チューリングマシンの拡大と複雑性

はじめに

小論は、複雑性理論の中で、計算のモデルとしてチューリングマシンが果たしている役割を概観したものです。

第一部では、「多項式時間」「時間計算量と空間計算量」「PとNP」といった複雑性理論の基本的な概念を説明します。

第二部では、いくつかの重要な複雑性クラスが、拡大されたチューリングマシンによって定義されることを見ていきます。

小論の第三部では、視点を変えて、現在の複雑性理論で起きている変化を紹介します。

はじめに

「多項式時間」に、理論の対象を基本的には制限したことが、その具体性・実効性で複雑性理論の発展を支えてきたのですが、複雑性理論自身の抽象化・高度化が、その「しばり」を越えようとしていると見ることができるかもしれません。

小論の第一部冒頭の「複雑性理論の背景を考える」と第三部最後の「有限の中の無限、無限の中の有限」は対応しています。

こうした循環は、果てのない「堂々巡り」に見えるかもしれません。

ただ、僕は、こうした「回帰」が、新しい認識への出発点になることを期待しています。

Agenda

チューリングマシンの拡大と複雑性

Part I

複雑性理論の基本概念

Part II

チューリングマシンと複雑性

Part III

複雑性理論の転換

Agenda Part I
複雑性理論の基本概念

- 複雑性理論の背景を考える
  - 無限と連続
  - 有限の限界
- 多項式時間
- 時間計算量と空間計算量
  - チューリングマシンで「時間複雑性」「空間複雑性」を定義する
- PとNP
  - NPというコンセプト
  - 「やさしい問題」と「難しい問題」
  - チューリングマシンでNPクラスを定義する

Part II
チューリングマシンと複雑性

- チューリングマシン
  - チューリングマシンの「命令」
  - 命令セット（遷移集合）と配置（Configuration）状態
- 決定性チューリングマシンと非決定性チューリングマシン
  - 非決定性チューリングマシン
  - 決定性チューリングマシンでのシミュレート
  - 非決定性チューリングマシンの受理と拒否
- 確率性チューリングマシンとBPPクラス
  - 非決定性チューリングマシンと確率性チューリングマシン
- チューリングマシンの拡大と複雑性
  - チューリングマシンとそれが受理する複雑性のクラス
  - 決定論と確率論 / 古典論と量子論

Agenda Part III
複雑性理論の転換

- Interactive Proof
  - Interactive Proofと証明概念の転換
  - Interactive Proofの基本的想定とパターン
  - PCP定理
- MIP*=RE定理
  - グラフの三色塗り分け問題の複雑性
  - N**Pクラスの階層とMIP*=RE定理
  - P,NPの関係の一般化とMIP*=RE定理
- 有限の中の無限、無限の中の有限

Part I

複雑性理論の基本概念

Agenda Part I
複雑性理論の基本概念

- はじめに -- 複雑性理論の背景を考える
  - 無限と連続
  - 有限の限界
- 多項式時間
- 時間計算量と空間計算量
  - チューリングマシンで「時間複雑性」「空間複雑性」を定義する
- PとNP
  - NPというコンセプト
  - 「やさしい問題」と「難しい問題」
  - チューリングマシンでNPクラスを定義する

複雑性理論の背景を考える

複雑さへのアプローチ

私たちの認識は単純なものから複雑なものに進みます。世界は複雑なものであふれているので、我々の認識もどんどん複雑になっていきます。ただ、こうした私たちの認識の前進がいつまでも続くとは限りません。私たちの認識は、しばしば、複雑さの前に立ちすくみます。複雑なものを理解するのには、長い時間と膨大な知識の集積が必要であることは、科学の歴史を見れば分かります。

僕は、対象の複雑さと認識の複雑さは同じものだと考えています。そうした認識は「複雑さ」そのものを対象とした認識が可能だということを意味します。現代の計算科学のもっとも活発な研究分野はの一つは「複雑性理論」なのですが、そうした理論が登場したことの意味は、そう考えれば、よく理解できると思います。

今回のセミナーでは、様々なチューリングマシンの拡大と複雑性のクラスの対応を紹介したいと思います。次のようなチューリングマシンを取りあげます。

- 決定性チューリングマシン
- 非決定性チューリングマシン
- 確率性チューリングマシン
- 量子チューリングマシン

先に、「複雑なものを理解するのには、長い時間と膨大な知識の集積が必要である」と書きましたが、このチューリングマシンによる複雑さのモデルでは、複雑なものを理解するのに必要な「長い時間」がチューリングマシンの「計算時間」に、「膨大な知識の集積」がチューリングマシンの「テープの長さ」に対応すると思って構いません。

「そんな単純化で大丈夫か？」

確かに。

もちろん、科学の歴史をチューリングマシンに喩えようと思っているわけではありません。

ただ、こうした単純化（「抽象化」は、単純化に他なりません）された切り口が、「認識の有限性と対象の無限性」「古典論と量子論」「決定論と確率論」といった認識の大きな問題にアプローチする一つの有効な手段を提供するという話ができればと思っています。

無限について

無限は、古来から、多くの哲学的・数学的な関心を引き付けてきました。

時間や空間が連続的で無限であると仮定すると引き起こされる「ゼノンの逆理（アキレスはかめに追いつけない）」は有名ですね。

ユークリッドは、素数が無限にあることを証明したのですが、その証明では、慎重に、「無限」という言葉を使うことを避けました。アリストテレスは、もっと踏み込んで、無限について語っています。彼は、無限を「実無限」と「可能的無限」に分けました。

数学では、連続と無限の問題は、ニュートンやライブニッツが微積分法を始めた時にも、「無限大」「無限小」の概念を巡って大きな議論が巻き起こりました。現代の数学の出発点の一つであるカントールの集合論は、無限についての数学といってもいいのですが、ここでも多くの論争がありました。

20世紀になっても、超限解析(Non-Standard Analysis)の登場や、連続体仮説の独立性証明といったトピックでも、議論は尽きません。

物理学についても、現代の物理学の歴史を「連続的なものとの格闘」という視点から振り返った数理物理学者のJohn Baezは、次のように述べています。

「すべての主要な物理学の理論は、時空が連続的なものであるという想定から生まれる数学的問題に対する挑戦であることを、この一連の投稿でみてきた。連続的なものは、無限によって、我々をおびやかす！　これらの無限は、これらの理論から予測を行う我々の能力を脅かすのだろうか？　あるいは、これらの無限は、こうした理論を正確な形で定式化する我々の能力さえも脅かしているのであろうか？」

"Struggles with the Continuum"

https://johncarlosbaez.wordpress.com/2016/09/08/struggles-with-the-continuum-part-1/

こうした無限と連続をめぐる議論は、とても興味深いものです。昨年発見された、MIP*＝RE定理も、こうした議論に一石を投じることになるのは確実です。

と言いますのは、今回フォーカスしたいのは、複雑性理論での無限と有限の捉え方だからです。誤解を恐れず単純化していうと、複雑性理論は無限を直接の対象とはしません。複雑性理論がまず関心を持つのは、有限の捉え方です。有限という概念は、決して単純ではないのです。

有限の限界を考える

有限の「限界」を考える

有限と無限は対の概念です。英語でも、"finite"と "infinite" と対になっていますね。ただ、有限と無限の対比の仕方は、日本語と英語とでは、少し違います。日本語では、有限は「限界があるもの」で、無限は「限界がないもの」ですが、英語では、「無限」は「有限でないもの」です。日本語の方が、定義が踏み込んでいます。

ここでは、「限界がある」という有限の定義で、有限には、どんな限界があるのかを考えてみましょう。

有限の身近なモデルとして、自然数を考えましょう。ある具体的な自然数nを考えて、nは有限であると考えることは自然なことのように思われます。

具体的な自然数を有限のモデルだとして、今度は、有限の「限界」を、どんな自然数nに対しても、n ＜ N なるNが存在することだと考えましょう。有限な自然数には、大きさで超えられない壁があって、それが有限の「限界」だと考えるということです。

ただ、この有限の「限界」の特徴づけは、いくつか奇妙なところがあります。

限界Nが存在して、それが自然数だとします。その時、N＋1も自然数で、N ＜ N＋1 ですので、限界はNではなく、N＋1になります。これは矛盾です。ですから限界Nが、自然数だと考えることはできません。これは、素直に考えれば、最大の自然数は存在しないということです。また、それは、具体的に与えられた自然数nより大きい自然数は「無限」に存在することを意味します。

先に、ユークリッドが素数が無限に存在することの証明を、無限という言葉を使わずに行ったと書きましたが、彼が行ったのは、次のような証明です。基本的には、上と同じ論理です。

最大の素数pが存在するとする。p以下の全ての素数の積に1を加えた数をPとする。Pは、全ての素数で割っても、1が余って割り切れない。よってPは素数で、p ＜ P で、pは最大の素数ではない。これは矛盾。よって、最大の素数は存在しない。

有限の「限界」1
自然数の「上限」ω を考える

色々奇妙なところもあるのですが、「具体的に与えられた自然数＝有限」というモデルを維持することにこだわりましょう。このモデルは、自然なものにおもえますので。

それでは、この有限のモデルで、その「限界」は、どう考えればいいのでしょう？

次のようなωを考えます（先に見たように、ωは自然数ではありません）。

「全ての自然数nについて、n ＜ ω」

そう考えれば、「自然数＝有限」というモデルの「有限の限界」は、ωで与えられることになります。

この解釈も、いささか奇妙なものです。ωは、無限個の自然数より「大きく」、かつ自然数ではないのですから、このモデルでは「有限」のものとは考えにくいのです。この解釈によれば、有限なものの「限界」は、「有限ではない」ものによって規定されることになります。n < ω という式は、「有限は、無限より小さいもの」とも解釈できます。"finite" で "infinite" を定義するのとは、まったく逆です。

「そもそも、n < ω というようなωが存在することが具体的にイメージできない」

そうかもしれません。ただ、イメージだけなら、ωをイメージするいい方法があります。

0以外の自然数nを1/nに対応させます。1は1/1に、2は1/2に、3は1/3に、4は1/4 .... nは1/n.の点に移ります。

この一対一の対応は強力です。無限に広がった無限個の自然数nは、全て0と1の間の1/nの形の有理数に閉じ込められます。nが無限に大きくなるにつれ、1/nは0に近づくのですが、この対応づけで、ωは0に対応します。

0の周辺には、無数の1/nの形の有理数が集まっています。こういう点を「集積点」と呼びます。

少し慣れれば、nと1/nの対応を意識しなくても、有限な自然数全体の「集積点」としてωをイメージすることができるようになると思います。

こうした構成は、カントールの「順序数」の構成の最初の一歩をなぞったものです。数学的には、ωのような無限を「可算無限」と呼びます。それについては、また別に説明したいと思います。

有限の「限界」 2
我々が具体的に構成できるもの「帰納的可算」

ただ、有限なものの限界をωで抑えるのは、ある意味では本質的ですが、あまりに広すぎます。「具体的に与えられた自然数 n」という言い方をしてきましたが、我々は、少なくとも、可算無限に属する自然数の集合に関して言えば、その集合を全て定義し計算できるとは限らないのです（もちろん、有限個の自然数からなる集合なら、それを計算することはできるのですが）。我々が具体的に構成できるωの部分集合を「帰納的可算」と言います。

計算可能であるという条件を付ければ、有限の「限界」の第二の候補として、「可算無限」ではなくこの「帰納的可算」が浮上します。この「帰納的可算」を与えるのが、実は、チューリングマシンなのです。この認識は、とても大事なことです。我々がチューリングマシンを学ぶべき最大の理由の一つは、ここにあります。

有限の「限界」 3
多項式で表現できる「有限」

ただ、悲しいことに、こうして有限の「限界」を狭めていっても、「帰納的可算」も広すぎて、我々の手には余ります。我々が現実的・具体的に計算できる有限な量の特徴を把握するのには、有限の「限界」の別の特徴づけが必要になります。

それには、どんなnに対しても、xが大きくなると、x^nより、n^xの方がずっと大きくなるという「多項式関数 ＜ 指数関数」という関係を利用します。x^{10000}は大きな数ですが、xが大きくなると、2^xの方がずっと大きくなります。

多項式時間へ

「多項式関数 ＜ 指数関数」という関係は、先に見た「n ＜ ω」のような「有限 ＜ 無限」という関係ではありません。どちらも有限です。ただ、「多項式関数＝有限で人間にとって扱いやすいもの」で「指数関数＝有限だが、人間には扱いにくいもの」を代表していると考えればいいと思います。

こうして、有限の「限界」の第三の候補として、「チューリングマシンで計算できて（「帰納的可算」に属するということです）、かつ、その計算が「多項式」で表される時間で 可能なもの」という「限界」が、経験的に生まれることになります。

多項式時間

「多項式時間」について

ここでは複雑性理論での「多項式時間」について話してみようと思います。

xもx^2もx^3もxの多項式です。もっと一般に、

$$a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_nx^n$$

の形の式を「xの多項式関数」と呼びます。それぞれの項が、x^nに係数a_nが掛かった形をしていて、それらの和で表される関数です。xの多項式関数をpoly(x)で表します。

ただ、2^xや3^xは、xの多項式とは呼びません。a^xは、aについてみればaの多項式ですがxについてみればxの多項式ではありません。変数xが「x乗」の形で、ある項の肩に現れる関数を「xの指数関数」と言います。

複雑性の理論では、多項式時間で計算される問題のクラスを Pで表します。もっとも基本的なクラスです。

入力の大きさの関数としての多項式

ただ、いくつか注意すべきことがあります。

第一に、何についての多項式かを決めておく必要があります。
1+1よりは、1000+1000の方が計算には時間がかかります。
一般に、入力に与えられるデータが単純なものより複雑なものに
計算には時間がかかるはずです。

入力の大きさをnで表した時、計算時間をnの関数f(n)で表します。
「多項式時間で計算される」というのは、入力の大きさをnとする時、
その計算時間が、多項式poly(n)で表せるということです。

「時間計算量」は、実際の「実行時間」ではない

第二に、注意すべきことは、計算量を時間ではかるには、二つのやり方があるということです。一つは、実際に実行時間を実時間ではかるやり方です。もう一つは、たとえて言えば、コンピュータで何クロックその処理にかかるのかをはかるやり方です。

複雑性理論での時間計算量は、後者のアプローチをとります。計算を、基本的な要素に分解して、その基本操作の何ステップで処理が終わるかで、時間をはかります。正確な定義は後で述べますが、直感的にいうとチューリングマシンの一命令の実行を時間の単位にします。

「多項式時間」という性質は、ハードに依存しない

そうすると面白いことが起きます。

ある問題が多項式時間で計算される複雑性のクラスPに属するとします。ムーアの法則で、コンピュータの処理速度が指数関数的に増大したとしても、基本動作が早くなるだけで、処理に必要なステップが短くなる訳ではありません。ですので、実際の実行時間がいくら速くなっても、その問題が多項式時間のクラスPに属することに変わりはありません。

「時間計算量」と実際の「実行時間」との関係

実行時間ではかられる計算量と計算複雑性の時間計算量は、アプローチは違うことを強調したのですが、第三に注意したいことは、そうした違いにもかかわらず、両者は結びついているということです。関心の向いている方向が違うと思っていいと思います。

そのことについては、次節で説明します。

時間計算量と空間計算量

時間・空間リソースによる複雑性の階層

先に、次のように書きました。

「複雑なものを理解するのには、長い時間と膨大な知識の集積が必要である」と。

それは、人間の認識の歴史について語ったものでしたが、具体的な計算の複雑さの尺度にも、この「長い時間と膨大な知識の集積」に対応する量があります。

それは、「その計算にどれぐらいの時間が必要か？」という時間の尺度と、「その計算にどのくらいのメモリー空間が必要か？」という空間の尺度です。

前者を「時間計算量」、後者を「空間計算量」と言います。

ある時間T_0では解けないけれど、T_0 より大きな時間 T_1では解ける 問題、あるいは、ある量のメモリー空間S_0では解けないけれど、S_0 より大きなメモリー空間 S_1では解ける 問題を考えて、多くの計算リソース（時間、空間）を要する問題を「複雑な問題」と考えることは自然なことです。

また、多くの計算リソースを要する問題を「むずかしい問題」、より少ない計算リソースを要する問題を「やさしい問題」と考えることができます。

こうした「時間複雑性」や「空間複雑性」といった階層が存在することは、ある問題をできるだけ効率的に、短いステップで少ないメモリーで書こうとするプログラマーが、日常的に意識していることです。

様々な計算複雑性があるということ

一つの問題には、それを解く複数のアルゴリズムが存在しえます。ですので、ある問題にとって、それを解くアルゴリズムの数だけ具体的な実行計算複雑性が存在しうることになります。

例えば、n個のものを大きさ順に並べ替えるSORT問題の実行計算複雑性は、bubble sortのアルゴリズムなら $O(n^2)$に、quick sort のアルゴリズムならO(nlogn)になります。

プログラマにとっての実践的な関心は、その下限を目指すことですが、下限を確定することは、理論的にも実践的にも簡単なことではありません。

アルゴリズムへの関心と複雑性理論の関心

こうした時、n^2 あるいはnlog(n)に対応した計算複雑性を考えることは、もちろんできます。n^2は、多項式時間のPに属しますが、nlog(n)は、nの多項式ではありません。Pとは別の、nlog(n)という計算複雑性を考えることは可能です。

ただ、アルゴリズムに注目する、実時間的アプローチの実践的な関心の中心は、その高速化、計算時間の下限の追求にあります。一方、多項式時間 Pというクラスを設定した複雑性理論の関心は、もう少し別のところ、もう少し抽象的な複雑性の性質に向けられています。

チューリングマシンで
「時間複雑性」「空間複雑性」を定義する

「時間計算量」と「空間計算量」を チューリングマシンで定義する

複雑性理論で、その計算モデルにチューリングマシンを用いる大きな理由の一つは、複雑性の尺度である「時間計算量」と「空間計算量」に、キチンとした定義を与えることができるからです。

「時間計算量」はある計算に必要なチューリングマシンの命令の「ステップ数」に、「空間計算量」はある計算に必要なチューリングマシンの「テープの長さ」だと考えることができます。

DTIME(f(x))とDSPACE(f(x))

「決定性チューリングマシン」で、O(f(x))で表される時間計算量の複雑性クラスをDTIME(f(x))で表し、同じく空間計算量の複雑性クラスをDSPACE(f(x))で表します。

「決定性チューリングマシン」がどのようなものかは、第二部で紹介します。

DTIME(f(x)),DSPACE(f(x))のままでもいいのですが、代表的な複雑性クラスについては、次のような別名を付けます。ここでp(n)は、入力の大きさnについての多項式です。

DTIME(f(x))とDSPACE(f(x))の別名

時間複雑性については、

P=DTIME(p(n))；「多項式時間」です。

EXP=DTIME($2^{p(n)}$)；「指数関数的時間」です。

2-EXP=DTIME($2^{2^{p(n)}}$)；「二重指数関数的時間」です。

空間複雑性については、

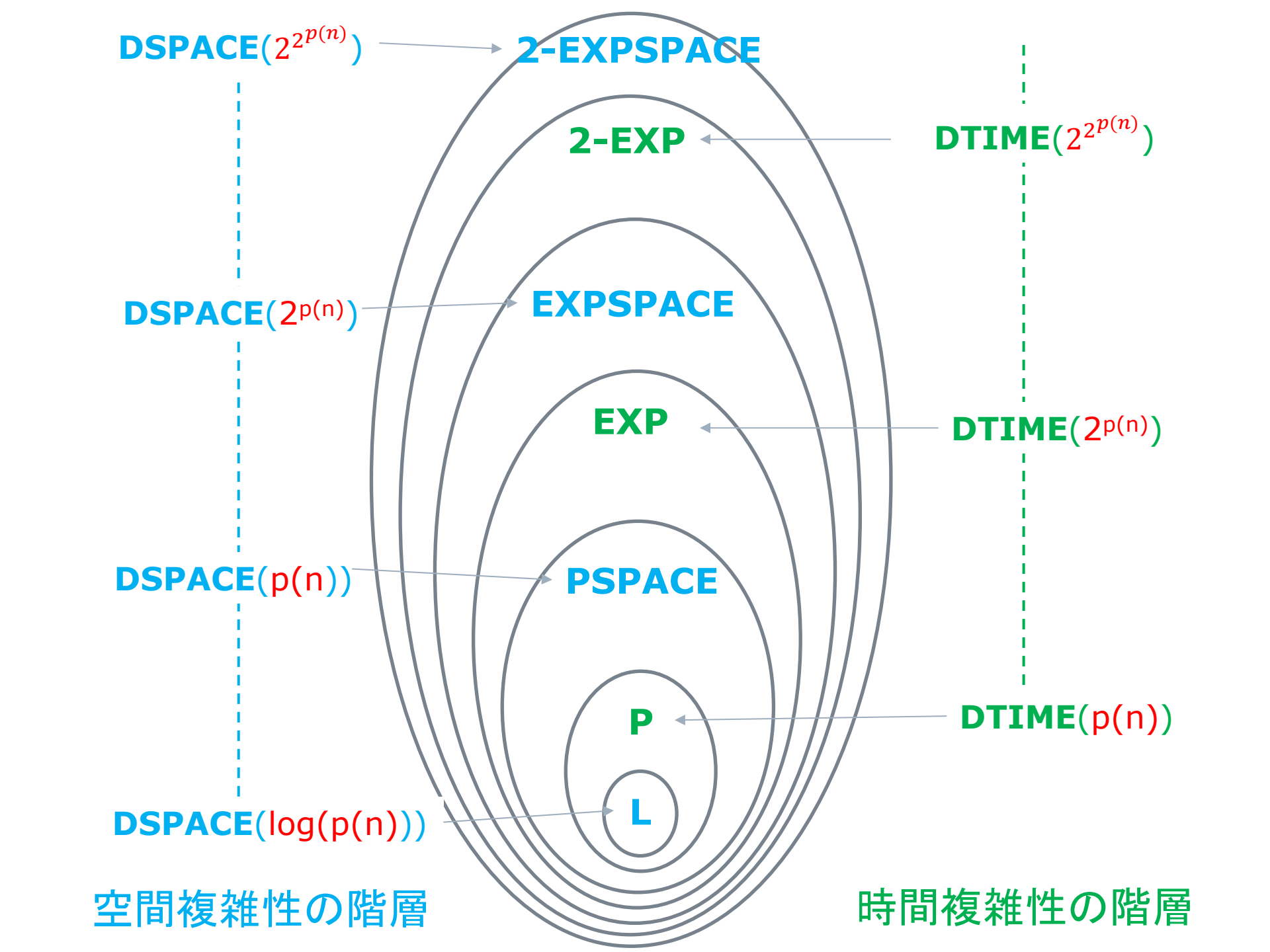
PSPACE=DSPACE(p(n))；「多項式空間」です。

EXPSPACE=DSPACE($2^{p(n)}$)；「指数関数的空間」です。

2-EXPSPACE=DSPACE($2^{2^{p(n)}}$)；

「二重指数関数的空間」です。

これらの複雑性の関係を図にしました。


DSPACE($2^{2^{p(n)}}$)
2-EXPSPACE
2-EXP
DTIME($2^{2^{p(n)}}$)
DSPACE($2^{p(n)}$)
EXPSPACE
EXP
DTIME($2^{p(n)}$)
DSPACE(p(n))
PSPACE
P
DTIME(p(n))
L
DSPACE(log(p(n)))
空間複雑性の階層
時間複雑性の階層

先の図が示すもの

時間計算量では、**P ⊂ EXP ⊂ 2-EXP** という関係が成り立ちます。空間計算量では、
PSPACE ⊂ EXPSPACE ⊂ 2-EXPSPACE という関係が成り立ちます。この辺りは、O(p(n))<O($2^{p(n)}$)<O($2^{2^{p(n)}}$)ですから、直観的にはわかりやすいかもしれません。（本当は、自明ではないのですが）

また、時間計算量と空間計算量の間では、
P ⊂ PSPACE ⊂ EXP ⊂ PSPACEです。これは面白いですね。
時間複雑性と空間複雑性が、交互に登場しています。

対数関数 ＜多項式関数 ＜ 指数関数

普段、図には入れてなかったのですが、**DSPACE**(log(p(n)))で定義される複雑性クラス**L**を入れています。

先に「多項式関数 ＜ 指数関数」であることを強調しましたが、この**L**のlog(p(n))のように、多項式の対数で表される関数について言えば、「対数関数 ＜多項式関数 ＜ 指数関数」が成り立ちます。これは、「多項式関数 ＜ 指数関数」の両辺の対数をとれば、「対数関数 ＜多項式関数」が現れることからわかります。

アルゴリズム的には複雑性をlogのオーダーに落とすことは重要ですが、複雑性理論でも、同じように重要です。

ただ、この図は、大事な複雑性クラスが抜けています。そうです。P vs.NP のNPクラスが抜けています。

次に、PとNPの問題をとりあげようと思います。

P≟NP

NPというコンセプト

NPというコンセプト

歴史的には、現代の「複雑性理論」は、NPという複雑性のクラスと「NP-完全」という概念の発見によって始まります。

NPは、ある問題とその答えが与えられた時、その答えが「正しい」ことを多項式時間で「検証」できる時間複雑性のクラスです。

この定義の双対で、その答えが「間違っている」ことを多項式時間で「検証」できるクラスをcoNPと呼びます。

先のSORT問題を NP問題として捉えると、その並べ替えが正しいことの検証は、n個のもの全てが順番に並んでいるかの n-1回のチェックで可能です。このn-1という値は下限です。最悪ケースを考えれば、これ以下にはできないことがわかります。それは、どのSORTアルゴリズムを採用するかには依存しません。

少しでも速いSORTアルゴリズムの開発に命をかけている人には、当たり前すぎてつまらないことのように思うかもしれませんが、n-1回のチェックで、ソートの正しさが検証できるというのは、ある意味、SORT問題の本質を捉えています。

「やさしい問題」と「難しい問題」

「やさしい問題」と「難しい問題」

前に見た「時間複雑性」や「空間複雑性」の階層では、「やさしい問題」と「むずかしい問題」の区別は、必要な計算リソース（時間、空間）の量的区別に帰着します。

しかし、NPというクラスが複雑性の階層に持ち込むものは、それとは違っています。それは、ある問題に答えを与えることと、その答えを検証することとは違うものだという質的区別です。

ある問題に答えを与えるのは「むずかしい問題」だとしても、その答えが正しいかをチェックするのは、それよりも「やさしい問題」になります。数学の証明を行うのは一般に「むずかしい」のですが、その証明が正しいかを検証するのは、証明することと比べれば「やさしい」のです。

「証明は「むずかしい」が検証は「やさしい」！」

このイメージはとても大事です。

もちろん、このことは、NP問題自身が、「やさしい問題」であることを意味しません。それは、問題自体の証明と比較して、その検証が「やさしい」ことを意味しているだけです。

NP問題というクラスは、とても豊かなものです。

例えば、全ての数学的証明は、NP問題です。なぜなら、その証明が「正しい」ことを、我々が理解できるものでなければ、証明としての意味を持たないからです。それは、その証明の各ステップを、我々が有限の時間の中で後追いでチェックできるということを意味します。ここでは、我々人間にとって有限の時間とは、指数関数ではなく多項式で表される時間だと考えています。

チューリングマシンでNPクラスを定義する

NPのもう一つの定義
Nondeterministic Polynomial time

先に、「NPは、ある問題とその答えが与えられた時、その答えが「正しい」ことを多項式時間で「検証」できる時間複雑性のクラスです。」という定義を述べたのですが、NPには、もう一つの定義があります。

それは、「非決定性チューリングマシンで、多項式時間で計算できるもの」という定義です。「非決定性チューリングマシン」がどのようなものであるかは、次の第二部で紹介します。

