGC	AGCAAGCTGTT	GGTGTACAC
1160	1170	1180	1190	1200
ACTTGATCTA	TATAACCACCA	ATGTGTCAATT	TATGGGGCGC	ACATATCGTC
1260	1270	1280	1290	1300
CTGTCATGT	ACCTTTGTAT	CCTATCAGCC	TTGGTTCCCA	GGGGGTGTCT
1360	1370	1380	1390	1400
TGTTTGAGGG	GGTGGTGCCA	GATGAGGTGA	CCAGGATAGA	TCTCGACCAG
1460	1470	1480	1490	1500
TCAGAGTCCT	AGTTCTATAT	TTAACCTTGG	CCCCAGACTG	CACGTGTATG
1560	1570	1580	1590	1600
CGATTTGAAG	CGGGGGGGGT	ATGGCGTCAT	CTGATATTCT	GTGGGTTGCA
1660	1670	1680	1690	1700
AAAAACTTACC	GTCTTACCTGC	CGGACACTGA	ACCCCTGGGTG	GTAGAGACCG
1760	1770	1780	1790	1800
AAGCTTCATC	GTGGTGCCCT	GCCCTCAAAT	TCTCACAAAG	GCTTGAGGAT

CTG.

Electronics



In Software Engineering

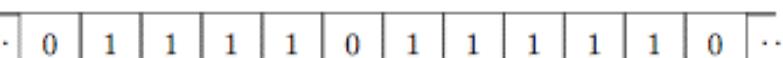
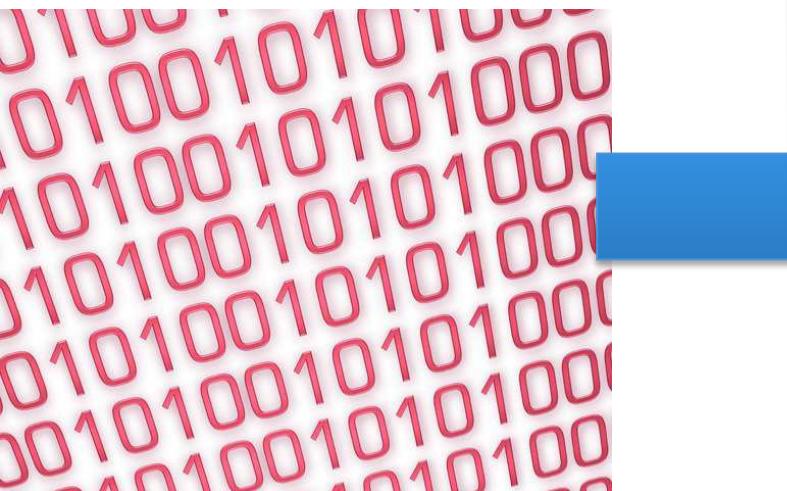
« Languages are the primary way in which system developers communicate, design and implement software systems »

General Purpose Languages

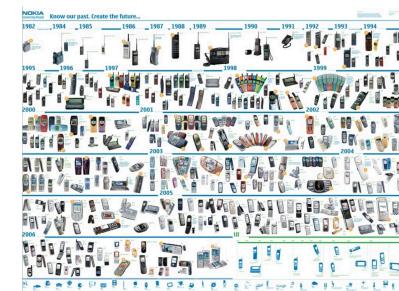
Assembly ?

COBOL ? LISP ? C ? C++ ?

Java? PHP ? C# ? Ruby ?



... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...



Google

twitter



Limits of General Purpose Languages (1)

- **Abstractions** and **notations** used are not natural/suitable for the stakeholders



```
if (newGame) resources.free();
s = FILENAME + 3;
setLocation(); load(s);
loadDialog.process();

try { setGamerColor(RED); }
catch(Exception e) { reset(); }
while (notReady) { objects.make();
if (resourceNotFound) break;

byte result; // сменить на int!
music();
System.out.print("");
```

Limits of General Purpose Languages (2)

- Not targeted to a **particular** kind of problem, but to any kinds of software problem.

