Domain-Specific Languages

Jean-Marc Jézéquel / Mathieu Acher

jezequel@irisa.fr, mathieu.acher@irisa.fr

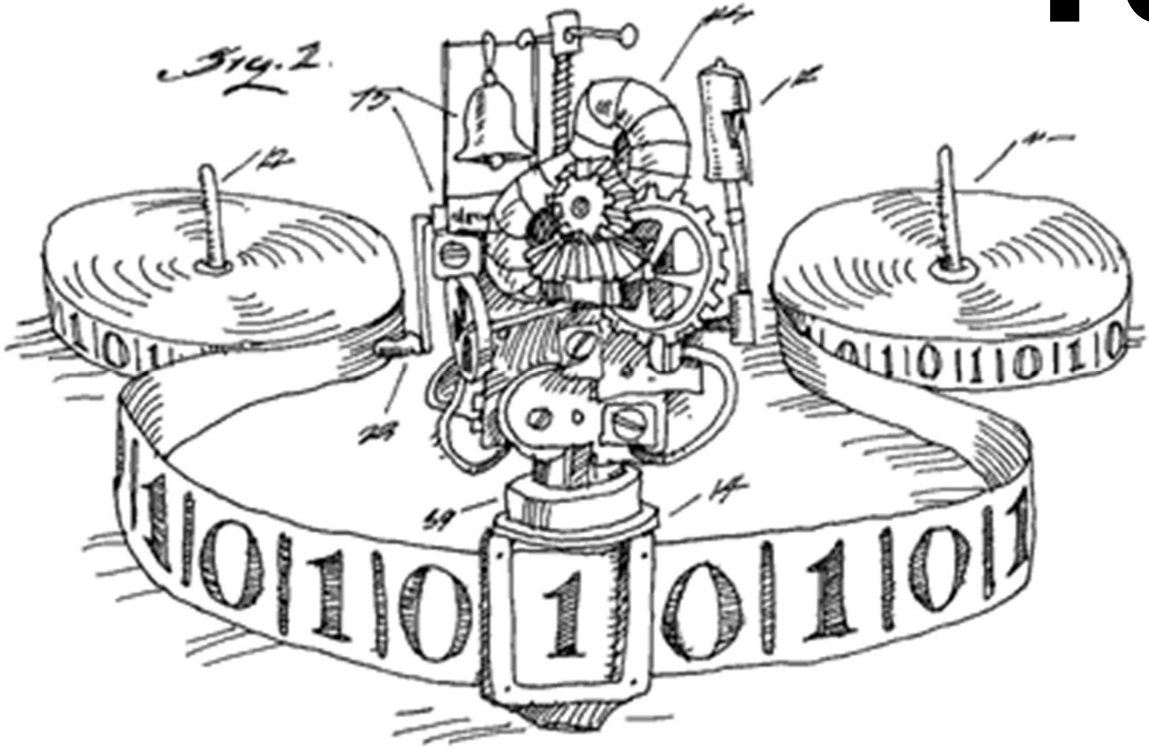
What are DSLs

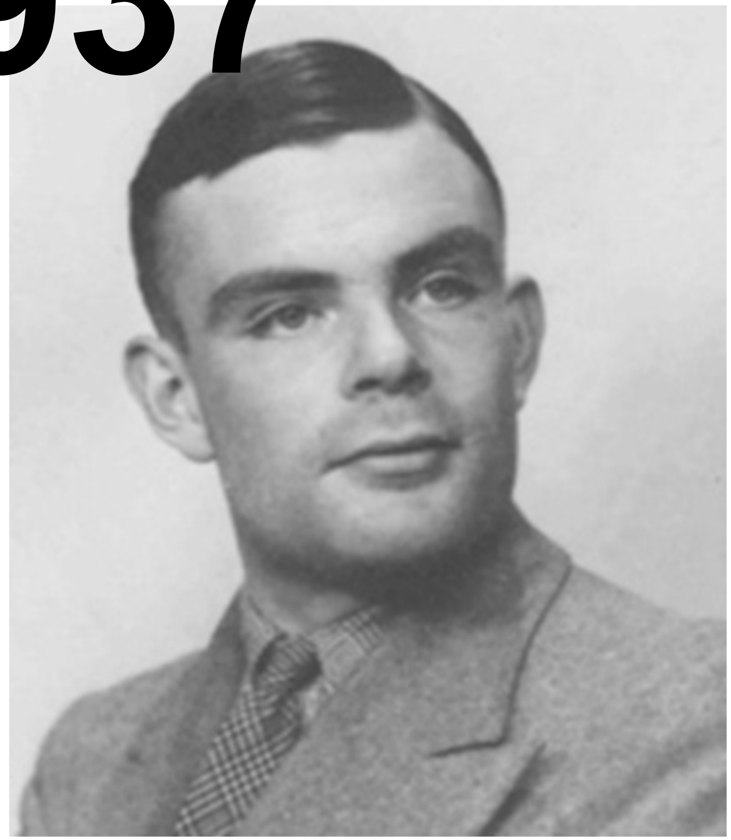
Where are DSLs

Why DSLs (will) matter

The (Hi)Story of Software Engineering / Computer Science

1937



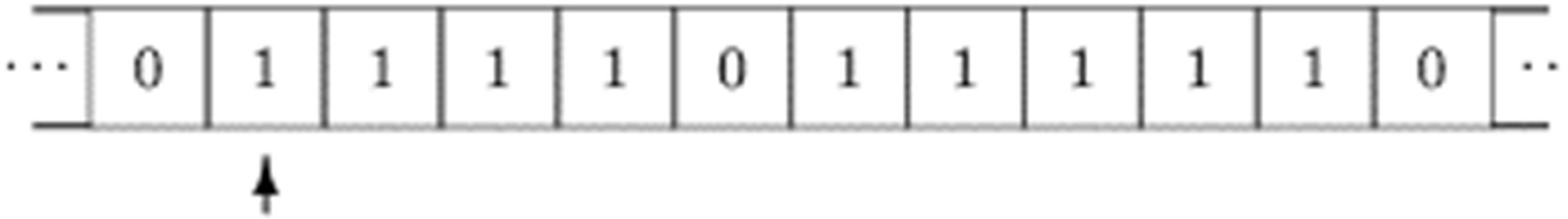


Turing Machine

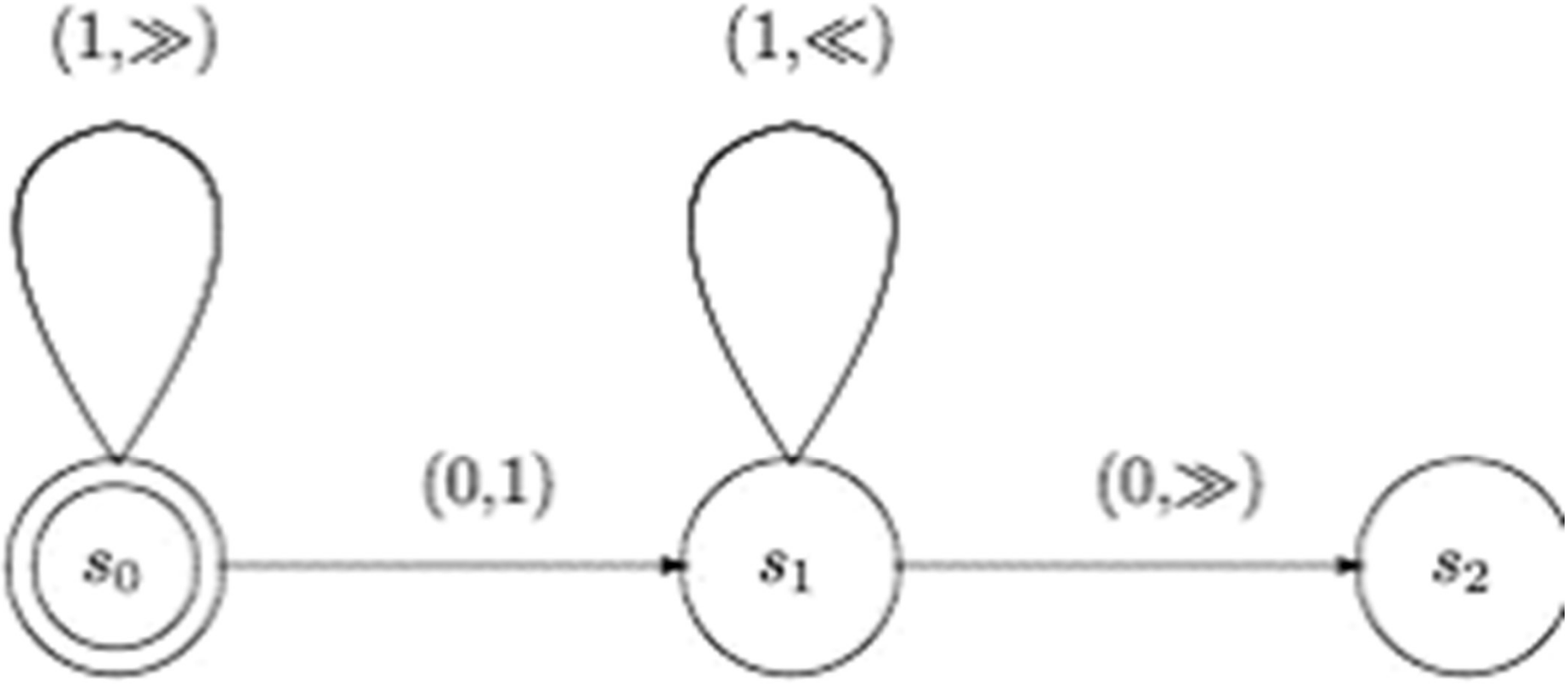
- Infinite tape divided into Cells (0 or 1)
- Read-Write Head
- Transitition rules

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

$$\langle\ State_{\text{current}}, Symbol, State_{\text{next}}, Action\ \rangle$$

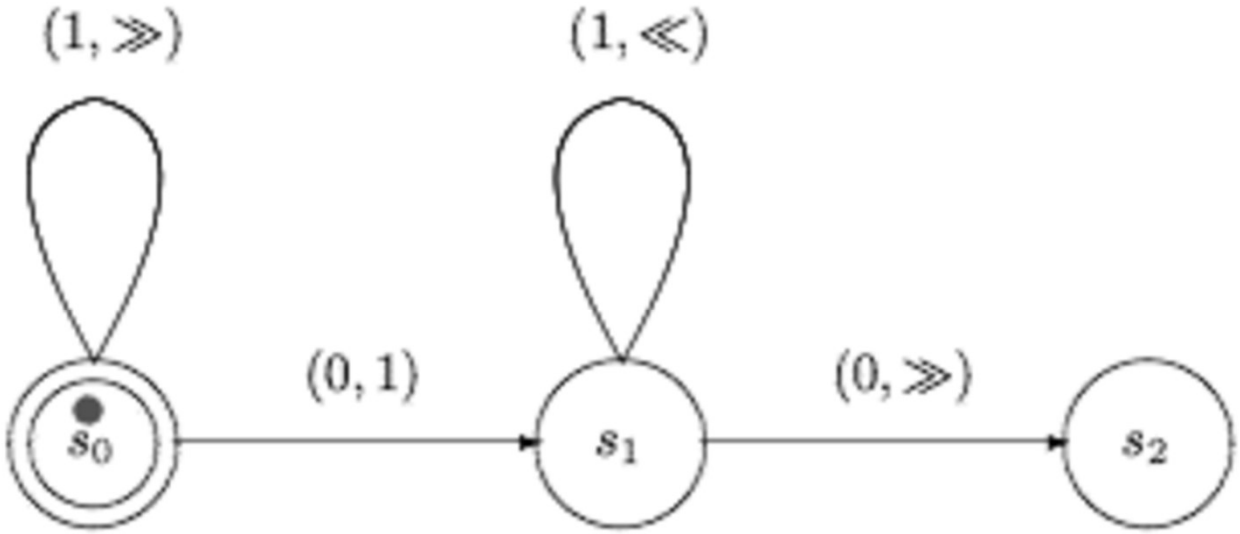


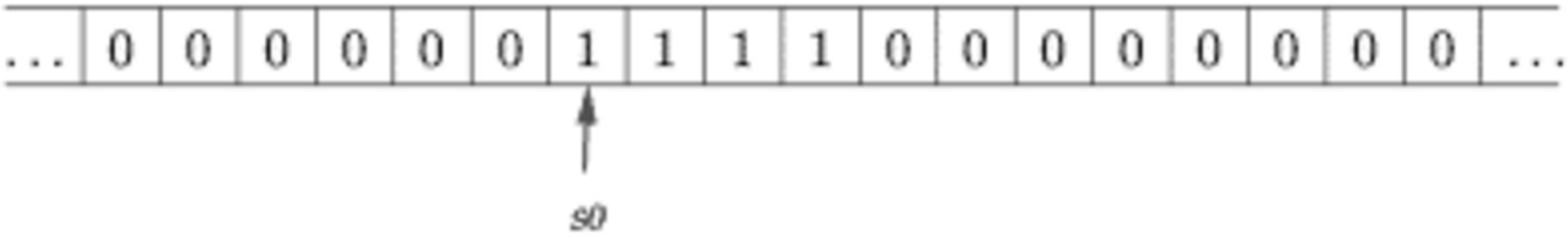
Turing Machine
~ kind of state machine

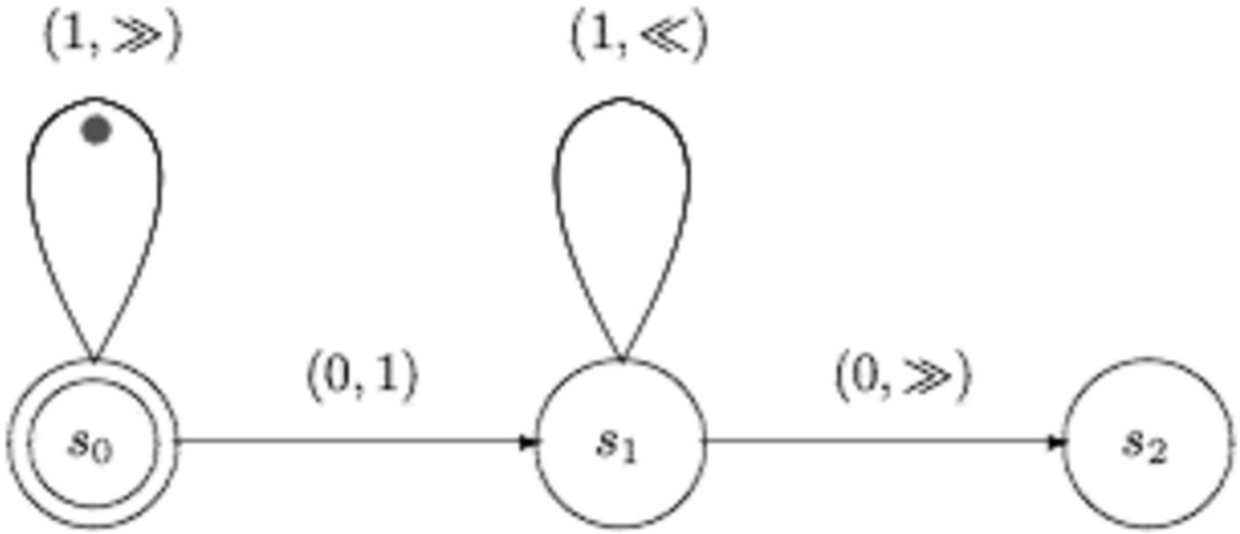


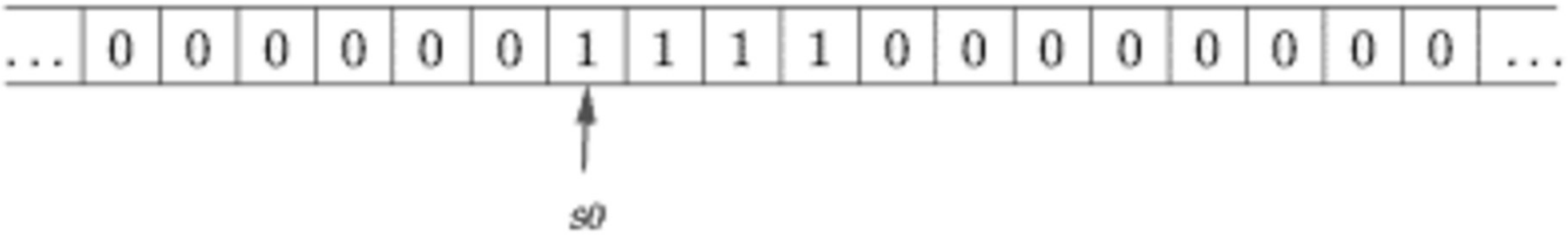
Successor (add-one) function

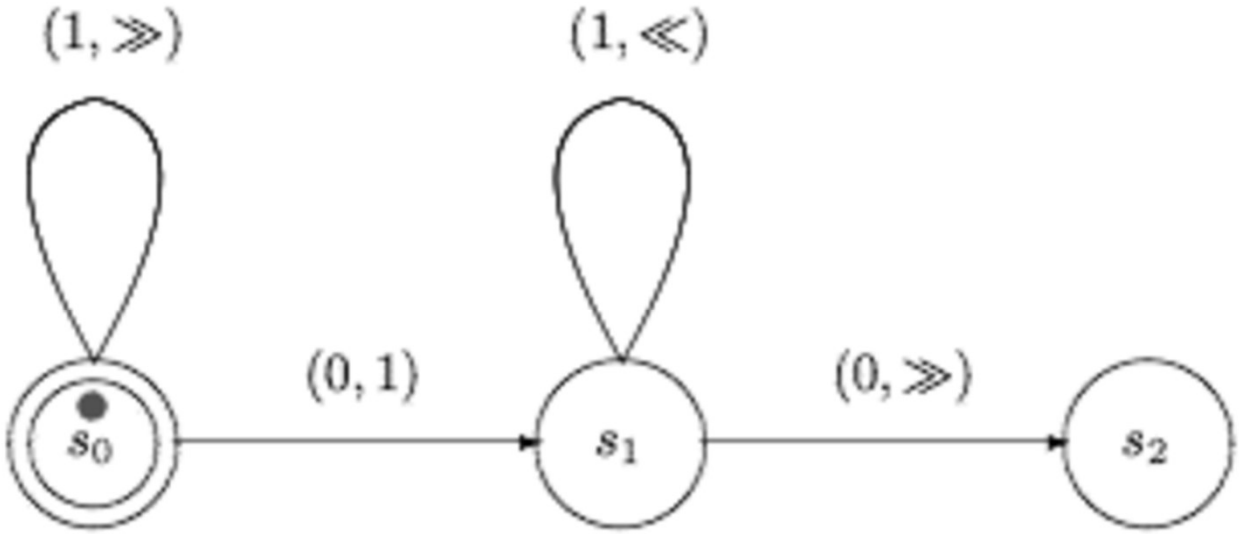
assuming that number n as a block of n+1 copies of the symbol ‘1’ on the tape (here, n=3)



