Computability and Logic
dBerLog

Q1 2007

dBerLog - Lectures

- Time: Tuesday 11 - 14, from August 27 to October 12
- Place: Store Auditorium, IT-Huset
- Lecturer
  - Mogens Nielsen IT-parken, Ada 216 mn
- Course secretary
  - Lene Kjeldsteen IT-parken, Ada 127 lenekj
- Course administrator
  - Doina Bucur IT-parken, Turing 227 doina

dBerLog - Tutoring:
STARTING SEPTEMBER 3

- Group 1 Monday 8-11 Shannon 159
  - Rune Thorbek, Turing 215 thorbek
- Group DA1 Thursday 8-11 Shannon 159
  - Jakob Truelsen, ?? u040819
- Group DA2 Wednesday 12-15 Shannon 164
  - Allan Jørgensen, Turing 124 jallan
- Group DA3 Thursday 8-11 Shannon 164
  - Rune Thorbek, Turing 215 thorbek
- Group DA4 Thursday 8-11 Shannon 156
  - Allan Jørgensen, Turing 124 jallan

dBerLog - Course material

- John Martin: Introduction to Languages and the Theory of Computation
  - McGraw Hill, 3rd edition
- John Kelly: The Essence of Logic
  - Prentice Hall
- Mogens Nielsen: Limitations of Program Verification
  - Course Notes

www.daimi.au.dk/dBerLog

- News
- About this class
- Weekly schedules
  - incl. exercises, handouts, slides etc
- Assignments
- Students
- Final Exam

- Newsgroup: daimi.dBerLog

dBerLog - Exam

- Oral exam

- Compulsory home works
  - 2 compulsory written home work assignments must all be handed in and accepted by the tutor - deadlines etc. will be made clear during the course

dBerLog - Contents

The course introduces

- *universal models* for computation, including Turing machines
- characterizations of *computable and semi-computable problem classes*, including presentations of a number of unsolvable problems, diagonalization, and reduction
- introduction to *propositional logic, predicate logic, and program logic, logical proof systems* with applications (program verification)
- *Gödel's completeness and incompleteness theorems*

dBerLog - Goals

- The goals of this course are to give the student the following capabilities
  - to be *familiar* with the basic *terminology* for computability and logic
  - to *describe* basic computability classes and fundamental logics
  - to *describe* basic *properties* of computability classes and logics
  - to *explain* constructive/algorithmic approaches to computability classes and logics
  - to *analyse* and to *prove* properties of computability classes and logics

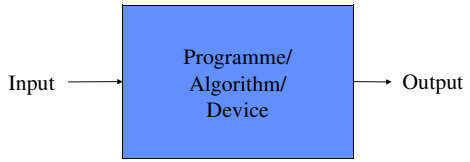
dBerLog - Goals

- The goals of this course are to give the student the following capabilities
  - to *be familiar with* the basic *terminology* for computability
    - problems as formal languages and operations on these, decidability, Turing machines
  - to *describe* basic computability classes and their properties
    - recursive and recursively enumerable languages, closure and decidability properties, from intuition and examples to formal notation and definitions
  - to *explain* algorithmic approaches to properties of computability classes
    - constructive arguments for closure and decidability properties, problem reductions
  - to *analyse* and to *prove* properties properties of computability classes
    - diagonalization, reduction

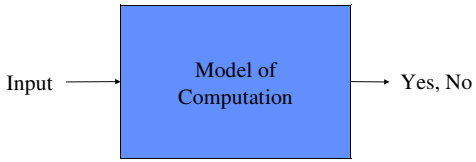
dBerLog - Goals

- The goals of this course are to give the student the following capabilities
  - to *be familiar with* the basic *terminology* for logic
    - truth, satisfaction, validity, syntax, semantics
  - to *describe* fundamental logics and their properties
    - propositional logic, truth tables, predicate logic, interpretations and valuations, program logics, proof systems
  - to *explain* algorithmic approaches to properties of logics
    - decidability, normal forms, proof systems and their proofs, from examples to formal definitions
  - to *analyse* and to prove properties properties of logics
    - soundness and completeness, existence and non-existence of proof systems, Gödel's theorems

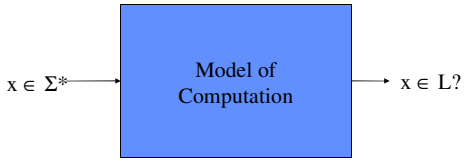
dBerLog contents



dBerLog contents



dBerLog contents



Problem = Language

- f: $\Sigma^* \rightarrow$ {Yes, No}, a problem over Σ
- $L \subseteq \Sigma^*$, a formal language over Σ

- We consider only **formal** languages and **well-defined** problems

Problems

- Given a text - is its LIX value greater than 10?
- Given a text - is it a syntactically correct Java program?
- Given a graph - is it connected?