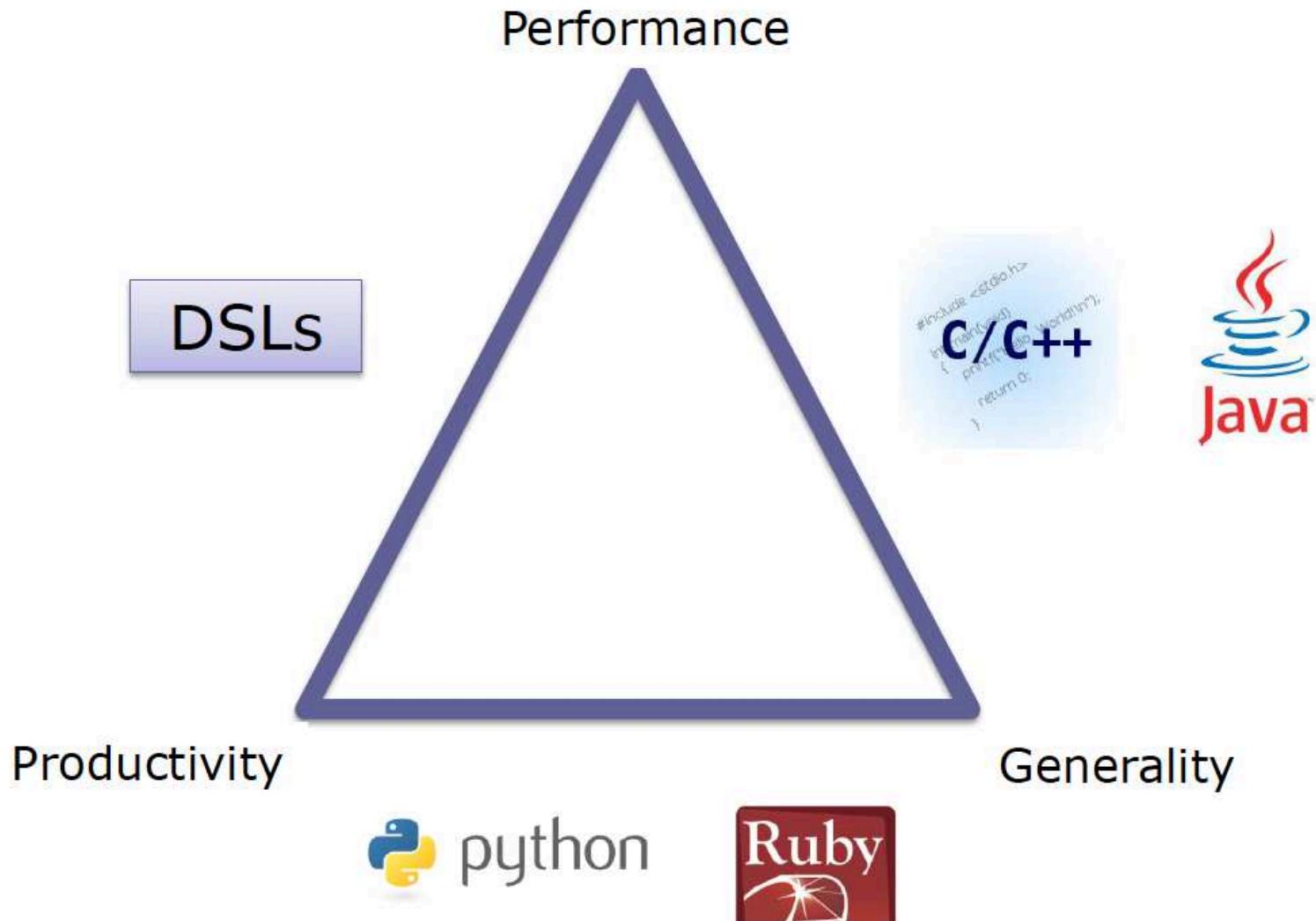


Domain Specific Languages

- Targeted to a **particular** kind of problem, with dedicated notations (textual or graphical), support (editor, checkers, etc.)
- Promises: more « efficient » languages for resolving a set of specific problems in a domain



A discussable view (slide “OptiML...” Sujeeth et al., ICML’11)



Domain Specific Languages (DSLs)

- Long history: used for almost as long as computing has been done.
- You're using DSLs in a daily basis
- You've learnt many DSLs in your curriculum
- Examples to come!

HTML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xml:lang="en" lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>My first Web page.</p>
  </body>
</html>
```

Domain: web (markup)

CSS

```
.CodeMirror {  
    line-height: 1;  
    position: relative;  
    overflow: hidden;  
}  
  
.CodeMirror-scroll {  
    /* 30px is the magic margin used to hide the element's real scrollbars */  
    /* See overflow: hidden in .CodeMirror, and the paddings in .CodeMirror-sizer */  
    margin-bottom: -30px; margin-right: -30px;  
    padding-bottom: 30px; padding-right: 30px;  
    height: 100%;  
    outline: none; /* Prevent dragging from highlighting the element */  
    position: relative;  
}  
.CodeMirror-sizer {  
    position: relative;  
}
```

Domain: web (styling)

SQL

```
SELECT Book.title AS Title,  
       COUNT(*) AS Authors  
  FROM Book  
 JOIN Book_author  
    ON Book.isbn = Book_author.isbn  
GROUP BY Book.title;
```

```
INSERT INTO example  
(field1, field2, field3)  
VALUES  
( 'test' , 'N' , NULL);
```

Domain: database (query)

Makefile

```
PACKAGE      = package
VERSION      = `date "+%Y.%m%d%" `
RELEASE_DIR  = ..
RELEASE_FILE = $(PACKAGE)-$(VERSION)

# Notice that the variable LOGNAME comes from the environment in
# POSIX shells.
#
# target: all - Default target. Does nothing.
all:
    echo "Hello $(LOGNAME), nothing to do by default"
    # sometimes: echo "Hello ${LOGNAME}, nothing to do by default"
    echo "Try 'make help'"

# target: help - Display callable targets.
help:
    egrep "^# target:" [Mm]akefile

# target: list - List source files
list:
    # Won't work. Each command is in separate shell
    cd src
    ls

    # Correct, continuation of the same shell
    cd src; \
    ls
```

Domain: software building

Lighttpd configuration file

```
server.document-root = "/var/www/servers/www.example.org/pages/"

server.port = 80

server.username = "www"
server.groupname = "www"