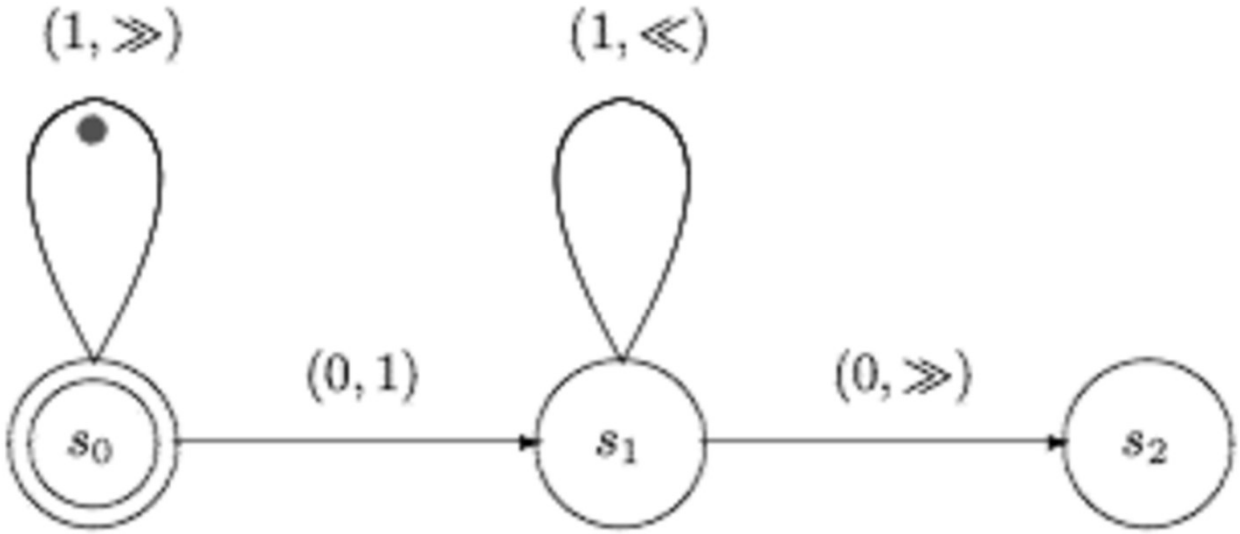



(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

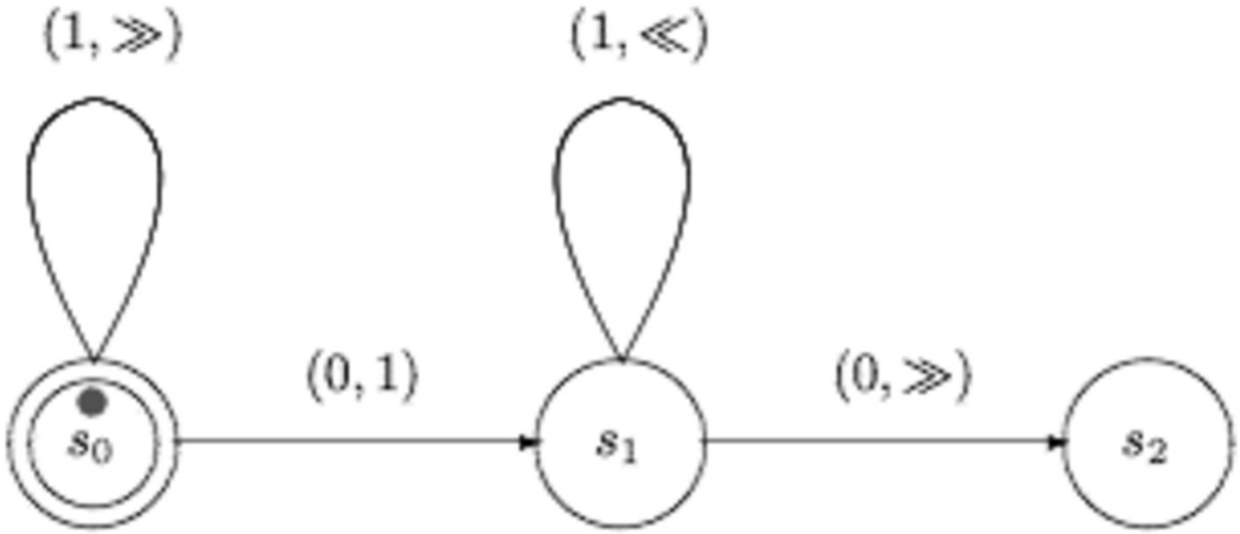

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

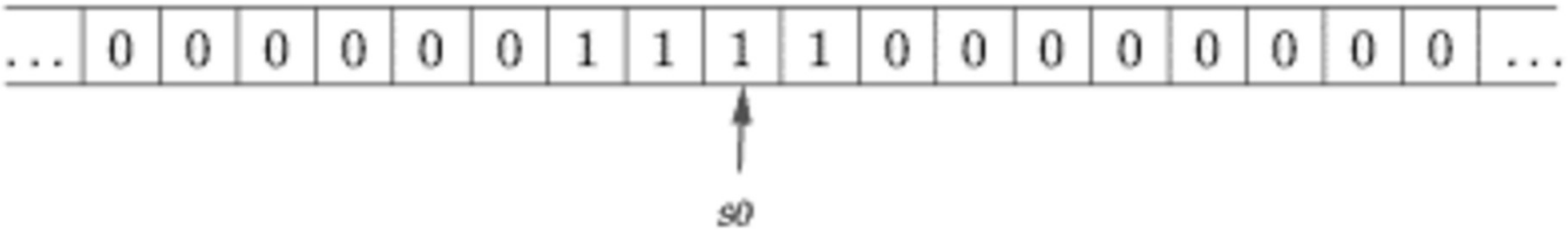

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

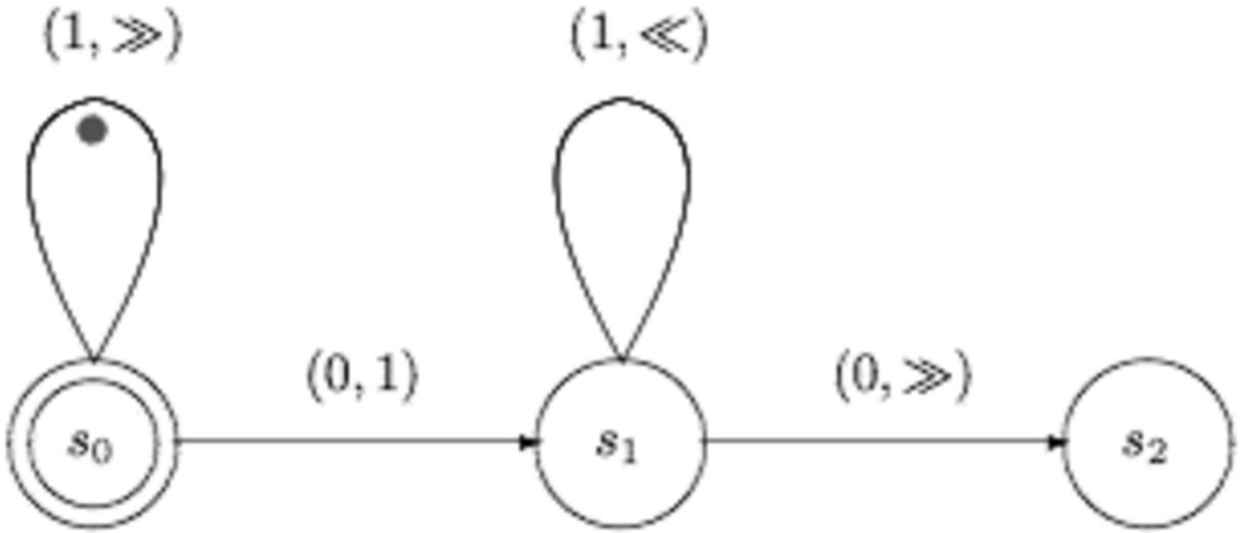

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

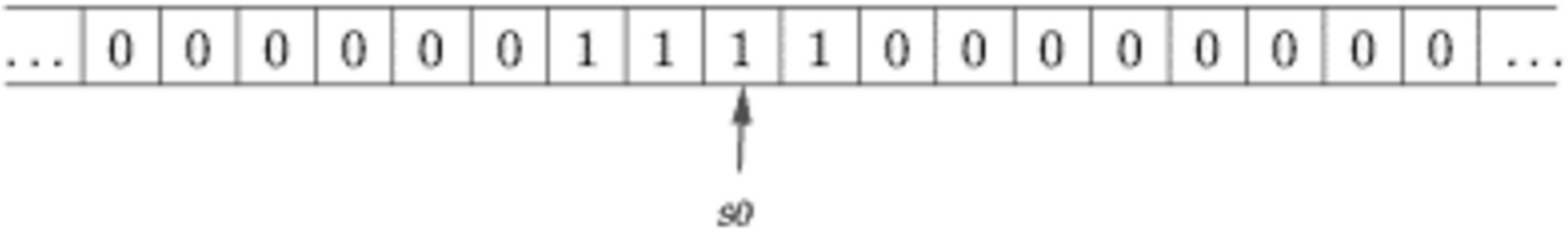

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

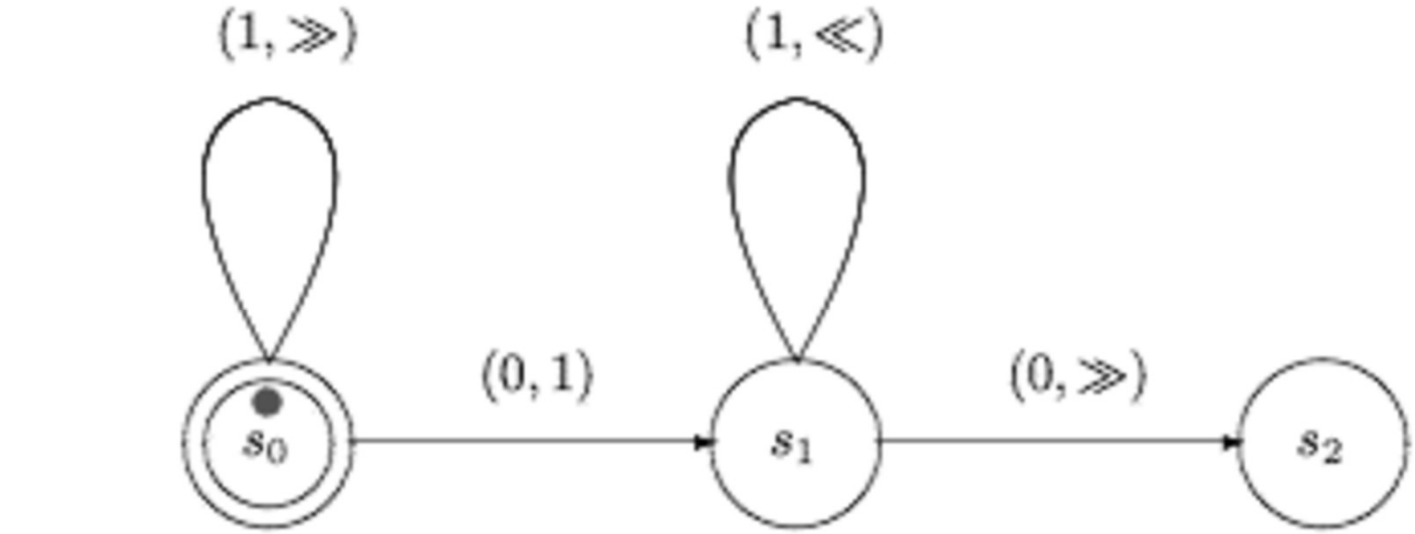

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

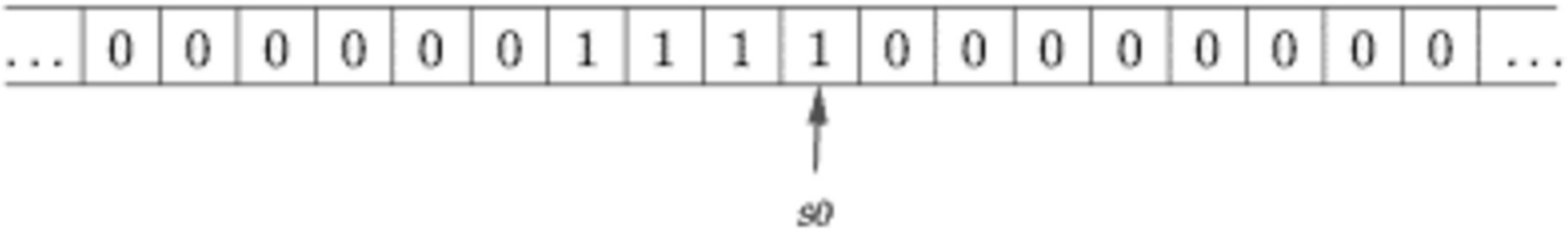

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

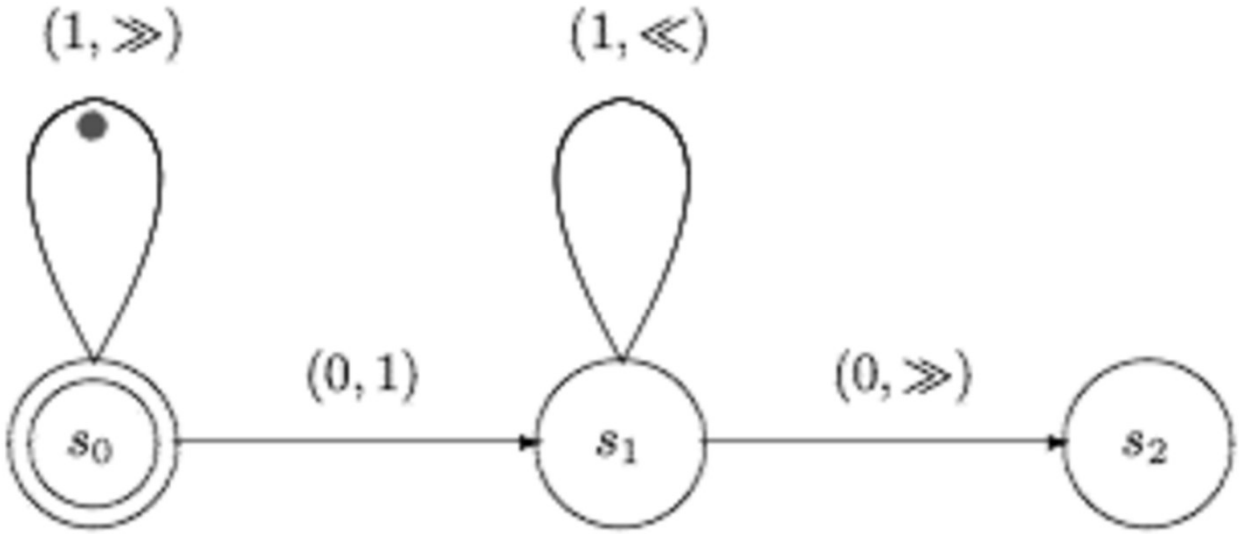

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

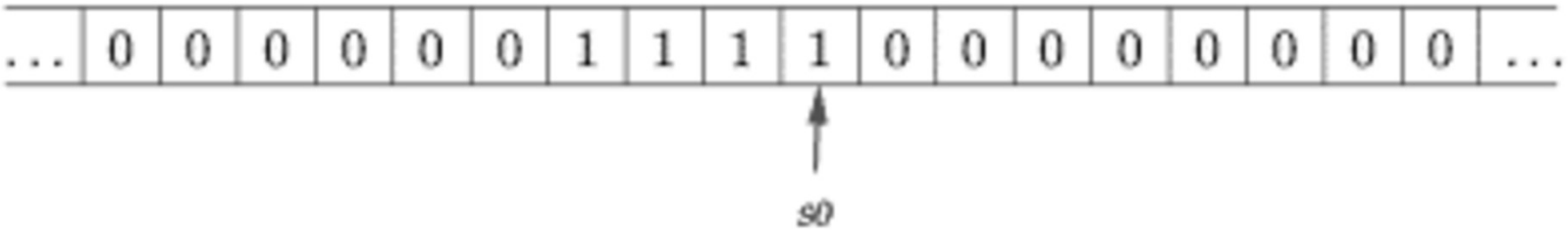

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

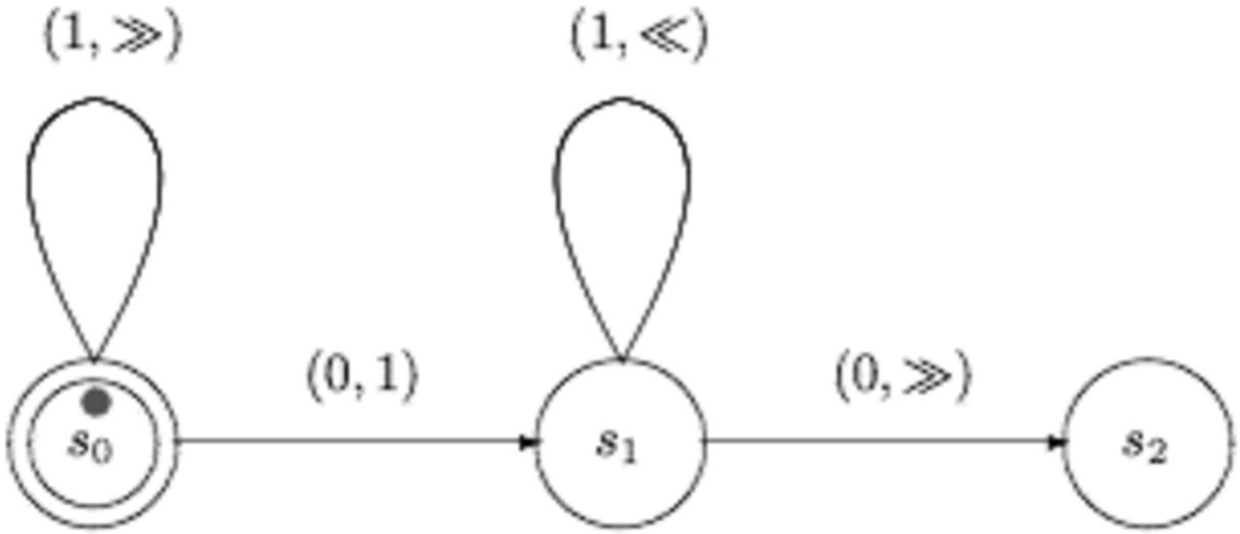