- Given a configuration in chess - is it winning for white?

- Given a text - is it a semantically correct Java program?

- Given a text - is it syntactically correct Danish?
- Given a text - is it art?

dBerLog - main results

- Only a tiny fraction of formally well-defined problems can be solved computationally!
- Concrete and important (in computing practice) problems can provably not be solved computationally!

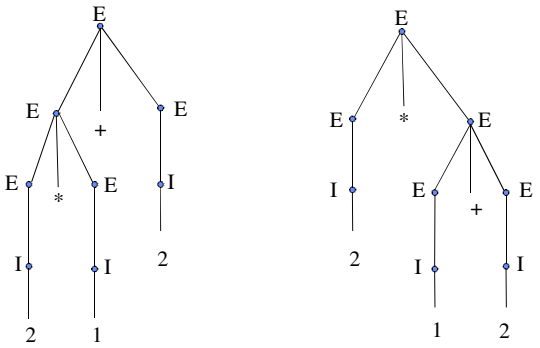
- Only a tiny fraction of math/logic may be formalized!
- Concrete and important (in computing practice) logics can provably not be formalized!

Context Freee Grammar for expressions

- G = ({E, I}, {1,2,+,*,(,)}, P, S)

P: E → I | E + E | E * E | (E)
I → 1 | 2 | I1 | I2

Ambiguity - example



Program correctness

Algoritme: Euklid (m, n)
Inputbetingelse: m, n ≥ 1
Outputkrav: r = sfd(m, n)
Metode: {m, n ≥ 1}
p ← m; q ← n;
I ={sfd(p, q) = sfd(m, n)} while p ≠ q do
if p > q then p ← p - q
else q ← q - p;
r ← p
{r = sfd (m, n)}

Program incorrectness

Algoritme: Euklid (m, n)
Inputbetingelse: m, n ≥ 1
Outputkrav: r = sfd(m, n)
Metode: {m, n ≥ 1}
p ← m; q ← n;
I ={sfd(p, q) = sfd(m, n)} while p ≠ q do
if p > q then p ← p - q
else q ← q - p;
r ← q
{r = sfd (m, n)}

Program termination

```
{x > 1}
while  x ≠ 1 do
        if  even (x)    then x := x div 2     else  x:=  3 × x + 1
```


Problems for Context Free Grammars

- Given a Context-free Grammar G over alphabet Σ

  is it unambiguous?

  is L(G) = Ø?

  is L(G) = Σ*?

Plans for the 7 weeks

- Model of Computation: Turing Machines
- Computability
- Non computable problems

- Propositional Logic
- Predicate Logic
- Program Logic - Gödel's theorems

- Summary - Exam

dBerLog - "Popular" Literature

- Gödel, Escher, Bach
  - Douglas R. Hofstadter - 1980
- The Emperor's New Mind
  - Roger Penrose - 1989
- Mærk Verden
  - Tor Nørretranders - 1991
- Computers Ltd.: What They *Really* Cannot Do
  - David Harel - 2000

Models of Computation

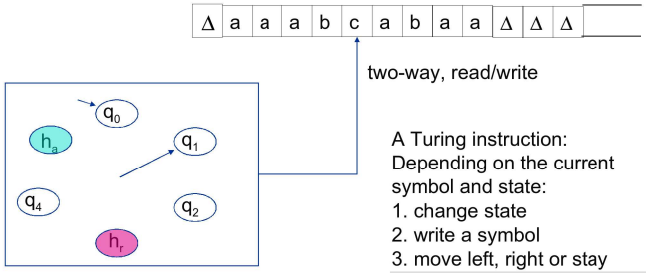
- Models of computation
  - DFA (the regular languages)
  - PDA (the context free languages)
  - ?? (any computable language)
- Turing machines is considered as a fundamental model for computation
  - any language that can be accepted by a a Turing machine can be accepted by any modern programming language (Java, Beta...)
  - AND VICE VERSA!