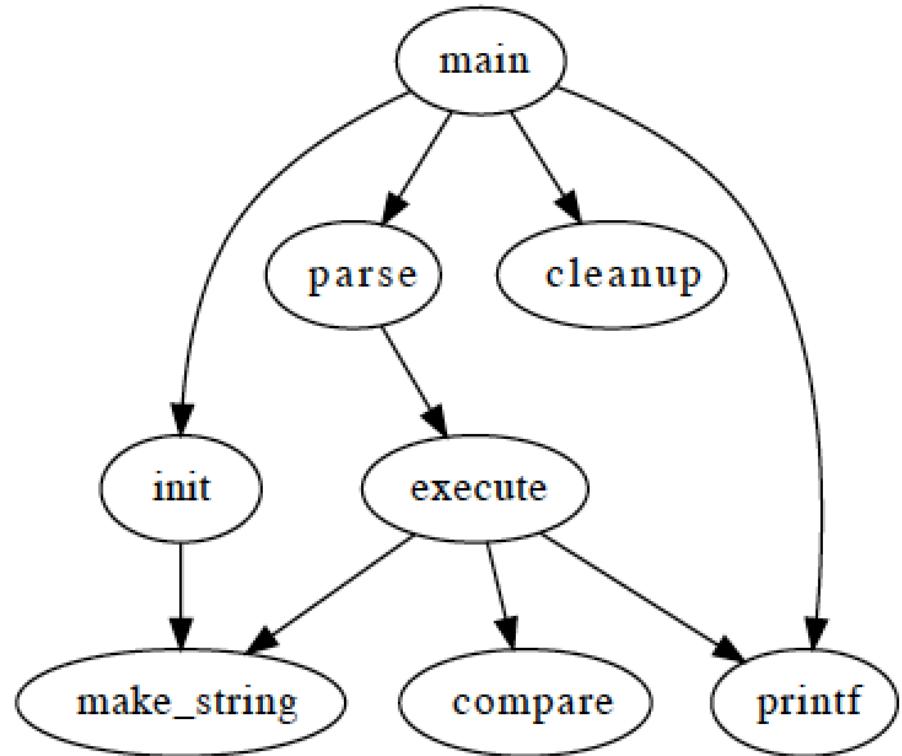
mimetype.assign = (
    ".html" => "text/html",
    ".txt" => "text/plain",
    ".jpg" => "image/jpeg",
    ".png" => "image/png"
)

static-file.exclude-extensions = ( ".fcgi", ".php", ".rb", "~", ".inc" )
index-file.names = ( "index.html" )
```

Domain: web server (configuration)

Graphviz