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0


(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

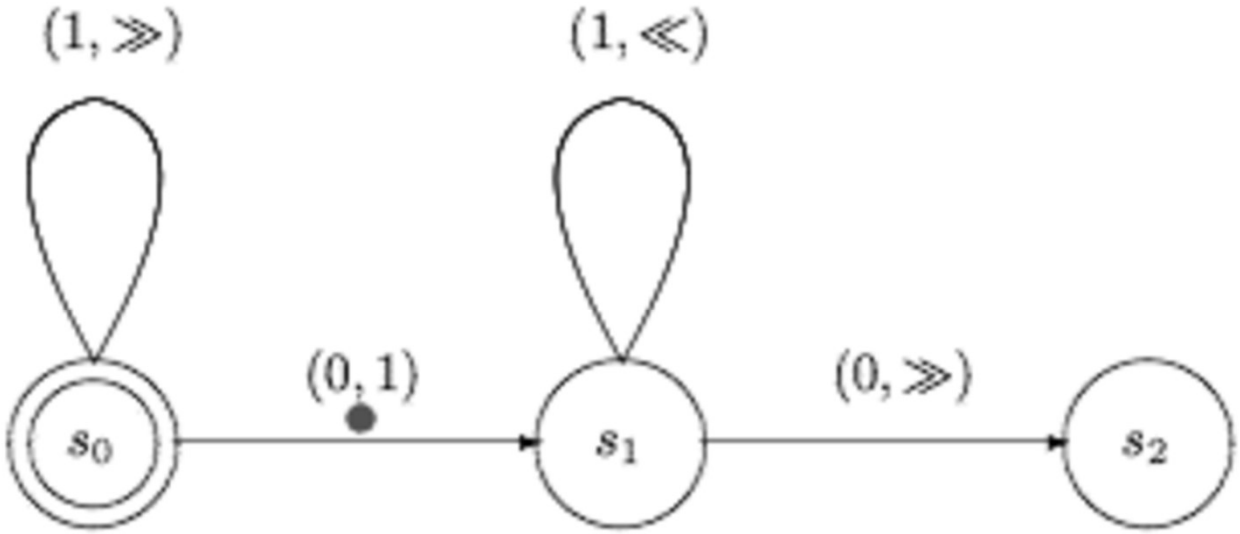

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0


(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

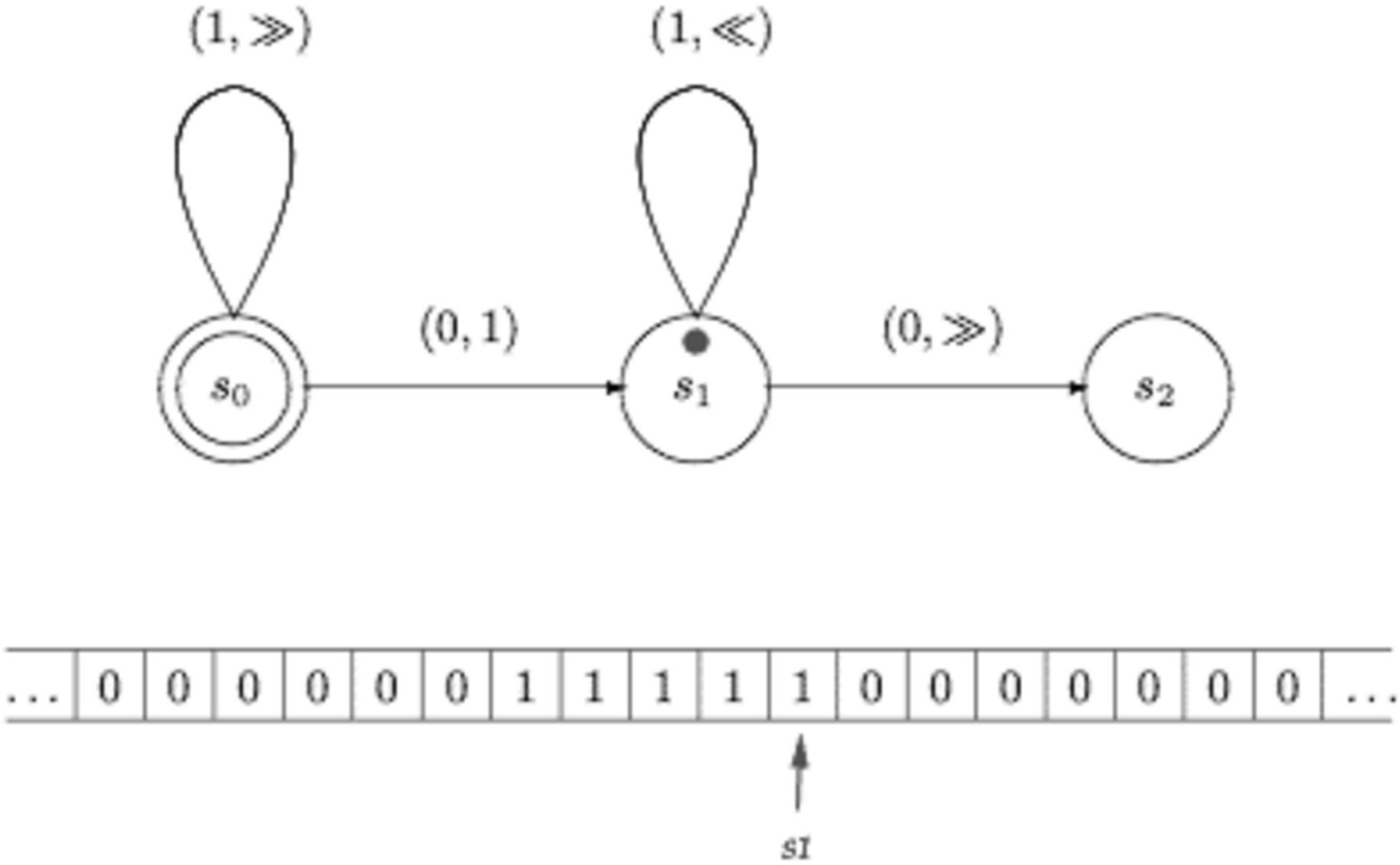

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

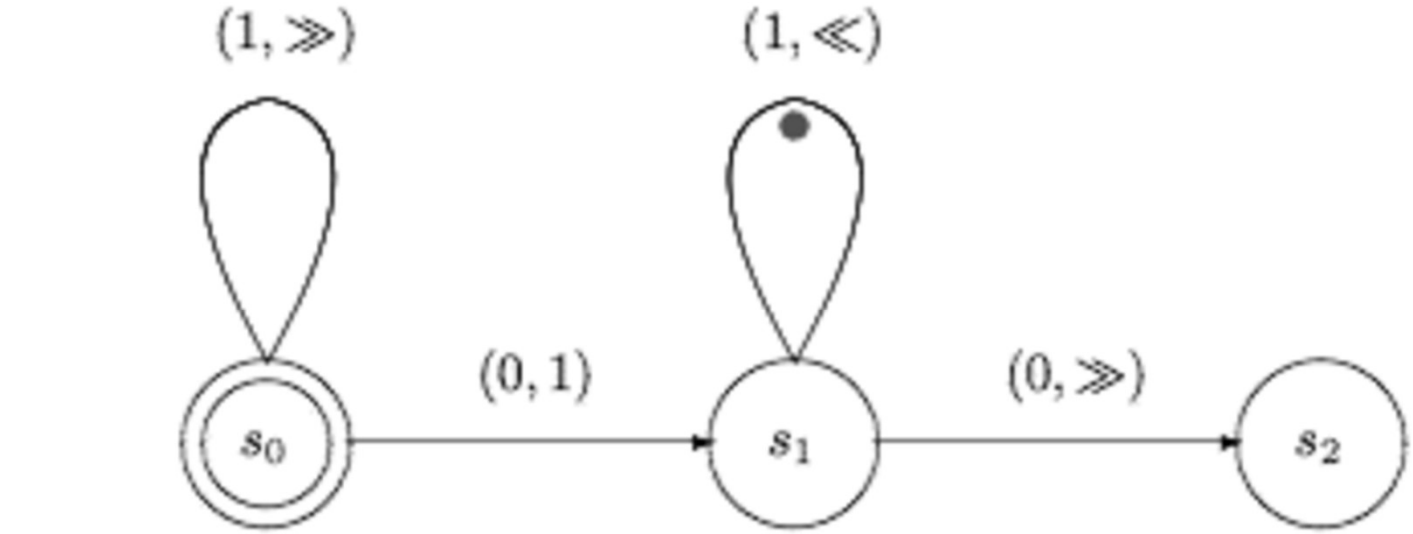

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

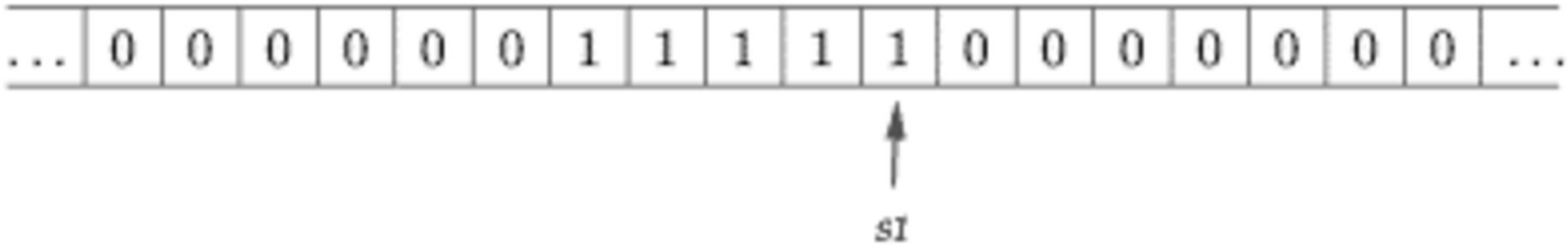

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

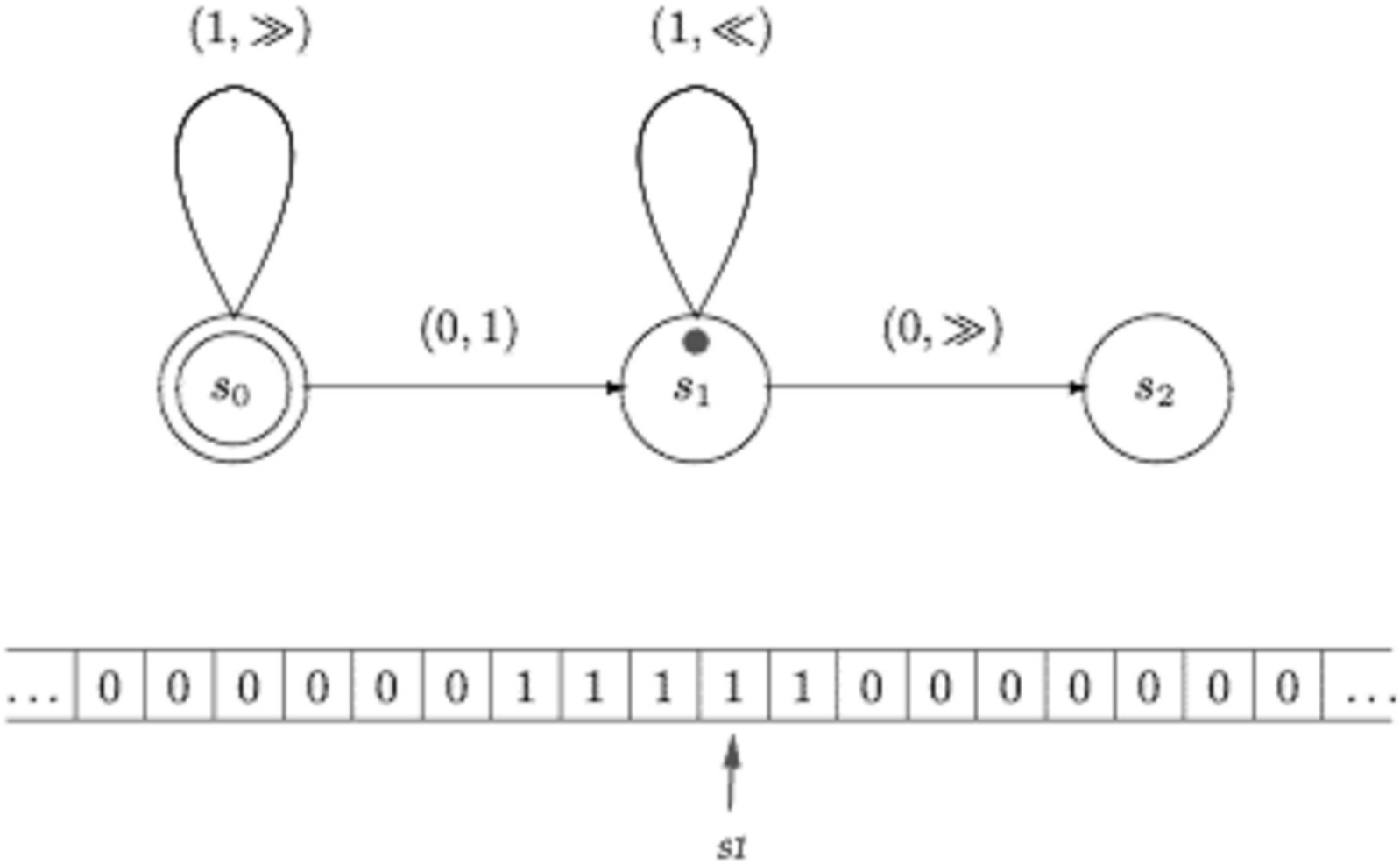

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

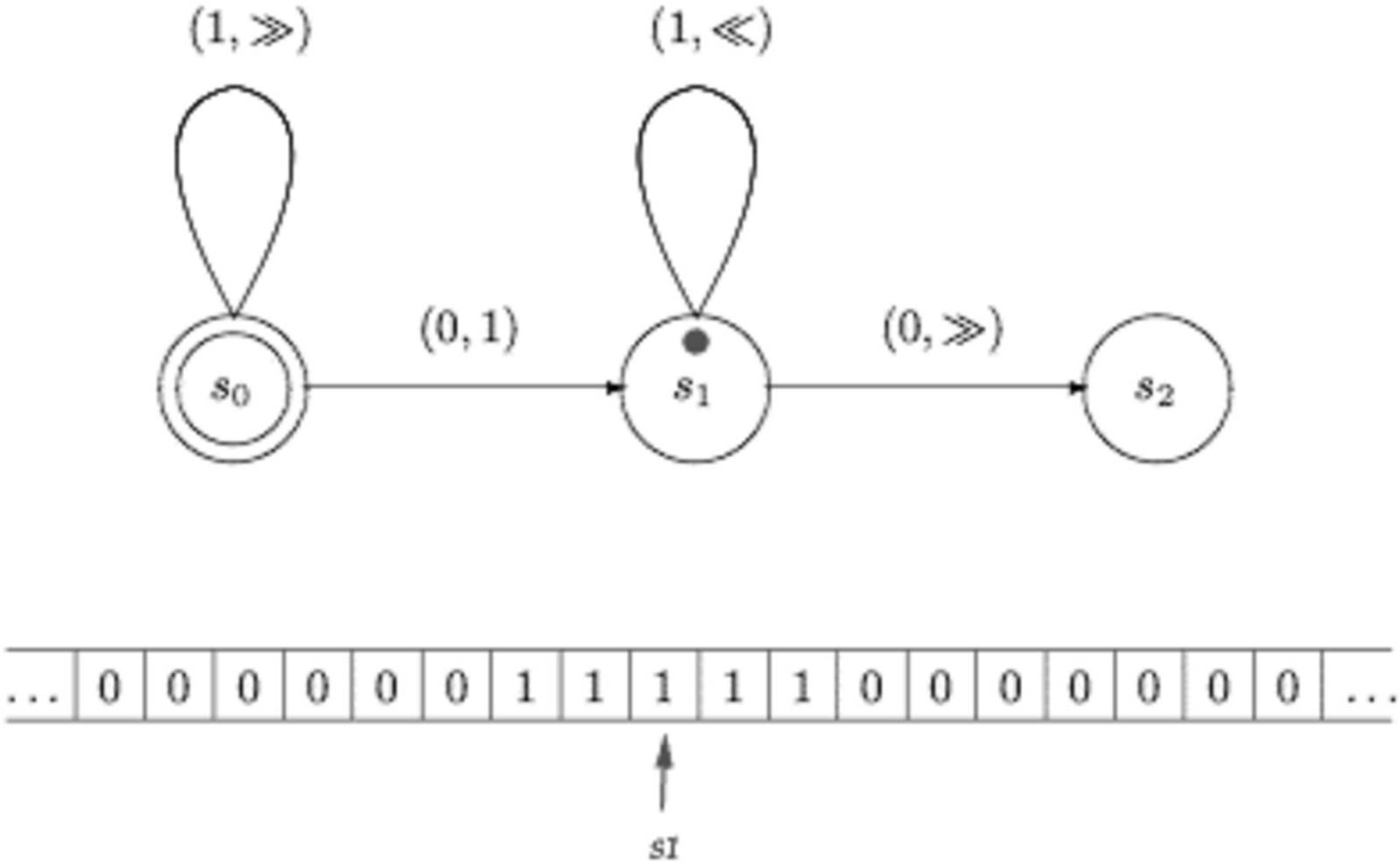

... 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 ...
s0

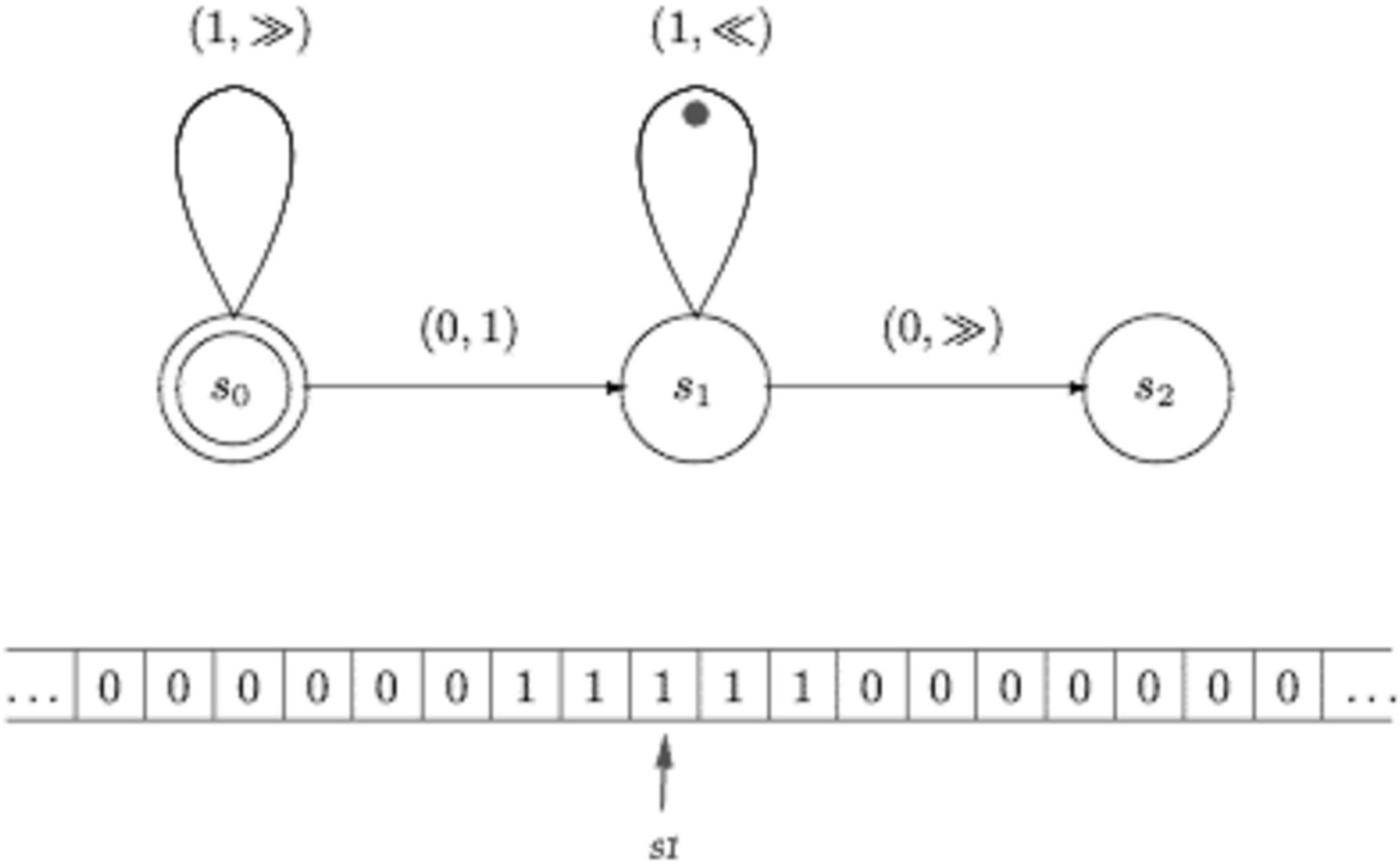

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

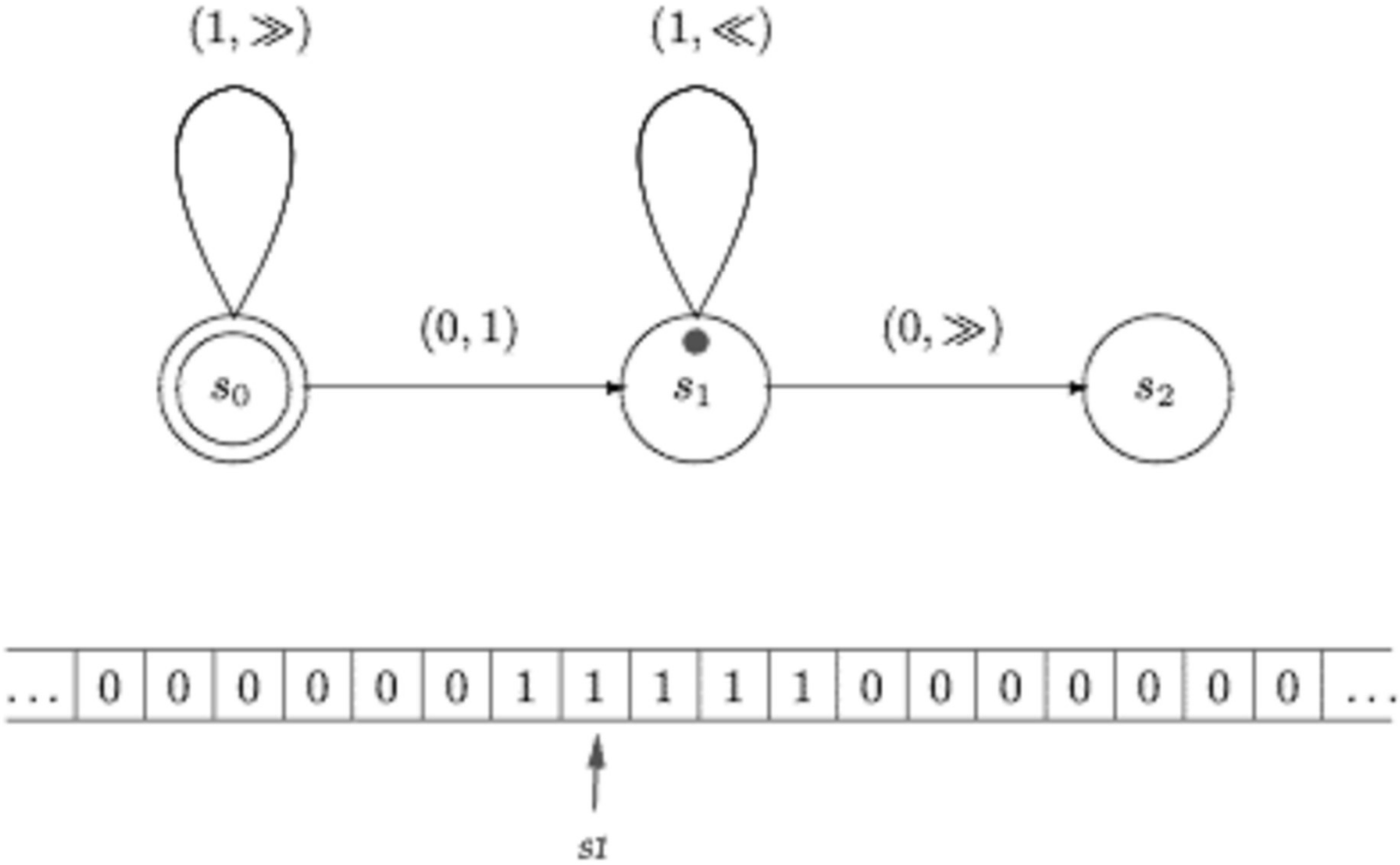

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