NPは、"Not P (多項式時間ではない)"ではありません。それは、**Nondeterministic Polynomial time** の略です。

DTIME(f(x))とNTIME(f(x))

「決定性チューリングマシン」で、O(f(x))で表される時間計算量の複雑性クラスを**DTIME**(f(x))で表したように、

「非決定性チューリングマシン」で、O(f(x))で表される時間計算量の複雑性クラスを**NTIME**(f(x))で表します。

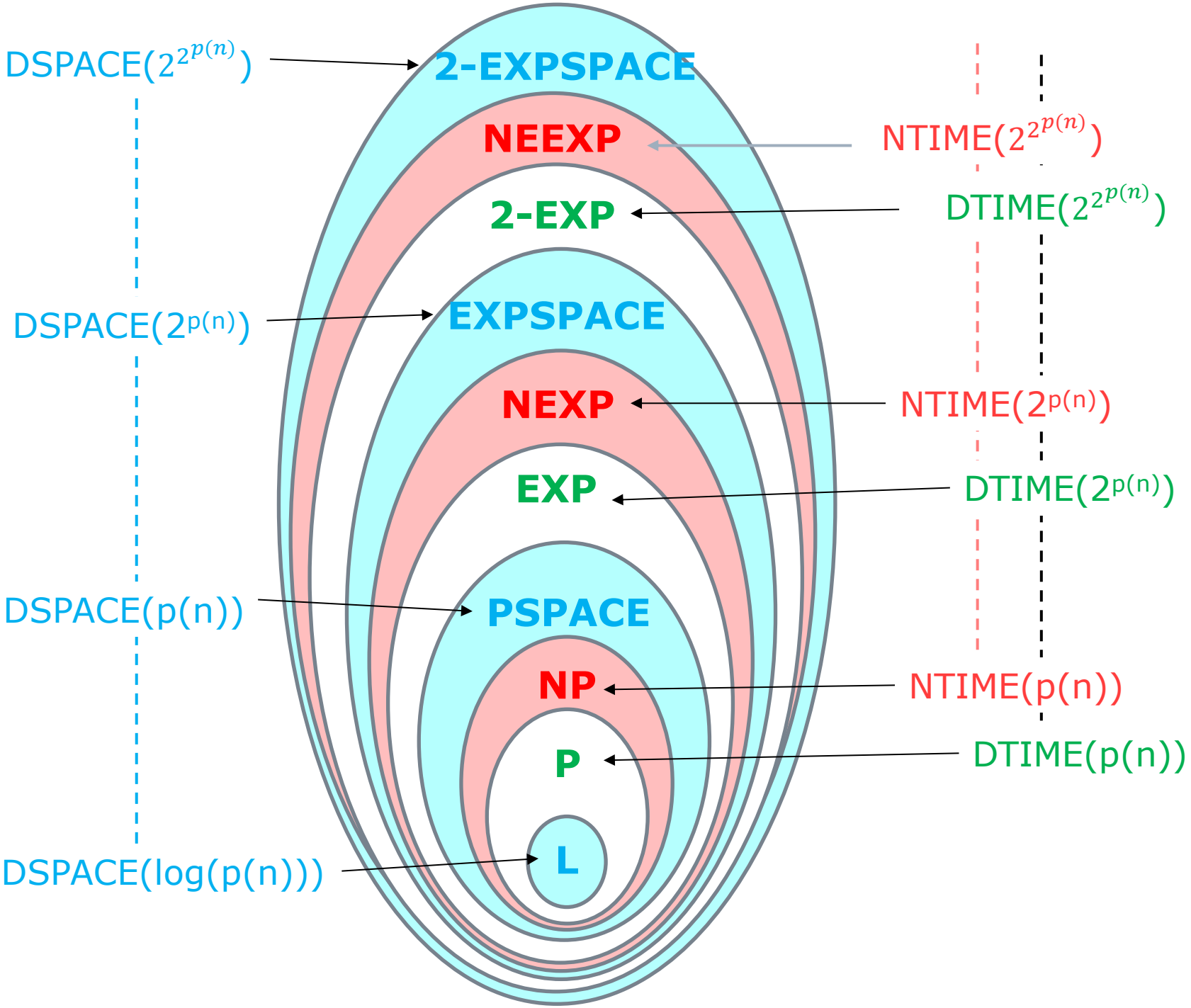
DTIMEの時と同じように、f(x)の形に応じて、別名を付けます。

NP=**NTIME**(p(n)) ;

NEXP=**NTIME**($2^{p(n)}$) ;

NEEXP=**NTIME**($2^{2^{p(n)}}$) ;

これらを先の図に付け加えてまとめてみました。次の図を見てください。


DSPACE($2^{2^{p(n)}}$)
2-EXPSPACE
NEEXP
NTIME($2^{2^{p(n)}}$)
2-EXP
DTIME($2^{2^{p(n)}}$)
DSPACE($2^{p(n)}$)
EXPSPACE
NEXP
NTIME($2^{p(n)}$)
EXP
DTIME($2^{p(n)}$)
DSPACE(p(n))
PSPACE
NP
NTIME(p(n))
P
DTIME(p(n))
L
DSPACE(log(p(n)))

「多項式時間」しばり

先の図は、可能な全ての複雑性の階層を表しているわけではありません。

DTIMEにしろDSPACEにしろNSPACEにしろ、それを定義している f(x)が、対数関数 ＜ 多項式関数 ＜ 指数関数 ＜ 二重指数関数 ＜ 三重指数関数 ＜ …. と複雑になるにつれて、図はどんどん上に伸びていきます。こうした拡張については、第三部で取りあげます。

ただ、我々人間が実際に計算可能なものからは、どんどん離れていきます。現実的には、次のような複雑性の階層図がよく用いられます。次の図は、先の階層図のPSPACEの「内側」に対応していることを確認してください。

複雑性クラスと問題の例

より難しい

n x *n* チェス

n x *n* 碁

箱詰め問題

地図の塗り分け

トラベリング・セールスマン

n x *n* 数独

グラフ同型問題

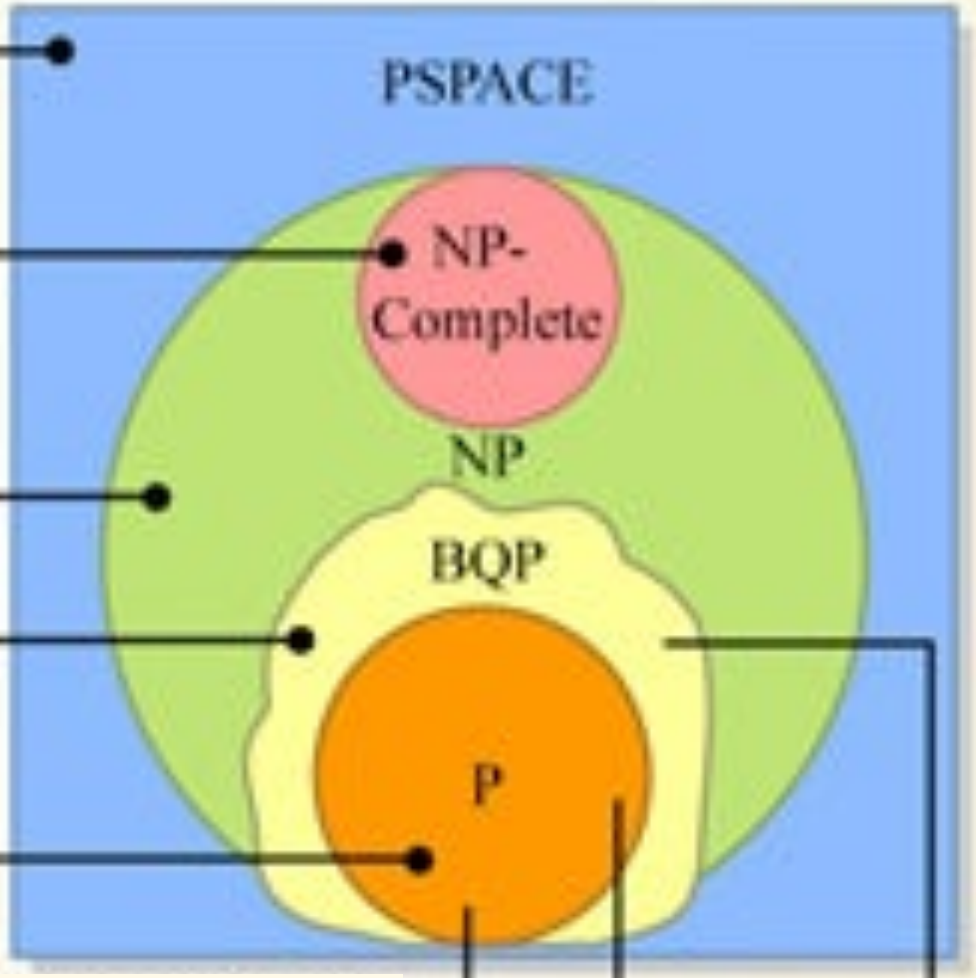
素因数分解

離散対数

グラフの接続性

素数判定

マッチ・メーキング



古典コンピュータで
効率的に解ける問題

量子コンピュータで
効率的に解ける
問題

Part II

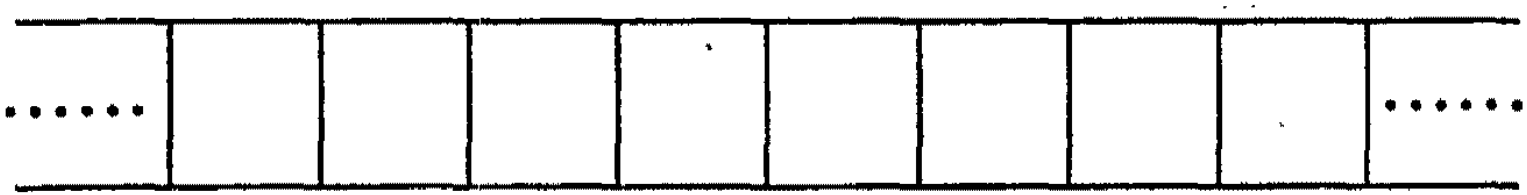
チューリングマシンと複雑性

Part II
チューリングマシンと複雑性

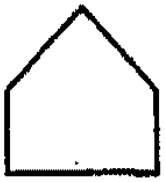
- チューリングマシン
  - チューリングマシンの「命令」
  - 命令セット（遷移集合）と配置（Configuration）状態
- 決定性チューリングマシンと非決定性チューリングマシン
  - 非決定性チューリングマシン
  - 決定性チューリングマシンでのシミュレート
  - 非決定性チューリングマシンの受理と拒否
- 確率性チューリングマシンとBPPクラス
  - 非決定性チューリングマシンと確率性チューリングマシン
- チューリングマシンの拡大と複雑性
  - チューリングマシンとそれが受理する複雑性のクラス
  - 決定論と確率論 / 古典論と量子論

チューリングマシン

チューリング・マシンのメカニズム



テープ（マスで区切られている）



ヘッド

テープ：ひとマスずつに区切られており、左右に移動する。長い。

ヘッド：テープのひとマスの記号を読み取り、マシンの「状態」に応じて、次の動作をする。

可能な動作：

1. マス目に記号を書き込む（消す＋書く）。あるいは、そのままにして何も書き込まない。
2. マシンの状態を変える。あるいは状態を同じに保つ。
3. ヘッドを、右あるいは左に移動する。（テープが動く）

チューリング・マシンのプログラムの例

Turingマシンのプログラムの例を示す。この表の、例えば、最初の State Aの行で、On '0' の欄の 'B1R' は、「状態Aで、ヘッドが'0'の上にあるなら、状態をBに変えて、1を書き込んで、ヘッドを右（R）に移動する」という命令を表す。

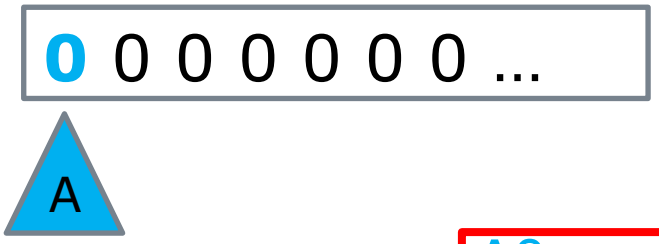
State	on 0	on 1	on 0			on 1		
			Print	Move	Goto	Print	Move	Goto
A	B1R	D0L	1	right	B	0	left	D
B	C1R	F0R	1	right	C	0	right	F
C	C1L	A1L	1	left	C	1	left	A
D	E0L	H1L	0	left	E	1	left	H
E	A1L	B0R	1	left	A	0	right	B

プログラム本体

プログラムの説明

Hは特別な「状態」で、この状態の時、マシンは「停止」する

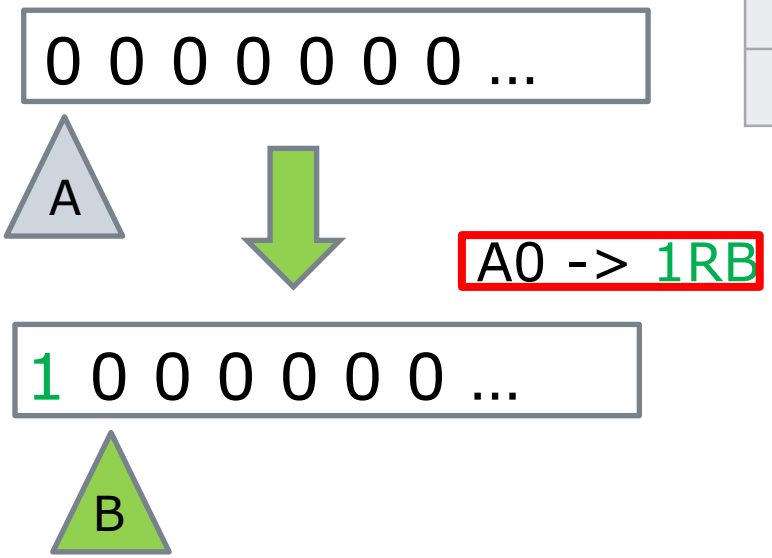
	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



A0 -> 1RB

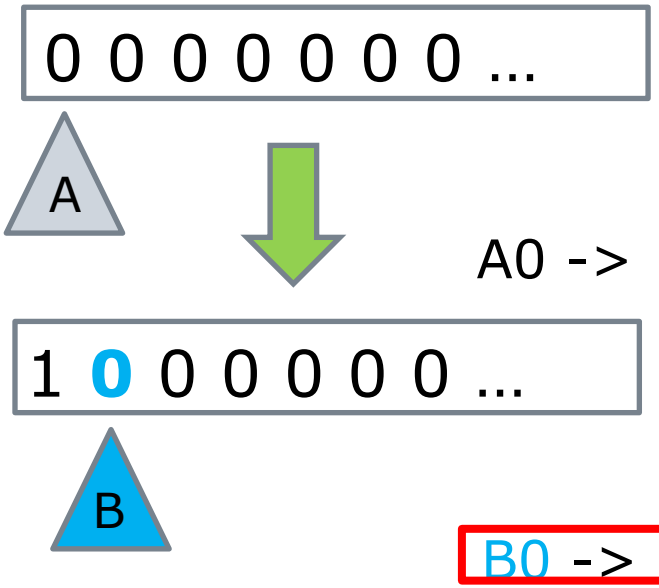
状態Aでヘッドが0の上にあるなら

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



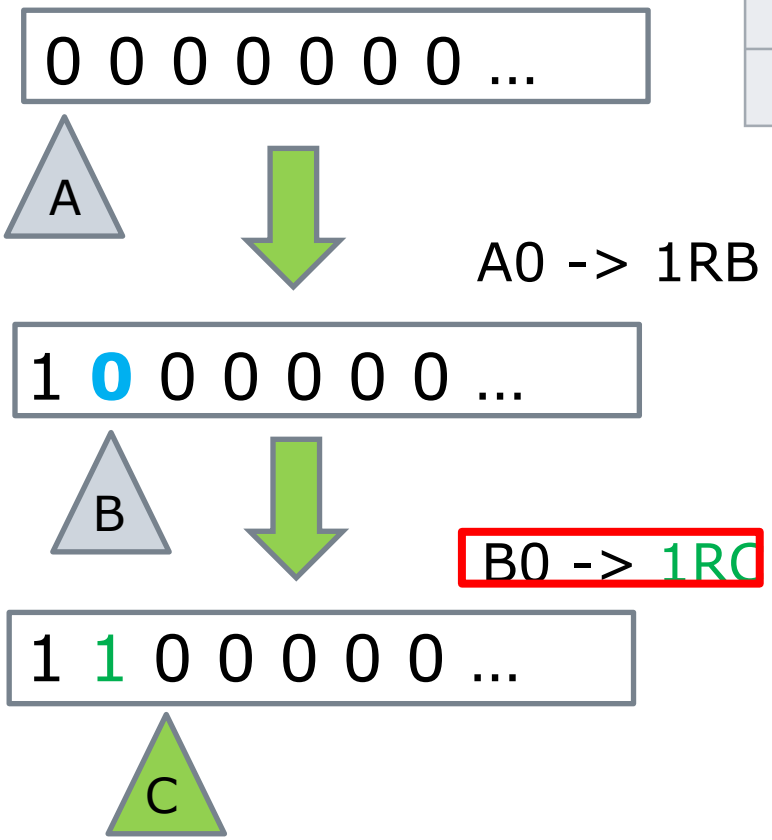
状態Aでヘッドが0の上にあるなら
1を書き込み、右にヘッドを移動し
状態をBに変える。

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



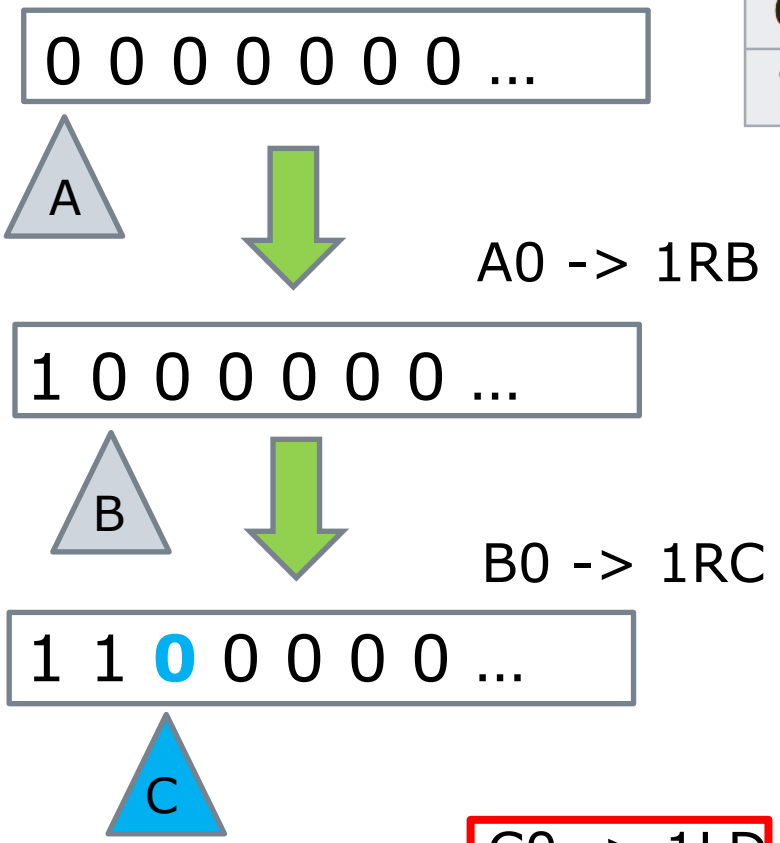
状態Bでヘッドが0の上にあるなら

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



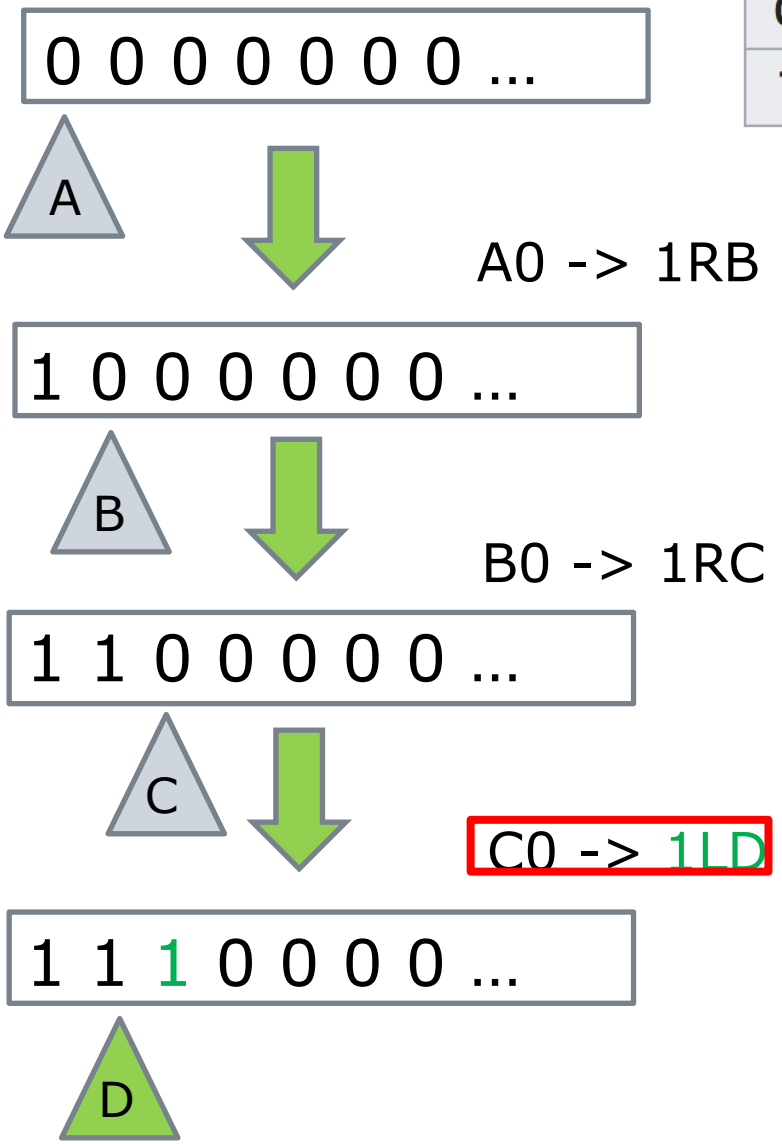
状態Bでヘッドが0の上にあるなら
1を書き込み、右にヘッドを移動し
状態をCに変える。

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



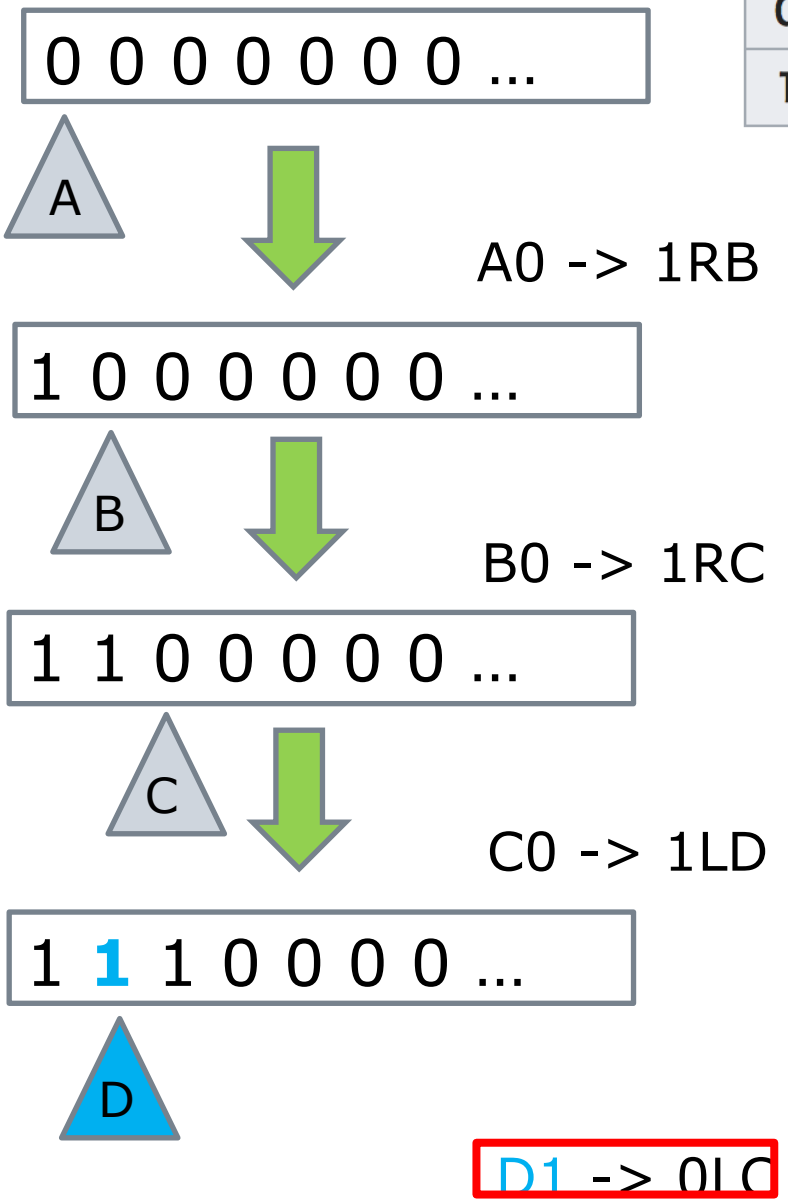
状態Cでヘッドが0の上にあるなら

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



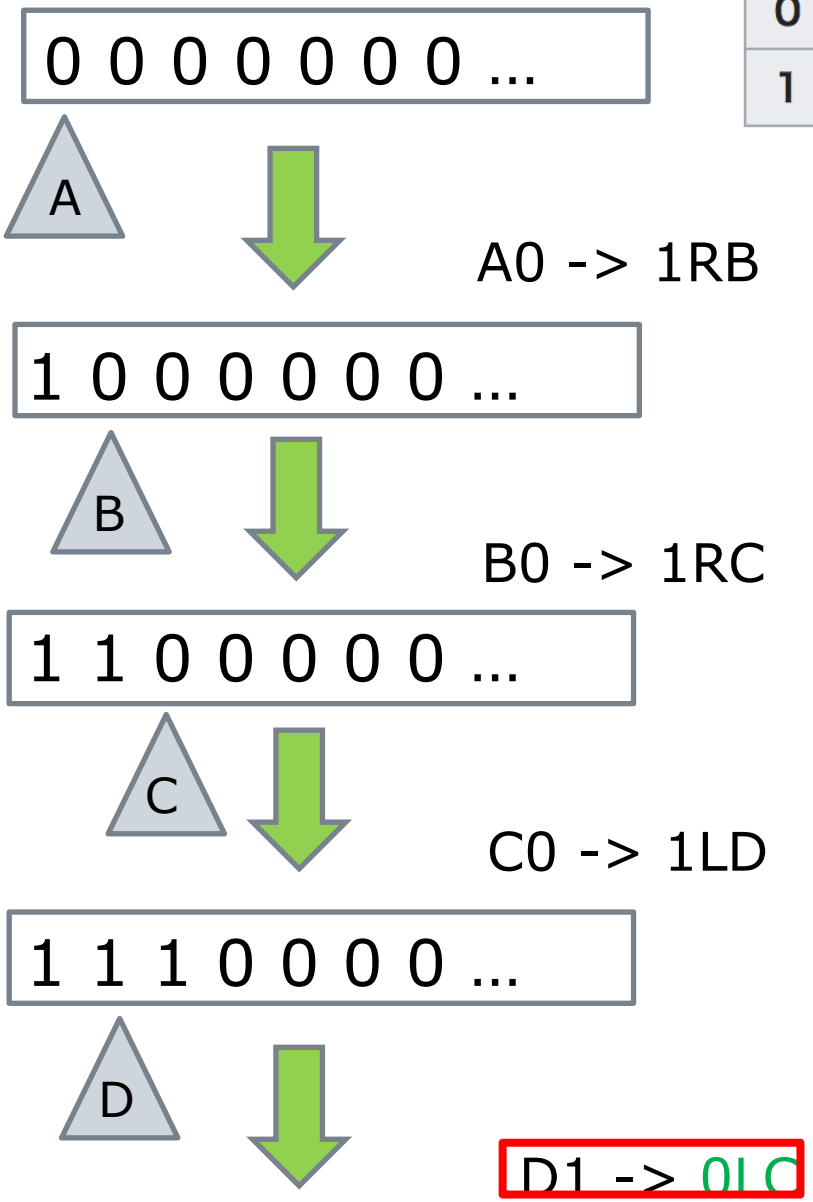
状態Cでヘッドが0の上にあるなら
1を書き込み、左にヘッドを移動し
状態をDに変える。

