Mathematical Foundation of Computer Sciences VIII

Theoretical Computer Sciences

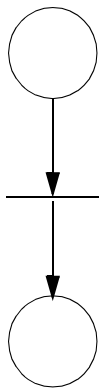
Guoqiang Li

School of Software, Shanghai Jiao Tong University

Petri Nets

Petri Nets

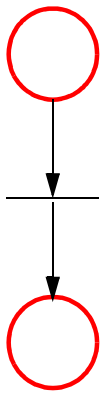
Systems are specified as a directed bipartite graph. The two kinds of nodes in the graph:



Petri Nets

Systems are specified as a directed bipartite graph. The two kinds of nodes in the graph:

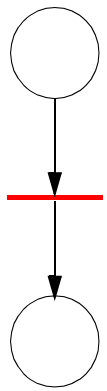
- Places: they hold the distributed state of the system expressed by the presence/absence of tokens in the places.



Petri Nets

Systems are specified as a directed bipartite graph. The two kinds of nodes in the graph:

- Places: they hold the distributed state of the system expressed by the presence/absence of tokens in the places.
- Transitions: denote the activity in the system

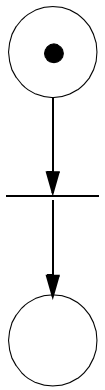


Petri Nets

Systems are specified as a directed bipartite graph. The two kinds of nodes in the graph:

- Places: they hold the distributed state of the system expressed by the presence/absence of tokens in the places.
- Transitions: denote the activity in the system

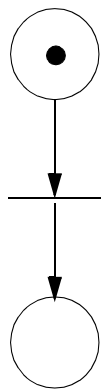
The state of the system: captured by the marking of the places (number of tokens in each place)



Petri Nets

The dynamic evolution of the system: determined by the firing process of transitions.

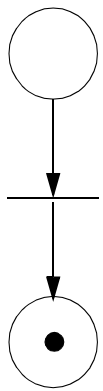
- A transition is enabled and may fire whenever all its predecessor places are marked.



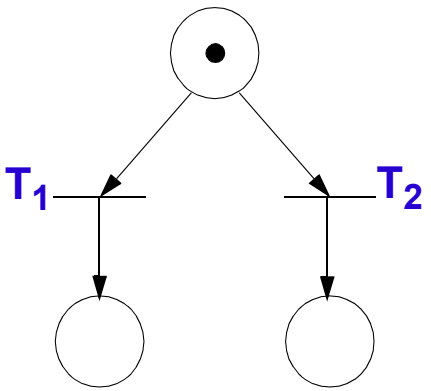
Petri Nets

The dynamic evolution of the system: determined by the firing process of transitions.

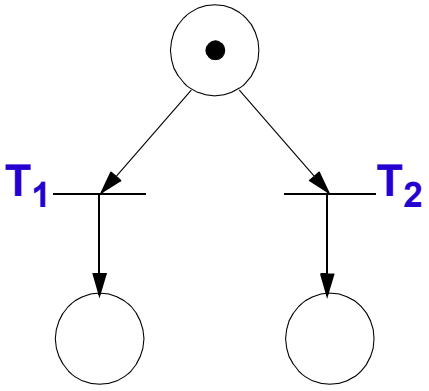
- A transition is enabled and may fire whenever all its predecessor places are marked.
- If a transition fires it removes a token from each predecessor place and adds a token to each successor place.