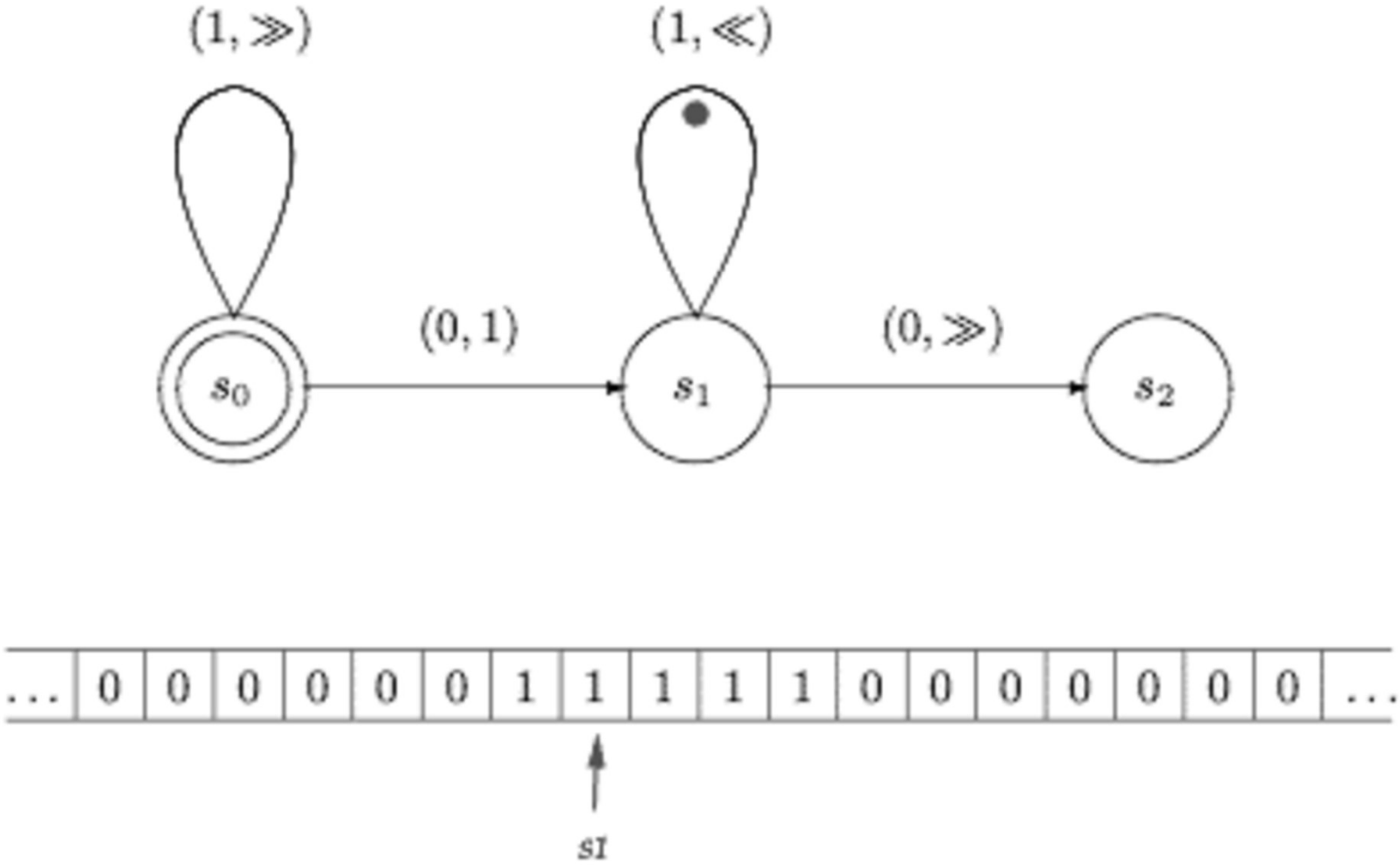

... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

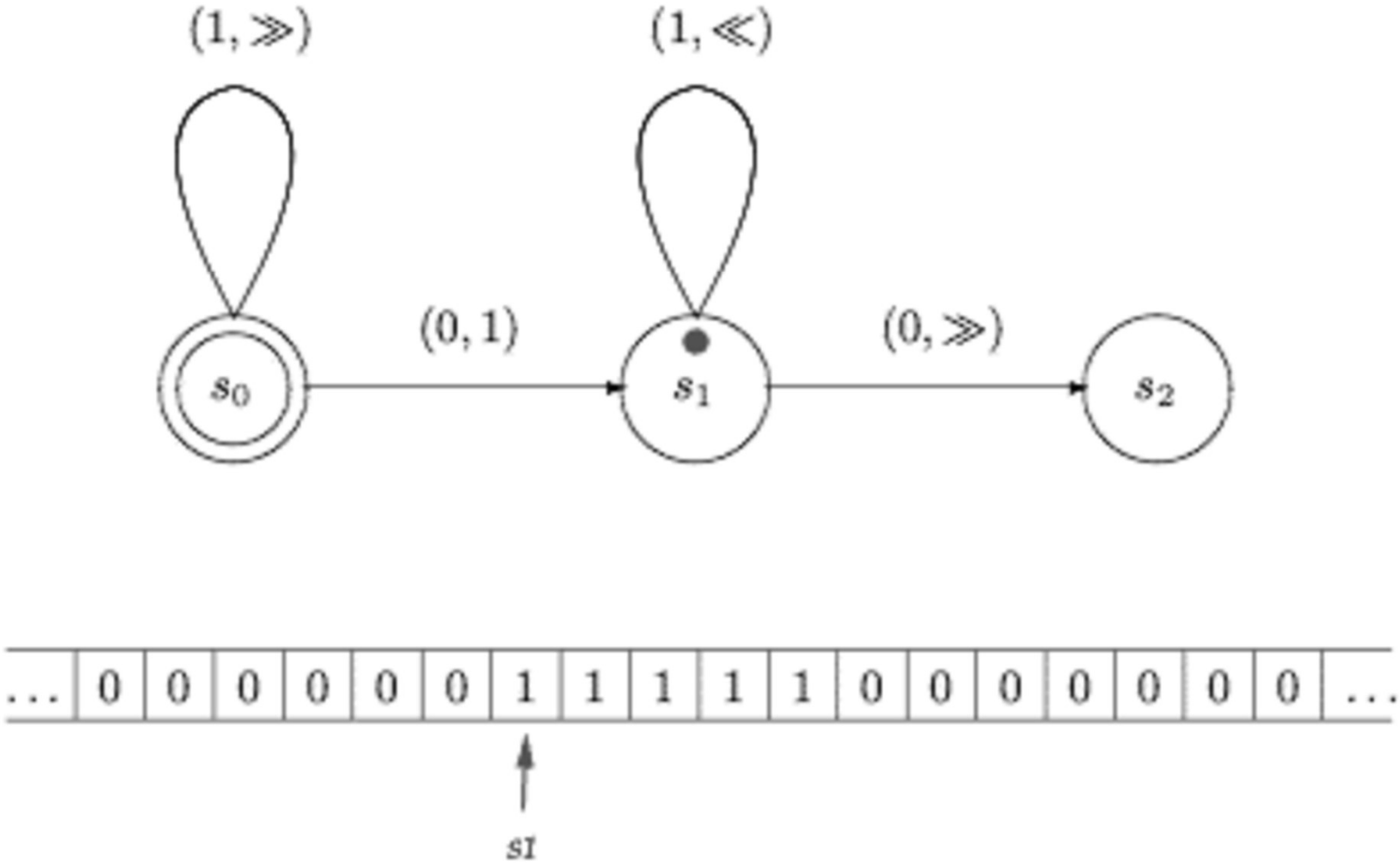

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

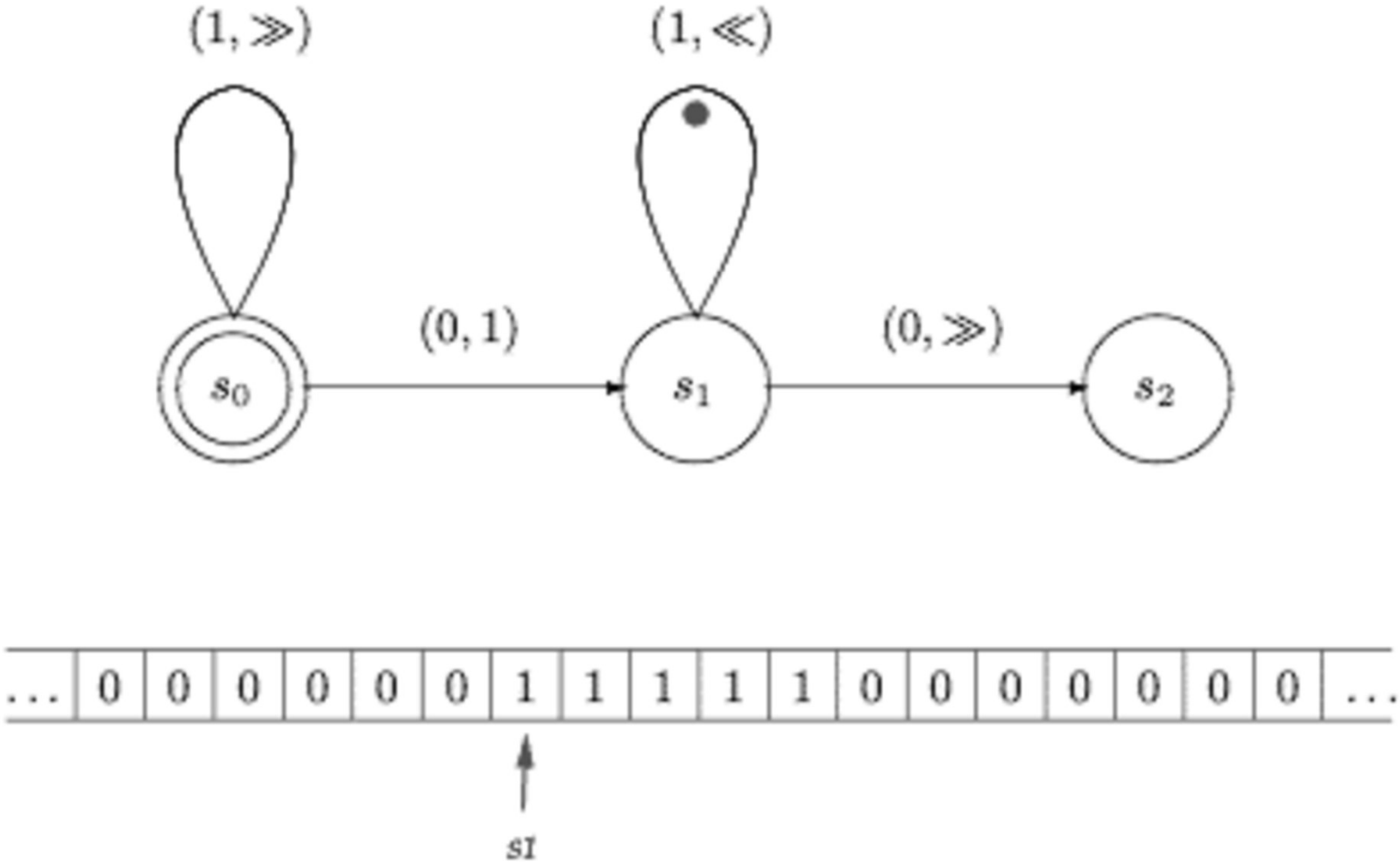

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

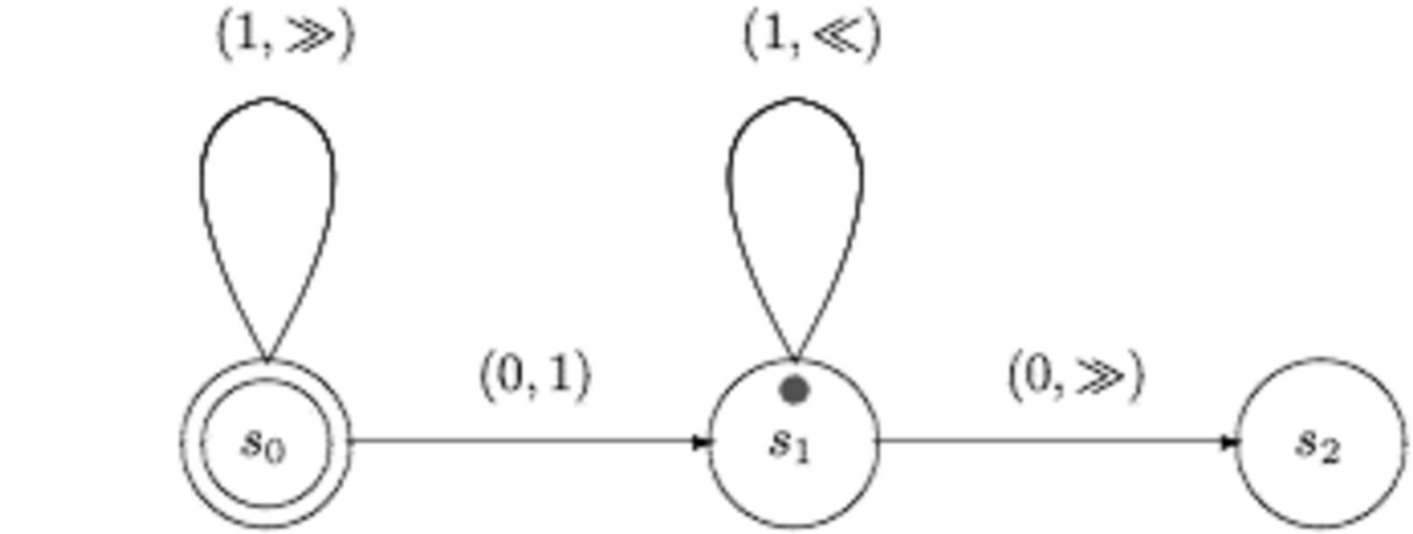

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI


(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

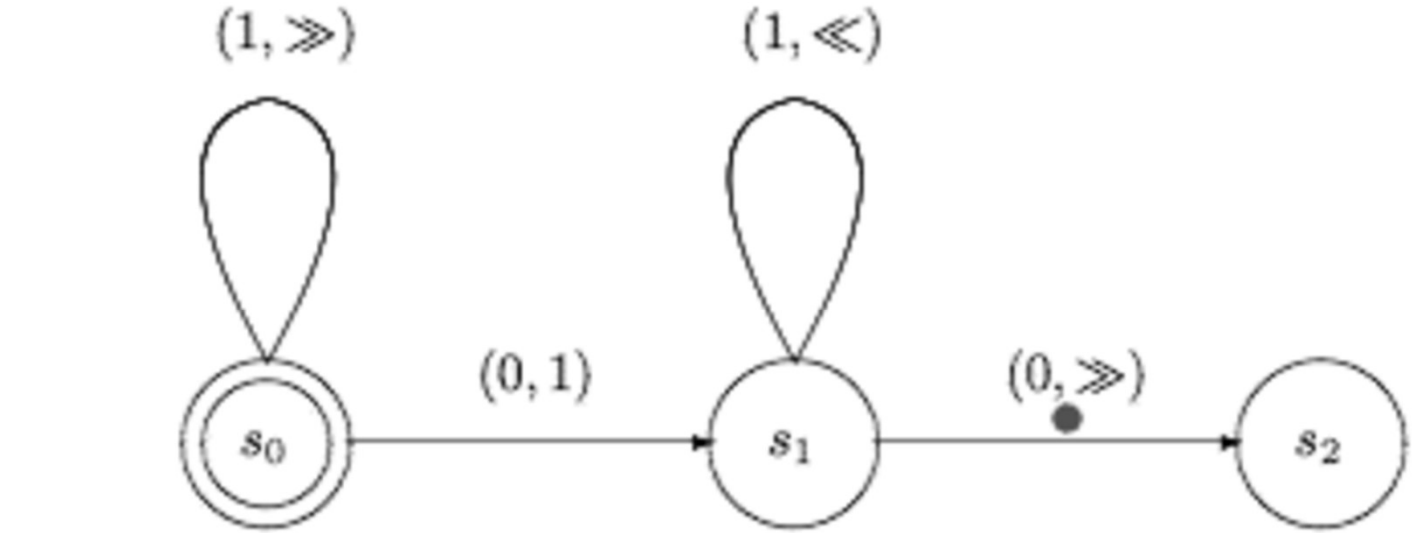

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI


(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

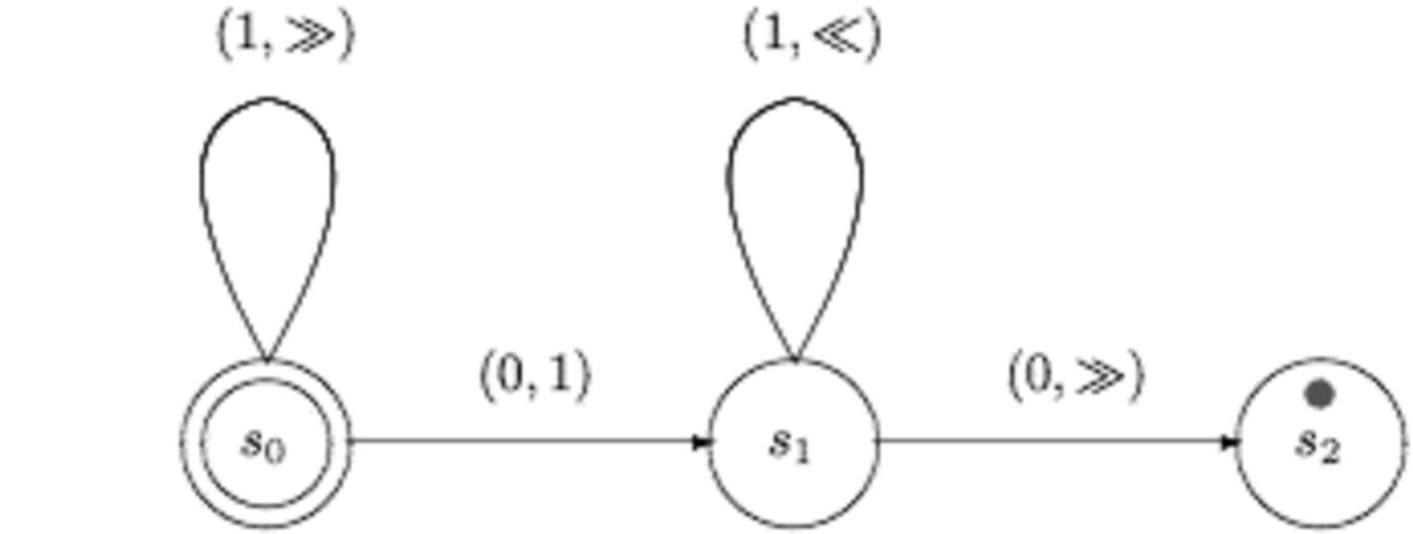

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2
... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI


(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2


... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI

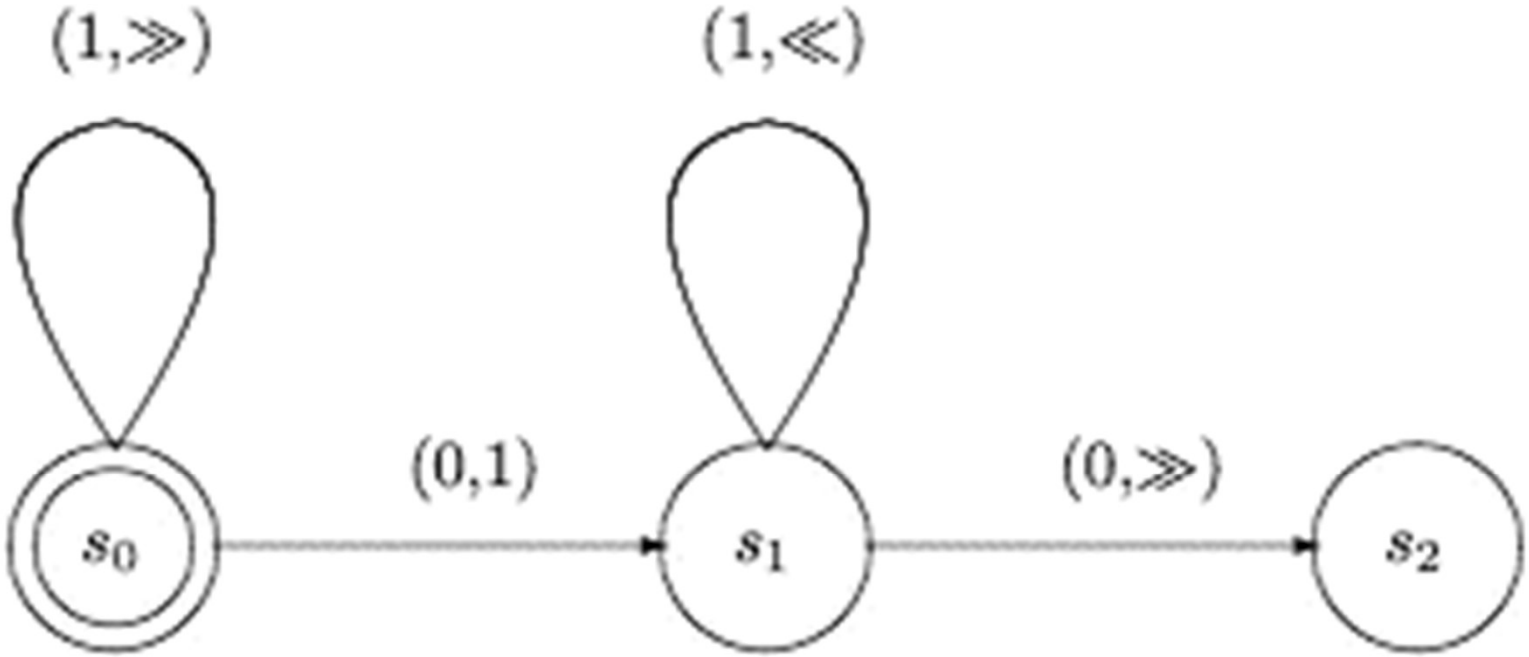

(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2

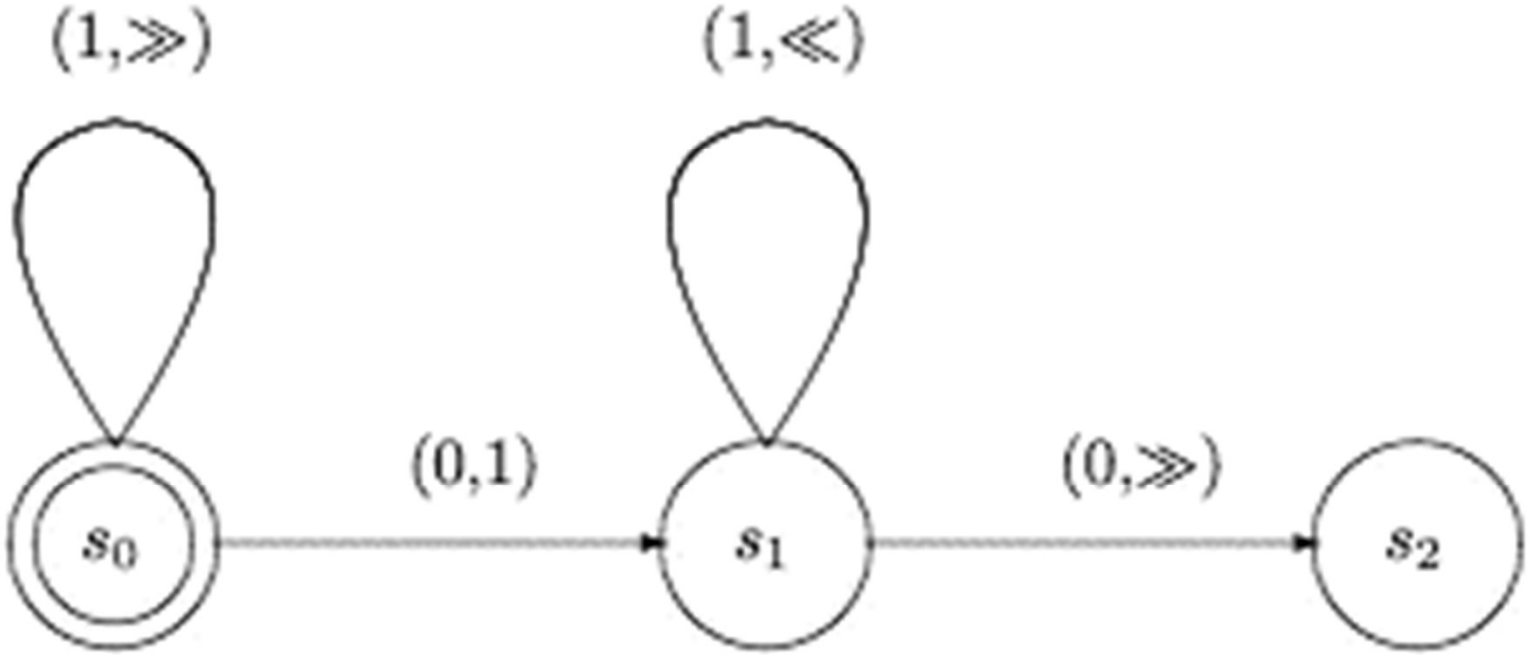

... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
sI


(1, ≫)
(1, ≪)
(0, 1)
(0, ≫)
s_0
s_1
s_2


... 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 ...
s2

Question: what does it compute?



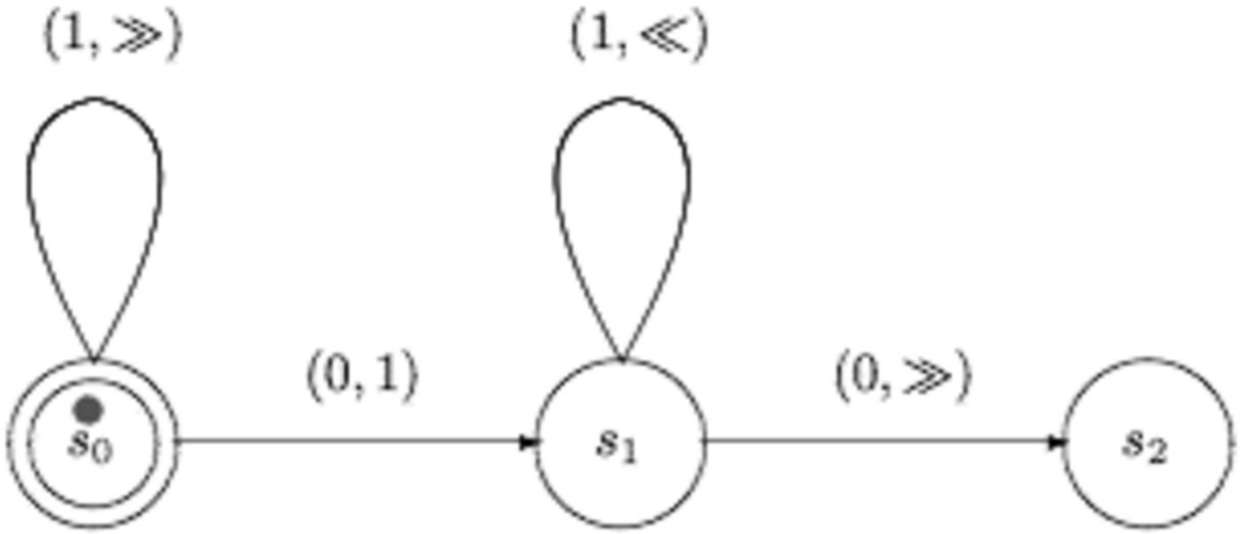


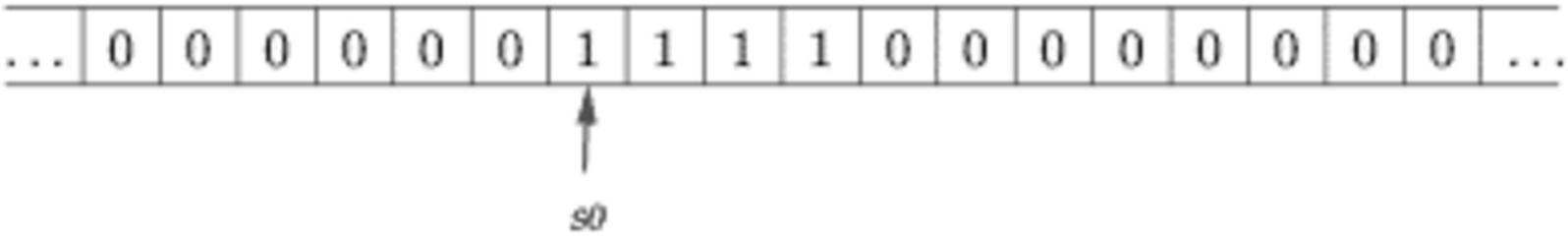

```
function succ (n) {
        return n + 1;
}
```

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

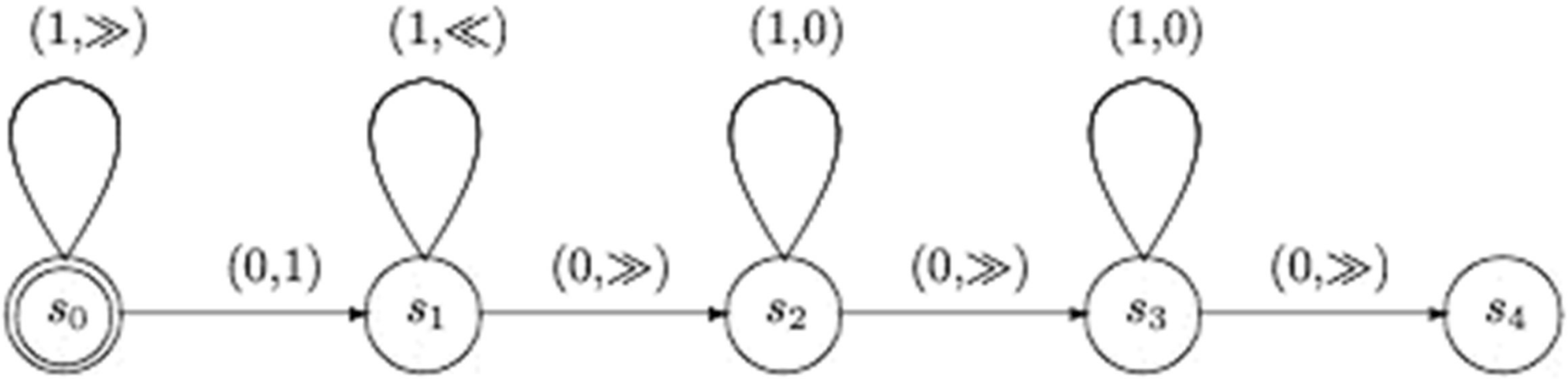
Successor (add-one) function

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



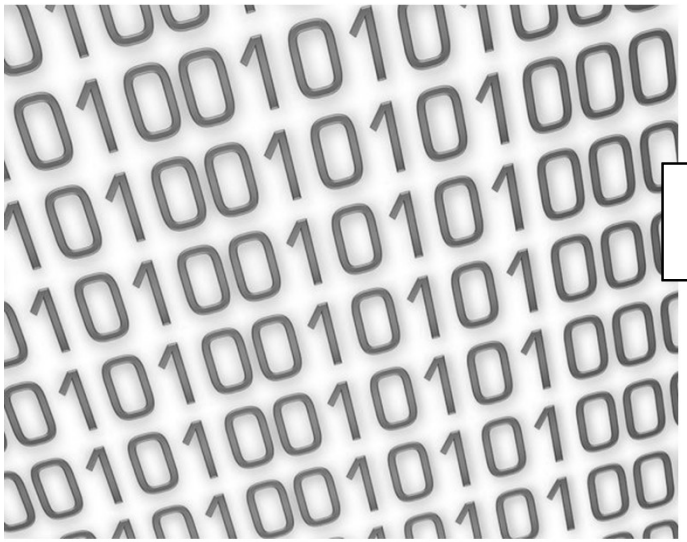


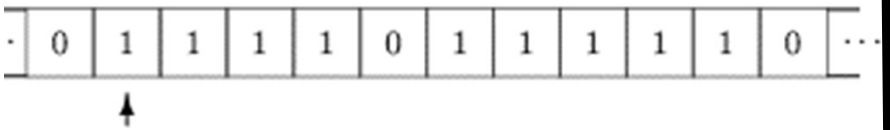
Addition of n+m

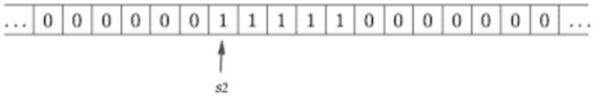


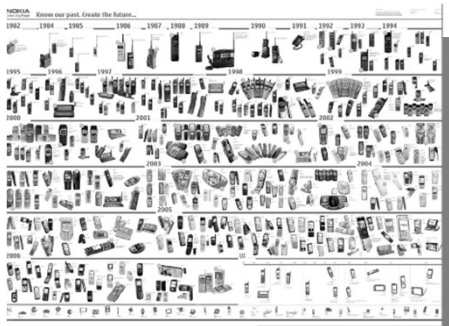
http://graphics.stanford.edu/~seander/bithacks.html
Maybe you prefer to use bit operations?