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



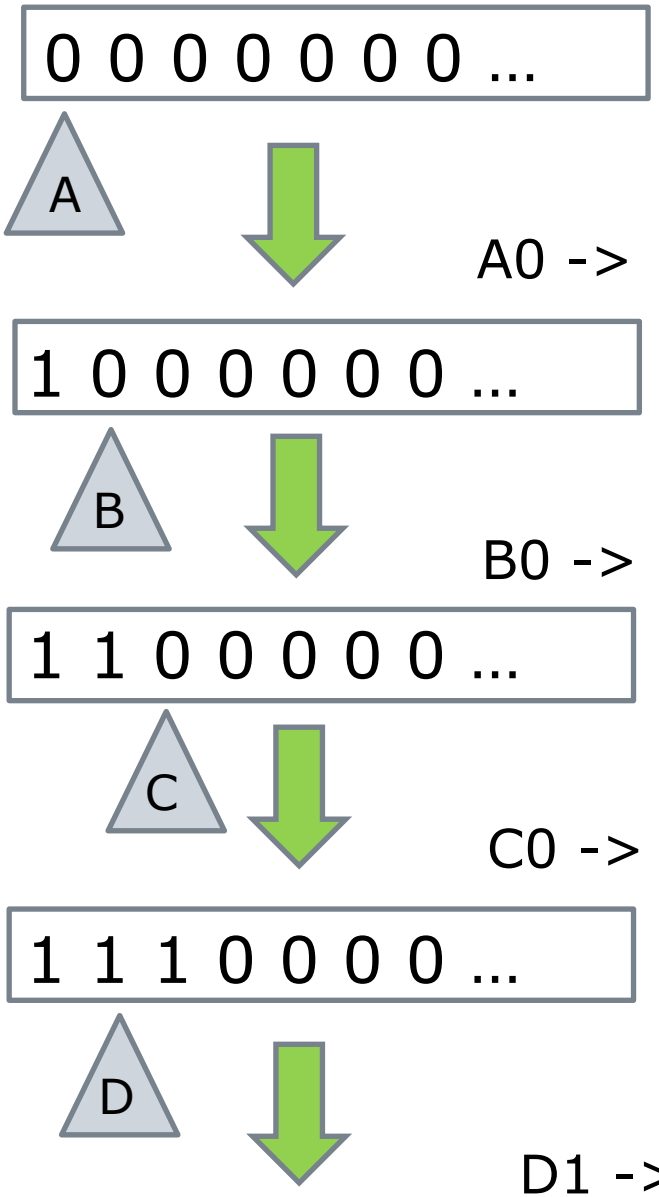
状態Dでヘッドが1の上にあるなら

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



状態Dでヘッドが1の上にあるなら
0を書き込み、左にヘッドを移動し
状態をCに変える。

	A	B	C	D	E	F
0	1RB	1RC	1LD	1RE	1LA	1LH
1	1LE	1RF	0RB	0LC	0RD	1RC



状態Aでヘッドが0の上にあるなら
1を書き込み、右にヘッドを移動し
状態をBに変える。

状態Bでヘッドが0の上にあるなら
1を書き込み、右にヘッドを移動し
状態をCに変える。

状態Cでヘッドが0の上にあるなら
1を書き込み、右にヘッドを移動し
状態をCに変える。

状態Dでヘッドが1の上にあるなら
0を書き込み、左にヘッドを移動し
状態をCに変える。

チューリングマシンの「命令」

チューリングマシンの「命令」
命令の構成要素

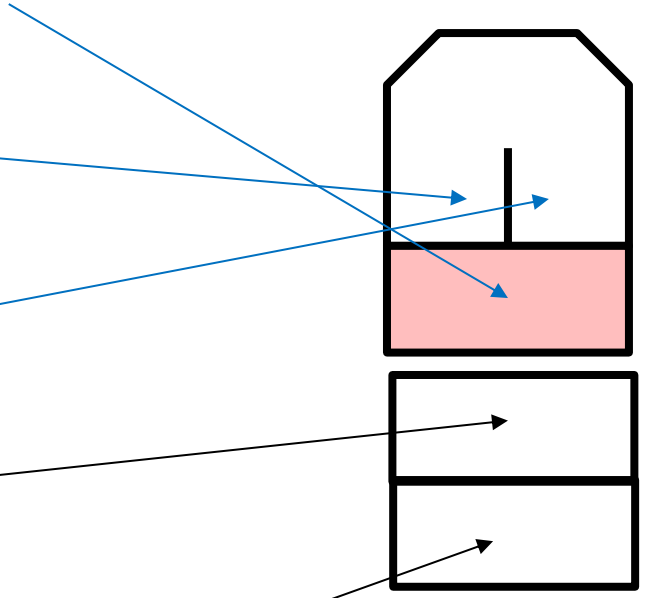
- チューリングマシンの命令は、次の要素から構成されている。

1. 現在のチューリングマシンの状態　(State-Now)
2. ヘッド位置のテープ上の一文字　　(Char-Read)
3. ヘッド位置に書き出される一文字　(Char-Write)
4. ヘッドが次に進む方向 (R, L)　　(Move-Direction)
5. 次のステップでのチューリングマシンの状態　(Next-State)

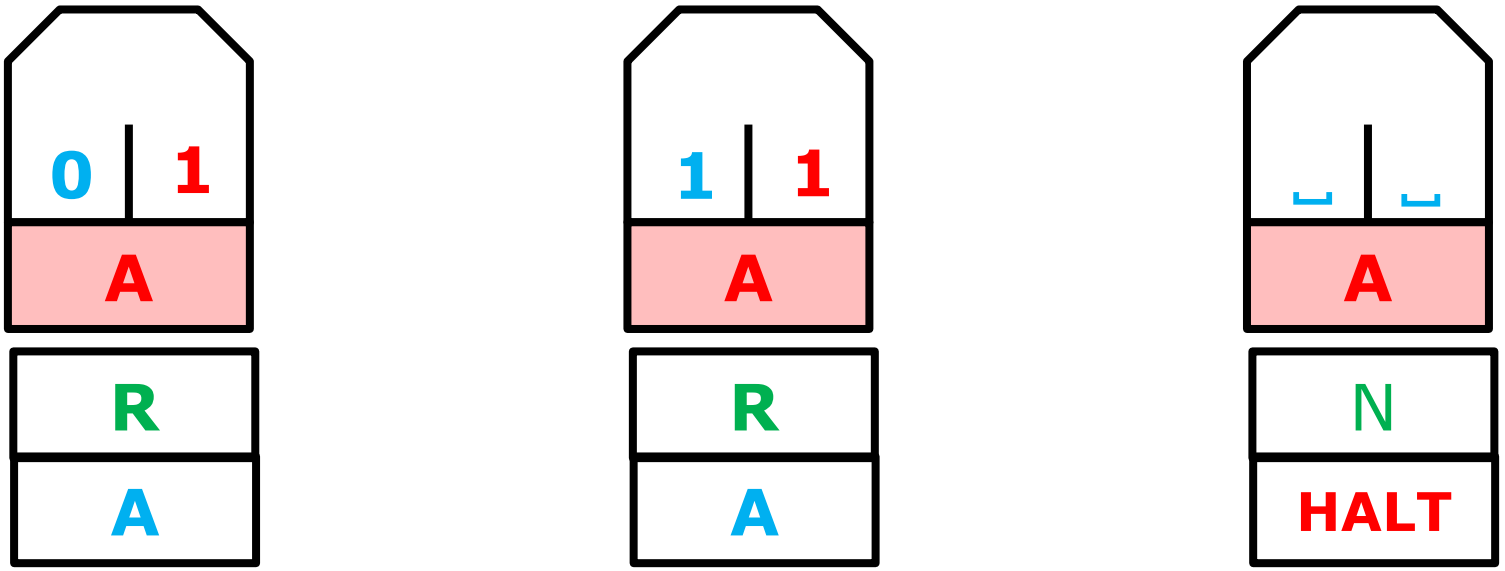
チューリングマシンの「命令」を図で表す

- チューリングマシンの命令を、次のように図で表すことができる。

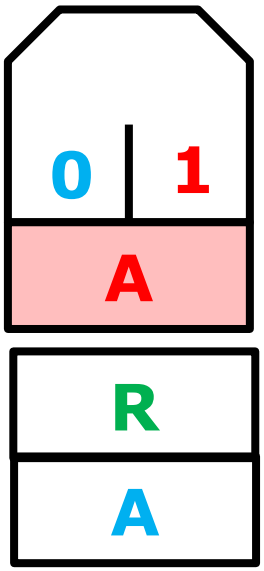
1. 現在のチューリングマシンの状態 (State-Now)
2. ヘッド位置のテープ上の一文字 (Char-Read)
3. ヘッド位置に書き出される一文字 (Char-Write)
4. ヘッドが次に進む方向 (R, L) (Move-Direction)
5. 次のステップでのチューリングマシンの状態 (Next-State)

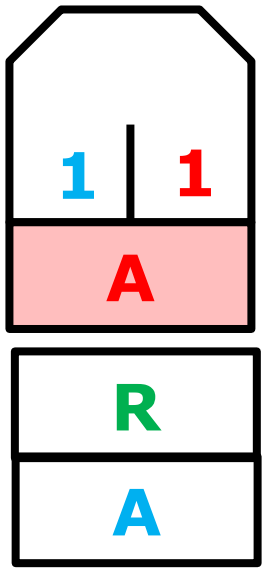


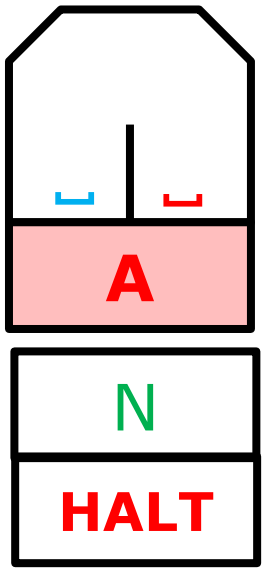
次のものは、図で表された
チューリングマシンの命令である



図で表された命令を
文字列で表す







A0→1:RA　　A1→1:RA　　A␣→␣:NHALT

文字列で表された命令の意味

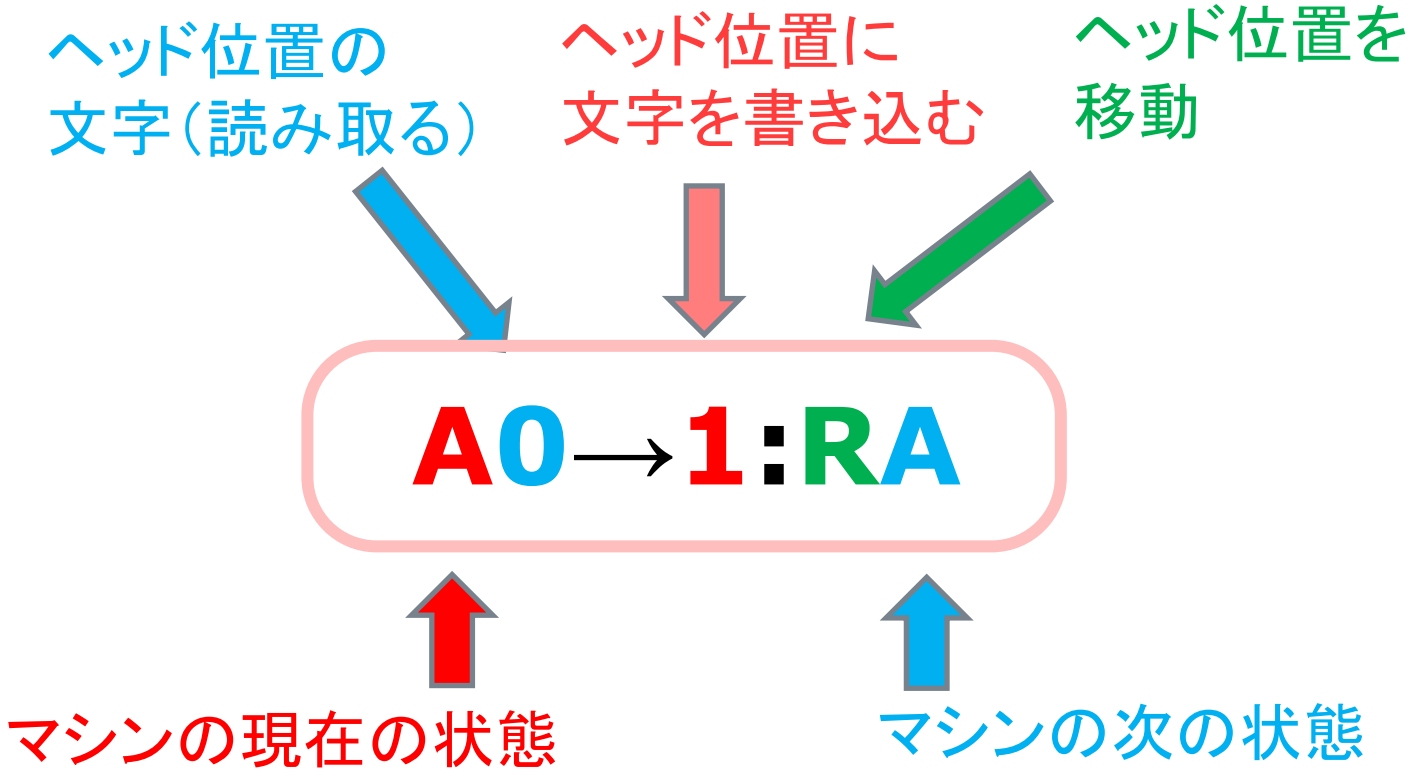
文字列で表された命令は、次のような意味を持つ

- **A0→1:RA**
  - 現在の状態が**A**でヘッドが**0**の上にあるなら
    ヘッド位置に**1**を書き込み、ヘッドを右に進め、状態を**A**にする。

- **A1→1:RA**
  - 現在の状態が**A**でヘッドが**1**の上にあるなら
    ヘッド位置に**1**を書き込み、ヘッドを右に進め、状態を**A**にする。

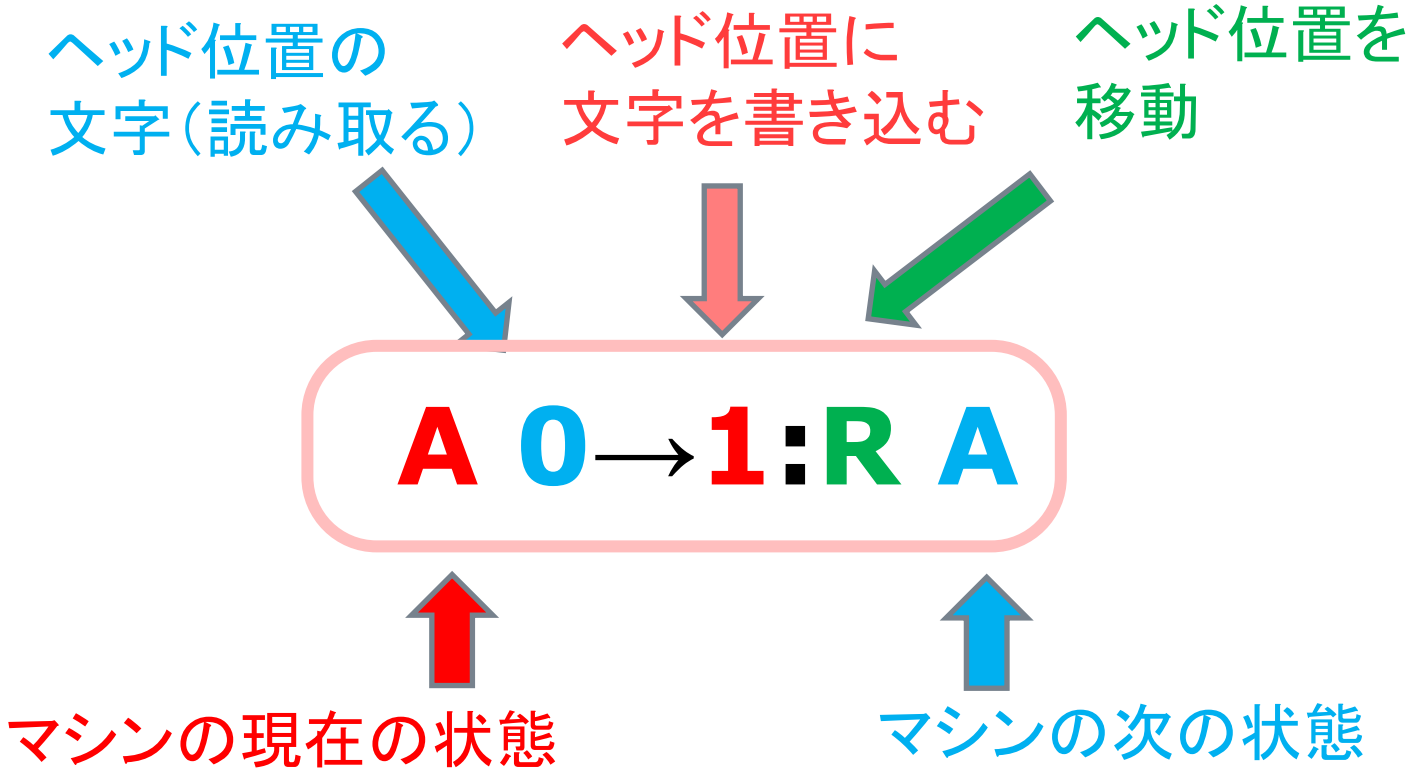
- **A␣→␣:NHALT**
  - 現在の状態が**A**でヘッドが␣（空白）の上にあるなら
    状態を**HALT**にして、停止する。

文字列で表された命令の意味



現在の状態がAでヘッドが0の上にあるなら、ヘッド位置に1を書き込み、ヘッドを右に進め、状態をAにする。

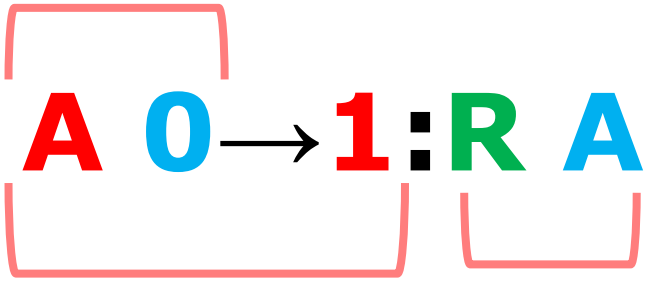
文字列で表された命令（修正版）



現在の状態がAでヘッドが0の上にあるなら、ヘッド位置に1を書き込み、ヘッドを右に進め、状態をAにする。

文字列で表された命令の意味

命令を特徴付ける
基本的な情報
状態＋ヘッド上の文字



現時点のヘッド位置
での状態とヘッド上の
文字の読み・書き

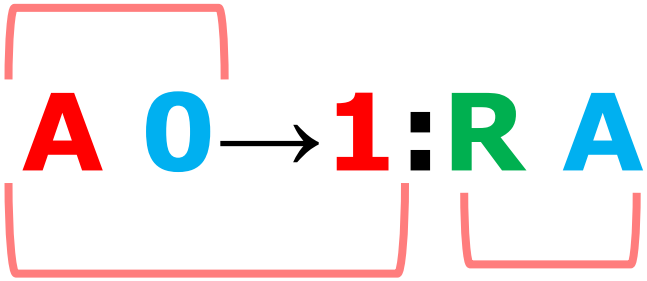
ヘッドの移動と
移動後の新しい
状態

現在の状態がAでヘッドが0の上にあるなら、ヘッド位置に1を書き込み、ヘッドを右に進め、状態をAにする。

命令セット（遷移集合）と配置（Configuration）状態

文字列で表された命令の意味

命令を特徴付ける
基本的な情報
状態＋ヘッド上の文字



現時点のヘッド位置
での状態とヘッド上の
文字の読み・書き

ヘッドの移動と
移動後の新しい
状態

現在の状態がAでヘッドが0の上にあるなら、ヘッド位置に1を書き込み、ヘッドを右に進め、状態をAにする。

命令δを関数として表す

Q :マシンがとりうる状態の集合
∑ :テープ上で読み書きされる文字の集合 アルファベット
D :移動方向 {R, L} とする

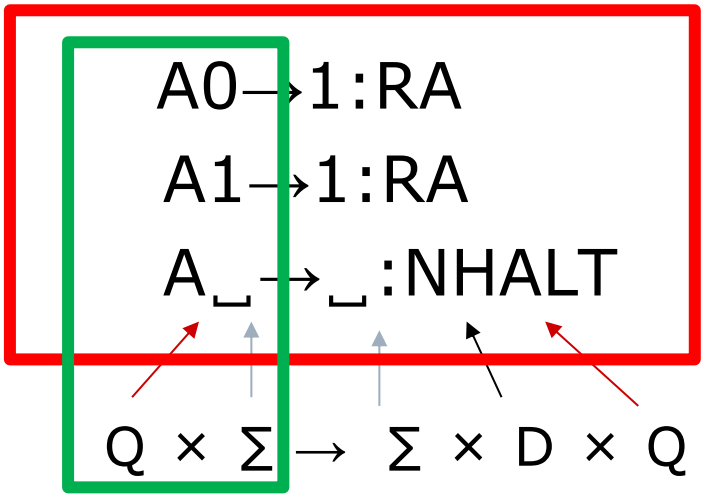
δ_i: **A** **0** → **1** : **R** **A**
Q ∑ ∑ D Q

$$\delta: Q \times \Sigma \rightarrow \Sigma \times D \times Q$$

命令セット（遷移表）の例

あるチューリングマシンの、Q × Σ → Σ × D × Q の形の関数の集まりを、命令セット（遷移表）δ と呼ぶ。

次は、三つの命令からなる命令セットである。



δのメンバーは、関数の二つの引数 Q と Σ、すなわち、マシンの状態とヘッド上の文字によって、一意に決まる。

命令実行のある時点でのチューリングマシンの配置状態（configuration）

命令の実行のある時点での、

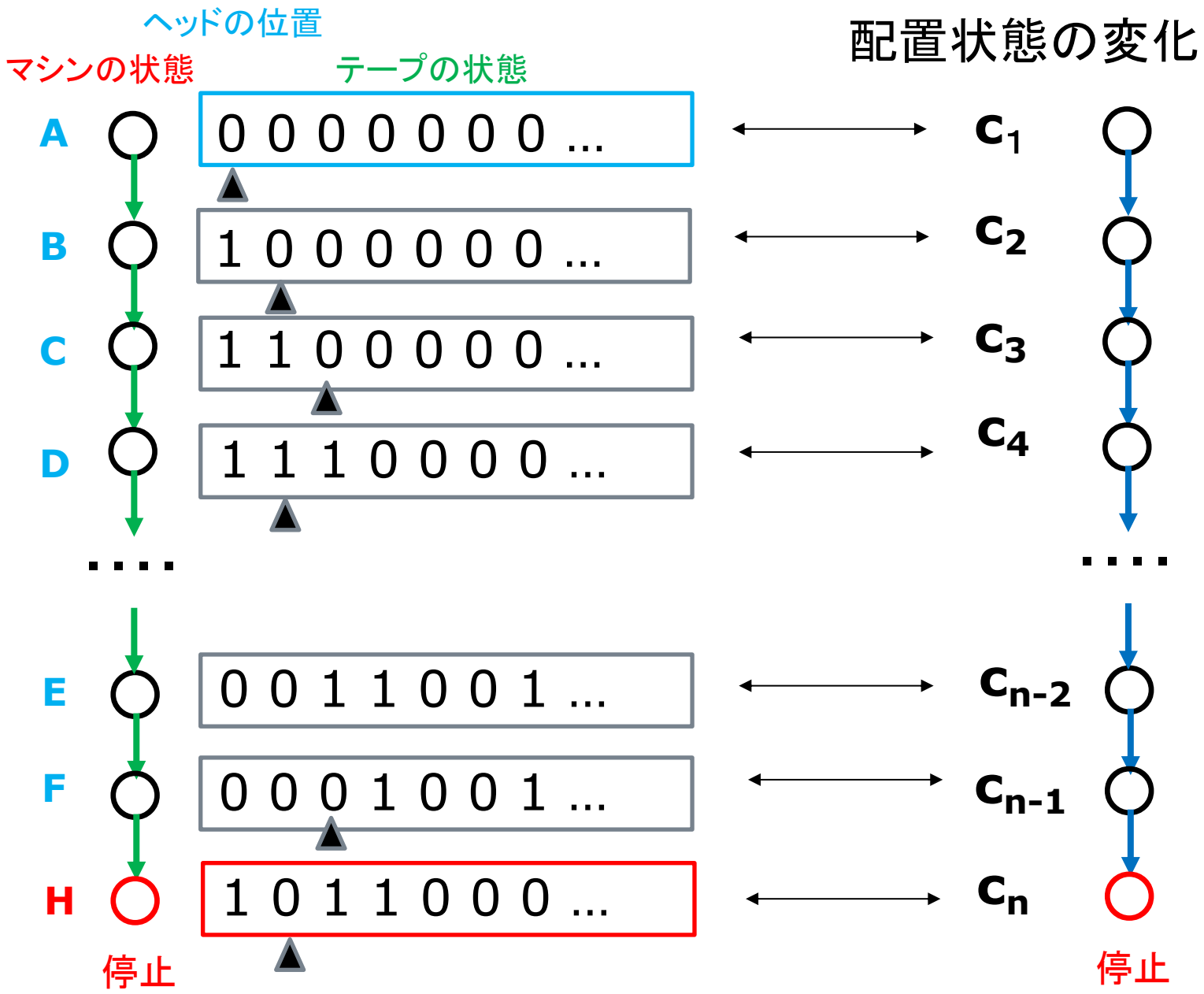
（1） マシンの状態Qと、

（2） テープ上の文字列の状態と、

（3） どの位置にヘッドがあるか

をあわせた、チューリングマシン全体の状態を「配置状態」と呼ぶ。

次の図のように、チューリングマシンの配置状態の変化は、一直線に進む。


ヘッドの位置
マシンの状態
テープの状態
配置状態の変化
A
B
C
D
E
F
H
0 0 0 0 0 0 0 0 …
1 0 0 0 0 0 0 0 …
1 1 0 0 0 0 0 0 …
1 1 1 0 0 0 0 0 …
0 0 1 1 0 0 1 …
0 0 0 1 0 0 1 …
1 0 1 1 0 0 0 …
c_1
c_2
c_3
c_4
c_{n-2}
c_{n-1}
c_n
停止
停止