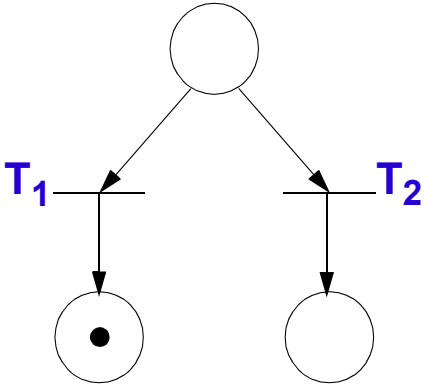


Nondeterminism

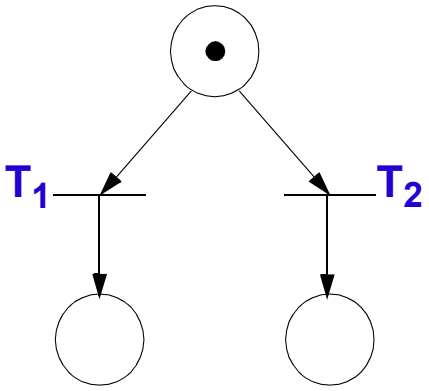


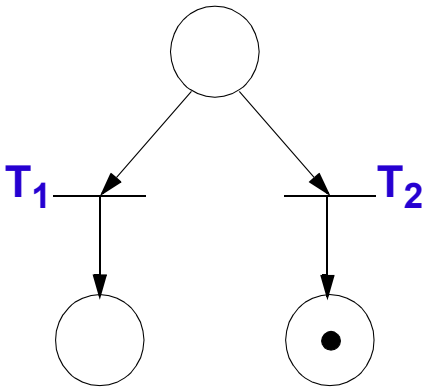
Nondeterminism



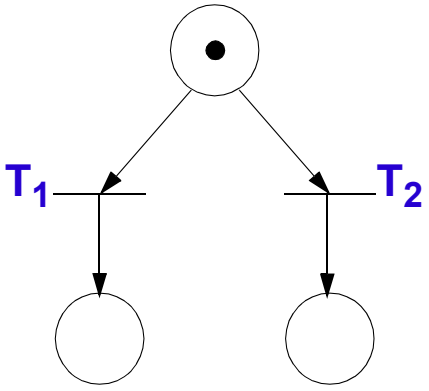


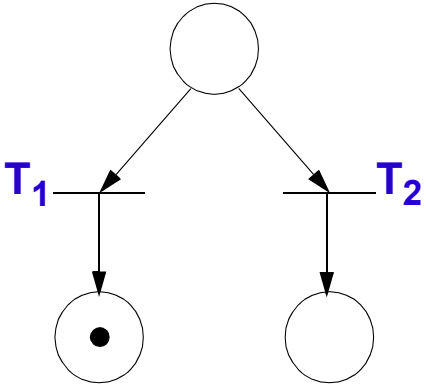
Nondeterminism