The (Hi)Story of Software Engineering & Computer Science







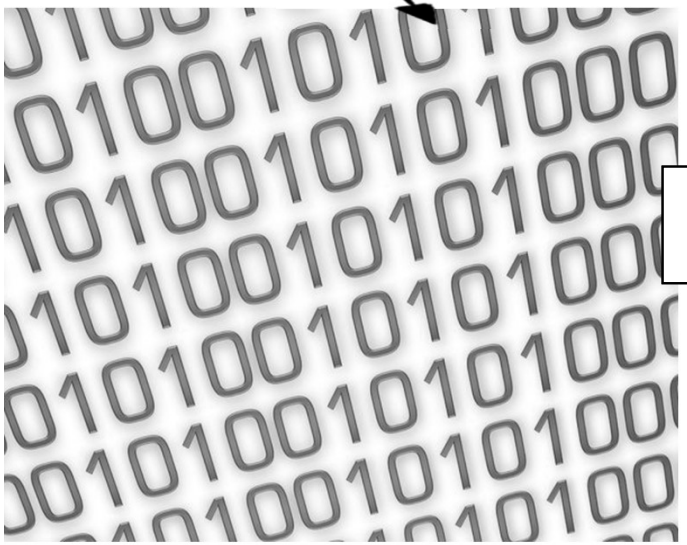


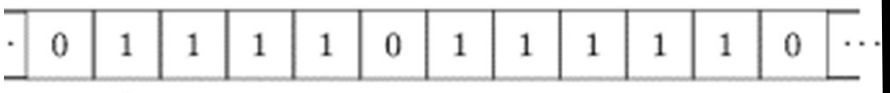
orange™

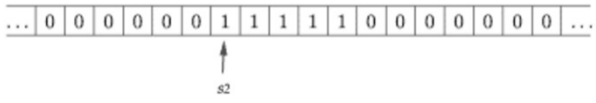
Google

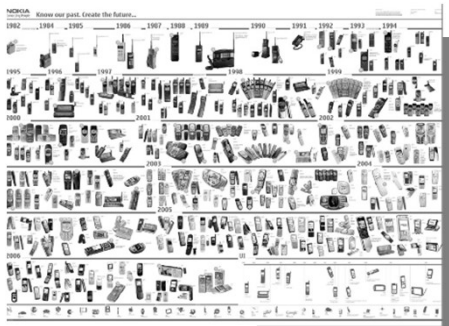
twitter

Software Languages









orange™

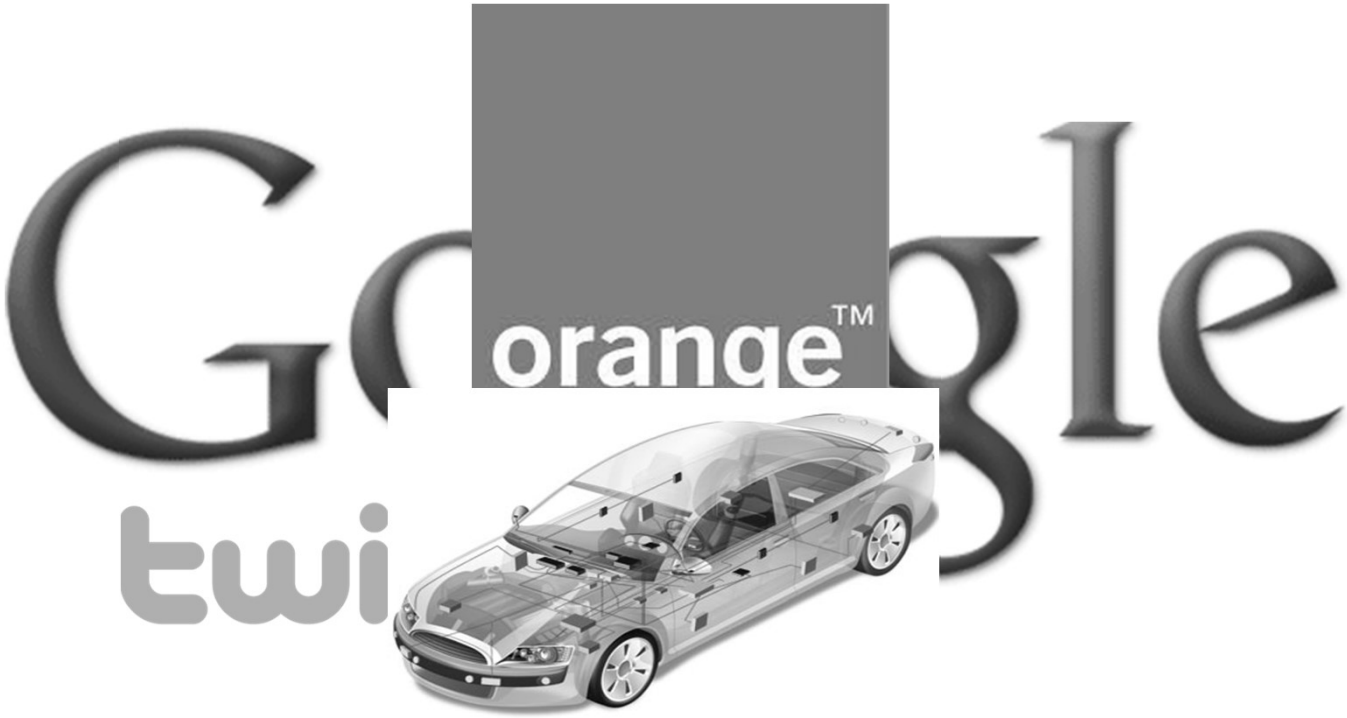
Google

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

Implementing a Turing m

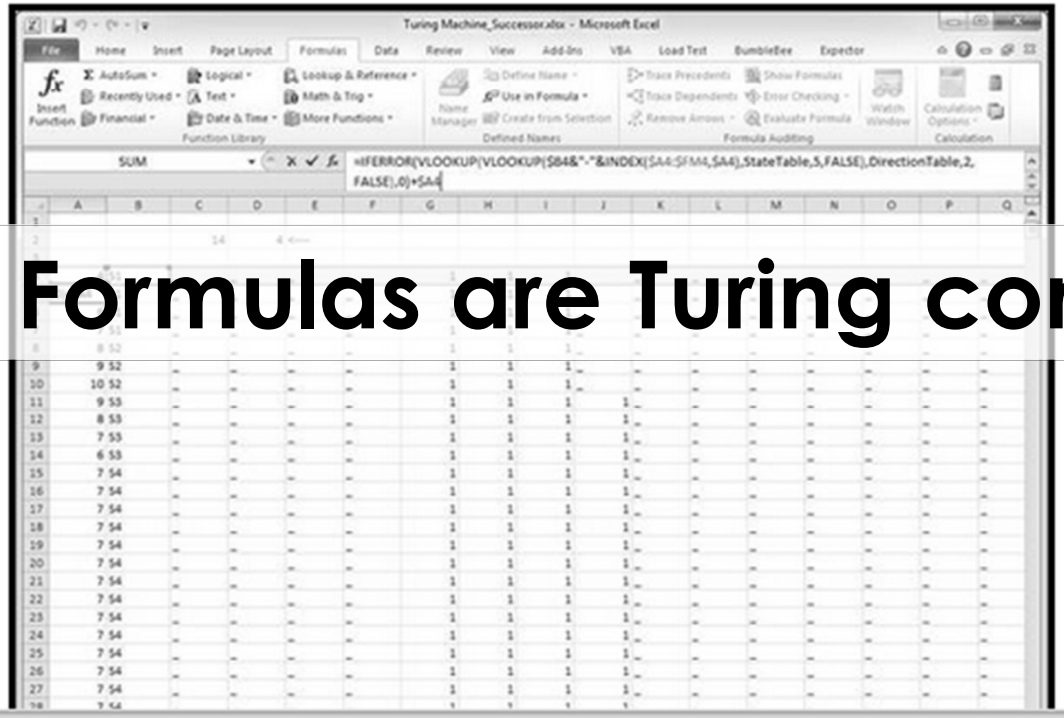
boingboing.net/2013/09/20/implementing-a-turing-machine.html

SUBMIT A LINK FEATURES REVIEWS PODCASTS VIDEO FORUMS MORE

boing boing

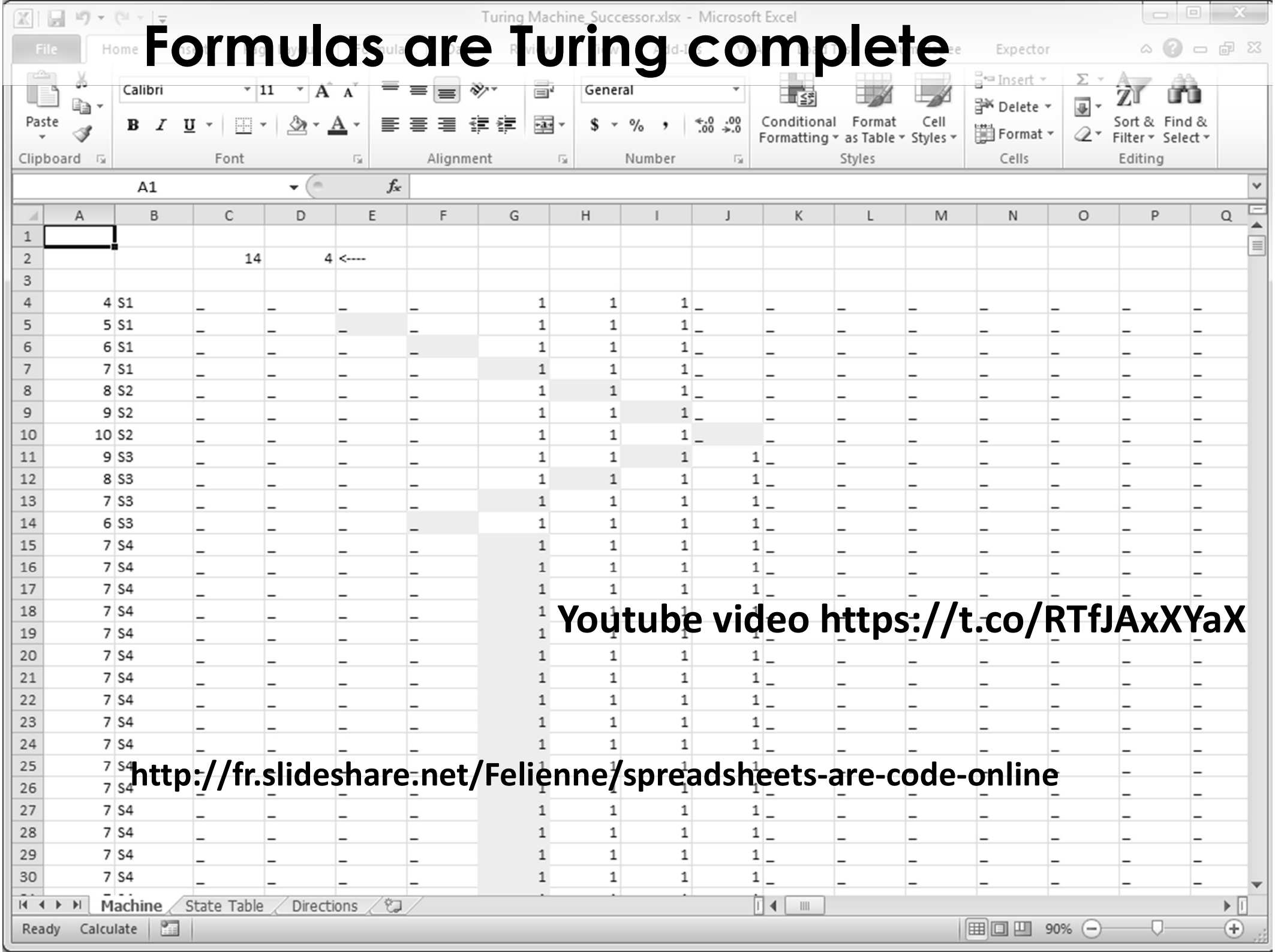
Implementing a Turing machine in Excel

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



Formulas are Turing complete

Formulas are Turing complete



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?

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

Questions to the audience

- 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?

The answer to such « though-provoking » questions seems obvious at first glance

  - Help to define the good properties of software languages we expect
  - Help to understand why there is still innovation in language design

Programming the Turing Machine

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

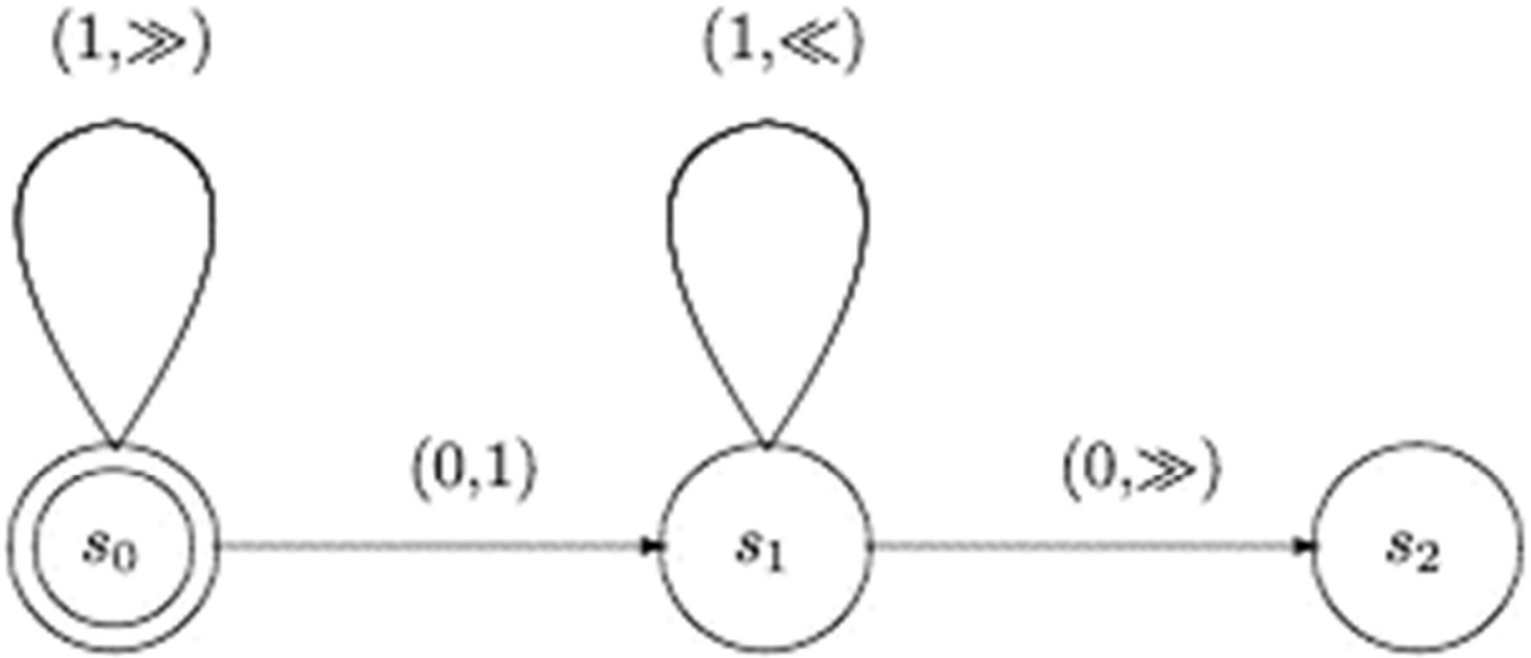
Software Languages

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?

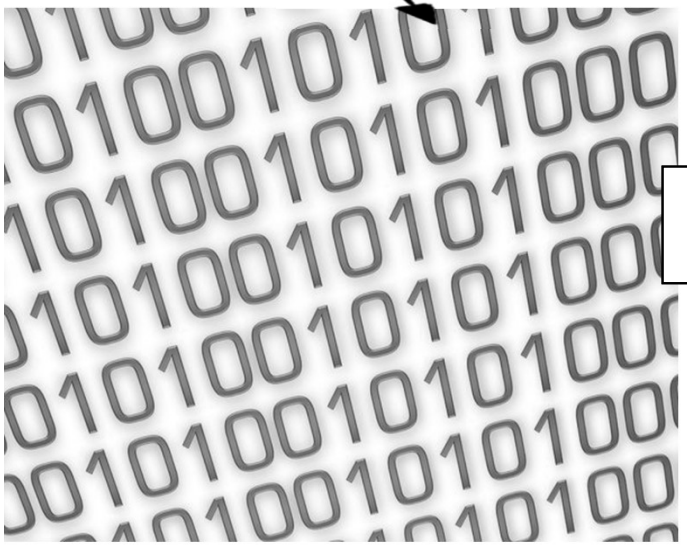


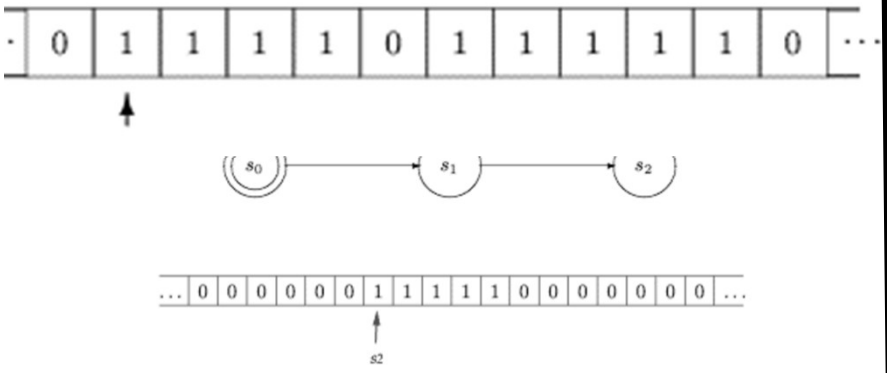
Performance, usability, productivity, reusability, safety, expressiveness, learnability.

Qualities and challenges

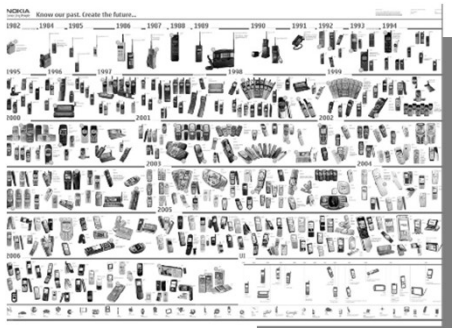
- Cognitive dimensions (see references after)
- Abstractions
  - Eg Kramer “Abstraction and Modelling - A Complementary Partnership” MODELS’08
- Separation of concerns/modularity
  - Eg Tarr et al., ICSE’99
- Scalability
  - Growing a language (like Scala)
- Performance
- …

Languages





Complex Systems



orange™

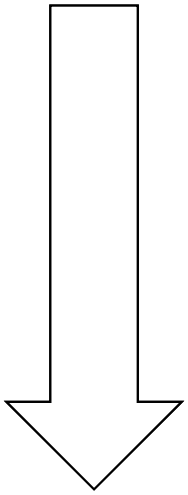
Google

Instagram

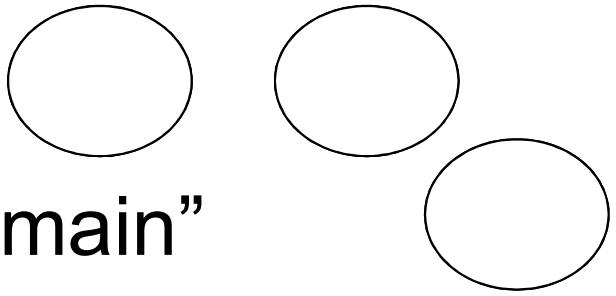
twitter

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”)

COGNITIVE PSYCHOLOGY

How Language Shapes Thought

The languages we speak affect our perceptions of the world

By Lera Boroditsky

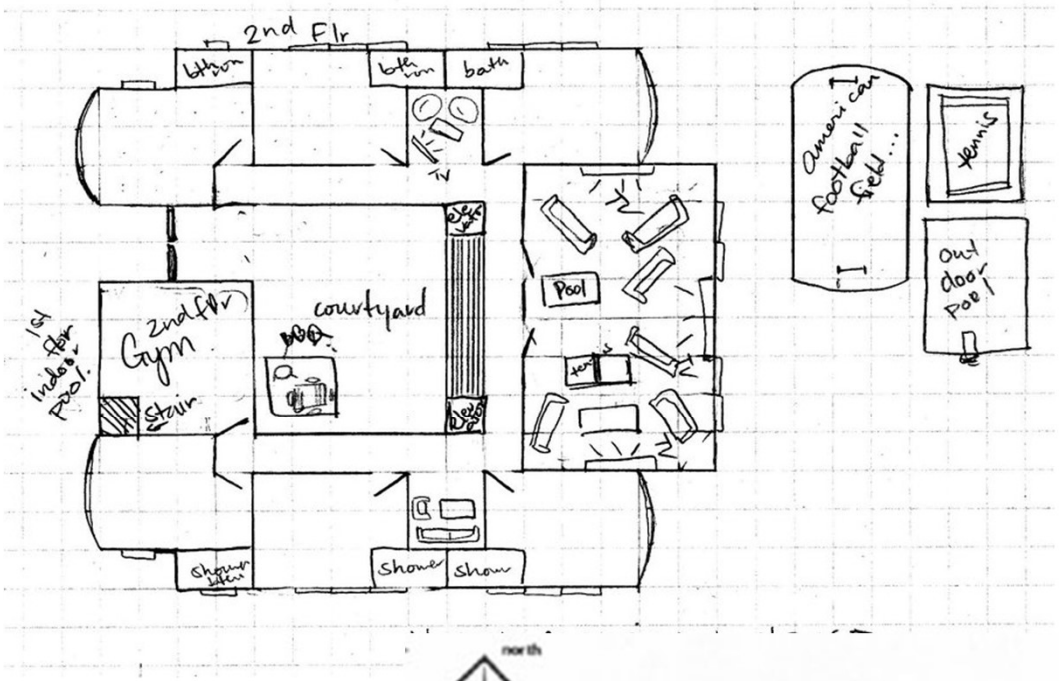
"Even variations in grammar can profoundly affect how we see the world."