決定性チューリングマシンと
非決定性チューリングマシン

チューリングマシンと帰納的可算集合

自然数の集合Sは、次の条件を満たす
チューリング・マシンMが存在する時、
「帰納的可算」と呼ばれる

- $n \in S$ なら、Mは停止する。

受理状態と拒否状態

マシンの状態がHALTの時、配置状態が、あらかじめ定義された
（1）受理状態　(Accept)
（2）拒否状態　(Reject)
の二つのどちらかの状態をとるチューリングマシンを
「決定性チューリングマシン」と呼ぶ。

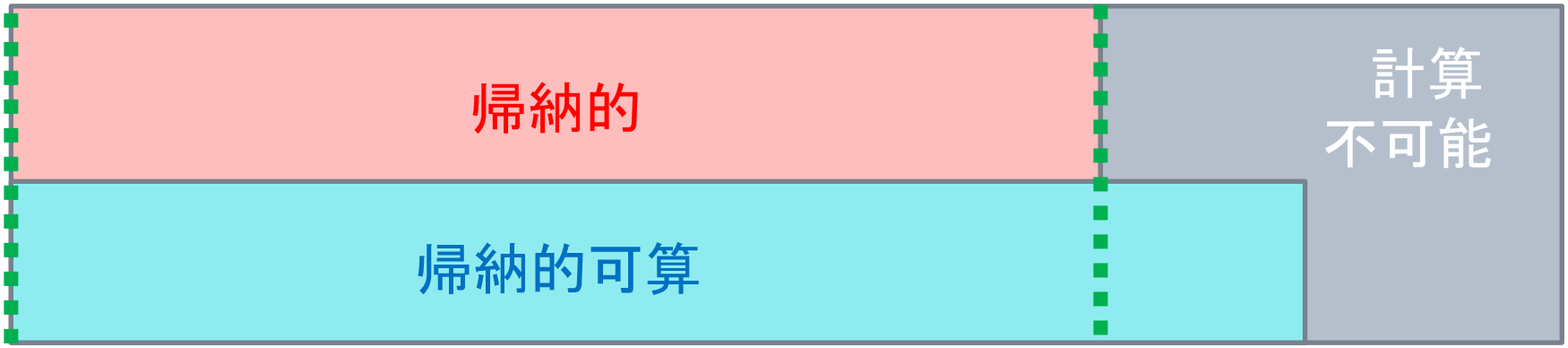
Mが決定性チューリングマシンの時、

$s \in L \iff$ Mは受理状態で停止

$s \notin L \iff$ Mは拒否状態で停止

決定性チューリングマシンと帰納的集合

自然数の集合Sは、次の条件を満たす
チューリング・マシンMが存在する時、
「帰納的」と呼ばれる

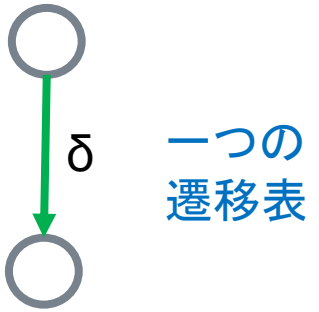


- $n \in S$ なら、Mは受理状態で停止する。
- $n \notin S$ なら、Mは拒否状態で停止する。

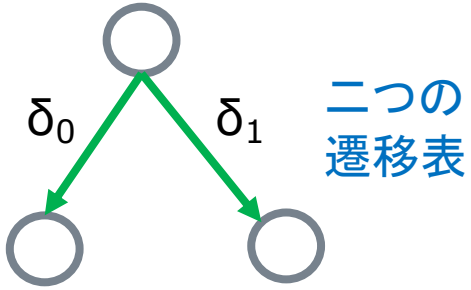
非決定性チューリングマシン

二つの遷移表（命令セット）を持つチューリングマシンを考える

- 決定性チューリングマシン）は、一つの遷移表（命令セット）δを持つ。この時、マシンの配置状態を一つのノードとみなしてマシンの実行を、ノードからノードへの遷移のグラフとして考えると、このグラフは、枝分かれのない、すべてのノードが一直線上に並んだものになる。

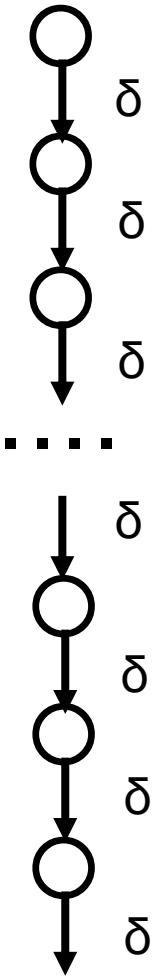


- 二つの遷移表（命令セット）δ_0とδ_1を持つチューリングマシンを考える。あるノード上で、δ_0とδ_1のどちらの命令セットを選ぶかは決まっていない。この時、このマシンで可能な実行のグラフは、木構造になる。こうしたマシンを非決定性チューリングマシンという。

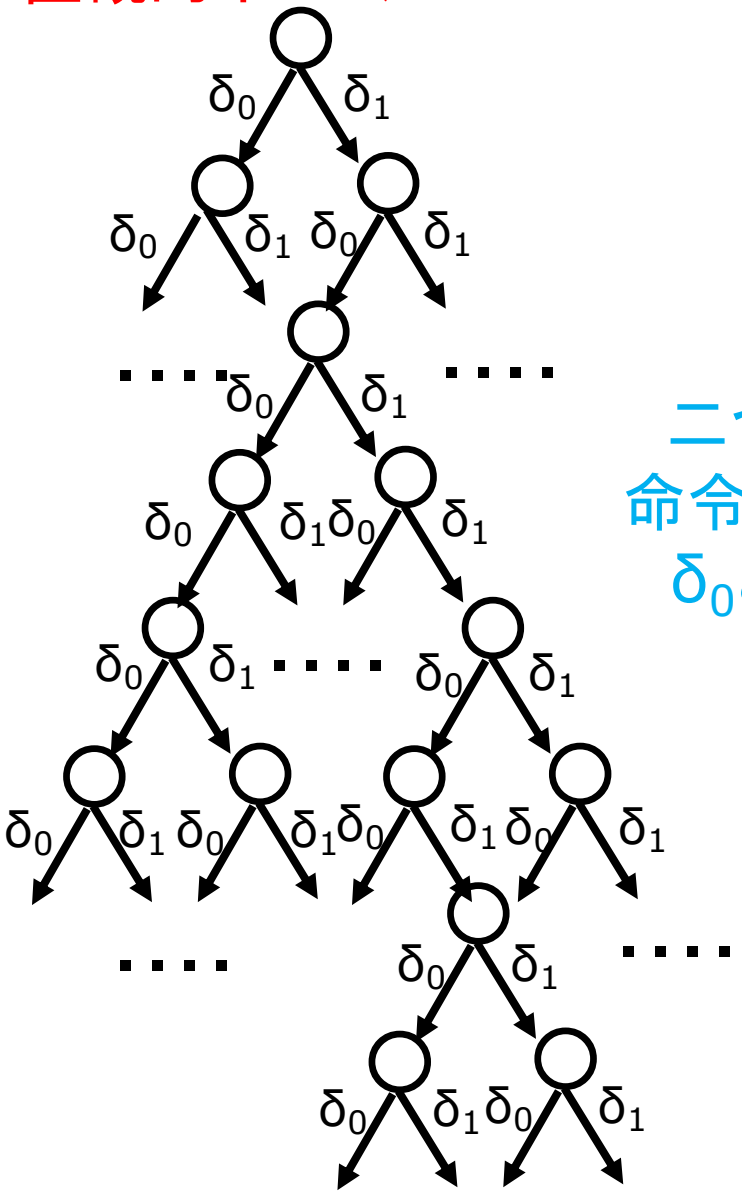


- 実は、遷移表は、二つ以上あってもいい。

配置状態の変化の直観的イメージ



一つの
命令セット
δ



二つの
命令セット
δ_0とδ_1

決定性チューリングマシン

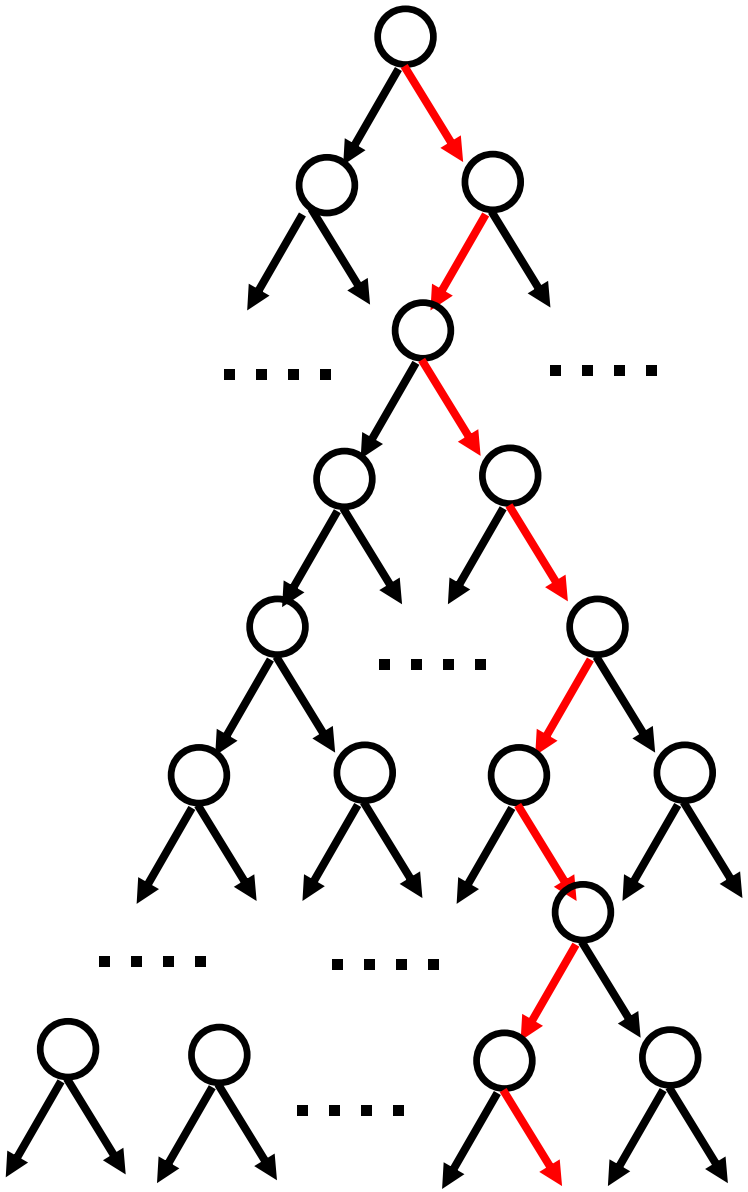
非決定性チューリングマシン

非決定性チューリングマシンでの受理と拒否

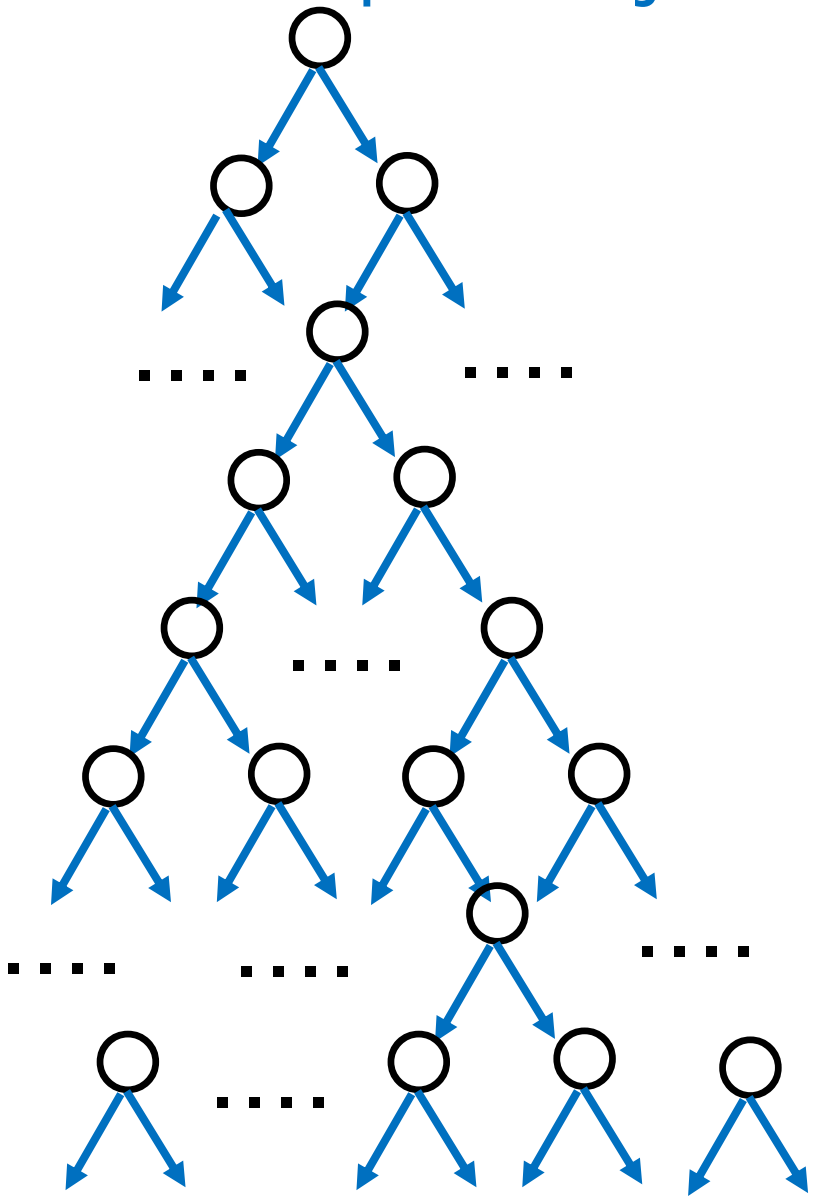
- 非決定性チューリングマシンNでの受理(accept)と拒否(reject)は、次のように定義される。

- Nがwを受理するのは、wを入力とするNの木構造で、accept状態で終わるパスが一つでもある場合である。
- Nがwを不受理とするのは、wを入力とするNの木構造で、全てのパスが reject状態で終わる場合である。