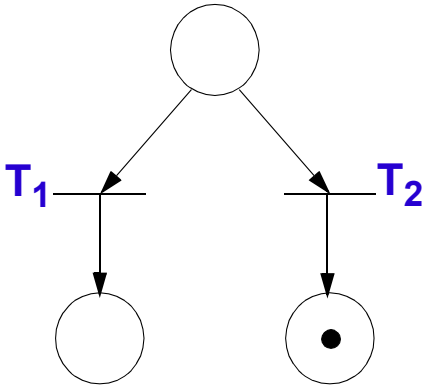




Nondeterminism





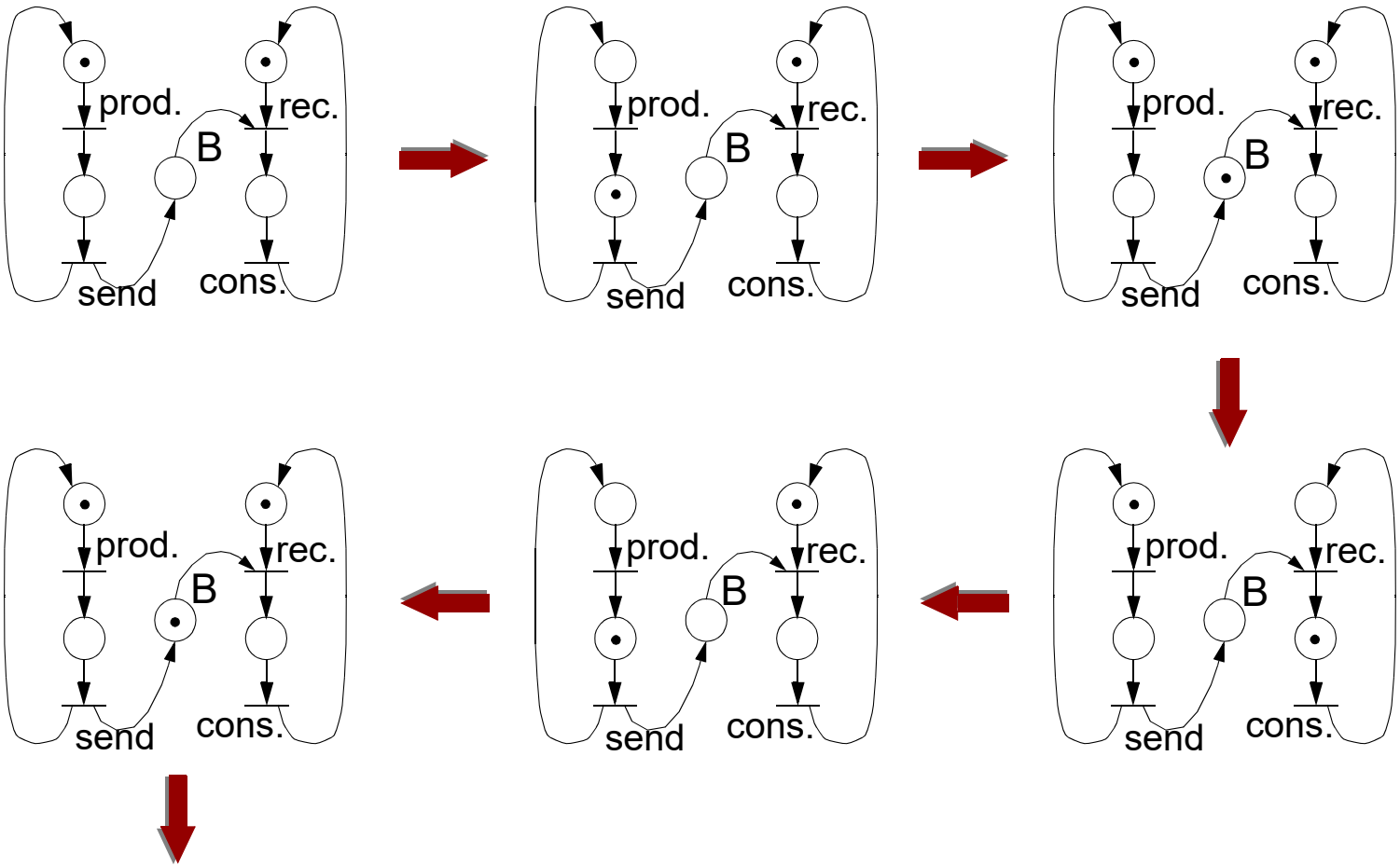


Example

A producer and a consumer process communicating through a buffer:

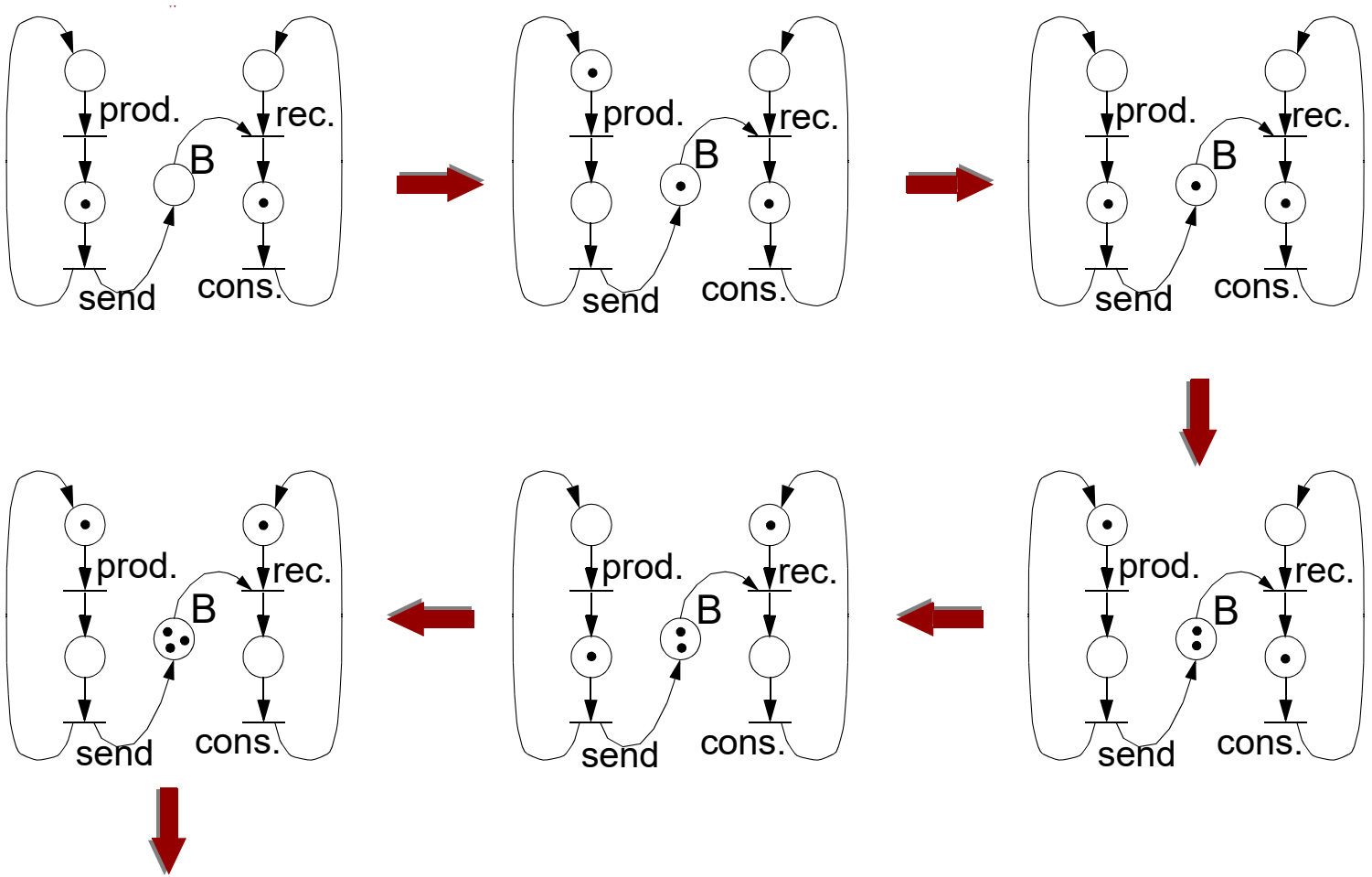
Example

A producer and a consumer process communicating through a buffer:



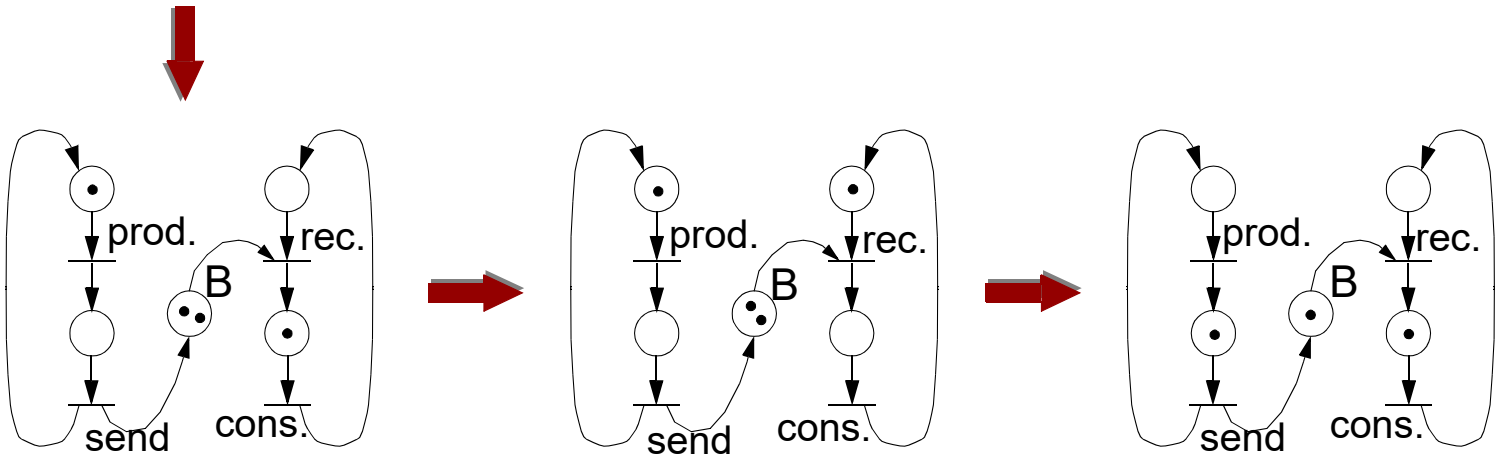
Example

A producer and a consumer process communicating through a buffer:



Example

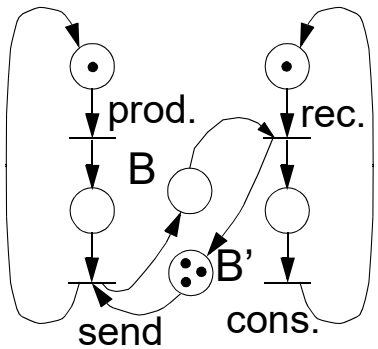
A producer and a consumer process communicating through a buffer:



Notice that the buffer is considered to be infinite (tokens accumulate in B).

Example

We have the same model, but with limited buffer.