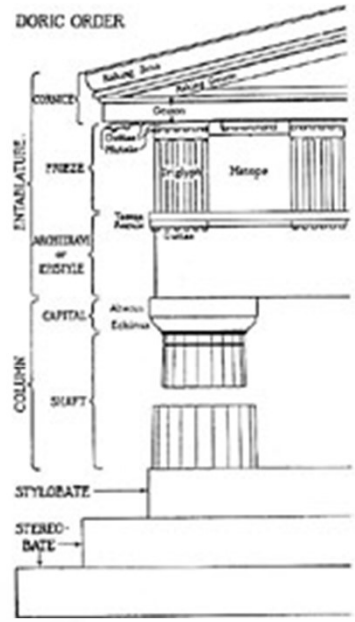
She's talking about real languages; **what about synthetic, programming languages?**

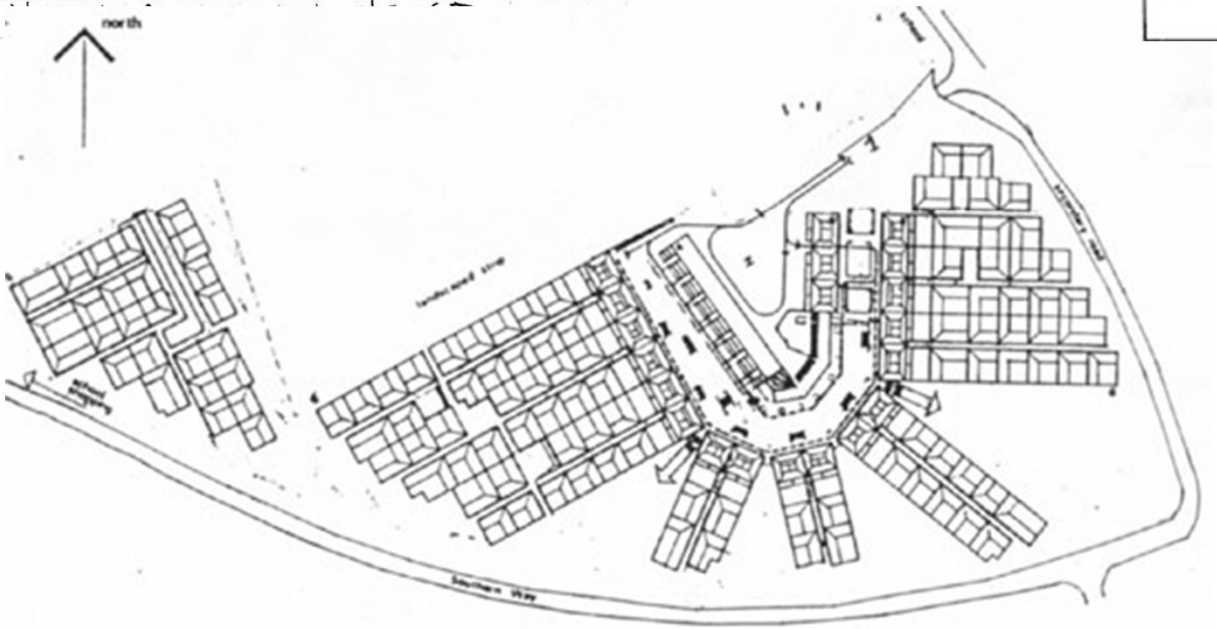
What is a language?

- « 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. »

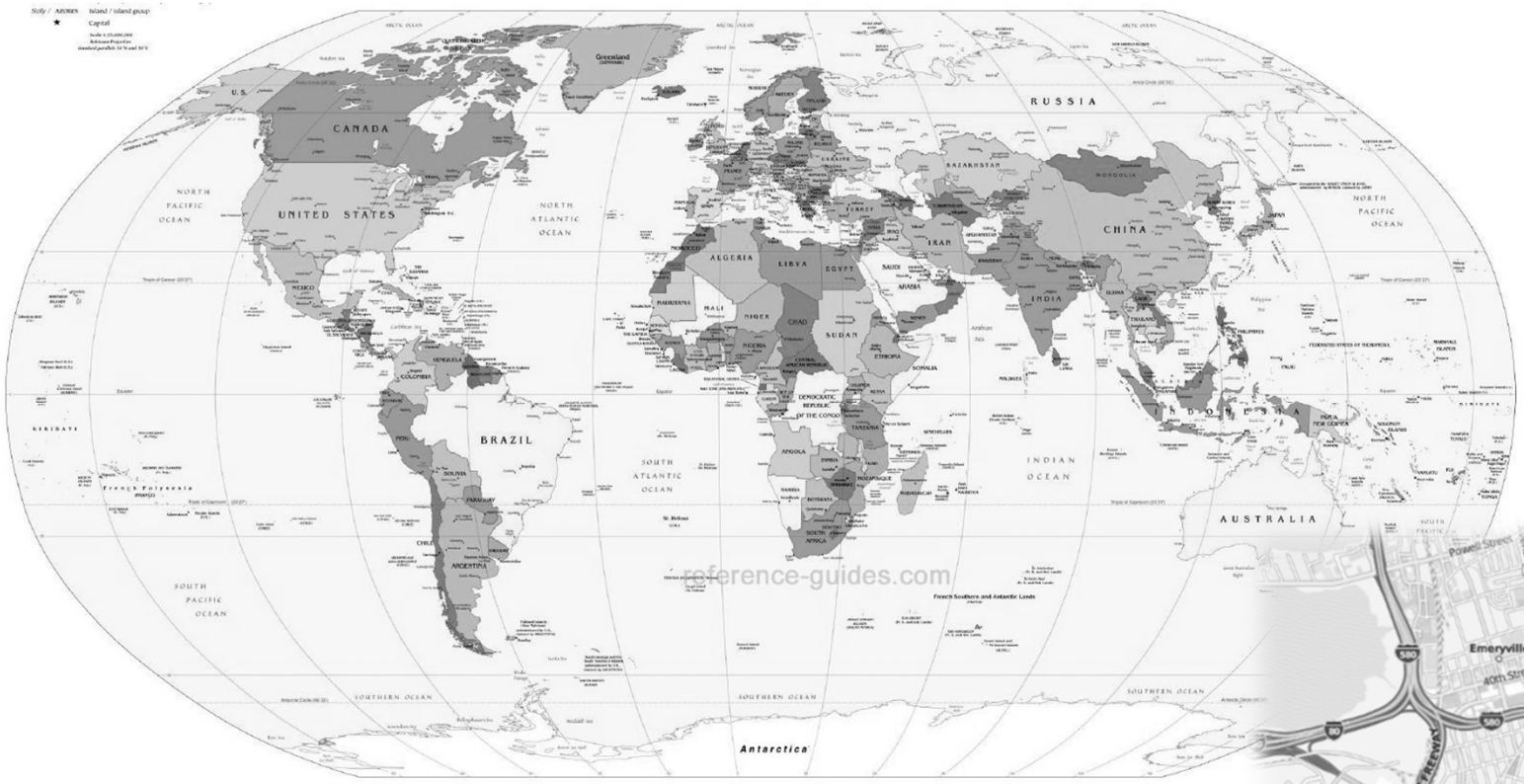
Architecture



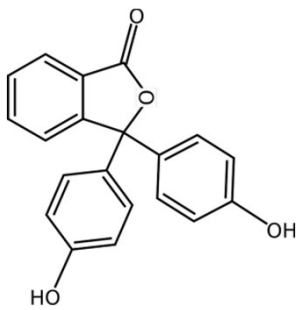


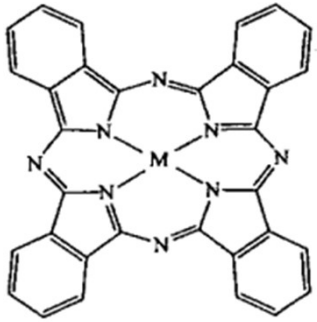


Cartography



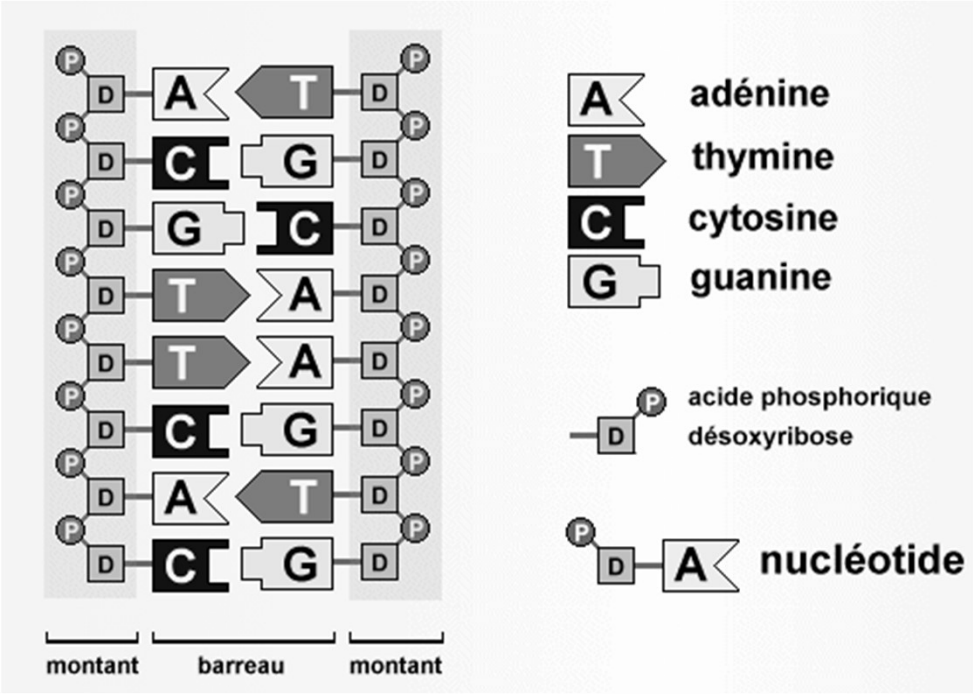
Biology



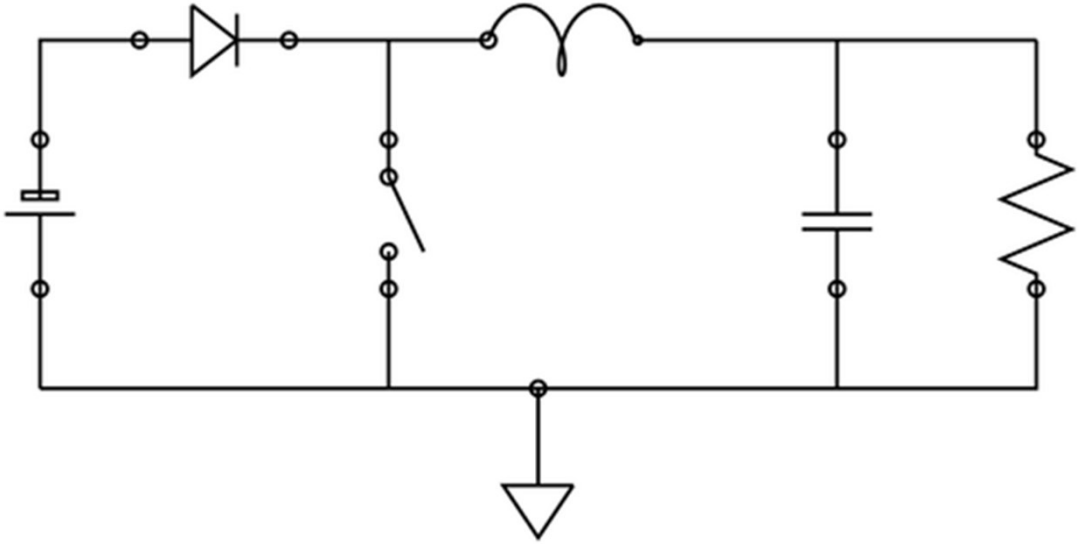


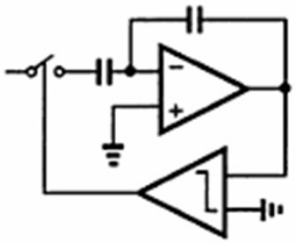
phthalocyanine

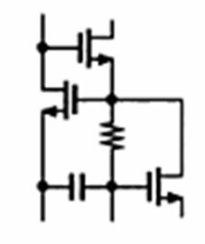
```
          60          70          80          90         100
AGACCCCCAG  CAACCCCCGG  GGGCGTGCGG  CGTCGGTCGT  GTCGTGTGAT
         160         170         180         190         200
AGACCCCGCG  TACGAATGCC  GGTCCACCAA  CAACCCGTGG  GCTTCGCAGC
         260         270         280         290         300
CTGCCGGGCA  TGTACAGTCC  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
GAAGAGGTGG  CGCCGATGAA  GAGACTATTA  AAGCTCGGAA  ACAAGGTGGT
         660         670         680         690         700
ATAGTGGTTA  ACTTCACCTC  CAGACTCTTC  GCTGATGAAC  TGGCCGCCCT
         760         770         780         790         800
AAAATATACA  GGCATTGGGC  CTGGGGTGCG  TATGCTCACG  TGAGACATCT
         860         870         880         890         900
CCTGGAGGAG  GTTCGCCCGG  ACAGCCTGCG  CCTAACGCGG  ATGGATCCCT
         960         970         980         990        1000
AGCAACACCC  AGCTAGCAGT  GCTACCCCCA  TTTTTTAGCC  GAAAGGATTC
        1060        1070                Pvu II site 1090        1100
TGCCGCAGCA  ACTGGGGCAC  GCTATTCTGC  AGCAGCTGTT  GGTGTACCAC
        1160        1170        1180        1190        1200
ACTTGATCTA  TATACCACCA  ATGTGTCATT  TATGGGGCGC  ACATATCGTC
        1260        1270        1280        1290        1300
CTGTCCATGT  ACCTTTGTAT  CCTATCAGCC  TTGGTTCCCA  GGGGGTGTCT
        1360        1370        1380        1390        1400
TGTTTGAGGG  GGTGGTGCCA  GATGAGGTGA  CCAGGATAGA  TCTCGACCAG
        1460        1470        1480        1490        1500
TCAGAGTCTC  AGTTCTATAT  TTAATCTTGG  CCCCAGACTG  CACGTGTATG
        1560        1570        1580        1590        1600
CGATTTGAAG  CGGGGGGGGT  ATGGCGTCAT  CTGATATTCT  GTCGGTTGCA
        1660        1670        1680        1690        1700
AAAAACTACC  GTCTACCTGC  CGGACACTGA  ACCCTGGGTG  GTAGAGACCG
        1760        1770        1780        1790        1800
AAGCTTCATC  GTGGTGCCCT  GCCCTCAAAT  TCTCACAACG  GCTTGAGGAT

CTG.
```

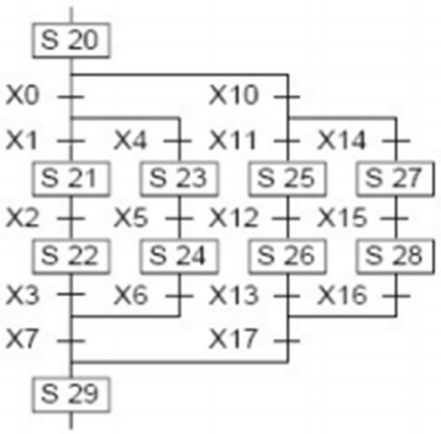


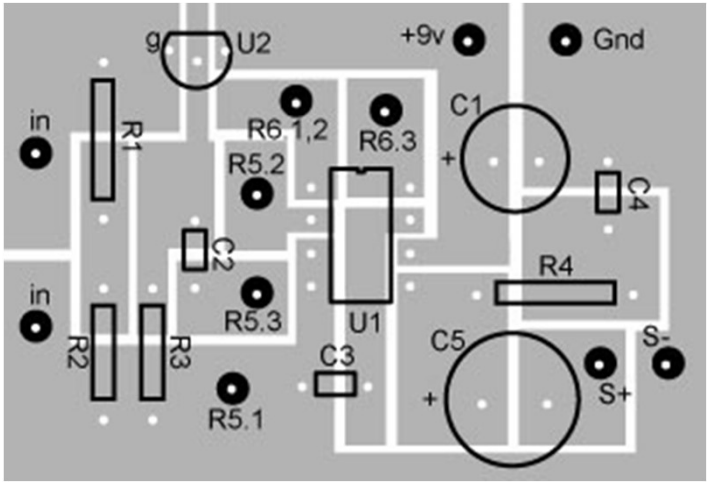
Electronics











In Software Engineering

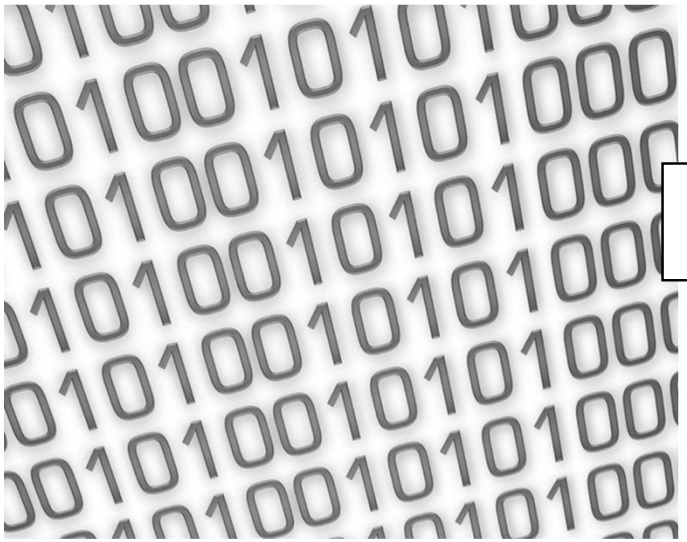
« Languages are the primary way in which system developers communicate, design and implement software systems »

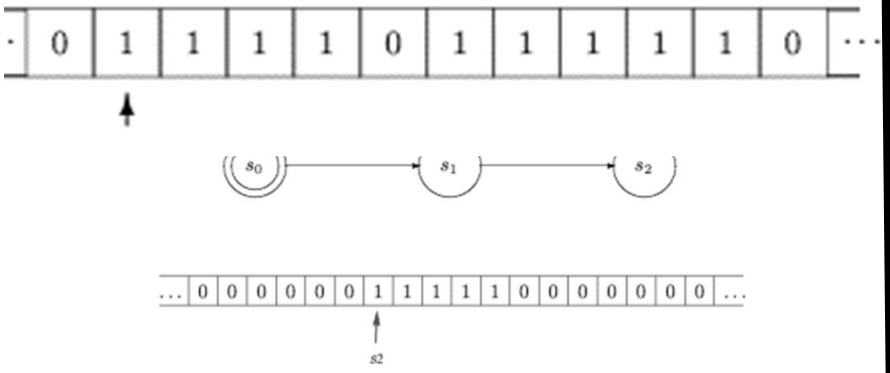
General Purpose Languages

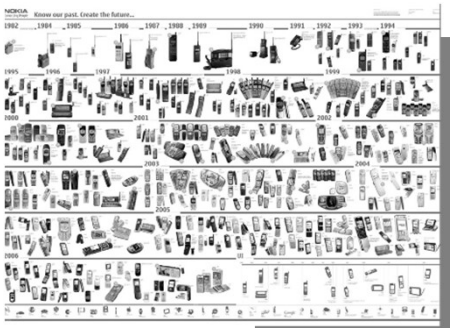
Assembly ?