```
digraph G {  
    main -> parse -> execute;  
    main -> init;  
    main -> cleanup;  
    execute -> make_string;  
    execute -> printf  
    init -> make_string;  
    main -> printf;  
    execute -> compare;  
}
```

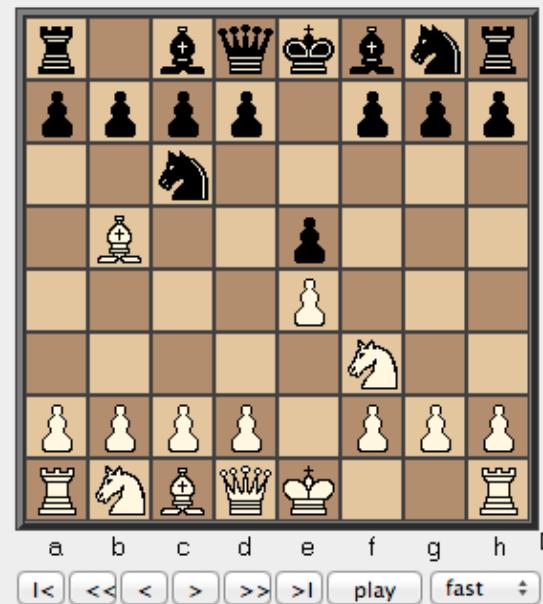


Domain: graph (drawing)

PGN (Portable Game Notation)

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia Yugoslavia|JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]
```

```
1. e4 e5 2. Nf3 Nc6 3. Bb5 {This opening is called the Ruy Lopez.} 3... a6
4. Ba4 Nf6 5. 0-0 Be7 6. Re1 b5 7. Bb3 d6 8. c3 0-0 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxel+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```



Domain: chess (games)

Regular expression

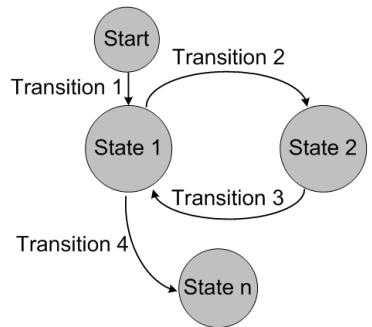
```
<TAG\b[^>]*>(.*)?</TAG>
```

Domain: strings (pattern matching)

BIBTEX



Graphviz



**Finite State
Machine**



Domain-Specific Languages (DSLs)



PGN



Make

Matlab



Quizz Time

- Give three other examples of domain-specific languages (DSLs)

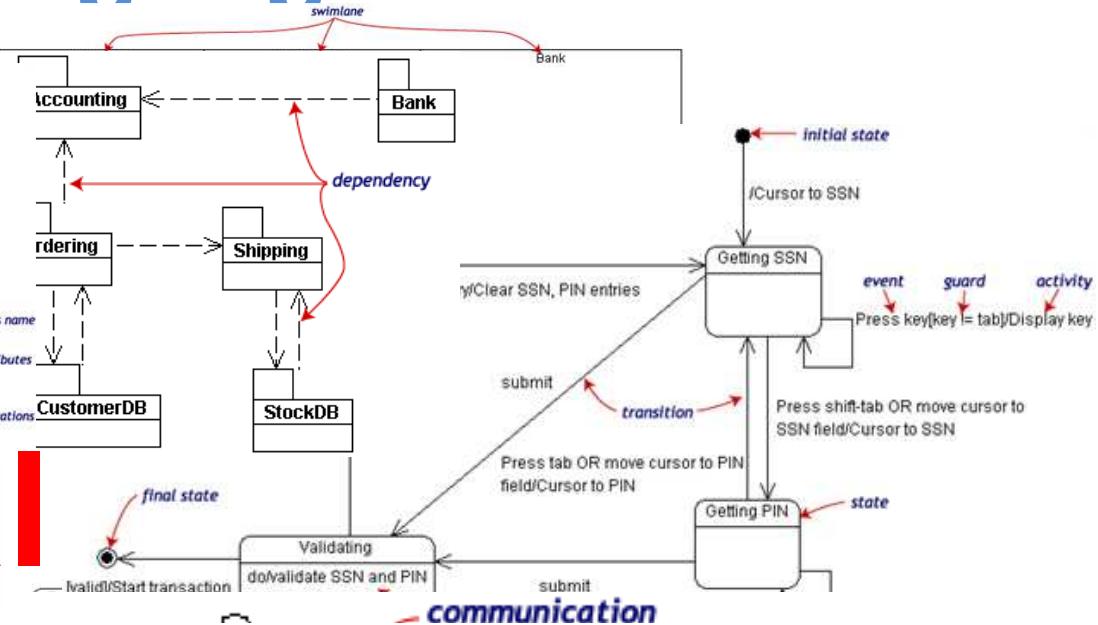
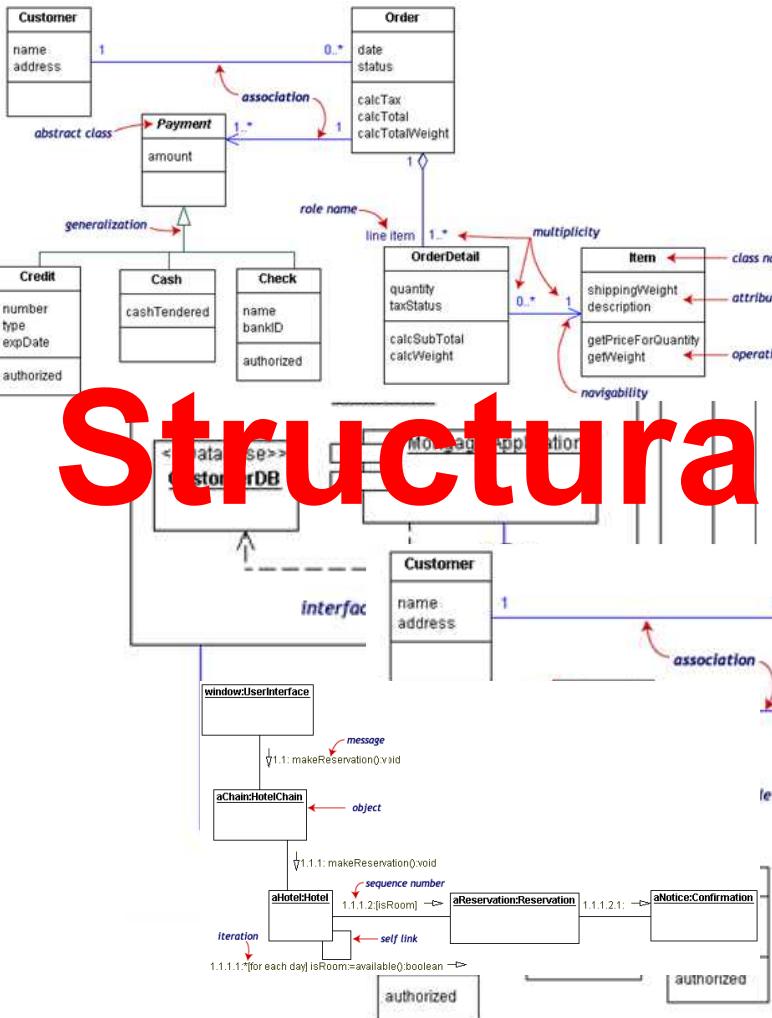
OCL