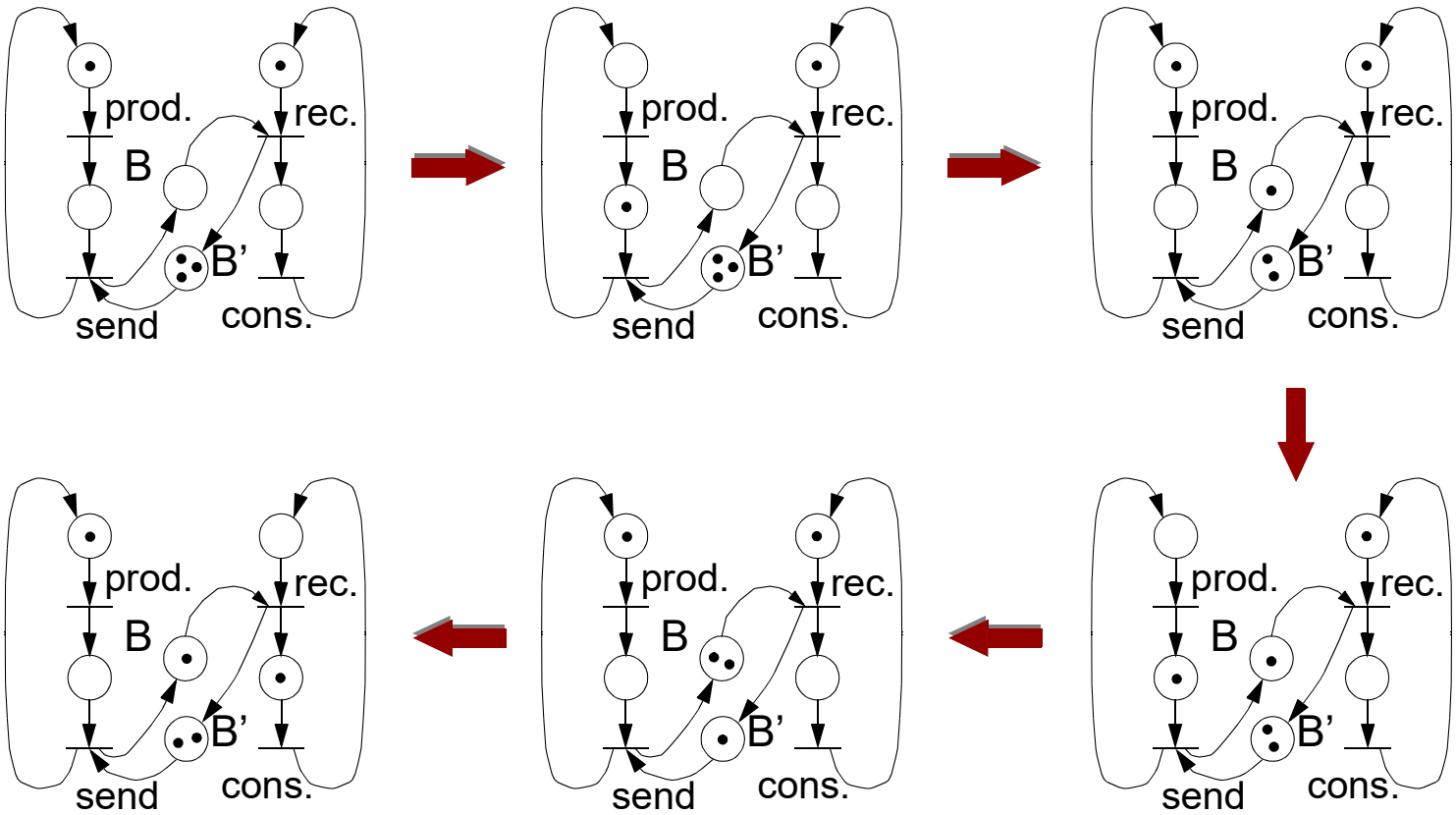
The buffer size is three (number of initial tokens in B')

- Number of tokens in B': how many free slots are available in the buffer;
- Number of tokens in B: how many messages (tokens) are in the buffer.

Total number of tokens in B and B' is constant (= 3).

Example

We have the same model, but with limited buffer.



Formal Definition

A Petri net is a four-tuple: $PN = \langle P, T, I, O \rangle$, where $P \cap T = \emptyset$

- P: a finite set of places, $\{p_1, p_2, \ldots, p_n\}$
- T: a finite set of transitions, $\{t_1, t_2, \ldots, t_s\}$
- I: an input function, $T \times P \to \{0, 1\}$
- O: an output function, $T \times P \to \{0, 1\}$

M_0: an initial marking, $P \to \mathbb{N}$

Some Features and Applications of Petri Nets

Intuitive:

- Easy to express concurrency, synchronisation, nondeterminism.
- Nondeterminism is an important difference between Petri nets and sequential models.

As an uninterpreted model, Petri Nets can be used for several, very different classes of problems.

Some Features and Applications of Petri Nets

Industrial production systems,

Information systems,

Computer architectures,

Operating systems,

Concurrent programs,

Distributed systems,

Hardware systems

Properties and Analysis of Petri Nets

Boundedness: number of tokens in a place does not exceed a limit.

- You can check that available resources are not exceeded.

Liveness: A transition t is called live if for every possible marking there exists a chance for that transition to become enabled.

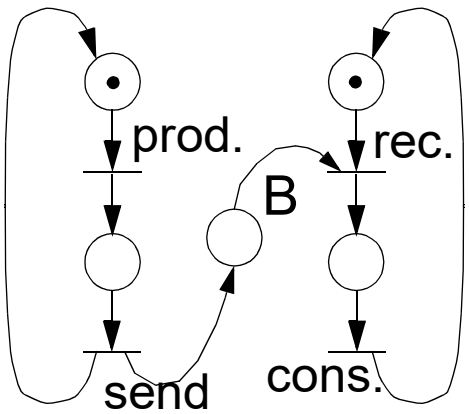
The whole net is live, if all its transitions are live.

- Important in order to check that the system is not deadlocked.

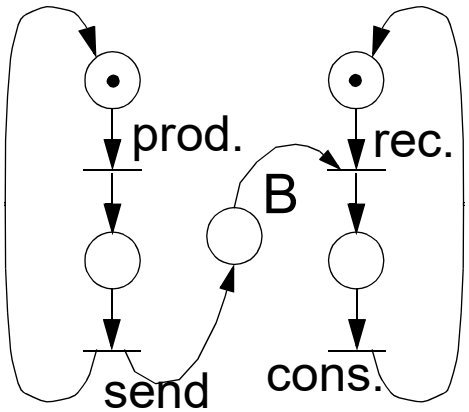
Reachability: given a current marking M and another marking M', does there exist a sequence of transitions by which M' can be obtained?