非決定性チューリングマシンのAccept と Reject



一つでも**accept**で終わるパスがある時

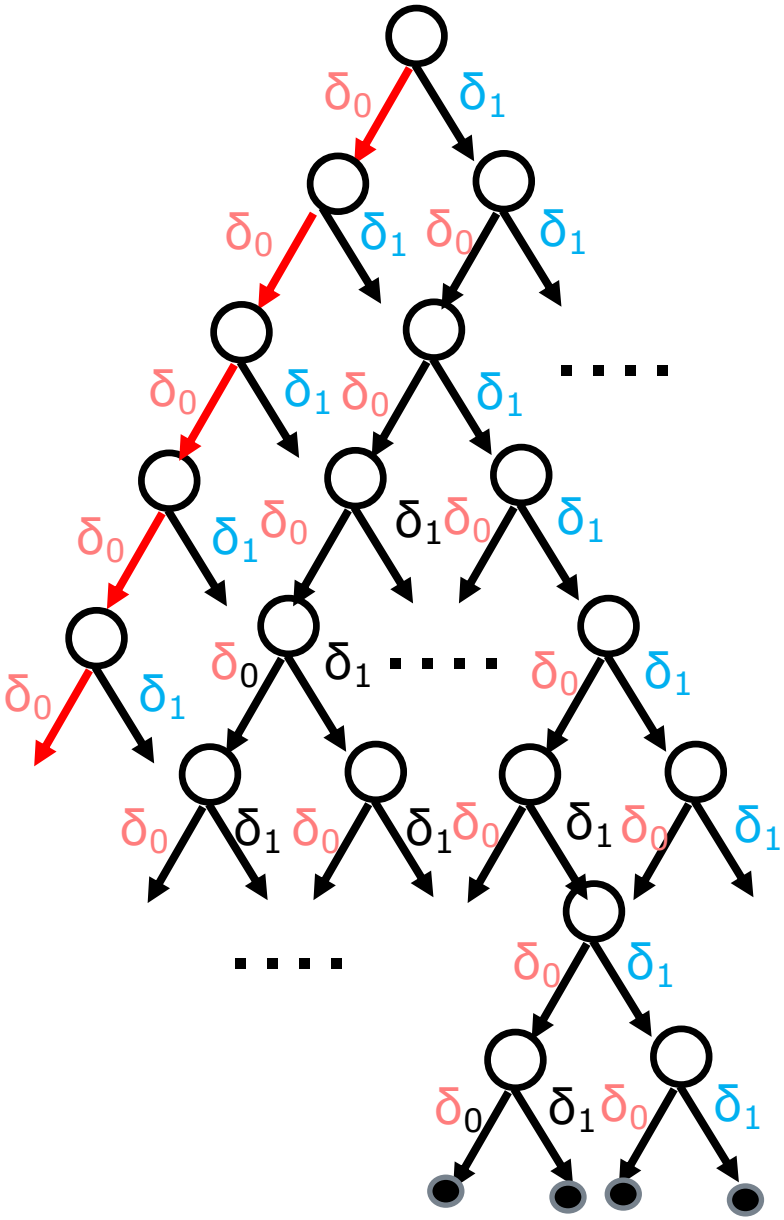


全てのパスが**reject** で終わる場合

非決定性チューリングマシンを決定性チューリングマシンでシミュレートする

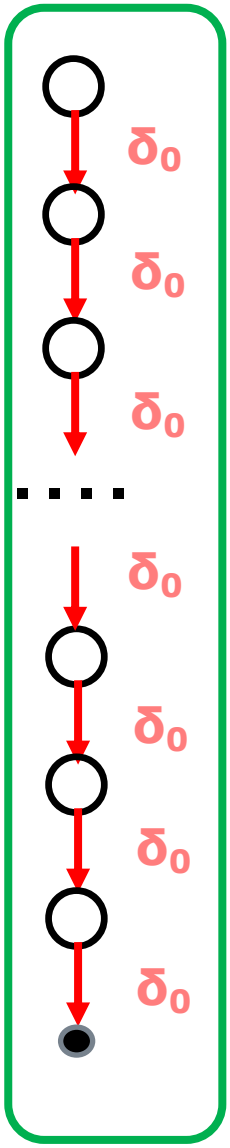
決定性チューリングマシン

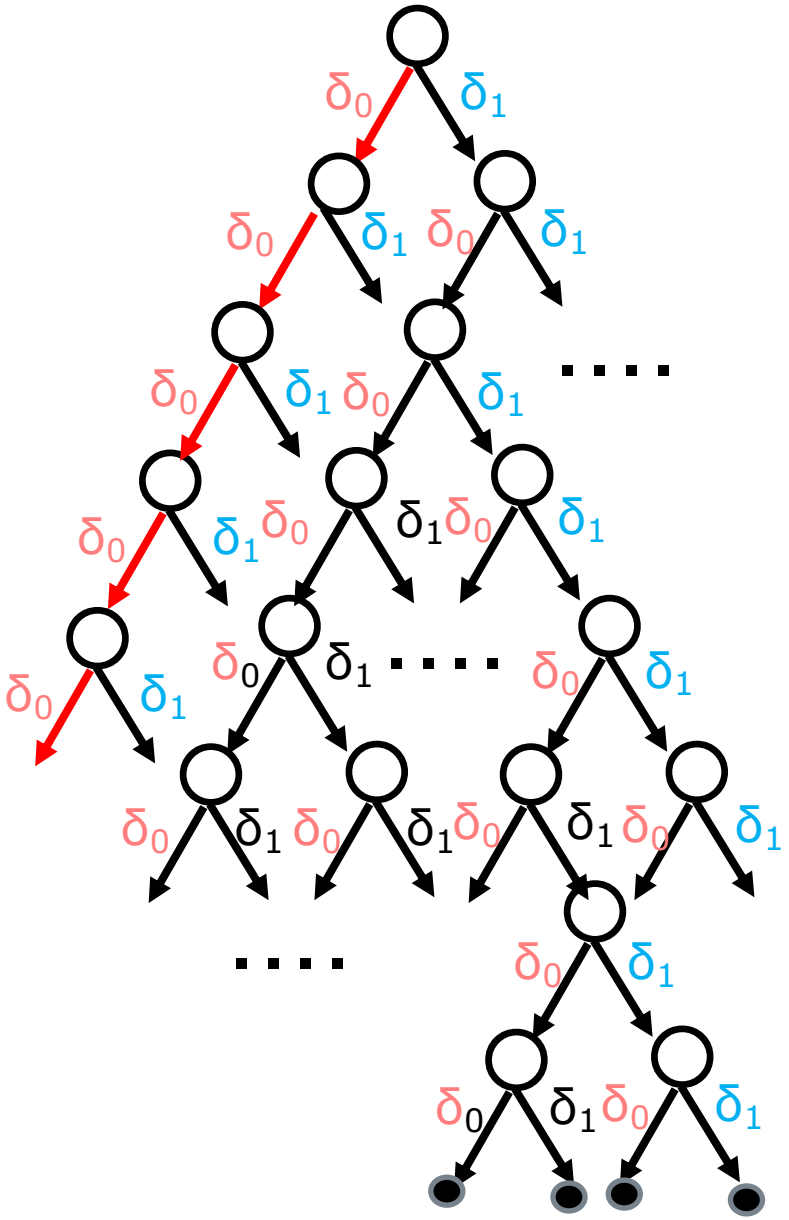
非決定性チューリングマシン



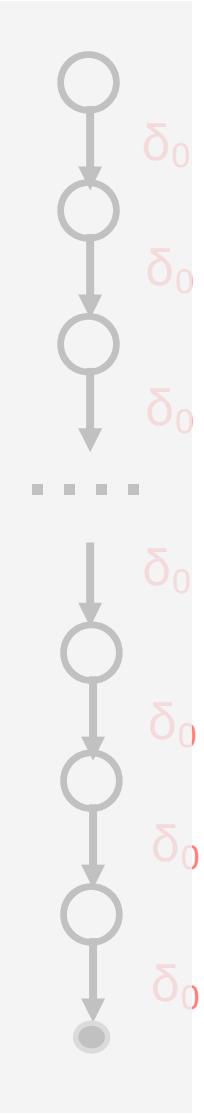
決定性チューリングマシン

非決定性チューリングマシン

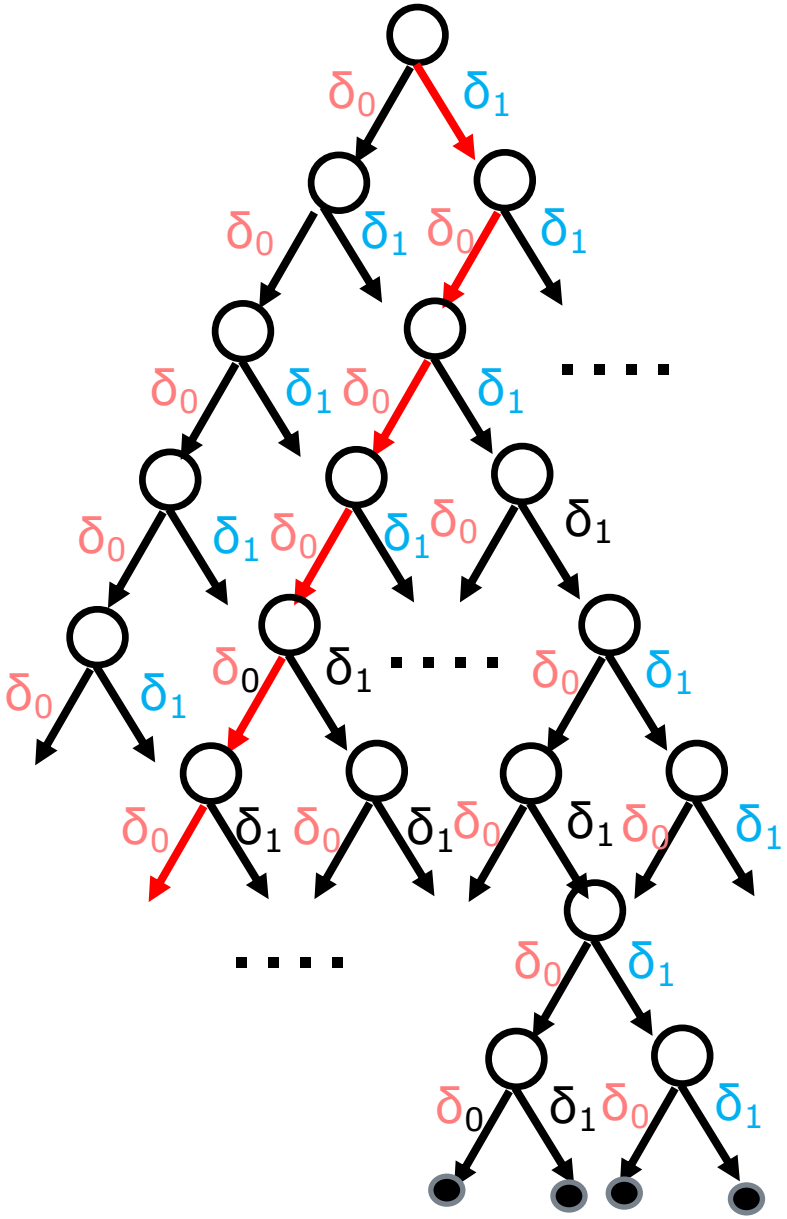




決定性チューリングマシン

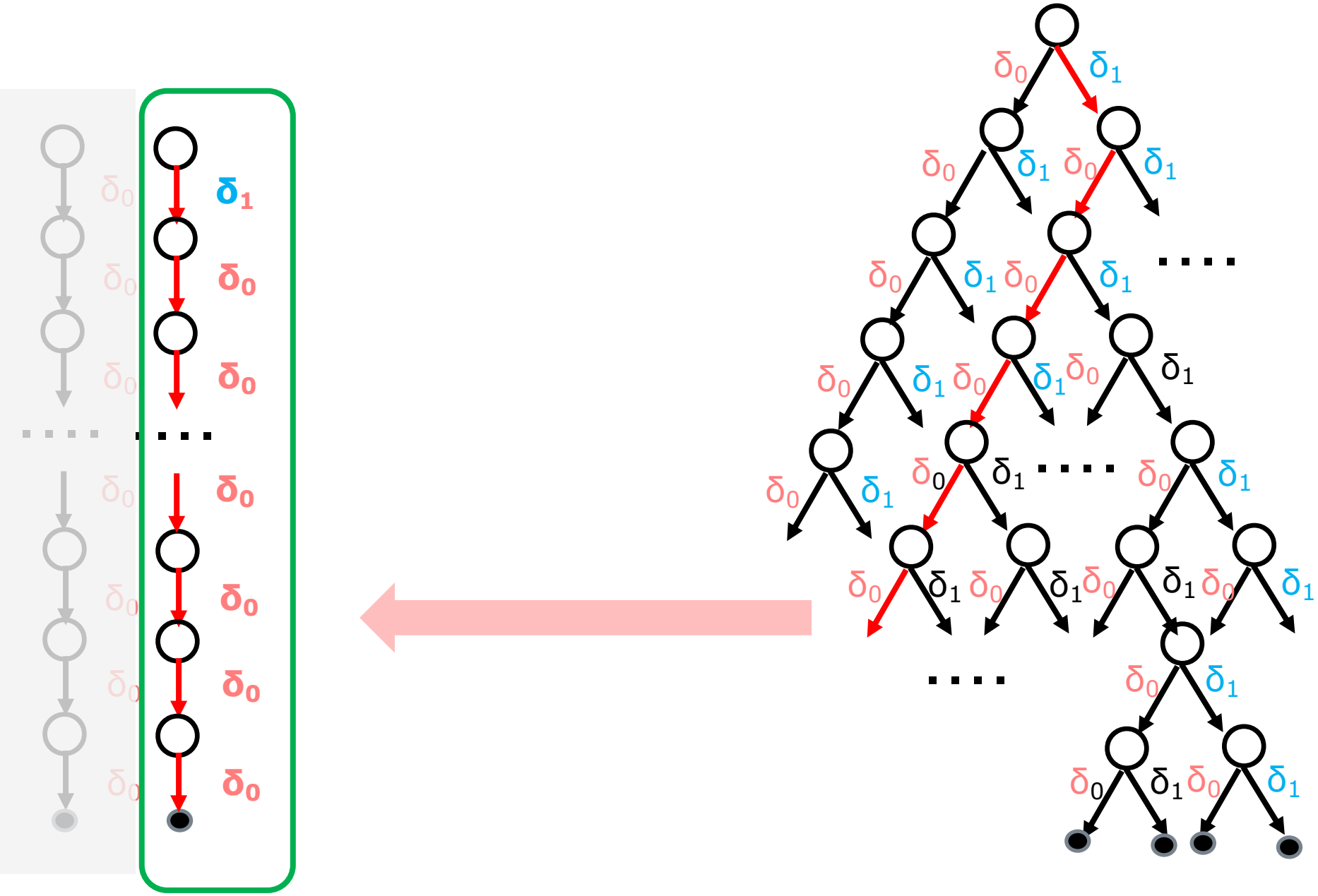


非決定性チューリングマシン

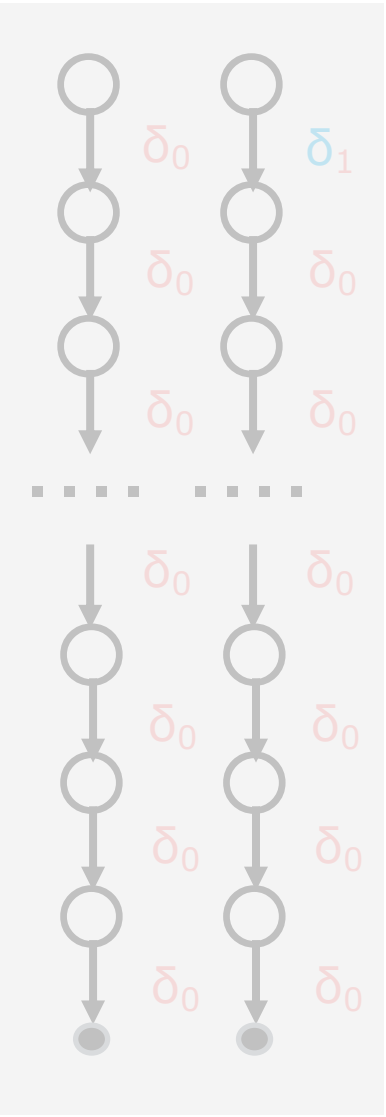


決定性チューリングマシン

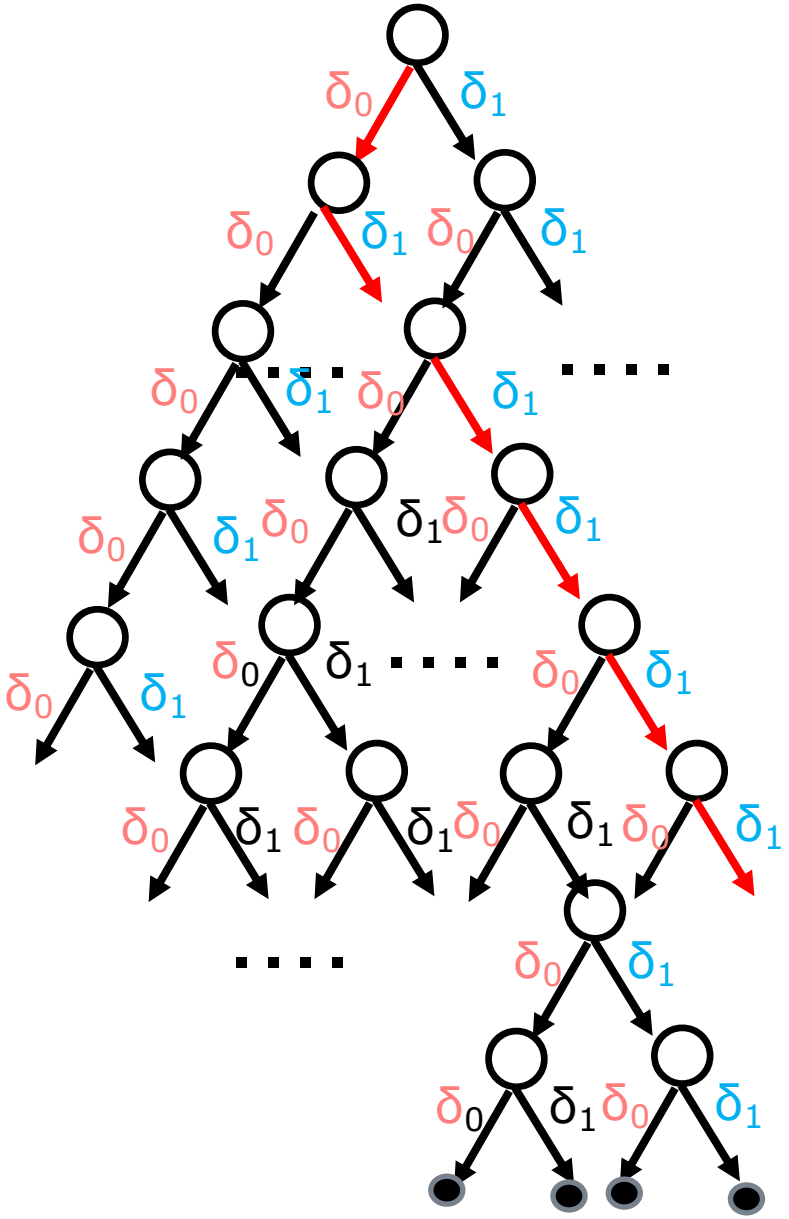
非決定性チューリングマシン



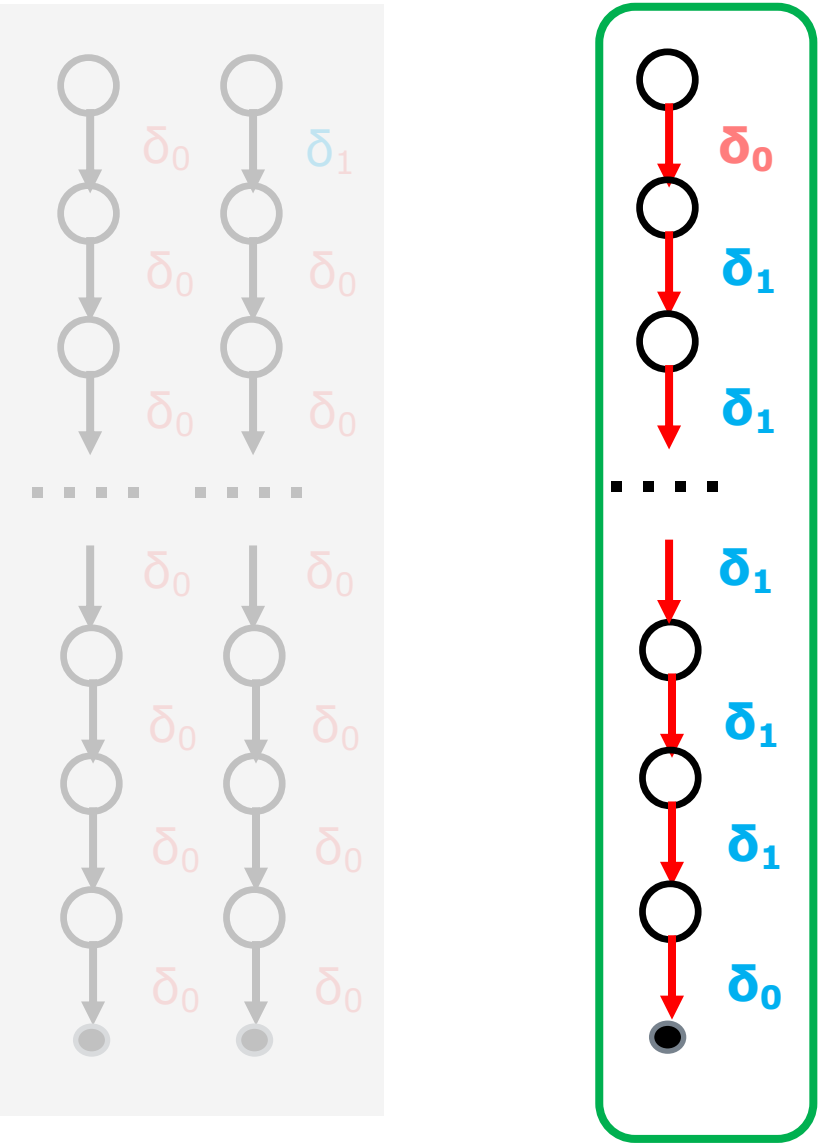
決定性チューリングマシン



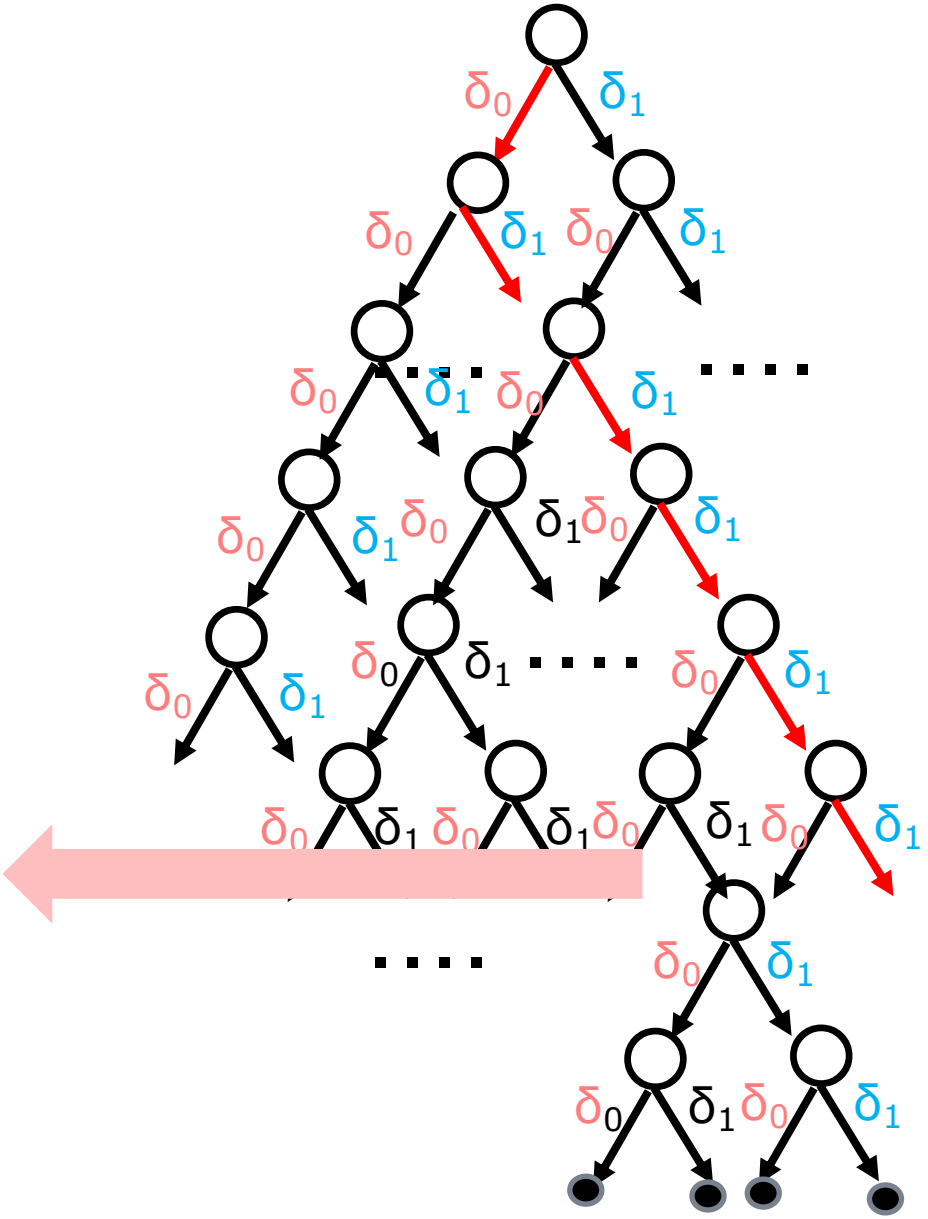
非決定性チューリングマシン



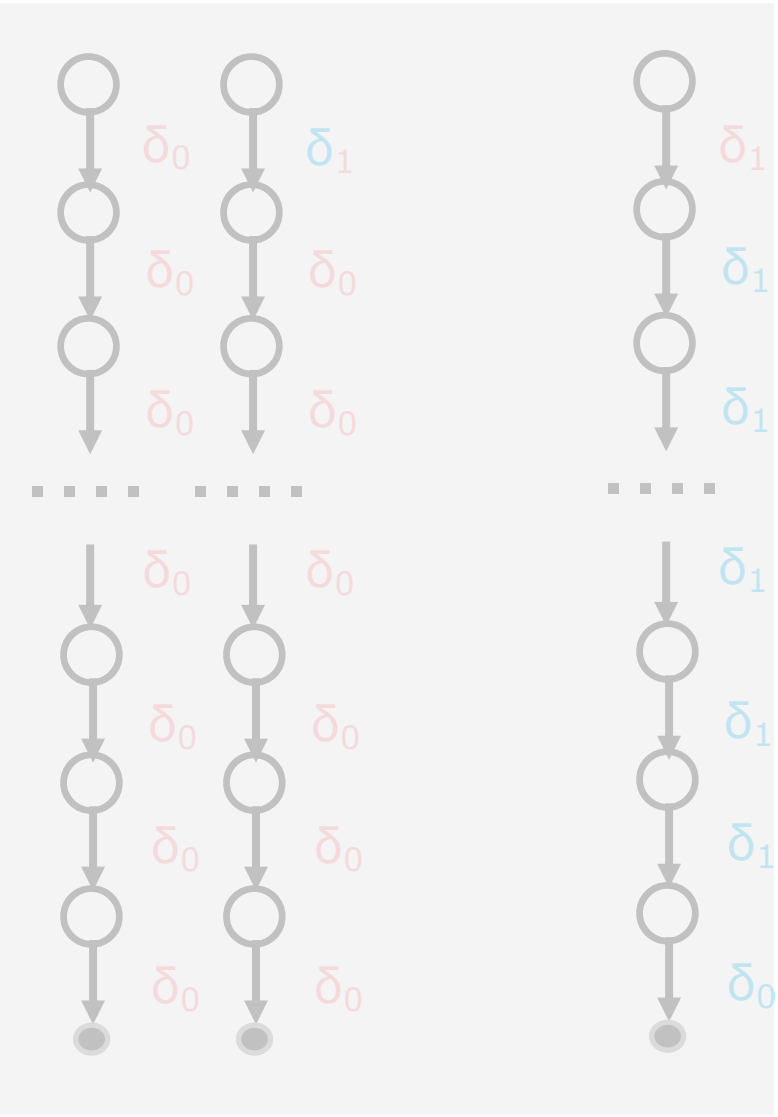
決定性チューリングマシン



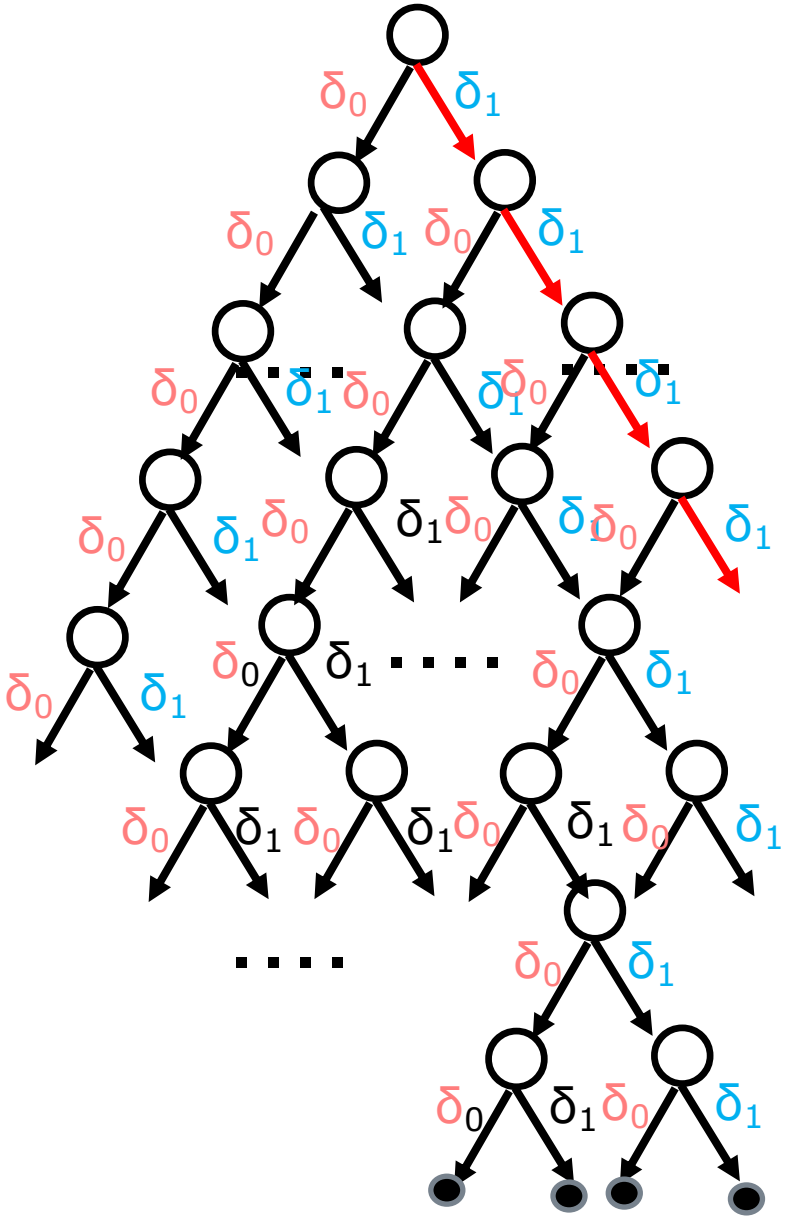
非決定性チューリングマシン



決定性チューリングマシン

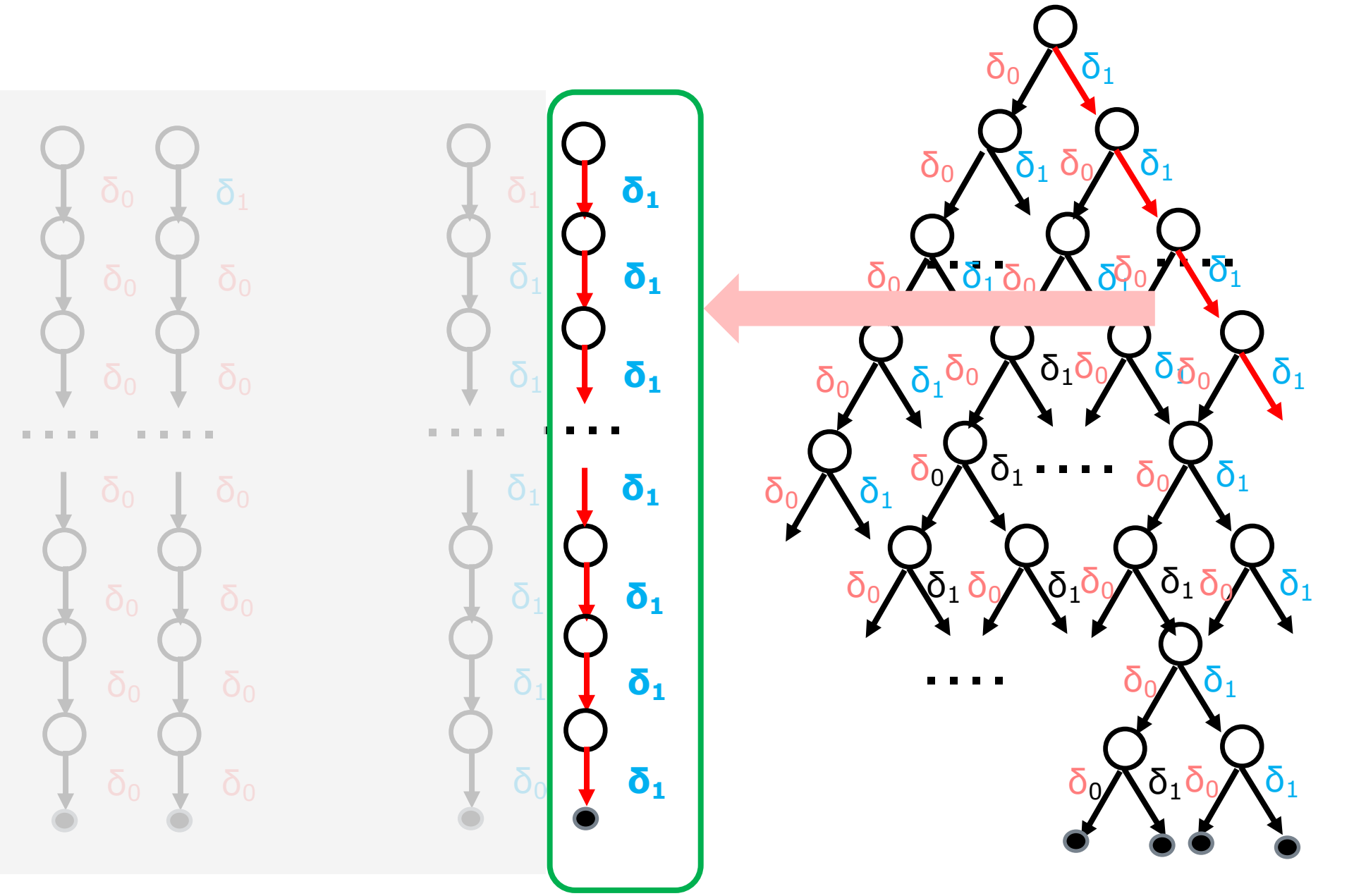


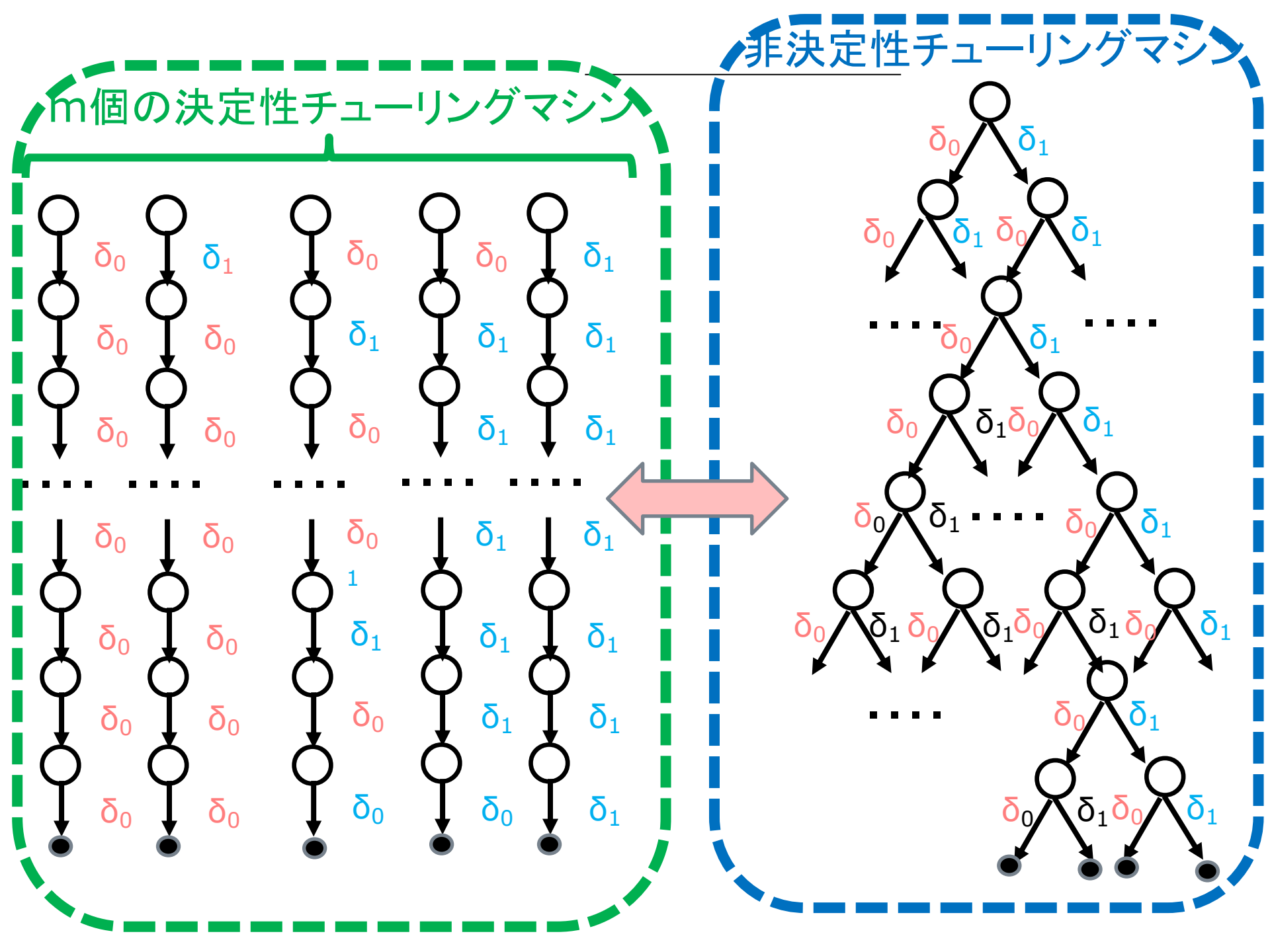
非決定性チューリングマシン



決定性チューリングマシン

非決定性チューリングマシン



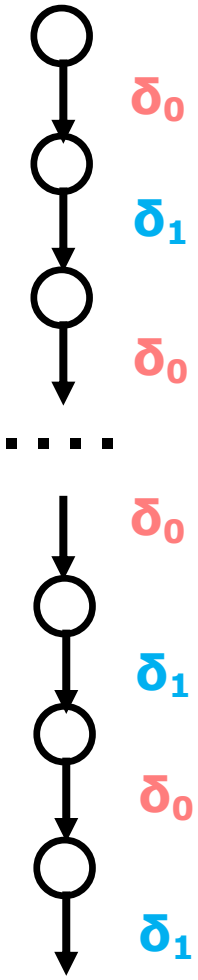

m個の決定性チューリングマシン
非決定性チューリングマシン

P ⊆ NP

決定性チューリングマシンは、
非決定性チューリングマシンの
特別な場合と考えることができる
ので、P ⊆ NPである。

三つのテープを持つ
決定性チューリングマシンでシミュレートする

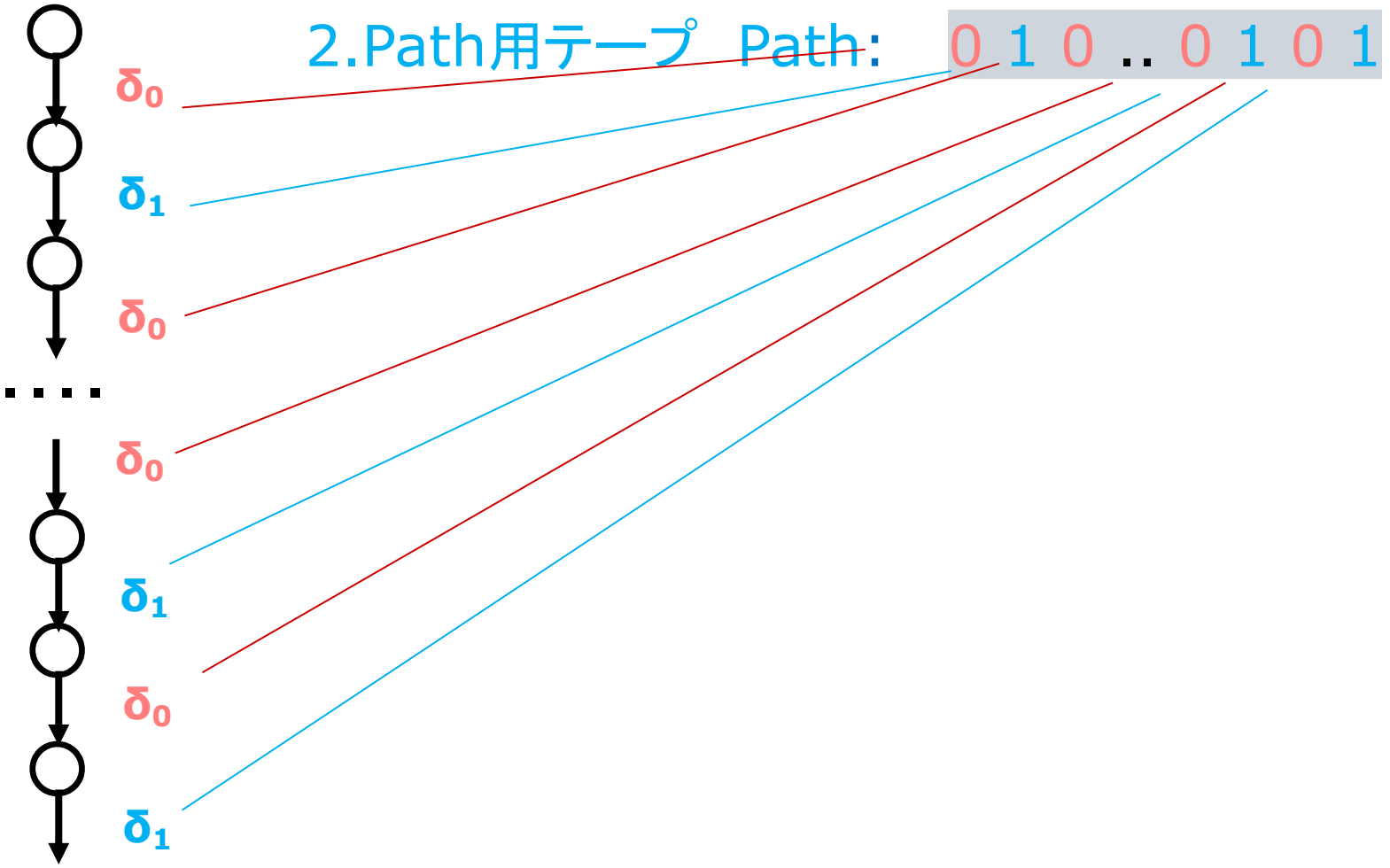
1.入力用テープ　Input:　1 0 1 1 0 ..