Plan

- Terminology - describe
  - Turing machines
  - Computing functions with TMs
  - Robustness of TM definitions - extensions
- Explain
  - A universal TM

[Martin, Chapter 9]

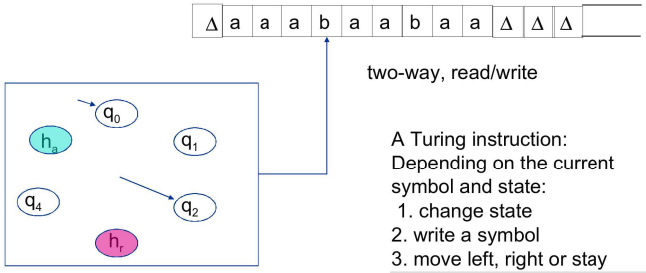
Turing machines – informally 1



A Turing instruction:
Depending on the current symbol and state:
1. change state
2. write a symbol
3. move left, right or stay

Ex: $\delta(q_1, c) = (q_2, a, L)$

Turing machines – informally 2



A Turing instruction:
Depending on the current symbol and state:
1. change state
2. write a symbol
3. move left, right or stay

Ex: $\delta(q_1, c) = (q_2, a, L)$

Turing machines - formally

- T = $(Q, \Sigma, \Gamma, q_0, \delta)$
  - Q is a finite set of *states*
    - NOT containing two special *halt* states: h_a, h_r

    Σ a finite set of *input* symbols

    Γ is a finite set of *tape* symbols, $\Sigma \subseteq \Gamma$
    - NOT containing the special *blank* symbol: Δ
  - $q_0 \in$ Q is the *initial* state

    δ is the *transition* function - a *partial* function:

    $\forall\delta$: $Q \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{R,L,S\}$

Configurations of a Turing machine

- A *configuration* is a global state (snapshot) of a Turing machine consisting of
  - the state
  - the content of the tape: finitely many non-blank symbols
  - the current position
- Formally:
  - an element of the form $(q, x\underline{a}y) \in Q \times (\Gamma^*\underline{\Gamma}\Gamma^*)$

The step function

- $(q, x\underline{a}y) \vdash (r, z\underline{b}w)$ defined in the obvious way
- Example: configuration = $(q, aa\underline{b}\Delta\Delta bb)$

  If $\delta(q, b) = (r, a, R)$ then
  - $(q, abb\underline{\Delta}\Delta bb) \vdash (r, aba\underline{\Delta}\Delta bb)$

  If $\delta(q, b) = (r, b, L)$ then
  - $(q, abb\underline{\Delta}\Delta bb) \vdash (r, ab\underline{b}\Delta\Delta bb)$

  If $\delta(q, b) = (r, a, S)$ then
  - $(q, abb\underline{\Delta}\Delta bb) \vdash (r, aba\Delta\Delta bb)$

The Language of a Turing machine

- $(q_0, \underline{\Delta}x)$ is the *start* configuration with input $x \in \Sigma^*$
- $x \in \Sigma^*$ is *accepted* iff
  - $(q_0, \underline{\Delta}x) \vdash^* (h_a, y\underline{a}z)$ for some $a \in (\Gamma \cup \{\Delta\})$ & $y, z \in (\Gamma \cup \{\Delta\})^*$
- $L(T) = \{ x \in \Sigma^* \mid x \text{ is accepted by } T\}$

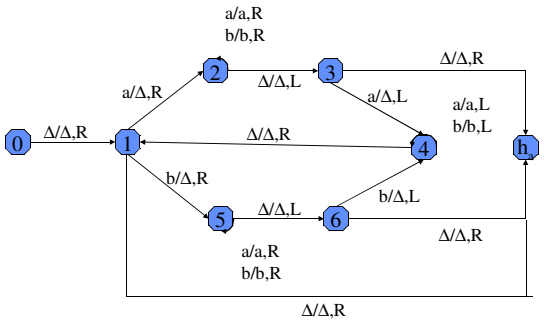
The Language of a Turing machine

- Types of non-accepting runs from $(q_0, \underline{\Delta}x)$:
  - eventually enters h_r
  - crashing (1) - machine in a non-halt state, but no outgoing transition; convention: the machine then enters h_r
  - crashing (2) - head at the leftmost tape square, and step function tells machine to go left; convention: the machine then enters h_r
  - infinite loop - the machine never enters h_a or h_r

Example of languages accepted by Turing machines

- { ww | w ∈ {a,b}* }
- $\{a^n b^n a^n \mid n > 0\}$
- { {a, b}^{n} | n prime }
- { w | w ∈ {a,b}* and reverse(w) = w} (palindromes)

A Turing Machine accepting?