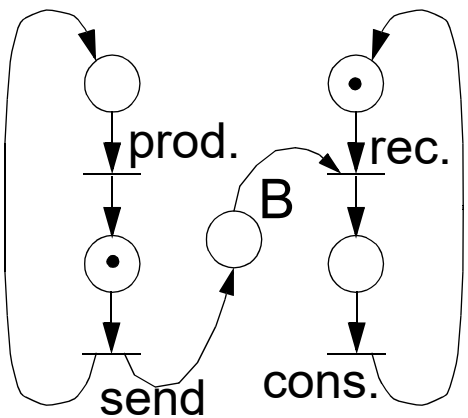
- You can check that a certain desired state (marking) is reached.
- You can check that a certain undesired state is never reached.

Vector Addition Systems



$$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$





$$\begin{pmatrix}1\\1\\0\\0\\0\end{pmatrix} \xrightarrow{\begin{pmatrix}-1\\0\\1\\0\\0\end{pmatrix}} \begin{pmatrix}0\\1\\1\\0\\0\end{pmatrix}$$

Formal Definition

Definition (d-VAS)

Assume $d \geq 1$, $A \subseteq \mathbb{Z}^d$. Given $u, v \in \mathbb{Z}^d$, define one step as $u \xrightarrow{a} v$, if $v - u = a \in A$. The transitive closure is: $\xrightarrow{w}$. The VAS language $L_A(u, v) = \{w \mid w \in A^*, u \xrightarrow{w} v\}$

Definition (Reachability problem)

Input: d-VAS A, and $u, v \in \mathbb{Z}^d$, u and v is reachable if $\exists w, u \xrightarrow{w} v$ or $L_A(u, v) \neq \emptyset$.

Example

1-VAS $A = \{-2, 2\}$

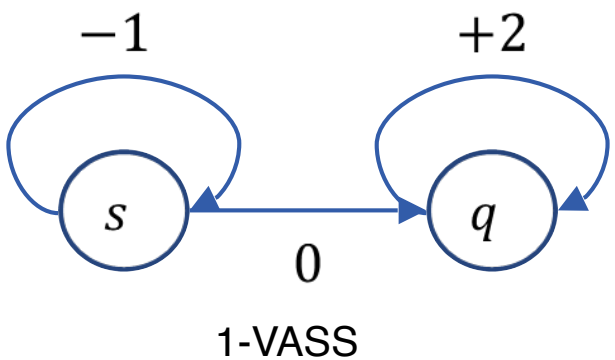
$L_A(0, 5) = \emptyset$ (unreachable), $L_A(0, 4) \neq \emptyset$(reachable)

VAS and VASS

Definition (Vector addition system with states(VASS))

, A triple $\langle Q, \Delta, d \rangle$, and $A \subseteq \mathbb{Z}^d$

- Q a finite set of states
- Transition rule $\Delta \subseteq Q \times A \times Q$



1-VASS

n-VASS can be encoded in n+3-VAS(Hopcroft & Pansiot 1979).