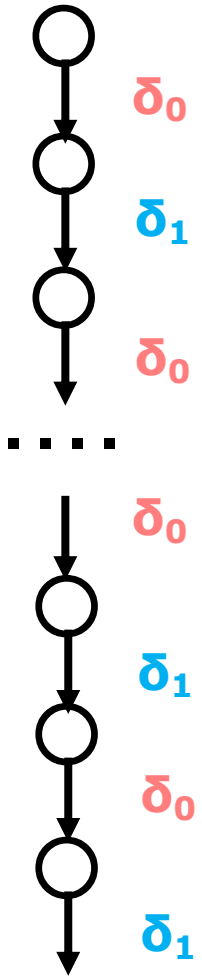
三つのテープを持つ
決定性チューリングマシンでシミュレートする

1.入力用テープ　Input:　1 0 1 1 0 ..

2.Path用テープ　Path:　0 1 0 .. 0 1 0 1



三つのテープを持つ
決定性チューリングマシンでシミュレートする



1.入力用テープ　Input:　1 0 1 1 0 ..

2.Path用テープ　Path:　0 1 0 .. 0 1 0 1

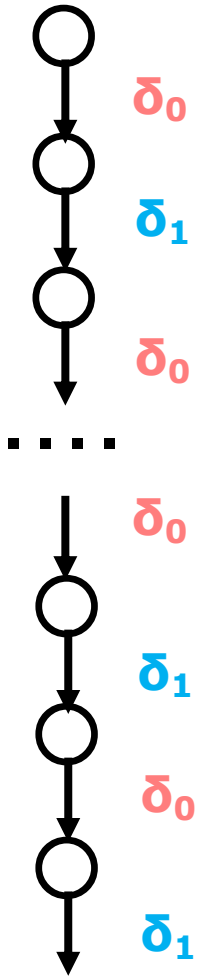
3.出力用テープ　Output:

....

<Path n>Path nの出力
<Path n’>Path n’の出力
<Path n’’>Path n’’の出力

....

三つのテープを持つ
決定性チューリングマシンでシミュレートする



1.入力用テープ　Input:　1 0 1 1 0 ..

2.Path用テープ　Path:　0 1 0 .. 0 1 0 1

3.出力用テープ　Output:

....

<Path n>Path nの出力
<Path n'>Path n'の出力
<Path n''>Path n''の出力

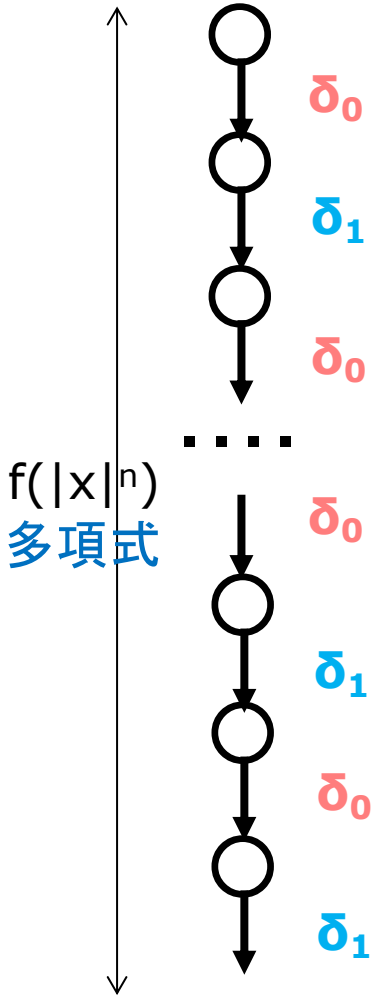
....

Pathテープが可能なパスを全て枚挙するようにプログラムし(幅優先)、
Inputが受理された時点で停止する。
全てのパスで不受理なら不受理とする。

NPクラス

入力用テープ Input:
x = 1 0 1 1 0 ..

Path用テープ Path:
p = 0 1 0 .. 0 1 0 1

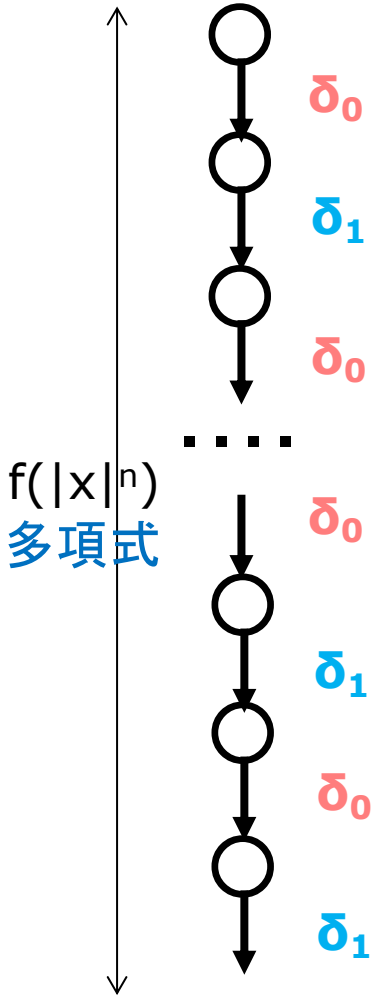


出力用テープ Output:
1:accept, 0:reject

NPクラス

入力用テープ Input: x = 1 0 1 1 0 ..

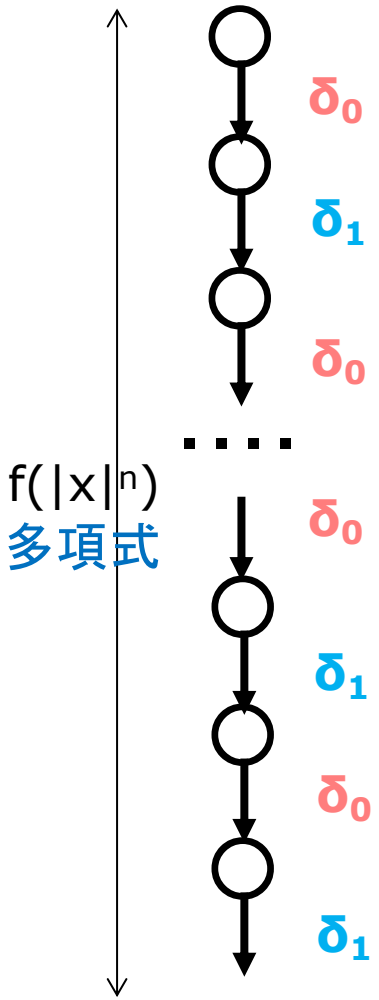
Path用テープ Path: p = 0 1 0 .. 0 1 0 1



xとpを入力とする決定性チューリングマシンMがあって、次の条件を満たすとき、LはNPクラスに属する。

出力用テープ Output:
1:accept, 0:reject

NPクラス

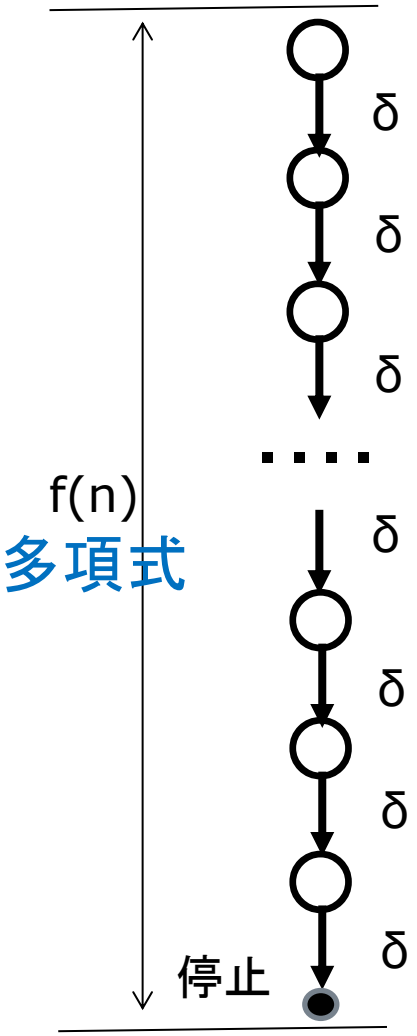


xとpを入力とする決定性チューリングマシンMがあって、次の条件を満たすとき、LはNPクラスに属する。

- 全ての入力(x, p)について、Mは多項式時間$f(|x|^n)$で走る。
- $x \in L$ なら、あるpが存在して、$M(x, p)=1$
- $x \notin L$ なら、全てのpに対して、$M(x, p)=0$