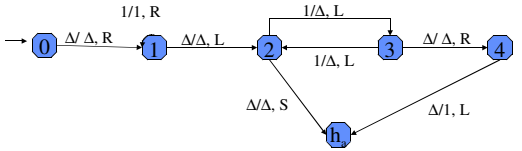
Computing a function Σ* -> Σ*

- T = (Q, Σ, Γ, q_0, δ) is said to compute the partial function f: Σ* -> Σ* iff
  - for all x∈ Σ*, s.t. f(x) is defined:
    $(q_0, \underline{\Delta}x) \vdash^* (h_a, \underline{\Delta}f(x))$
  - for all x∈ Σ*, s.t. f(x) is undefined:
    T does not accept x

Computing a function N -> N

- T = (Q, {1}, Γ, q_0, δ) is said to compute the partial function f: N -> N iff
  - for all n, s.t. f(n) is defined:
    $(q_0, \underline{\Delta}1^n) \vdash^* (h_a, \underline{\Delta}1^{f(n)})$
  - for all n. s.t. f(n) is undefined:
    T does not accept 1^n

A Turing Macgine computing?



Extensions

The basic Turing machine may be extended in many ways:

- two-way infinite
- multiple tapes
- non-determinism
- ...

None of these extensions extends the essential power of the Turing machine: they all can be simulated by a standard Turing machine!

Multiple tape TMs

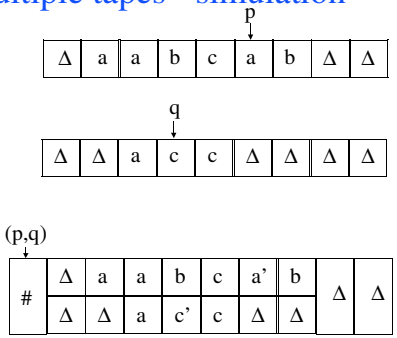
- Separate head for each tape
- Input (and output) on tape 1
- Every transition makes a step for each tape

(Formally defined in [Martin])

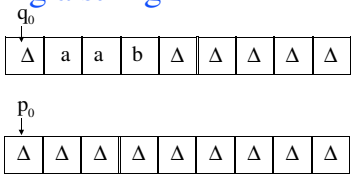
Theorem

A language is accepted by a 2-tape TM iff it is accepted by a 1-tape TM

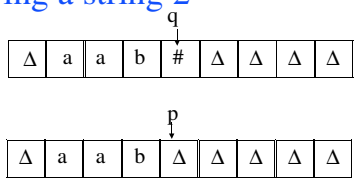
Multiple tapes - simulation



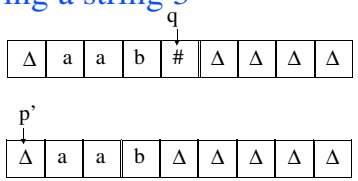
Copying a string 1



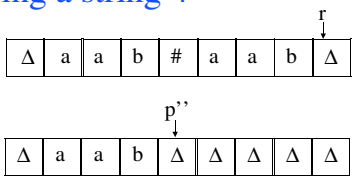
Copying a string 2



Copying a string 3



Copying a string 4



Nondeterministic Turing Machines

- Generalize transition function (like FA -> NFA)
- Accept a string if *some* run leads to accept

(Formally defined in [Martin])

Theorem

A language is accepted by a nondeterministic TM iff it is accepted by a deterministic TM

Nondeterministic TMs - simulation

Simulating a NTM T_1 by a (deterministic) TM T_2,
try all possible runs, searching for an accepting one

Use a 3-tape TM:

- tape 1: contains original input string (never changed)
- tape 2: encoding of the current run
- tape 3: working tape (corresponds to T_1's tape)

(Details in [Martin])

The Universal Turing Machine

- **Theorem**

There exists a TM U which inputs:
a Turing machine T and an input x to T
such that
U accepts (T, x) iff T accepts x