COBOL ? LISP ? C ? C++ ?

Java? PHP ? C# ? Ruby ?







orange™

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

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

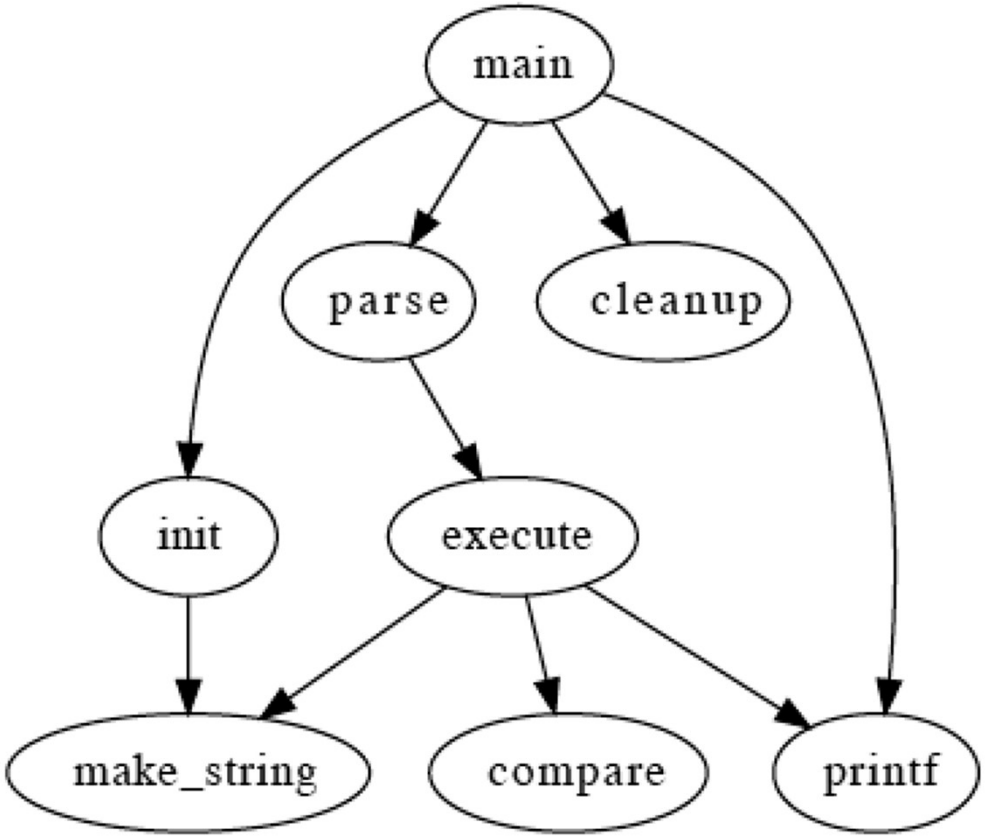
Lighthttpd configuration file

```
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)

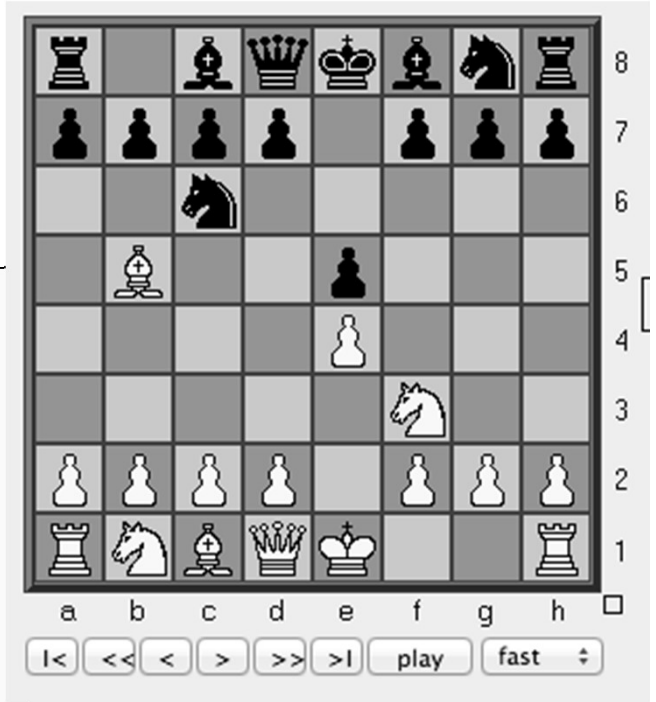
Graphviz

```
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. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 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 Rxe1+ 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)

Question to the audience

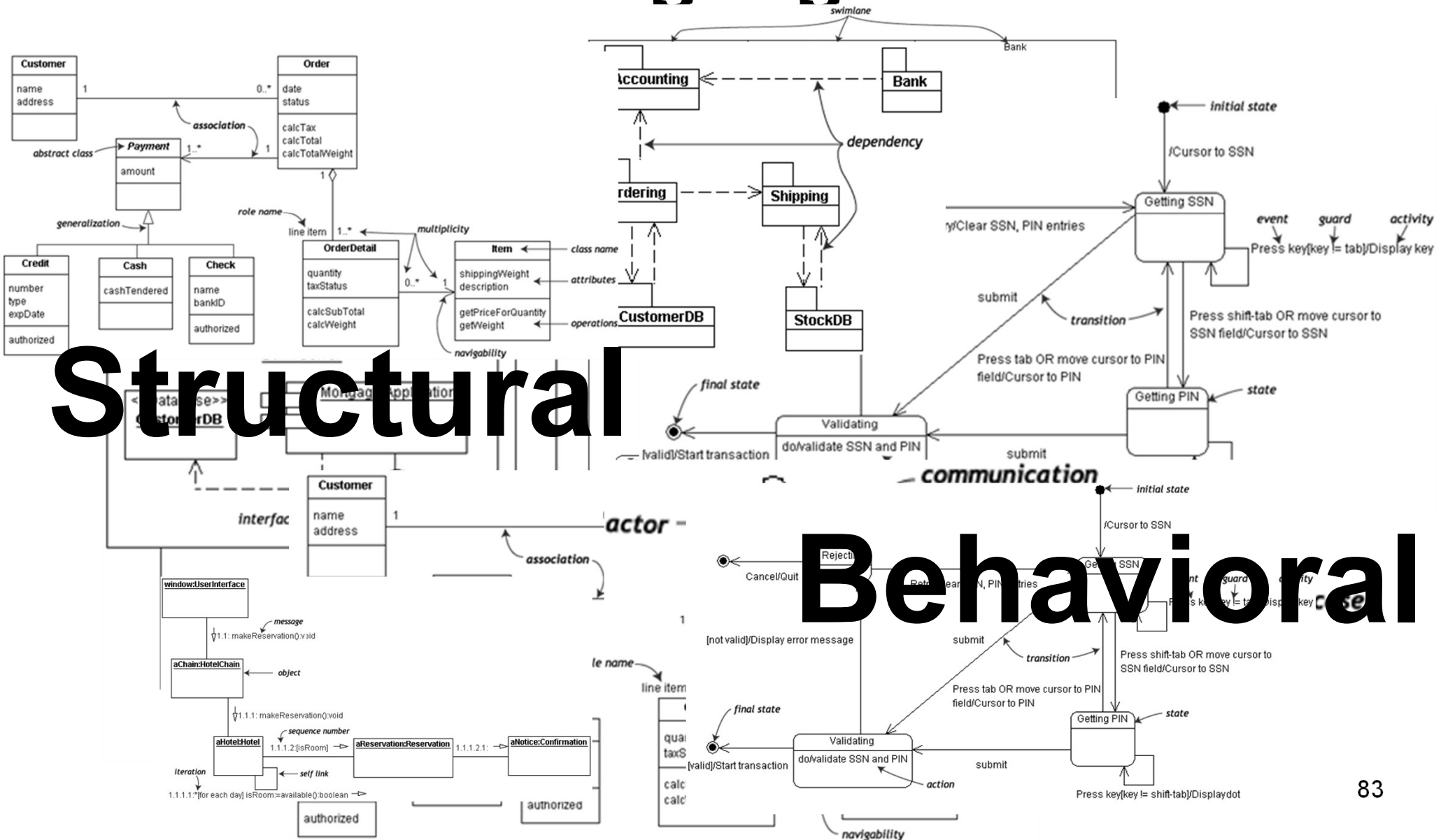
Give three examples of domain-specific languages (DSLs)

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



Abstraction Gap

Assembler

C, Java

DSLs

Problem Space

Solution Space

Google

twitter

« 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 »

aka **Domain-Specific Languages**

1972

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

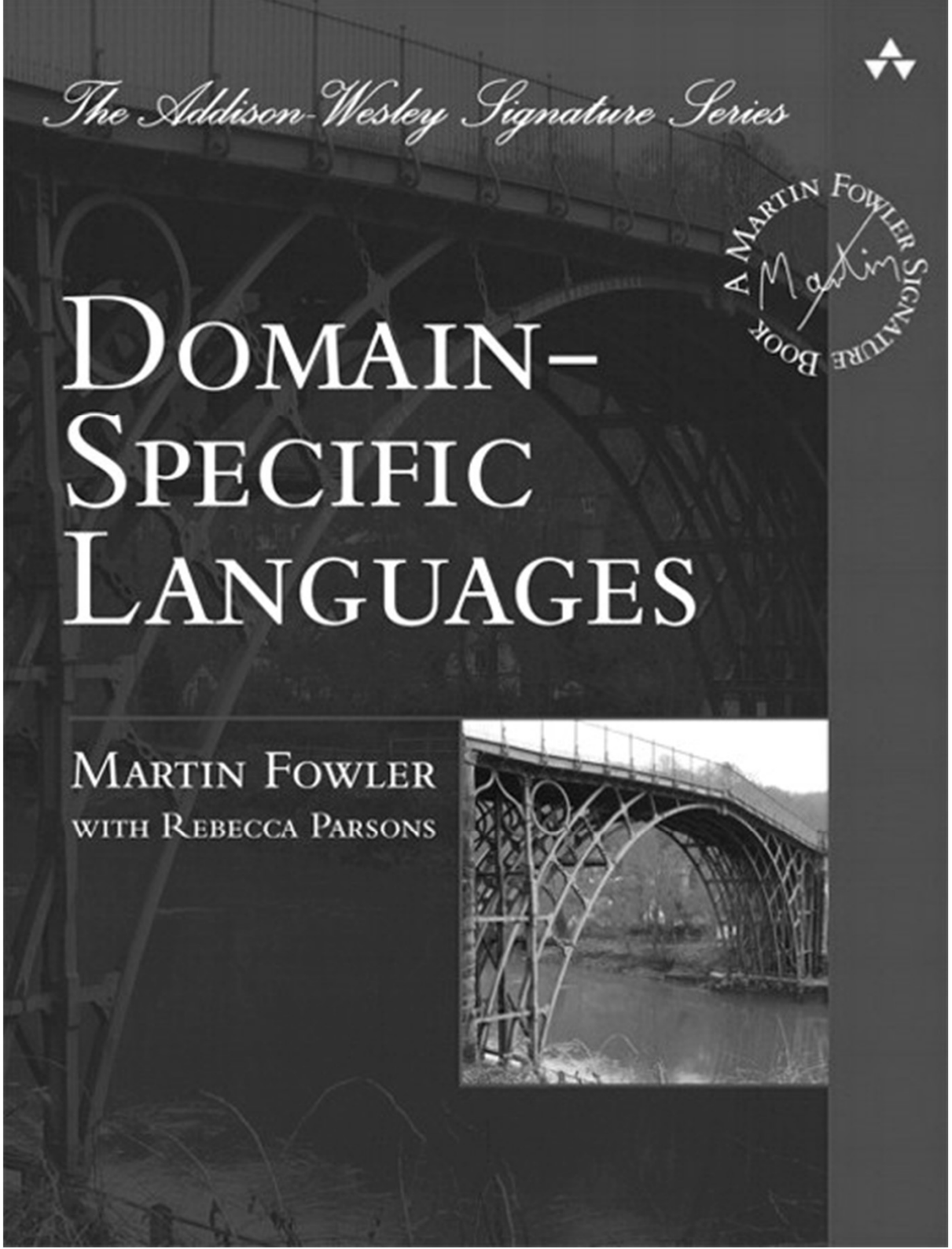
John Hutchinson
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.hutchinson@lancaster.ac.uk}

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
A Martin Fowler Signature Book
Domain-Specific Languages
Martin Fowler
with Rebecca Parsons

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

What is offered?

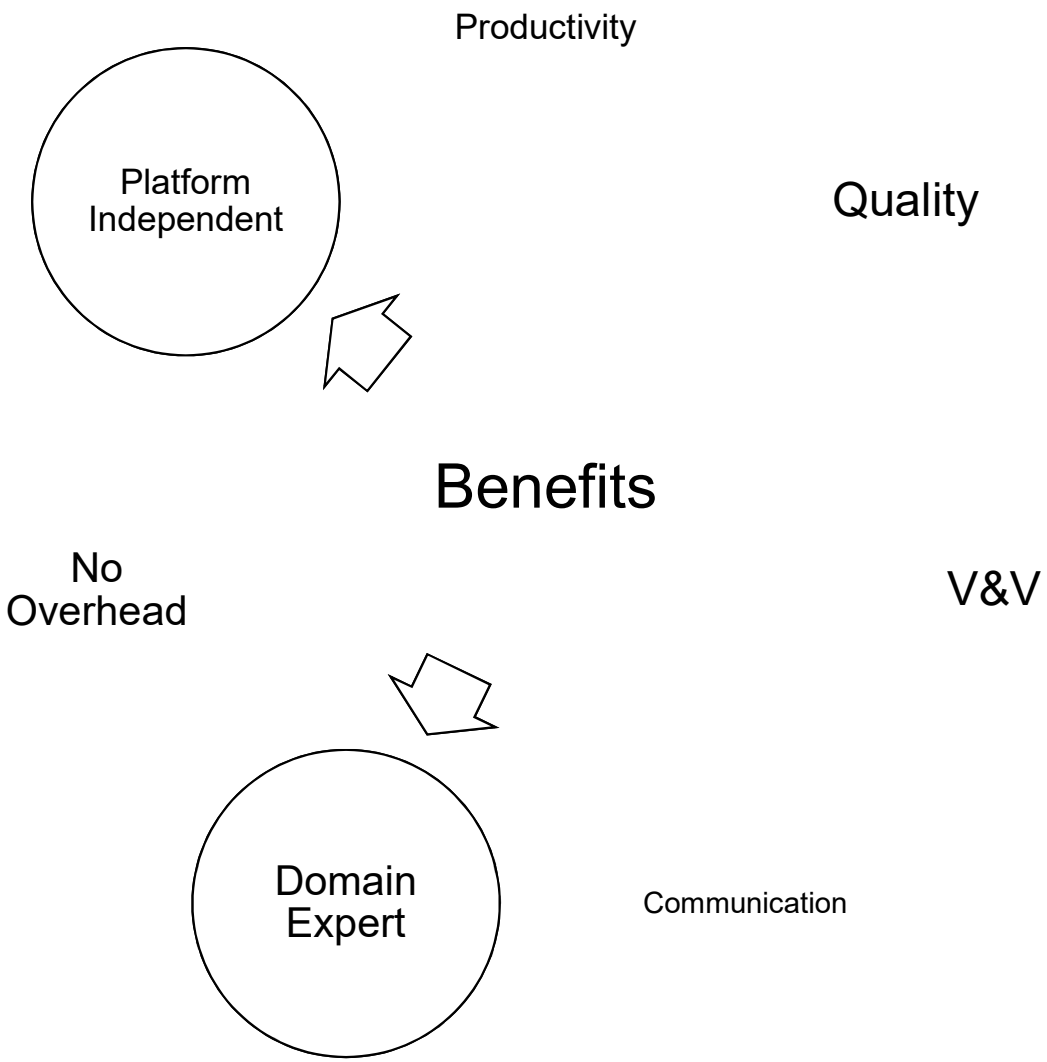
Higher abstractions

Avoid redundancy

Separation of concerns

Use domain concepts

Promises of domain-specific languages



General PLs vs Domain SLs

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

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

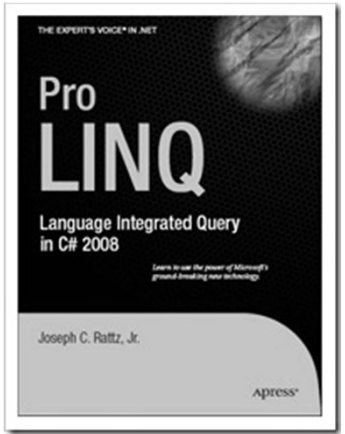
External vs Internal DSL (SQL example)

```
-- 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 »

```
-- 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();
```

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"));
  }
}
```

Internal DSLs vs External DSL

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

Question to the audience

Find a DSL that is both internal and external

HTML

- External DSL: <html>….
- Internal DSLs
  - PHP
  - 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

```
// Import the Glitter DSL
import glitter._

object Templates {

  // Define a reusable layout
  def layout(body: Xml) =
    html5dtd | 'html (
        'head :: 'title :: "Glitter is amazing!"
      | 'body :: body
    )

  // Define a template taking one String argument and using the layout defined above
  def show(name: String) =
    layout (
        'h1 :: "Show user"
      | 'p :: ("Hello " | 'strong(name) | "!")
    )

  // Define a template taking a List of Strings, using the layout defined above
  def index(users: List[String]) =
    layout (
        'h1 :: "User list"
      | 'ul % 'class~"user-list" :: (for (user <- users) yield ('li :: user))
    )
}
```

https://github.com/julienrf/glitter

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

```
let webpage : HTML = HTMLElement(Dict.empty(), [BodyElement(Dict.empty(),
  [H1Element(Dict.empty(), [TextNode("Results for " + keyword)]),
  ULElement((Dict.add Dict.empty() ("id","results")), to_list_items(query(db,
    SelectStmt(["title", "snippet"], "products",
      [WhereClause(InPredicate(StringLit(keyword), "title"))])))])])
```

```
let webpage : HTML = <html><body><h1>Results for {keyword}</h1>
  <ul id="results">{to_list_items(query(db,
    SELECT title, snippet FROM products WHERE {keyword} in title))}
  </ul></body></html>
```

```
let webpage : HTML = parse_html("<html><body><h1>Results for "+keyword+"</h1>
  <ul id=\"results\">" + to_string(to_list_items(query(db, parse_sql(
    "SELECT title, snippet FROM products WHERE '"+keyword+"' in title")))) +
  "</ul></body></html>")
```

SQL

Plain SQL
(external DSL)

shape
#1

```sql
-- SQL
SELECT * FROM journal
   WHERE published_year = 2013
     AND publisher = 'IEEE'
ORDER BY title
```

Java
(internal DSL)

shape
#2

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

Scala
(internal DSL)

shape
#3

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

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.