```
self.questions->size  
self.employer->size  
self.employee->select (v | v.wages>10000 )->size  
Student.allInstances  
->forAll( p1, p2 |  
           p1 <> p2 implies p1.name <> p2.name )
```

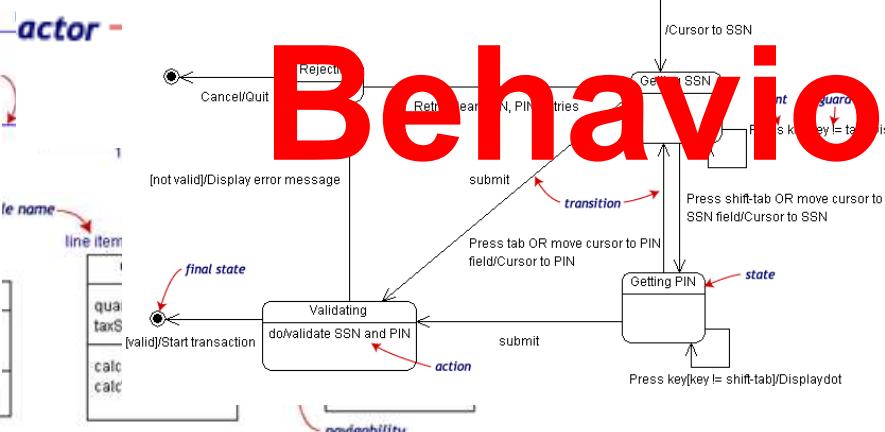
Domain: model management

UML can be seen as a collection of domain-specific modeling languages

Structural



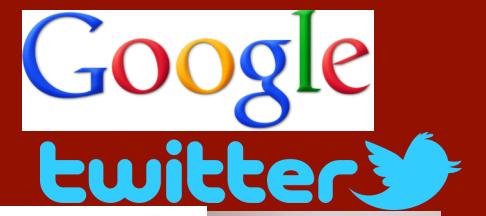
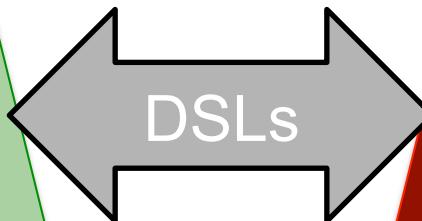
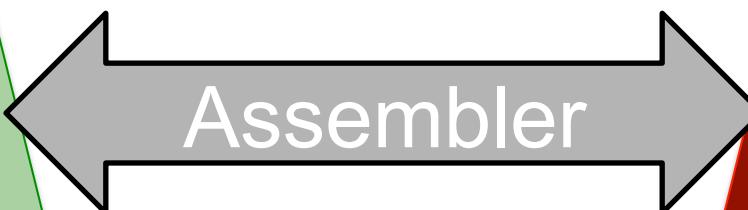
Behavioral



Abstraction Gap

Problem
Space

Solution
Space



« Another lesson we should have learned from the recent past is that the development of 'richer' or 'more powerful' programming languages was a mistake in the sense that these baroque monstrosities, these conglomerations of idiosyncrasies, are really unmanageable, both mechanically and mentally.

aka General-Purpose Languages

I see a great future for very systematic and very modest programming languages »

1972



aka Domain-Specific Languages

ACM Turing Lecture, « The Humble Programmer »
Edsger W. Dijkstra

Empirical Assessment of MDE in Industry

John Hutchinson, Jon Whittle, Mark Rouncefield

School of Computing and Communications
Lancaster University, UK
+44 1524 510492

{j.hutchinson, j.n.whittle,
m.rouncefield}@lancaster.ac.uk

Steinar Kristoffersen

Østfold University College and Møreforskning Molde AS
NO-1757 Halden
Norway
+47 6921 5000

steinar.kristoffersen@hiof.no

Model-Driven Engineering Practices in Industry

John Hutchinson
School of Computing and
Communications
Lancaster University, UK
+44 1524 510492

{j.hutchinson@lancaster.ac.uk}

Mark Rouncefield
School of Computing and
Communications
Lancaster University, UK
+44 1524 510492

{m.rouncefield@lancaster.ac.uk}

Jon Whittle
School of Computing and
Communications
Lancaster University, UK
+44 1524 510492

{j.n.whittle@lancaster.ac.uk}

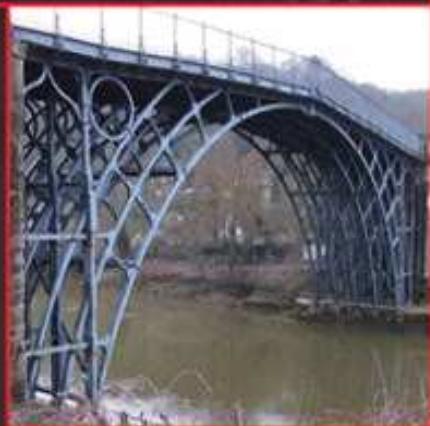
2011

**« Domain-specific
languages are far more
prevalent than
anticipated »**

The Addison-Wesley Signature Series

DOMAIN-SPECIFIC LANGUAGES

MARTIN FOWLER
WITH REBECCA PARSONS



A MARTIN FOWLER
Signature Book
Martin Fowler



2011



What is a domain-specific language ?

- « Language **specially** designed to perform a task in a **certain domain** »
- « A formal processable language targeting at a **specific viewpoint or aspect** of a software system. Its **semantics and notation** is designed in order to support working with that viewpoint as good as possible »
- « A computer language that's targeted to a particular kind of problem, **rather than a general purpose language** that's aimed at any kind of software problem. »