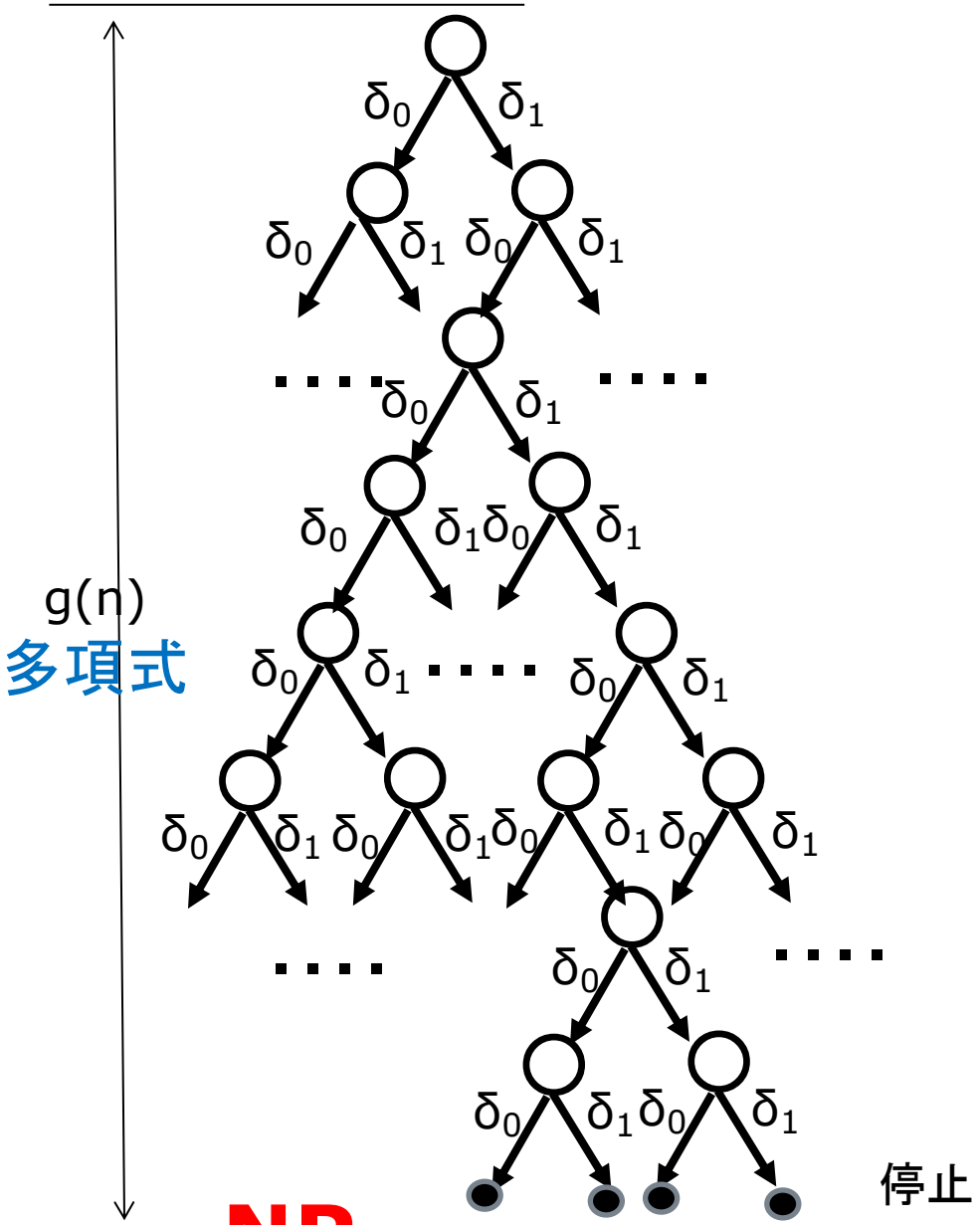
非決定性チューリングマシンの受理と拒否

決定性チューリングマシン



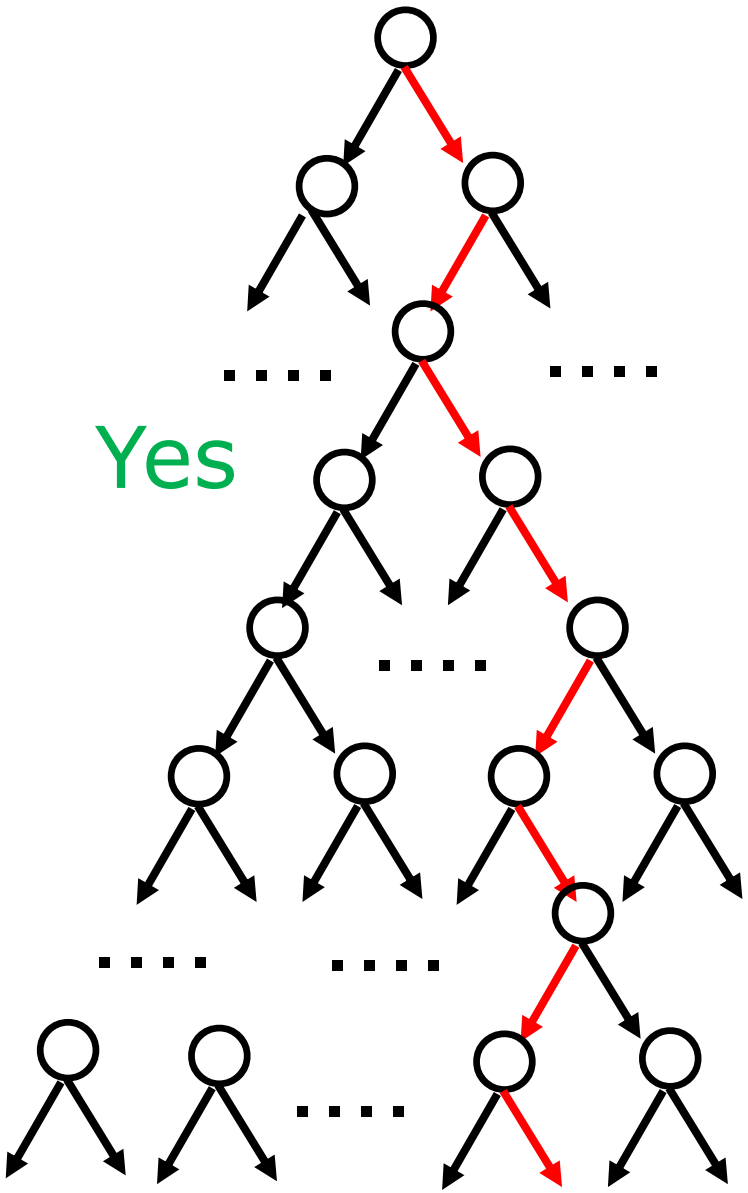
P

非決定性チューリングマシン

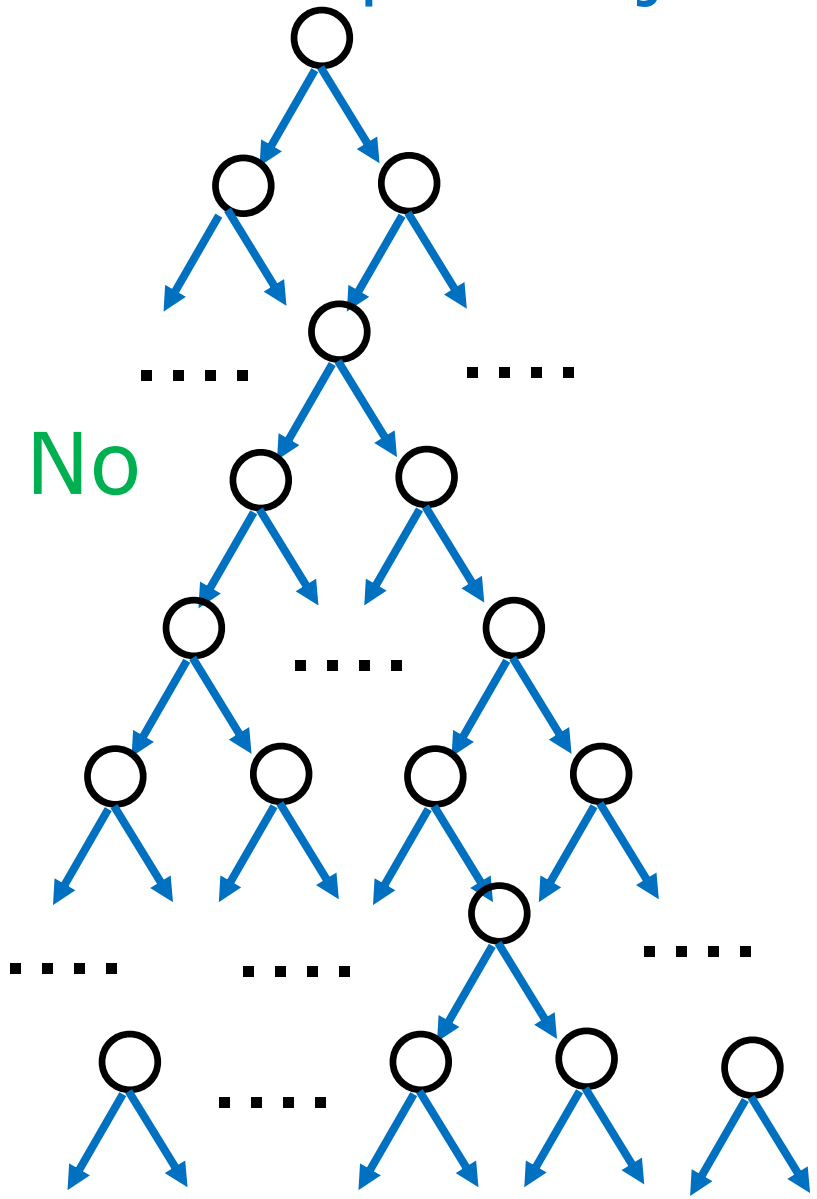


NP

非決定性チューリングマシンのAccept と Reject

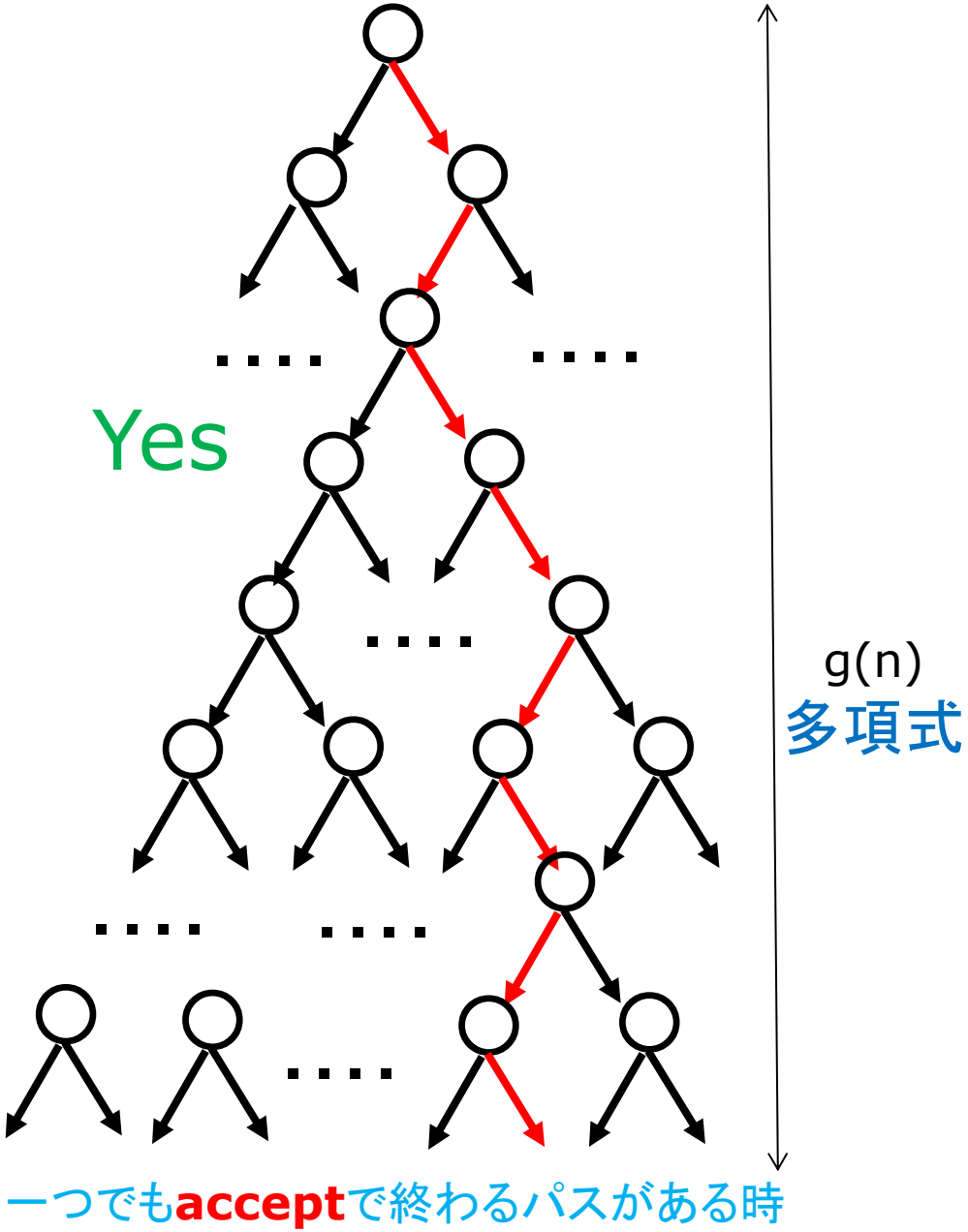


一つでも**accept**で終わるパスがある時



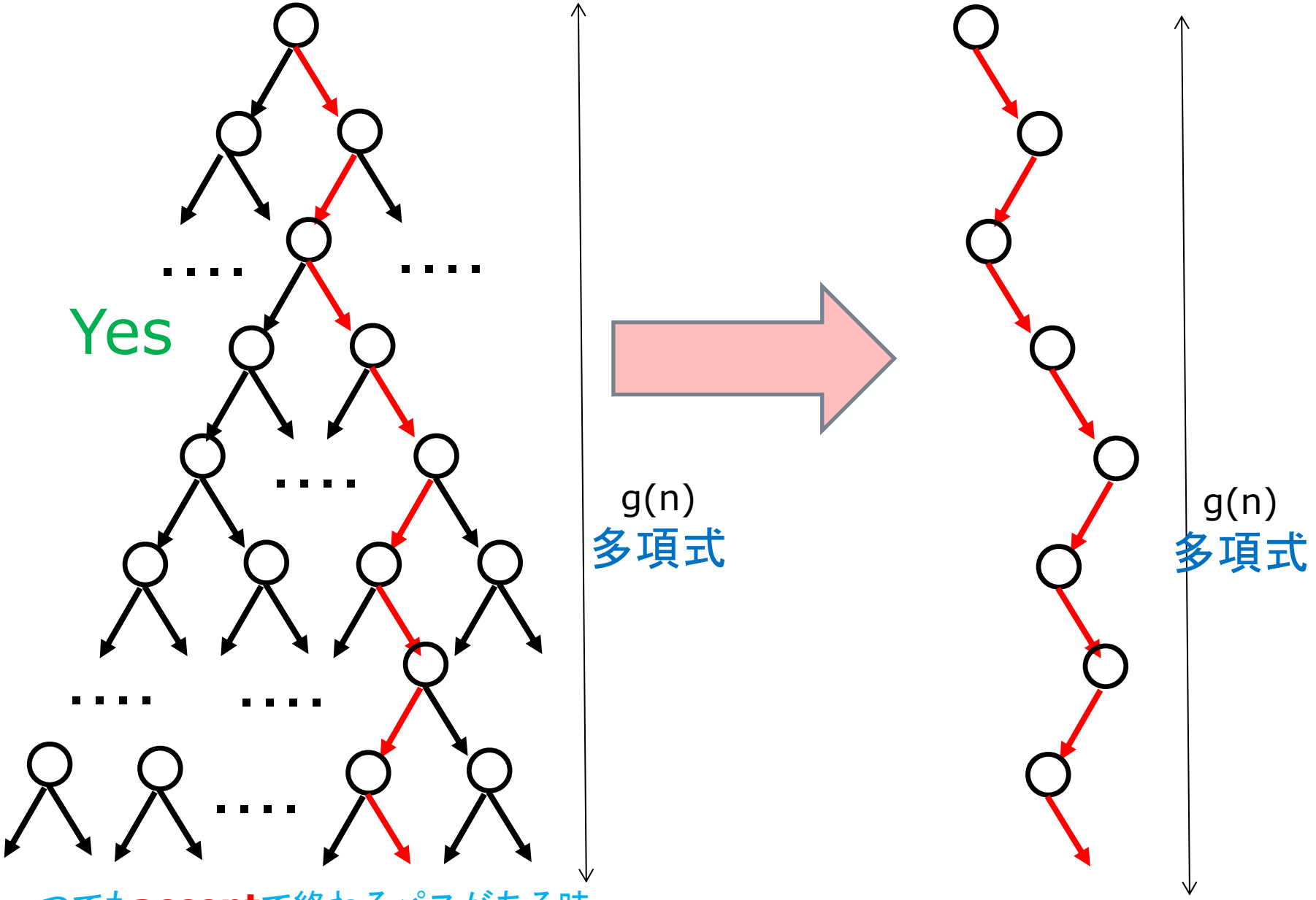
全てのパスが**reject** で終わる場合

NP：非決定性チューリングマシンで受理される場合

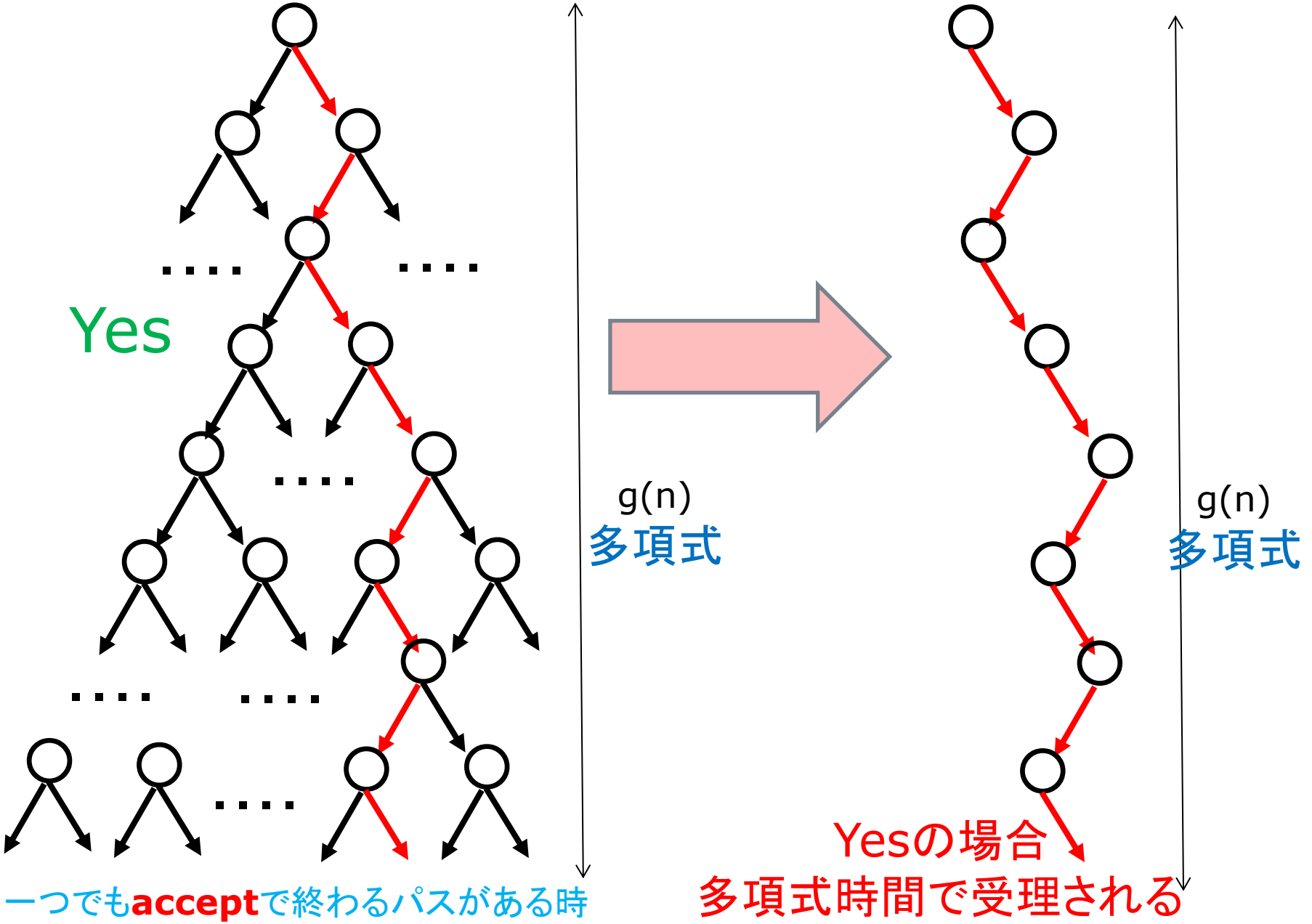


一つでも**accept**で終わるパスがある時

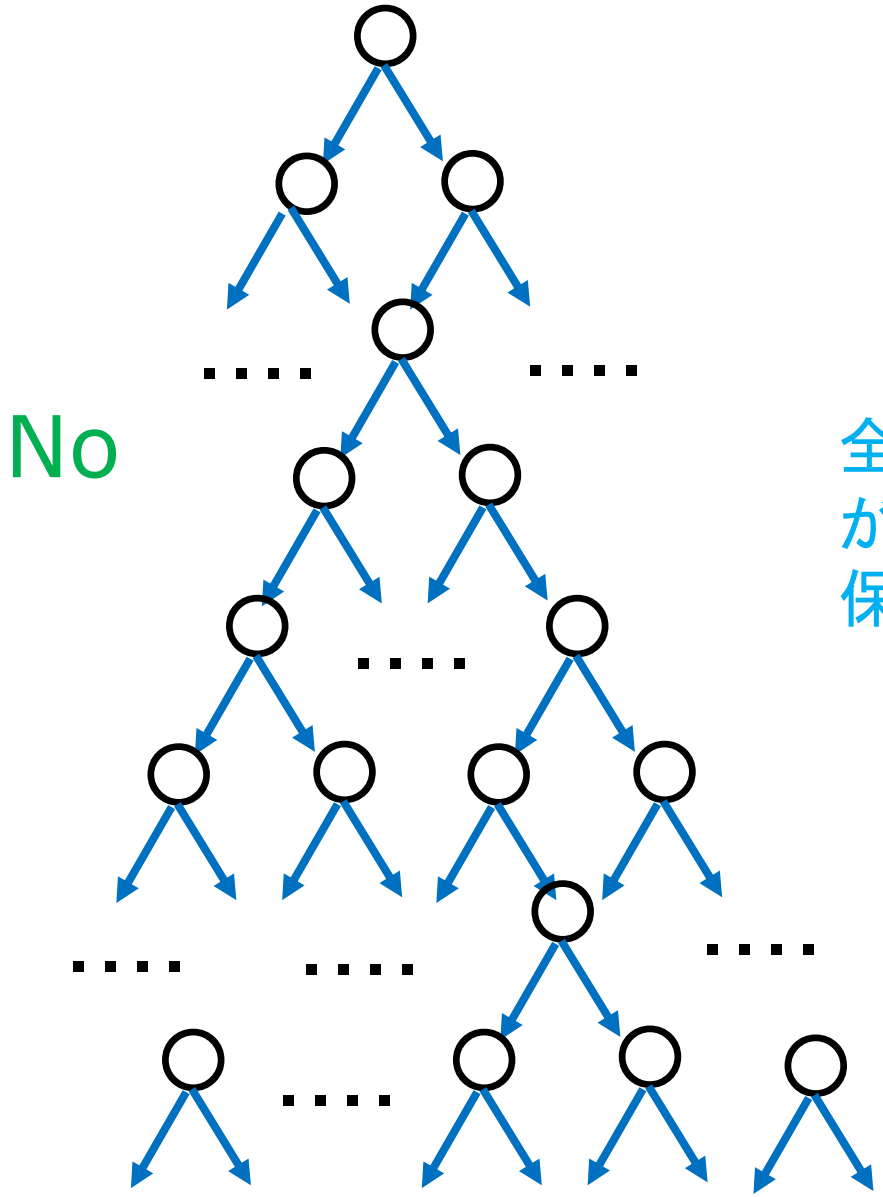
NP：非決定性チューリングマシンで受理される場合



NP：非決定性チューリングマシンで受理される場合



非決定性チューリングマシンで受理されない場合

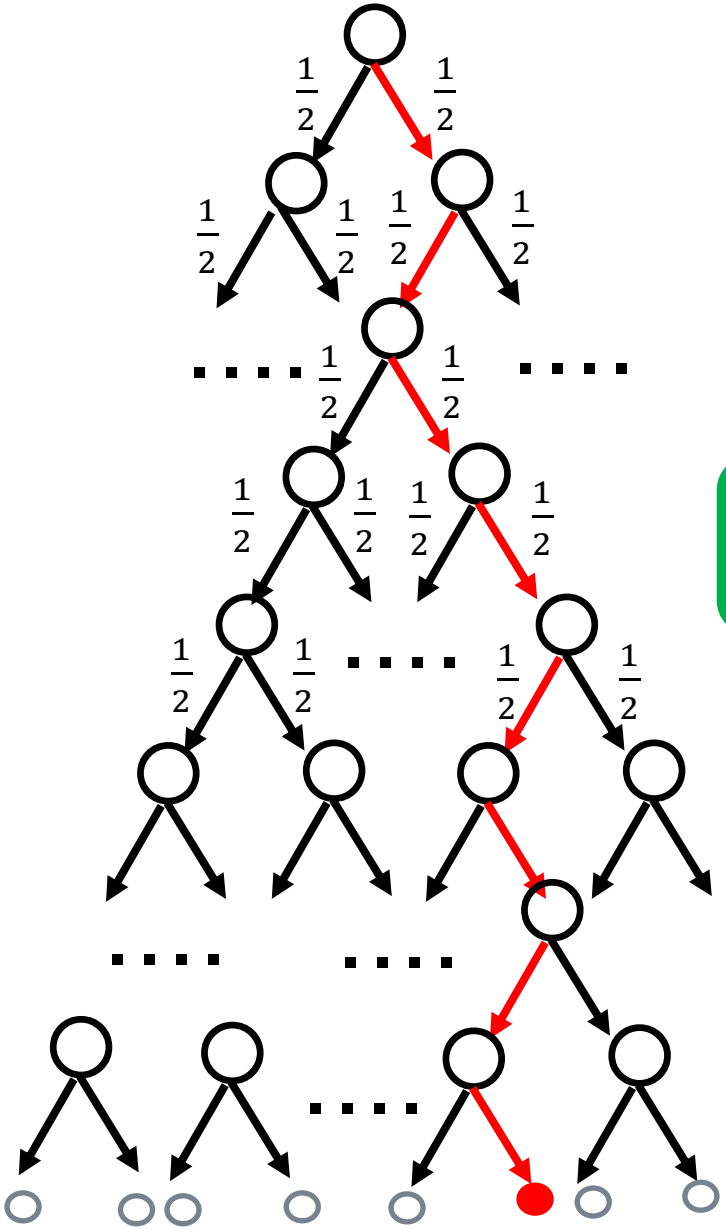


Noの場合
全てのパスのチェック
が多項式時間で終わる
保証はない

No: 全てのパスが**reject** で終わる場合

確率性チューリングマシンと BPPクラス

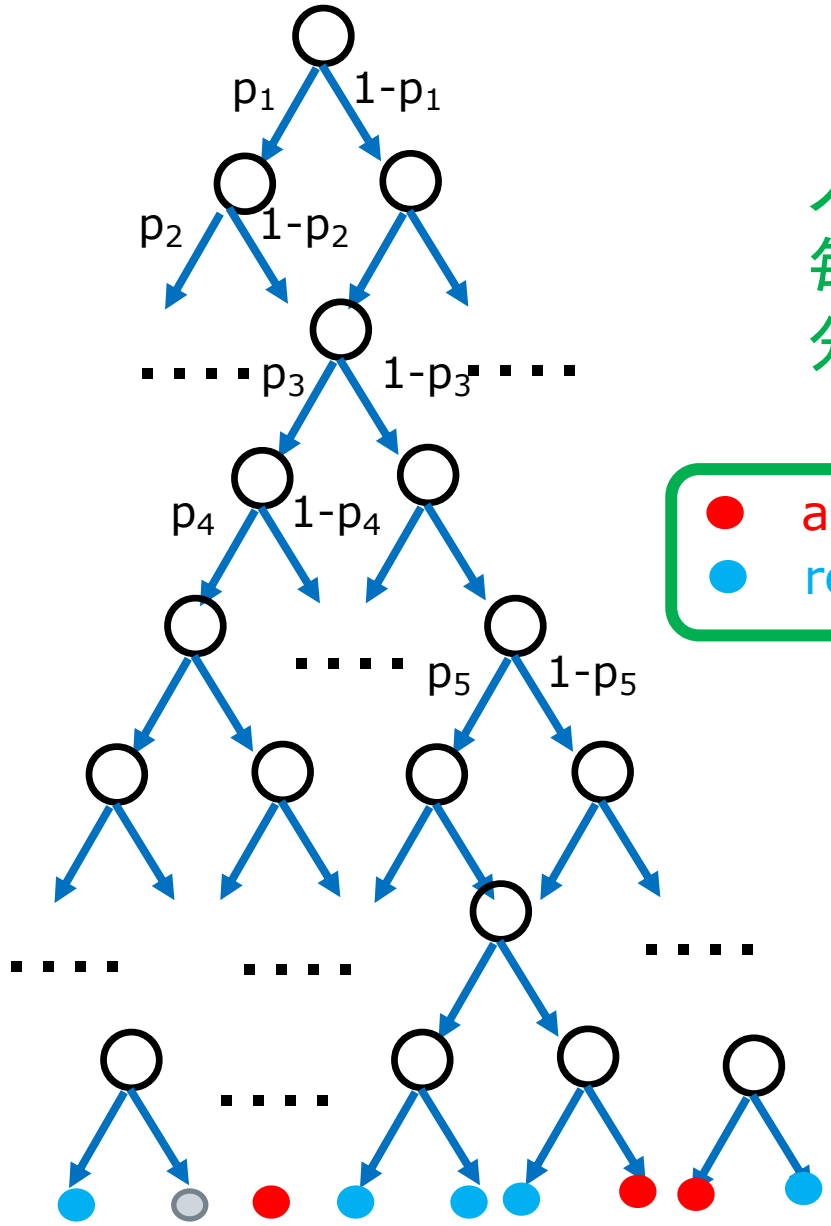
非決定性チューリングマシン



ノードでの枝分かれの確率は
等しく、どの分岐でも 1/2 である

一つでもacceptされれば、accept

確率性チューリングマシン



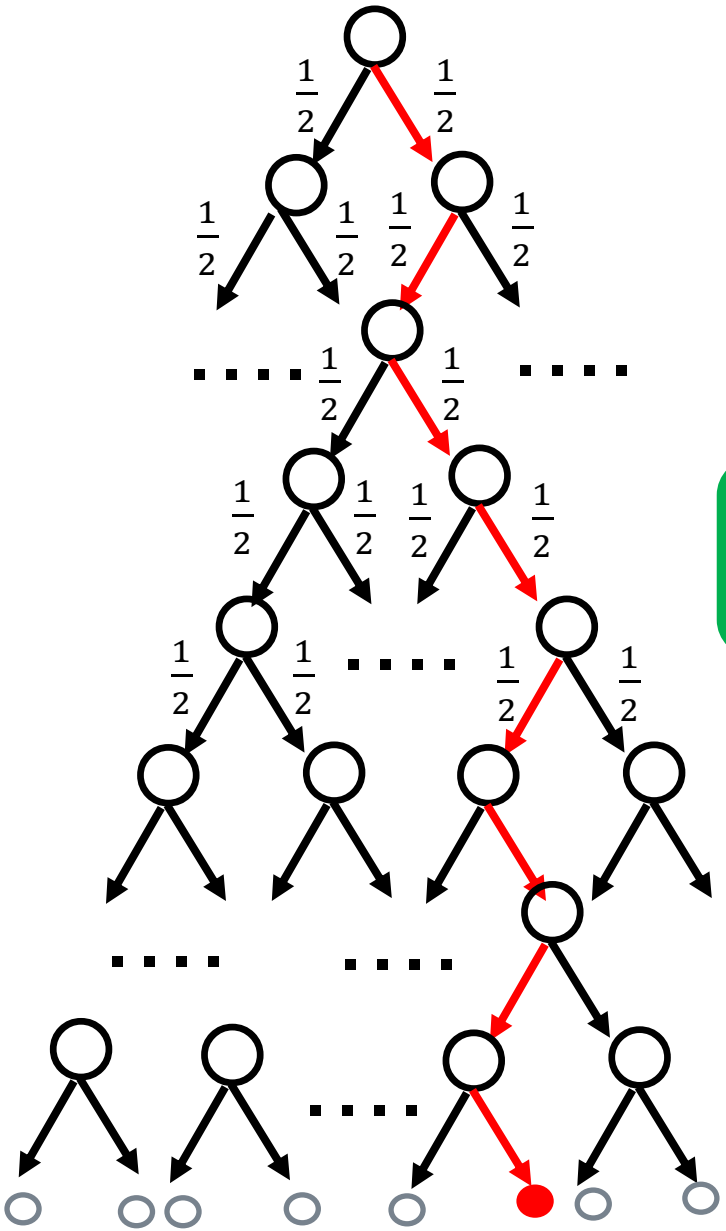
ノードでの枝分かれの確率は
毎回異なる。しかも、先行した
分岐の確率の影響を受けない

accept
reject

計算は、ある深さで終了する。
終了時点で、accept状態と
reject状態の数を数える。
ここから、acceptの確率と
rejectの確率を計算できる。

acceptとrejectの確率を考える

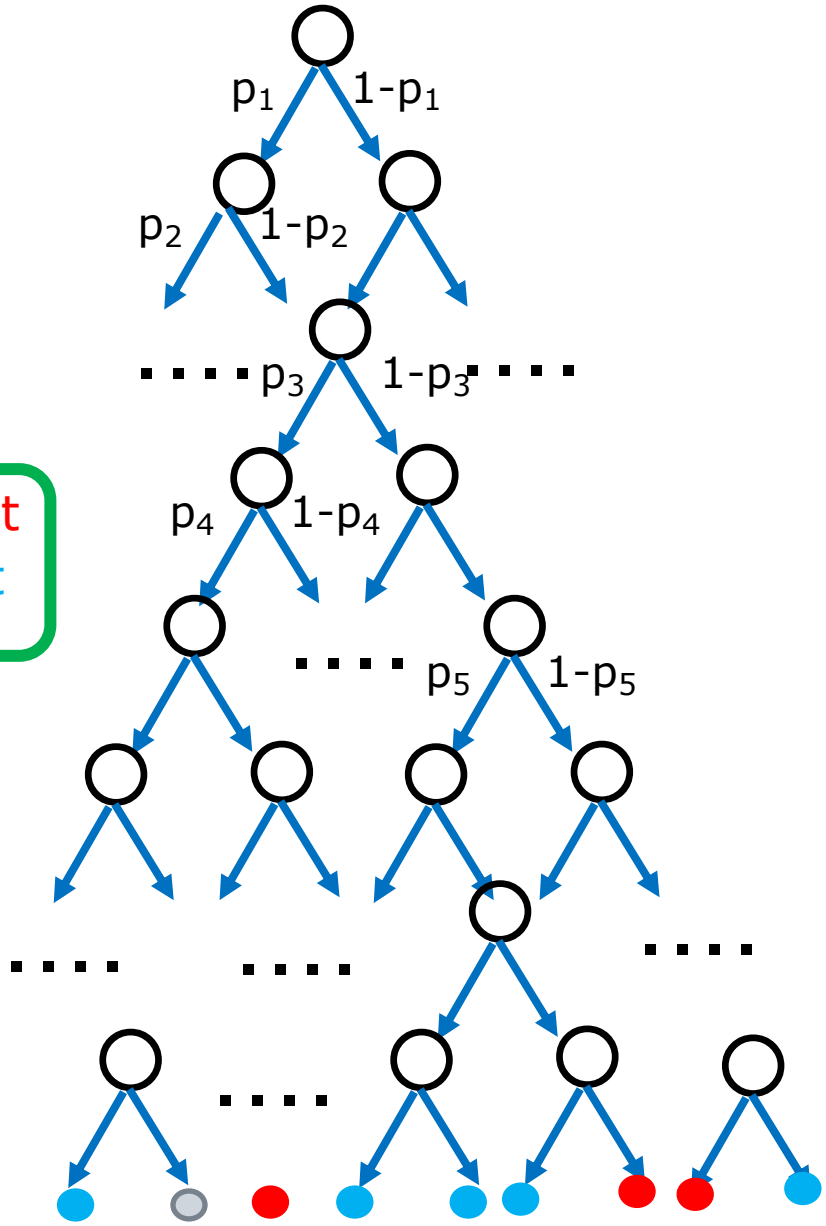
非決定性チューリングマシン



一つでもacceptされれば、accept

accept
reject

確率性チューリングマシン

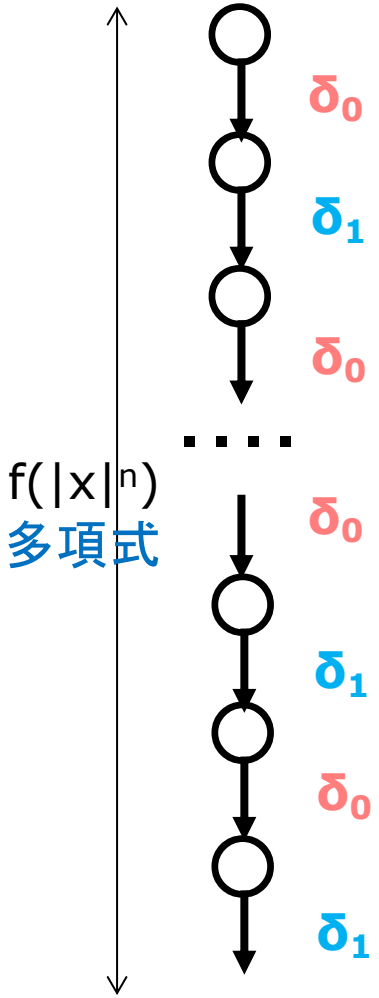


acceptとrejectの確率を考える

BPPクラス

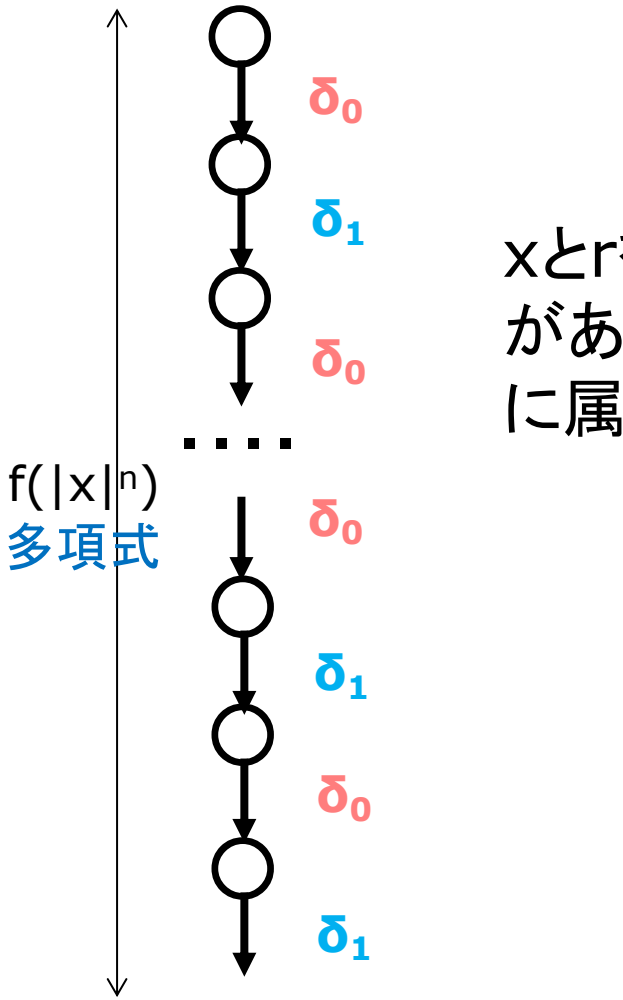
入力用テープ Input:
x = 1 0 1 1 0 ..

乱数用テープ Rand:
r = 0 1 0 .. 0 1 0 1



出力用テープ Output:
1:accept, 0:reject

BPPクラス

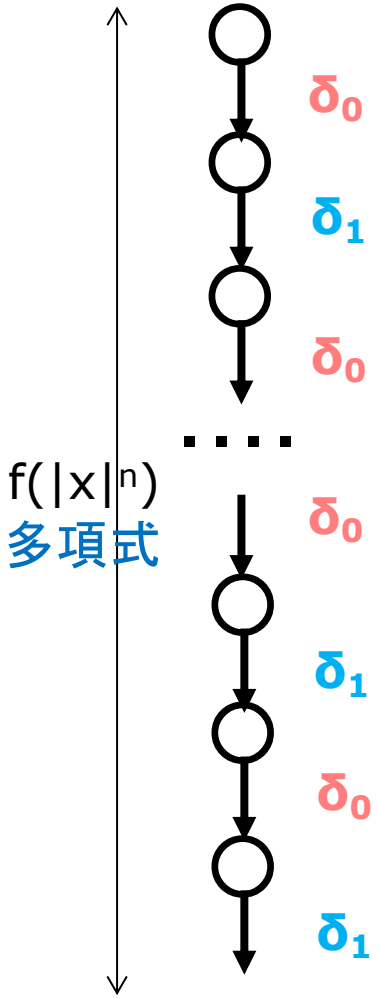


xとrを入力とする確率性チューリングマシンMがあって、次の条件を満たすとき、LはBPPクラスに属する。

BPPクラス

入力用テープ Input: x = 1 0 1 1 0 ..