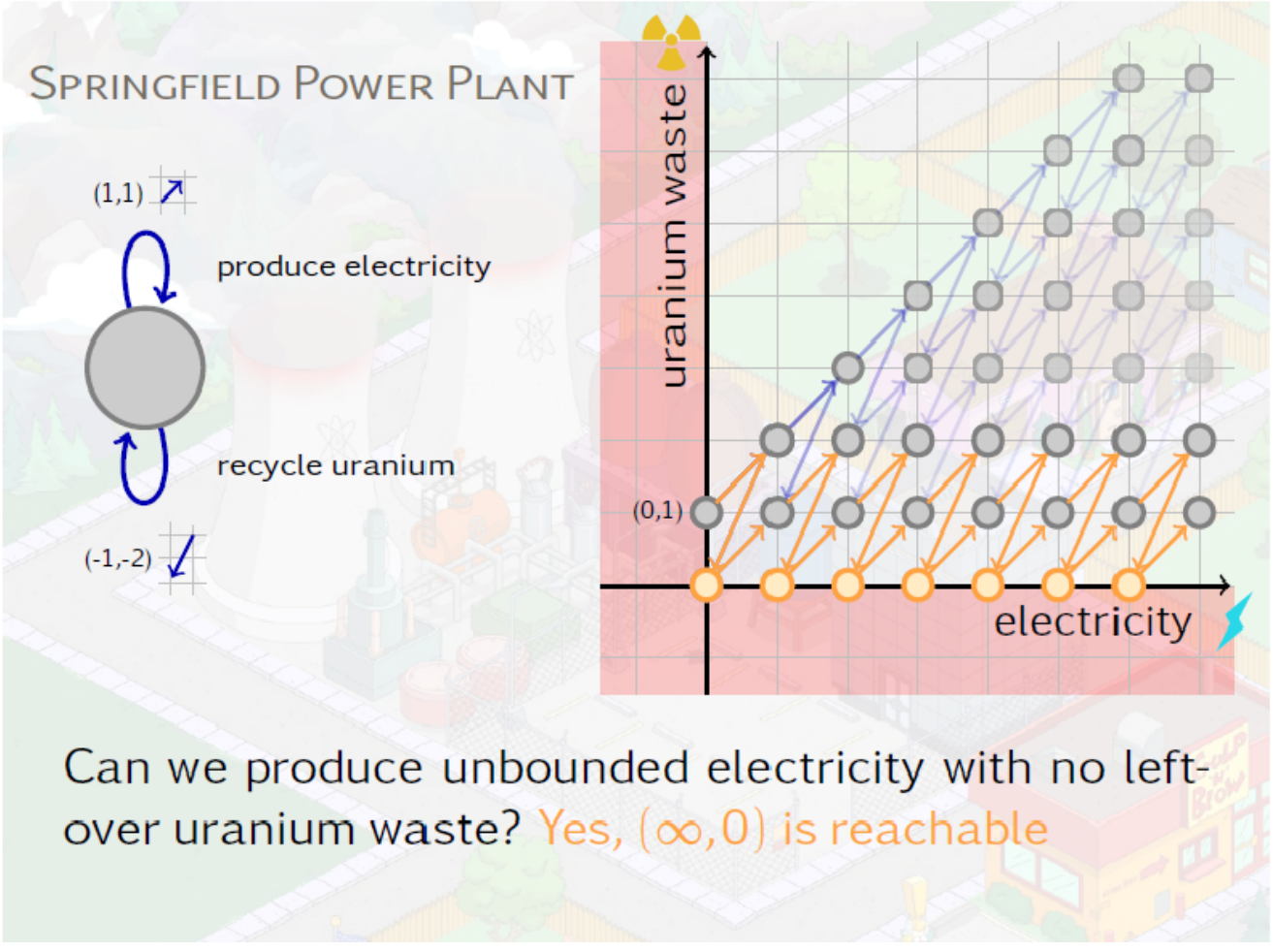
Three Equivalent Models

Vector addition system(VAS), ideal for proof

Vector addition system with states(VASS), ideal for examples

Petri nets, ideal for applications

VAS as Coordinate



from slides by Sylvain Schmitz

Theoretical Computer Sciences

What is the core in computing science?

Computation!

The First Question

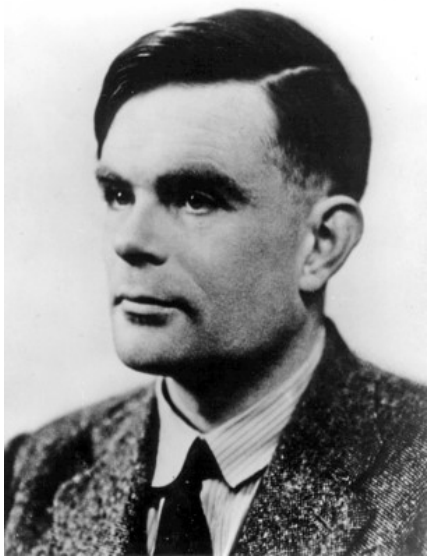
Q1: Given a model, which problem can be solved by the model?

Theory of Computation

Theory of Computation.

Church-Turing Thesis

Everything algorithmically computable is computable by a Turing machine.



Church-Turing Thesis

Church-Turing Thesis

李停舟 V
2月21日 22:32 来自 微博搜索
#理论计算机科学# 无限寄存器机(unlimited register machine; URM)
现代通用计算机的机器模型。计算能力上等价于图灵机。1963年由约翰·谢皮德森和霍华德·斯特吉斯提出，作为一个比图灵机更加易于理解的计算模型。一个无限寄存器机拥有一定数量的用于存储自然数的寄存器，并利用相应的程序来操作这些寄存器。一个无限寄存器机的程序由一个有限长度的基本指令序列构成，其中基本指令包括寄存器清零，后继，复制和跳转指令。当程序运行时，一个特殊的指针指向当前的指令。#寒假不打烊# 收起全文
阅读 1960 推广 转发 评论 1

李停舟 V
2月21日 17:30 来自 微博 weibo.com
#理论计算机科学#图灵机(Turing machine)
现代通用计算机的数学模型。1936年由图灵提出。是一种完全忽略硬件状态，考虑的焦点是逻辑结构的虚拟 "计算机"。不是一个具体机器，而是一种数学模型，由有限多个状态、一个可以左右移动的读写头和一条无限长的"磁带"构成。据此可制造一种十分简单但运算能力极强的计算装置。可以用来计算所有的可以被计算的函数，同时，所有可以被计算的函数都可以被图灵机所刻画。
#寒假不打烊# 收起全文
阅读 3019 推广 转发 评论 1

李停舟 V
2月17日 10:18 来自 微博 weibo.com
#理论计算机科学# λ演算(lambda calculus)
计算机程序语言的数学模型。由数学家阿隆佐·丘奇在20世纪30年代首次提出。后来成为函数式语言的逻辑模型。包括了一条变量替换规则和一条将函数抽象化规则。可以证明λ演算和图灵机等价，它们共同刻画了可以计算的函数。#寒假不打烊#
阅读 2606 推广 转发 评论 4

The Second Question

Q2: Given a solvable problem, can it be solved efficiently?