Encoding of (T, x) in alphabet {0, 1}

- Assume states of T: $\{q_1, q_2, \ldots, q_m\}$
- Assume alphabet of T: $\{a_1, a_2, \ldots, a_n\}$

- $s(\Delta) = 0 \quad s(a_i) = 0^{i+1}$
- $s(h_a) = 0 \quad s(h_r) = 00 \quad s(q_i) = 0^{i+2}$
- $s(S) = 0 \quad s(L) = 00 \quad s(R) = 000$

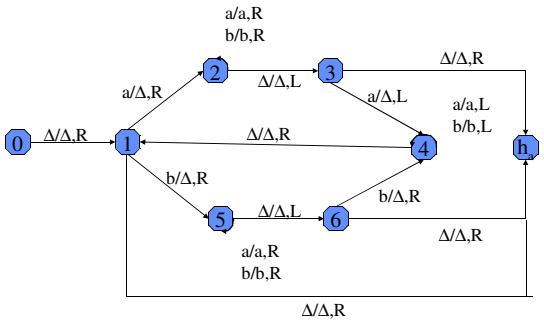
Encoding of (T, x) in alphabet {0, 1}

- Each move m: $\delta(p, a) = (q, b, D)$ encoded by
  $e(m) = s(p)\ 1\ s(a)\ 1\ s(q)\ 1\ s(b)\ 1\ s(D)\ 1$

- Turing machine T with moves $\{m_1, m_2, .., m_k\}$ encoded by
  $e(T) = 1\ s(q_0)\ 1\ e(m_1)\ 1\ e(m_2)\ 1 \ldots .\ e(m_k)\ 1$

- Input $x = a_1 a_2 \ldots a_l$ encoded by
  $e(x) = 1\ s(a_1)\ 1\ s(a_2)\ 1 \ldots s(a_l)\ 1$

TM for palindromes



The Universal Turing Machine

- **Theorem**

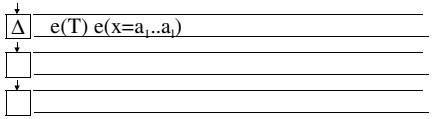
  There exists a TM U with input alphabet $\Sigma = \{0, 1\}$ such that

  U accepts $e(T)e(x) \in \{0, 1\}^*$
  iff
  T accepts x

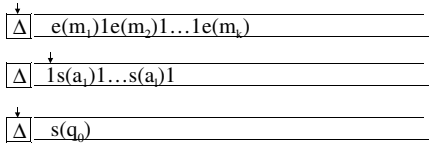
Constructing a Universal Turing machine

- Use a 3-tape TM:
  - tape 1: initially contains input string e(T)e(x)
  - tape 2: working tape (corresponds to T's tape)
  - tape 3: encoding of current state of T

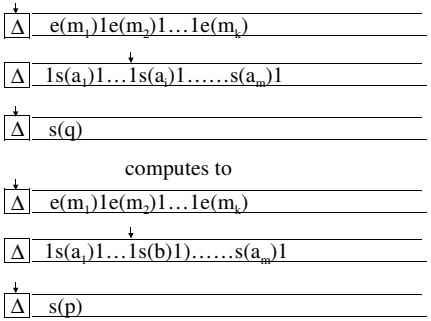
The Universal TM - initial moves



initially computes to



The Universal TM - simulating moves



$\delta(q, a_i) = (p, b, S)$

computes to

Summary

- Terminology - describe
  - Turing machines
  - Computing functions with TMs
  - Robustness of TM definitions - extensions
- Explain
  - A universal TM

Turing Machine simulator: http://ironphoenix.org/tril/tm

[Martin, Chapter 9]

Tutorials

- Familiarity with terminology
  - Martin 9.2
    - use http://ironphoenix.org/tril/tm to check your solution
  - Martin 9.11
  - Martin 9.35
- Describe computability classes
  - Martin 9.5
  - Martin 9.6(b), 9.15 (a, b)
    - notice the language from 9.6(b) is non context-free - for these exercises use again a one-tape TM and use http://ironphoenix.org/tril/tm to check your solution
  - Martin 9.18 and 9.19
    - for these exercises you are encouraged to use multitape TMs
- Describe properties of computability classes
  - Martin 9.33
    - see the use of non-determinism (informal arguments only)