GPL (General Purpose Language)

A GPL provides notations that are used to describe a computation in a human-readable form that can be translated into a machine-readable representation.

A GPL is a formal notation that can be used to describe problem solutions in a precise manner.

A GPL is a notation that can be used to write programs.

A GPL is a notation for expressing computation.

A GPL is a standardized communication technique for expressing instructions to a computer. It is a set of syntactic and semantic rules used to define computer programs.

Promises of domain-specific languages

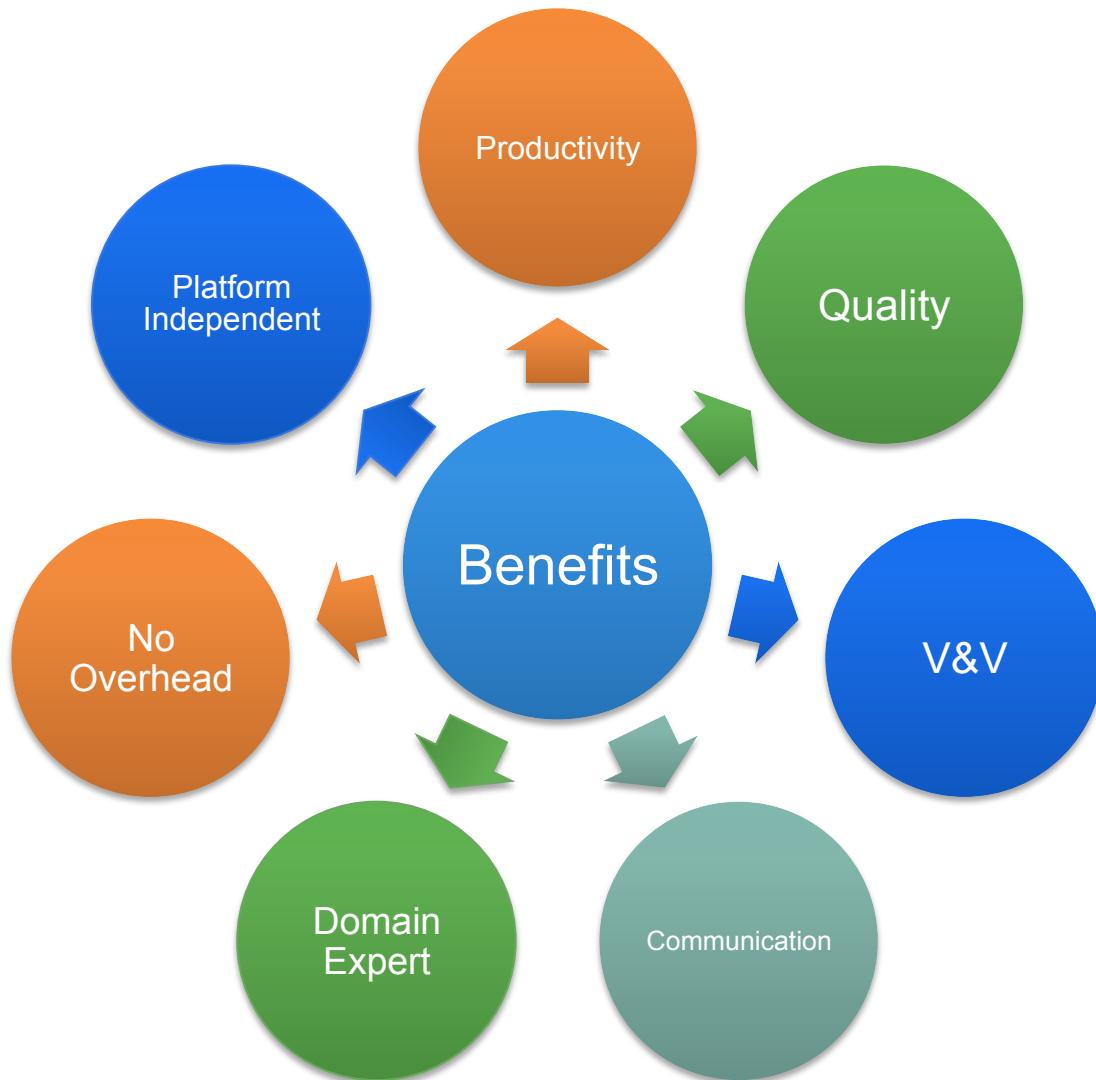
Higher abstractions

Avoid redundancy

Separation of concerns

Use domain concepts

Promises of domain-specific languages



GeneralPL vs DomainSL

The boundary isn't as clear as it could be. Domain-specificity is not black-and-white, but instead gradual: a language is more or less domain specific



	GPLs	DSLs
Domain	large and complex	smaller and well-defined
Language size	large	small
Turing completeness	always	often not
User-defined abstractions	sophisticated	limited
Execution	via intermediate GPL	native
Lifespan	years to decades	months to years (driven by context)
Designed by	guru or committee	a few engineers and domain experts
User community	large, anonymous and widespread	small, accessible and local
Evolution	slow, often standardized	fast-paced
Deprecation/incompatible changes	almost impossible	feasible

Specializing syntax and environment pays off?

- Promises of DSL « improvement » in terms of
 - usability, learnability, expressiveness, reusability, etc.
- Empirical study on the role of **syntax**
 - C-style syntax induces problems in terms of usability for novices; language more or less intuitive for (non-)programmers (Stefik et al. 2014)
 - Syntax issues with Java for students (Denny et al. 2011)
 - PL usability: method namings/placement, use of identifiers, API design (Ellis et al., Styllos et al., Clarke, Montperrus et al., etc.)
- More **specialized/sophicated tools/IDE** can be derived from a DSL
 - editors, compilers, debuggers

Quizz Time

- Take one DSL and formulate assumptions on their qualities (and superiority to a GPL-based solution)
- Imagine an experience for providing evidence that the DSL has such qualities

External DSLs vs Internal DSLs

- An **external** DSL is a completely separate language and has its own custom syntax/tooling support (e.g., editor)
- An internal DSL is more or less a set of APIs written on top of a host language (e.g., Java).
 - Fluent interfaces

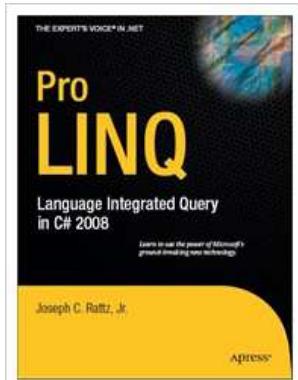
External vs Internal DSL (SQL example)