Theory of Complexity

Theory of Complexity.

Complexity Classes

Time: $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$

Space: $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}$

Hybrid: $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$

Counting: $\mathbf{NP} \subseteq \sharp\mathbf{P} \subseteq \mathbf{EXP}$

Probability: $\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{RP} \subseteq \mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$ $\mathbf{RP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$

Quantum: $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$

Complexity Zoo

https://complexityzoo.net/Complexity_Zoo

545 classes now!

A

A_0PP - AC - AC^0 - $AC^0[m]$ - AC^1 - ACC^0 - AH - AL - ALL - ALOGTIME - AlgP/poly - Almost-NP - Almost-P - Almost-PSPACE - AM - AM^{cc} - AM_{EXP} - AM ∩ coAM - AM[polylog] - AmpMP - AmpP-BQP - AP - APP - APSPACE - AUC-SPACE(f(n)) - AuxPDA - AVBPP - AvgE - AvgP - AW[P] - AWPP - AW[SAT] - AW[*] - AW[t] - AxP - AxPP

B

βP - $BC_=P$ - BH - $BP_{\|}(P)$ - BPE - BPEE - $BP_HSPACE(f(n))$ - BPL - BP•NP - BPP - BPP^{cc} - BPP_k^{cc} - BPP^{KT} - BPP/log - BPP/mlog - BPP//log - BPP/rlog - BPP-OBDD - BPP_{path} - BPQP - BPSPACE(f(n)) - BPTIME(f(n)) - BQNC - BQNI BQP/poly - BQP/mlog - BQP/mpoly - BQP/qlog - BQP/qpoly - BQP-OBDD - BQPSPACE - BQP_{CTC} - BQP_{tt}/poly - BQTIME(f(n)) - k-BWBP

C

$C_=AC^0$ - $C_=L$ - $C_=P$ - CC - CC^0 - CFL - CLOG - CH - Check - CL#P - C_kP - CNP - coAM - $coC_=P$ - cofrIP - Coh - coMA - $coMod_kP$ - compIP - compNP - coNE - coNEXP - coNL - coNP - $coNP^{cc}$ - coNP/poly - coNQP - coRE - c coSPARSE - coUCC - coUP - CP - cq-Σ_2 - CSIZE(f(n)) - CSL - CSP - CZK

D

D#P - DCFL - Δ_2P - δ-BPP - δ-RP - DET - $DiffAC^0$ - DisNP - DistNP - DP - DQC1 - DQP - DSPACE(f(n)) - DTIME(f(n)) - DTISP(t(n),s(n)) - Dyn-FO - $Dyn\text{-}TC^0$

E

E - EE - EEE - EESPACE - EEXP - EH - ELEMENTARY - EL_kP - EP - EPTAS - k-EQBP - EQP - EQP_K - EQTIME(f(n)) - ESPACE - ∃BPP - ∃NISZK - EXP - EXP/poly - EXPSPACE

F

FBQP - FERT - FPERT - Few - FewEXP - FewP - FH - FIXP - FNL - FNL/poly - FNP - FO - FO(DTC) - FO(LFP) - FO(PFP) - FO(TC) - FO($t(n)$) - FOLL - FP - $FP^{NP[log]}$ - FPL - FPR - FPRAS - FPT - FPT_{nu} - FPT_{su} - FPTAS - FQMA - frIP

G

GA - GAN-SPACE(f(n)) - $GapAC^0$ - GapL - GapP - GC(s(n),C) - GCSL - GI - GLO - GPCD(r(n),q(n)) - G[t]

H

HalfP - HeurBPP - HeurBPTIME(f(n)) - $HeurDTIME_g$(f(n)) - HeurP - HeurPP - $HeurNTIME_g$(f(n)) - H_kP - HVSZK

I

IC[log,poly] - IP - IPP - IP[polylog]

L

L - LC^0 - LH - LIN - L_kP - LOGCFL - LogFew - LogFewNL - LOGLOG - LOGNP - LOGSNP - L/poly - LWPP

M

MA - MA^{cc} - MA' - MAC^0 - MA_E - MA_{EXP} - mAL - $MA_{POLYLOG}$ - MaxNP - MaxPB - MaxSNP - $MaxSNP_0$ - mcoNL - MinPB - MIP - MIP* - MIP^{ns} - MIP_{EXP} - $(M_k)P$ - mL - MM - MMSNP - mNC^1 - mNL - mNP - Mod_kL - ModL - mP - MP - MPC - mP/poly - mTC^0

The Third Question

Q2: Given an (efficient) problem, how to solve it?

Theory of Programming

Theory of Programming.

Theory of Programming

data structures

algorithms

Process Rewriting Systems