```
-- Select all books by authors born after 1920,  
-- named "Paulo" from a catalogue:  
SELECT *  
  FROM t_author a  
  JOIN t_book b ON a.id = b.author_id  
WHERE a.year_of_birth > 1920  
    AND a.first_name = 'Paulo'  
ORDER BY b.title
```

```
Result<Record> result =  
create.select()  
  .from(T_AUTHOR.as("a"))  
  .join(T_BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))  
  .where(a.YEAR_OF_BIRTH.greaterThan(1920))  
  .and(a.FIRST_NAME.equal("Paulo"))  
  .orderBy(b.TITLE)  
  .fetch();
```

Internal DSL (LINQ/C# example)

```
// DataContext takes a connection string
DataContext db = new DataContext("c:\\northwind\\northwnd.mdf");
// Get a typed table to run queries
Table<Customer> Customers = db.GetTable<Customer>();
// Query for customers from London
var q =
    from c in Customers
    where c.City == "London"
    select c;
foreach (var cust in q)
    Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);
```



Internal DSL

- « Using a host language (e.g., Java) to give the host language the feel of a particular language. »
- Fluent Interfaces**
 - « The more the use of the API has that language like flow, the more fluent it is »

```
Result<Record> result =
    create.select()
        .from(T_AUTHOR.as("a"))
        .join(T_BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
        .where(a.YEAR_OF_BIRTH.greaterThan(1920))
        .and(a.FIRST_NAME.equal("Paulo")))
        .orderBy(b.TITLE)
        .fetch();
```

```
-- Select all books by authors born after 1920,
-- named "Paulo" from a catalogue:
SELECT *
FROM t_author a
JOIN t_book b ON a.id = b.author_id
WHERE a.year_of_birth > 1920
    AND a.first_name = 'Paulo'
ORDER BY b.title
```

SQL in... Java

DSL in GPL

```
Connection con = null;

// create sql insert query
String query = "insert into user values(" + student.getId() + ","
+ student.getFirstName() + "','" + student.getLastName()
+ "','" + student.getEmail() + "','" + student.getPhone()
+ "')";

try {
    // get connection to db
    con = new CreateConnection().getConnection("checkjdbc", "root",
        "root");

    // get a statement to execute query
    stmt = con.createStatement();

    // executed insert query
    stmt.execute(query);
    System.out.println("Data inserted in table !");
}
```

Regular expression in... Java

DSL in GPL

```
public class RegexTestStrings {  
    public static final String EXAMPLE_TEST = "This is my small example "  
        + "string which I'm going to " + "use for pattern matching."  
  
    public static void main(String[] args) {  
        System.out.println(EXAMPLE_TEST.matches("\\w.*"));  
        String[] splitString = (EXAMPLE_TEST.split("\\s+"));  
        System.out.println(splitString.length); // Should be 14  
        for (String string : splitString) {  
            System.out.println(string);  
        }  
        // Replace all whitespace with tabs  
        System.out.println(EXAMPLE_TEST.replaceAll("\\s+", "\t"));  
    }  
}
```

Terminology

- Traditional dichotomy between internal DSL and external DSL (Fowler et al., 2010)
 - Fluent APIs
 - Internal DSLs
 - (deeply) embedded DSLs
 - External DSLs
 - What's LINQ?
- Boundary between DSL and GPL is not that clear (Voelter et al., 2013)
 - What is and what is not a DSL is still a debate

Internal DSLs vs External DSL

- Both internal and external DSLs have strengths and weaknesses (Fowler et al., 2010)
 - learning curve,
 - cost of building,
 - programmer familiarity,
 - communication with domain experts,
 - mixing in the host language,
 - strong expressiveness boundary
- Focus of the next course
 - **external DSL** a completely separate language with its own custom syntax and tooling support (e.g., editor)

Quizz Time

- Find a DSL that is both internal and external (but not HTML, not SQL)

HTML

- External DSL: <html>....
- Internal DSLs
 - LISP
 - Scala (XML support included in the language)

```
object XMLTest1 extends Application {  
    val page =  
        <html>  
            <head>  
                <title>Hello XHTML world</title>  
            </head>  
            <body>  
                <h1>Hello world</h1>  
                <p><a href="scala-lang.org">Scala</a> talks XHTML</p>  
            </body>  
        </html>;  
    println(page.toString())  
}
```

```

object XMLTest1 extends Application {
  val page =
<html>
  <head>
    <title>Hello XHTML world</title>
  </head>
  <body>
    <h1>Hello world</h1>
    <p><a href="scala-lang.org">Scala</a> talks XHTML</p>
  </body>
</html>;
  println(page.toString())
}

```

Scala

```

1 // Import the Glitter DSL
2 import glitter._
3
4 object Templates {
5
6   // Define a reusable layout
7   def layout(body: Xml) =
8     html5dtd | 'html (
9       'head :: 'title :: "Glitter is amazing!"
10      | 'body :: body
11    )
12
13   // Define a template taking one String argument and using the Layout defined above
14   def show(name: String) =
15     layout (
16       'h1 :: "Show user"
17       | 'p :: ("Hello " | 'strong(name) | "!")
18     )
19
20   // Define a template taking a List of Strings, using the Layout defined above
21   def index(users: List[String]) =
22     layout (
23       'h1 :: "User list"
24       | 'ul % 'class~"user-list" :: (for (user <- users) yield ('li :: user))
25     )
26 }

```

<https://github.com/julienrf/glitter>

TCS Wyvern (Omar et al., OOPSLA'14)

```

1 let webpage : HTML = HTMLElement(Dict.empty(), [BodyElement(Dict.empty(),
2   [H1Element(Dict.empty(), [TextNode("Results for " + keyword)]),
3    ULElement((Dict.add Dict.empty() ("id", "results")), to_list_items(query(db,
4      SelectStmt(["title", "snippet"], "products",
5        [WhereClause(InPredicate(StringLit(keyword), "title"))]))))))]
1 let webpage : HTML = <html><body><h1>Results for {keyword}</h1>
2   <ul id="results">{to_list_items(query(db,
3     SELECT title, snippet FROM products WHERE {keyword} in title))}
4   </ul></body></html>
1 let webpage : HTML = parse_html("<html><body><h1>Results for "+keyword+"</h1>
2   <ul id=\"results\">" + to_string(to_list_items(query(db, parse_sql(
3     "SELECT title, snippet FROM products WHERE '"+keyword+"' in title")))) +
4   "</ul></body></html>")

```

SQL

Plain SQL
(external DSL)

shape
#1

```
1 |-- SQL
2 SELECT * FROM journal
3   WHERE published_year = 2013
4     AND publisher = 'IEEE'
5 ORDER BY title
```

Java
(internal DSL)

shape
#2

```
// JOOQ fluent API
ResultQuery q = create.selectFrom(JOURNAL)
    .where(PUBLISHED_YEAR.equal(2013)
        .and(PUBLISHER.equal("IEEE")))
    .orderBy(TITLE);
```

Scala
(internal DSL)

shape
#3

```
journals
  .filter(journal => journal.published_year === 2013
    & journal.publisher === "IEEE")
  .sortBy(_.title)
```

Basic exercise

Jhipster (<http://www.jhipster.tech/>), let us consider the “last” version

How many languages are used in such a contemporary project?

1. List all software languages used in Jhipster
2. Classify them (GPL? External DSL? Internal DSL?)

Don't forget APIs that look like (internal) DSLs

(for the next session)

Project

- “A DSL for SAT solver variants”
- See Google doc. describing the project:
 - <https://docs.google.com/document/d/1v1YUDhcCjx42-amBqXqNzrI5LF9HfjXxku7kvG9zM0E/edit?usp=sharing>
- First milestone: 3 October 2018!
- Please send by email (mathieu.acher@irisa.fr) your git repository before 3 October

References

- Martin Fowler. Domain Specific Languages. Addison-Wesley Professional, 2010.
- Markus Voelter et al. “DSL Engineering: Designing, Implementing and Using Domain-Specific Languages.” dslbook.org, 2013.
- Kramer “Abstraction and Modelling - A Complementary Partnership” MODELS’08
- Tarr et al. “N Degrees of Separation: Multi-Dimensional Separation of Concerns” ICSE’99
- Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert France, Jean-Marc Jézéquel, and Jeff Gray. « Globalizing Modeling Languages.” Computer, 2014.

References

- Leo A Meyerovich and Ariel S Rabkin. “Empirical analysis of programming language adoption” OOPSLA’13
- Felienne Hermans, Martin Pinzger, and Arie van Deursen. “Domain-Specific languages in practice: A user study on the success factors.” MODELS’09
- Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. “Understanding the syntax barrier for novices.” ITiCSE ’11
- Tiark Rompf et al . “Optimizing Data Structures in High-Level Programs: New Directions for Extensible Compilers based on Staging” POPL’13

References

- Mathieu Acher, Benoît Combemale, Philippe Collet: “Metamorphic Domain-Specific Languages: A Journey into the Shapes of a Language.” Onward! 2014
- Jeffrey Stylos and Brad A. Myers. “The implications of method placement on api learnability” FSE’08
- Martin Monperrus, Michael Eichberg, Elif Tekes, and Mira Mezini. “What Should Developers Be Aware Of? An Empirical Study on the Directives of API Documentation”. *Empirical Software Engineering*, 17(6):703–737, 2012.

Plan

- Domain-Specific Languages (DSLs)
 - Languages and abstraction gap
 - Examples and rationale
 - DSLs vs General purpose languages, taxonomy
- External DSLs
 - Grammar and parsing
 - Language workbenches, Xtext
- DSLs, DSMLs, and (meta-)modeling

Contract

- Better understanding/source of inspiration of software languages and DSLs
 - Revisit of history and existing languages
- Foundations and practice of Xtext
 - State-of-the-art language workbench (Most Innovative Eclipse Project in 2010, mature and used in a variety of industries)
- Models and Languages
 - Perhaps a more concrete way to see models, metamodels and MDE