乱数用テープ Rand: r = 0 1 0 .. 0 1 0 1



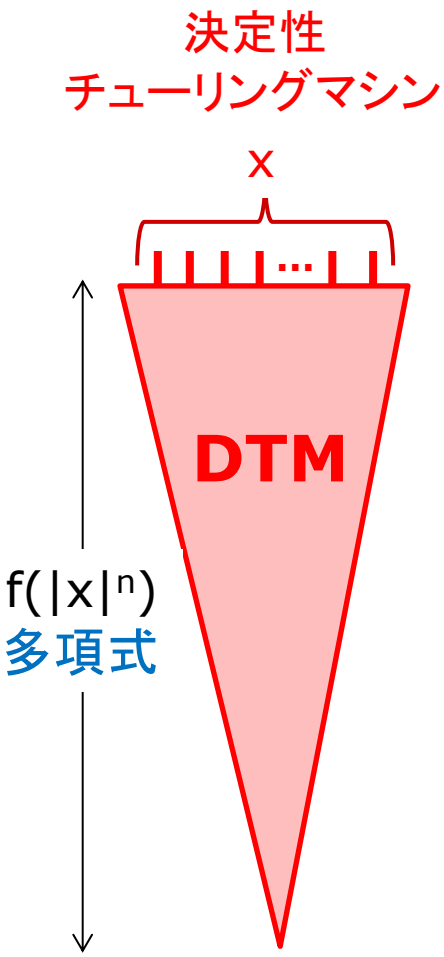
出力用テープ Output:
1:accept, 0:reject

xとrを入力とする確率性チューリングマシンMがあって、次の条件を満たすとき、LはBPPクラスに属する。

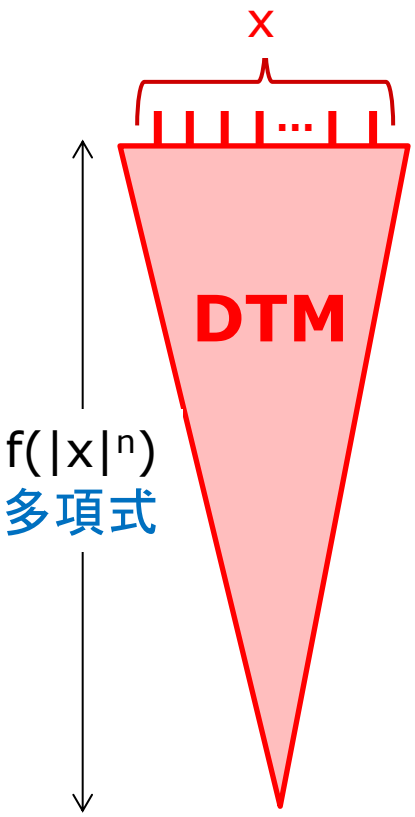
- 全ての入力(x, r)について、Mは多項式時間$f(|x|^n)$で走る。
- $x \in L$ なら、$M(x, r)=1$である確率は2/3以上である。
- $x \notin L$ なら、$M(x, r)=0$である確率は1/3以下である。

チューリングマシンの拡大と複雑性

チューリングマシンと
それが受理する複雑性のクラス

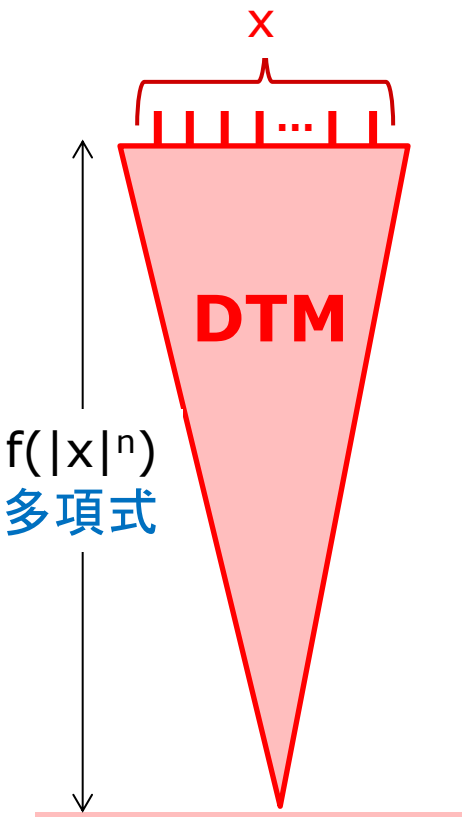

決定性
チューリングマシン
x
DTM
f(|x|ⁿ)
多項式

決定性
チューリングマシン



$x \in L$
$\rightarrow \mathrm{DTM}(x)=1$
$x \notin L$
$\rightarrow \mathrm{DTM}(x)=0$

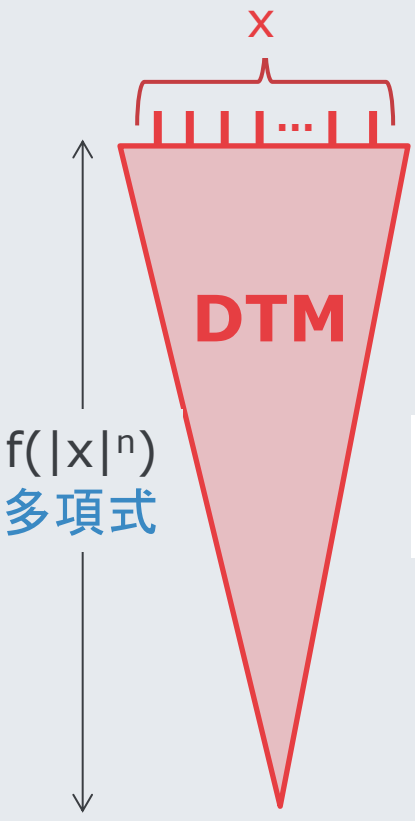
決定性
チューリングマシン



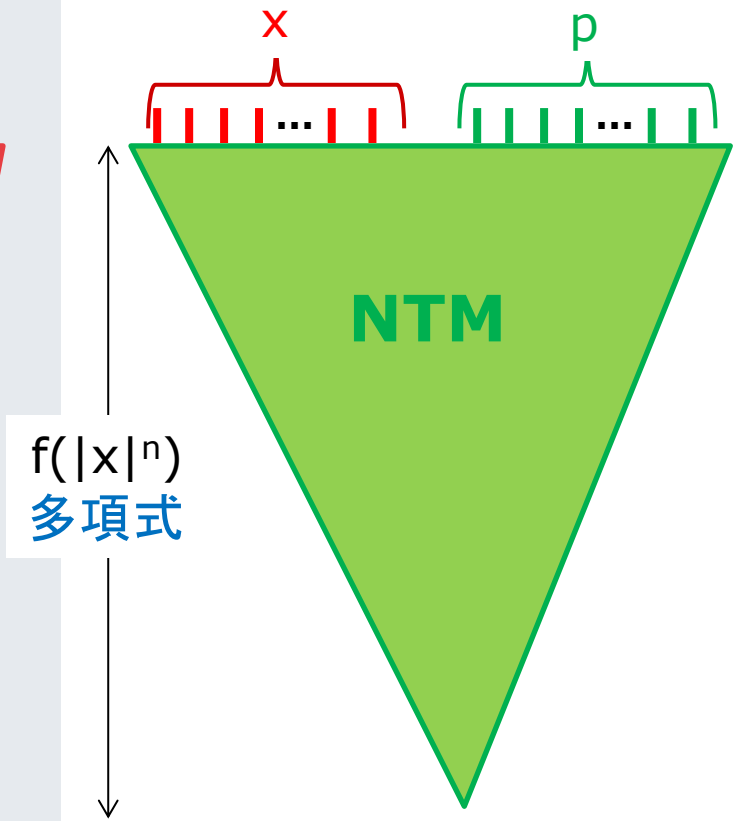
$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

$L \in P$

決定性
チューリングマシン



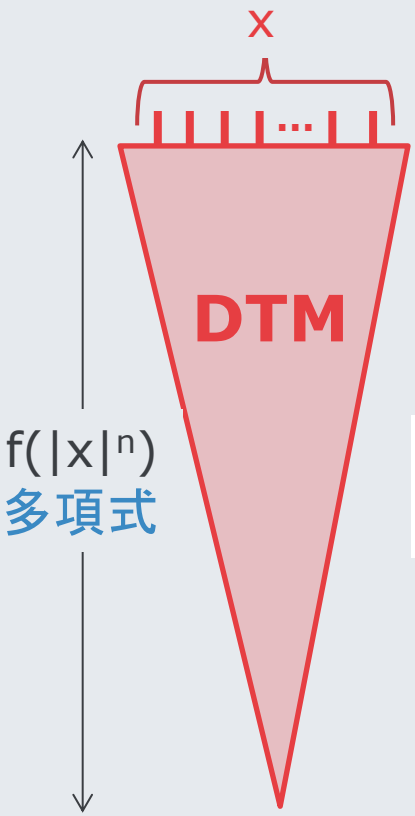
非決定性
チューリングマシン



$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

$L \in P$

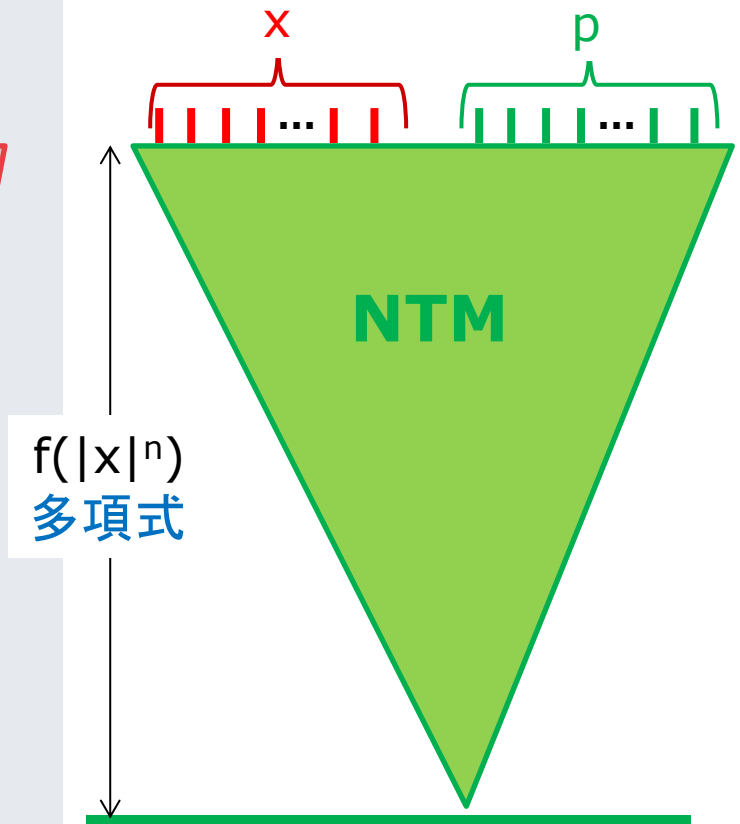
決定性
チューリングマシン



$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

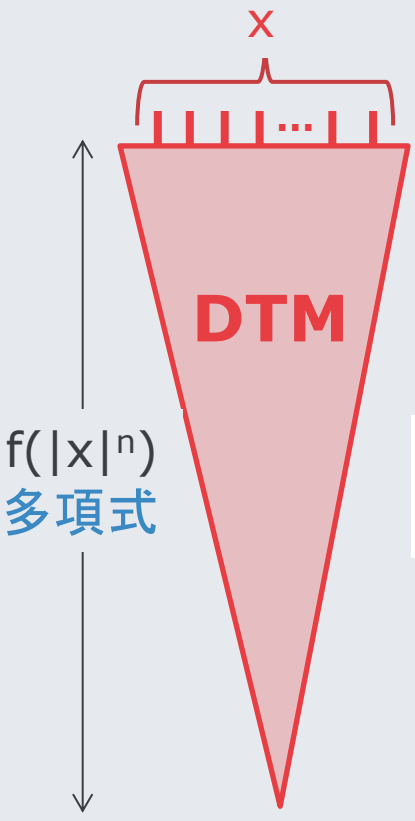
$L \in P$

非決定性
チューリングマシン



$x \in L$
$\rightarrow \exists p\{NTM(x, p)=1\}$
$x \notin L$
$\rightarrow \forall p\{NTM(x, p)=0\}$

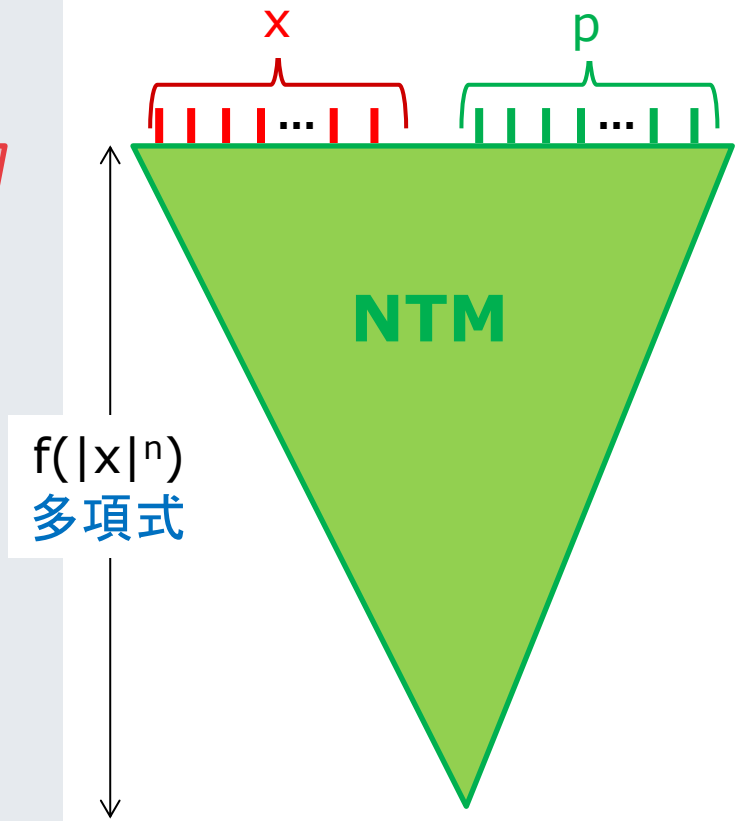
決定性
チューリングマシン



$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

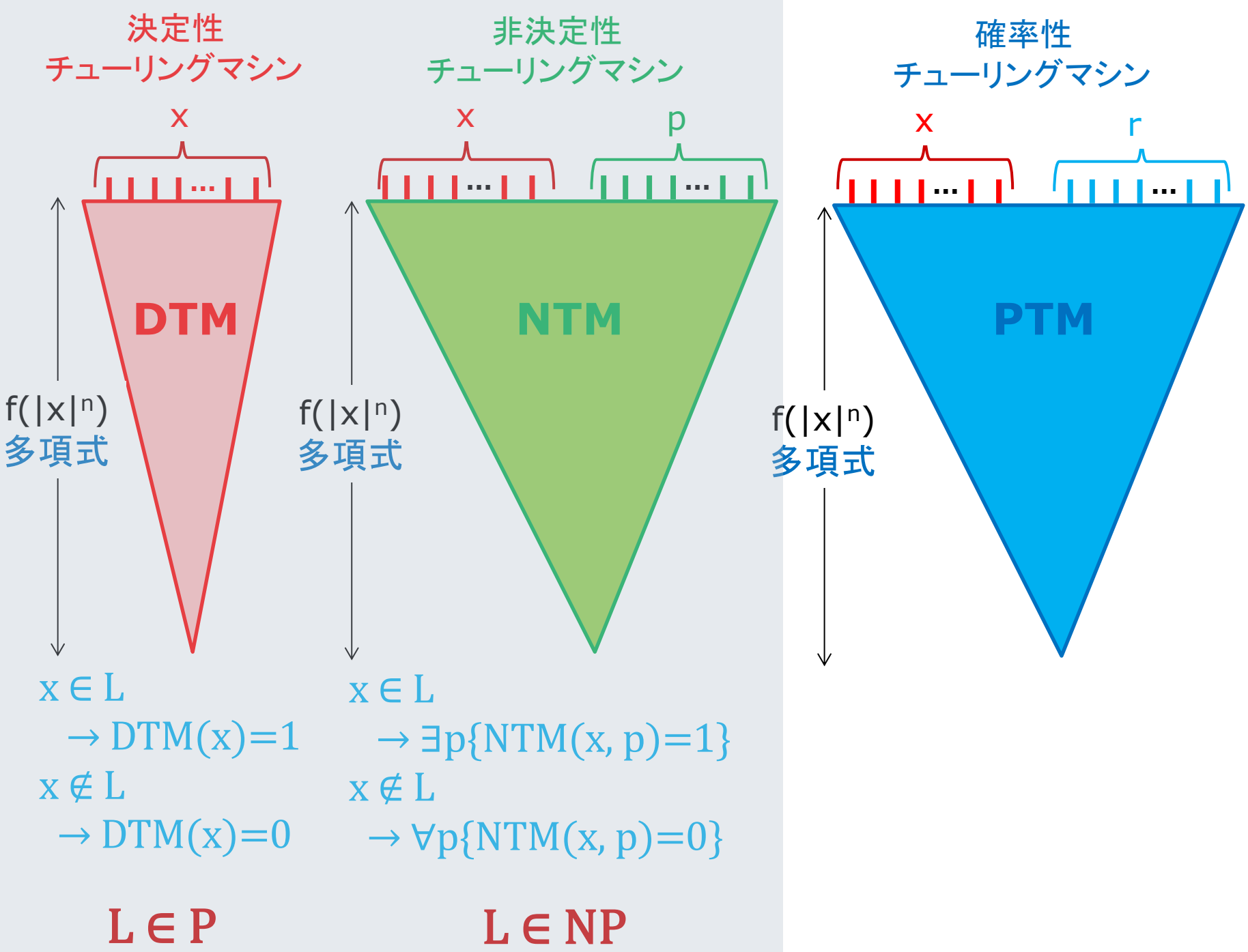
$L \in P$

非決定性
チューリングマシン

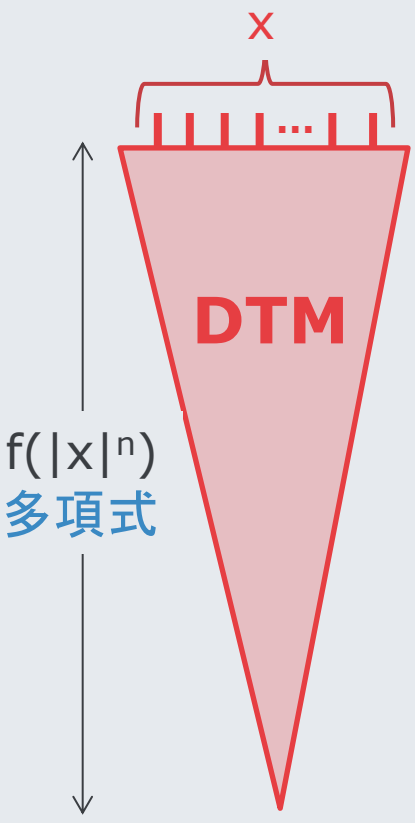


$x \in L$
$\rightarrow \exists p\{NTM(x, p)=1\}$
$x \notin L$
$\rightarrow \forall p\{NTM(x, p)=0\}$

$L \in NP$


決定性
チューリングマシン
x
DTM
f(|x|ⁿ)
多項式
x ∈ L
→ DTM(x)=1
x ∉ L
→ DTM(x)=0
L ∈ P
非決定性
チューリングマシン
x
p
NTM
f(|x|ⁿ)
多項式
x ∈ L
→ ∃p{NTM(x, p)=1}
x ∉ L
→ ∀p{NTM(x, p)=0}
L ∈ NP
確率性
チューリングマシン
x
r
PTM
f(|x|ⁿ)
多項式

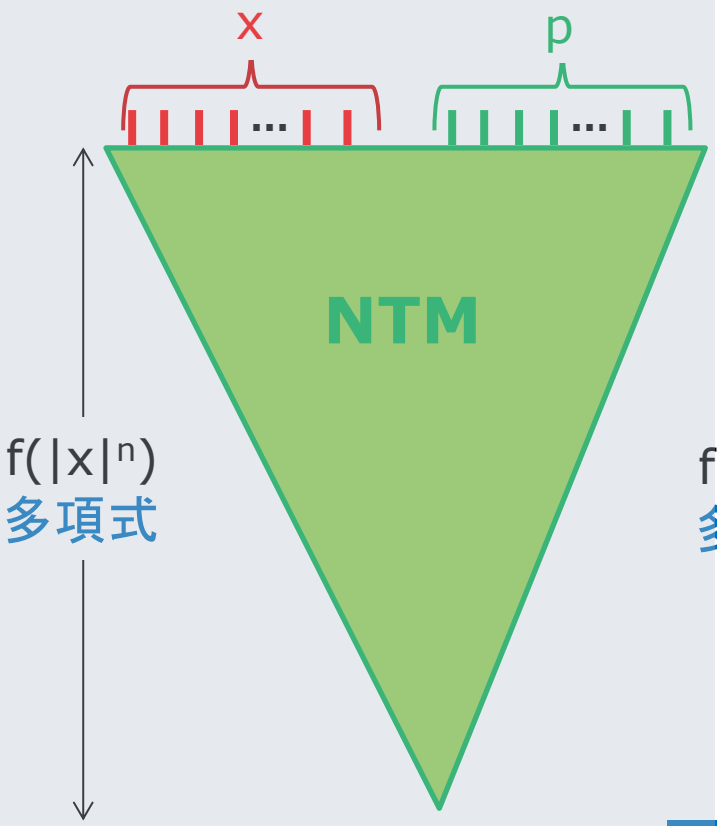
決定性
チューリングマシン



$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

$L \in P$

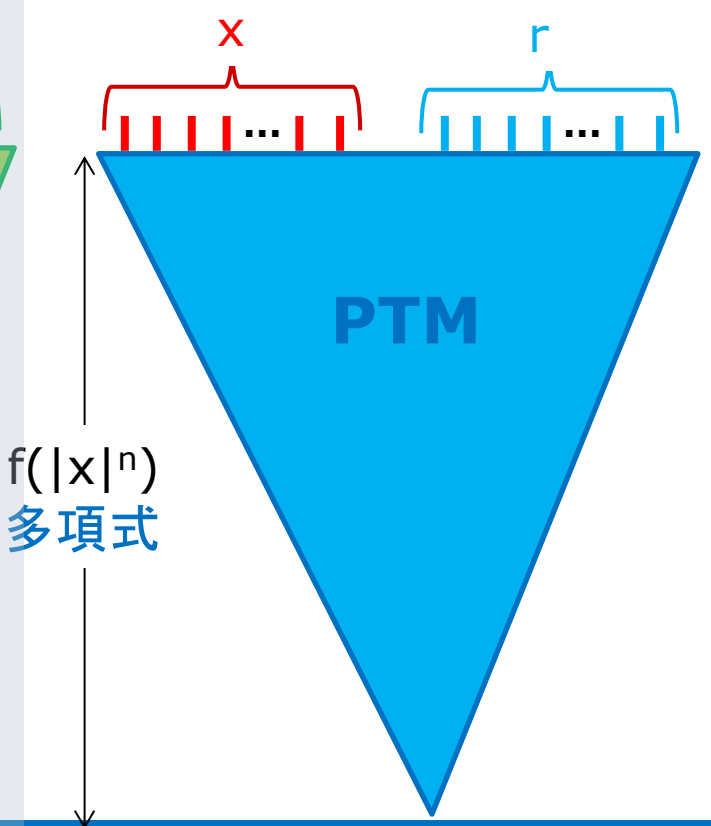
非決定性
チューリングマシン



$x \in L$
$\rightarrow \exists p\{NTM(x, p)=1\}$
$x \notin L$
$\rightarrow \forall p\{NTM(x, p)=0\}$

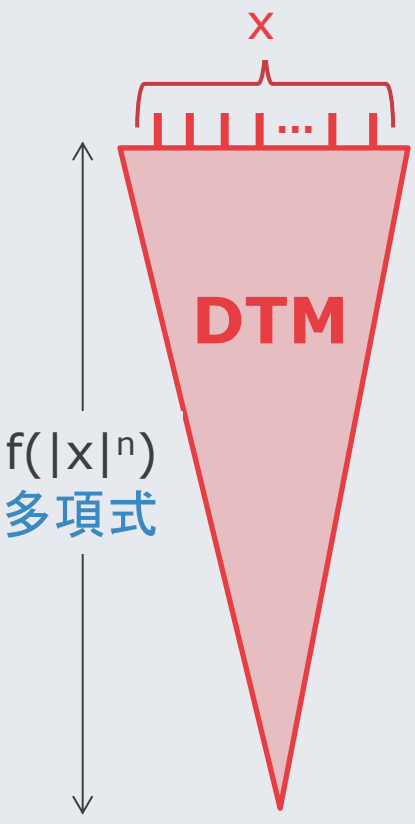
$L \in NP$

確率性
チューリングマシン



$x \in L$
$\rightarrow Prob(PTM(x, r)=1) \geq 2/3$
$x \notin L$
$\rightarrow Prob(PTM(x, r)=0) \leq 1/3$

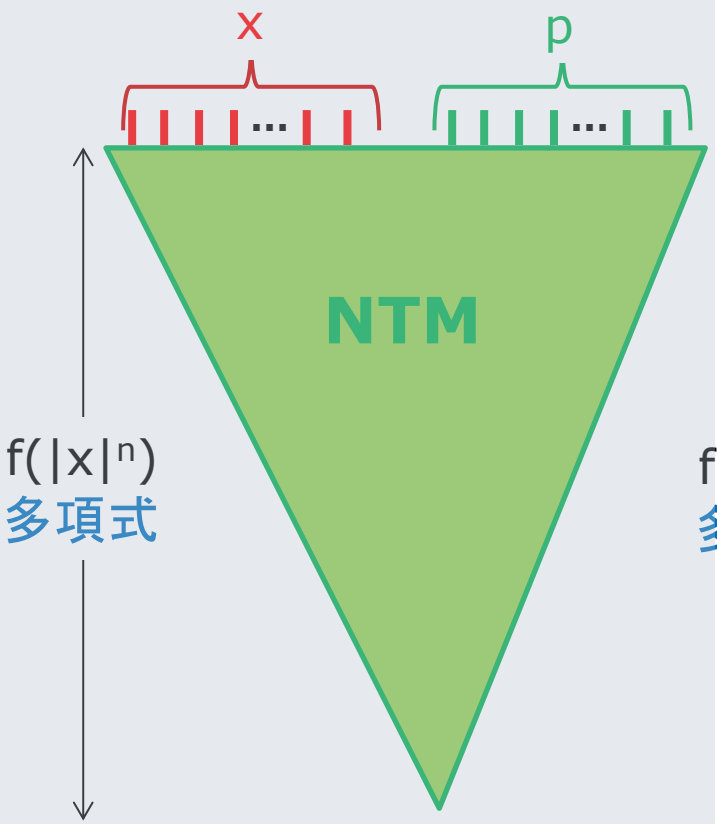
決定性
チューリングマシン



$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

$L \in P$

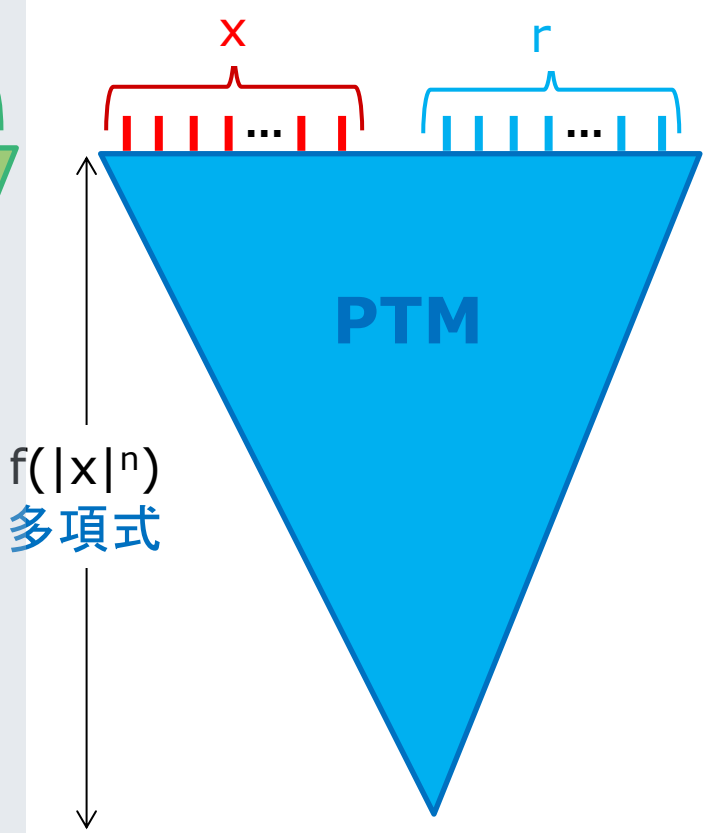
非決定性
チューリングマシン



$x \in L$
$\rightarrow \exists p\{NTM(x, p)=1\}$
$x \notin L$
$\rightarrow \forall p\{NTM(x, p)=0\}$

$L \in NP$

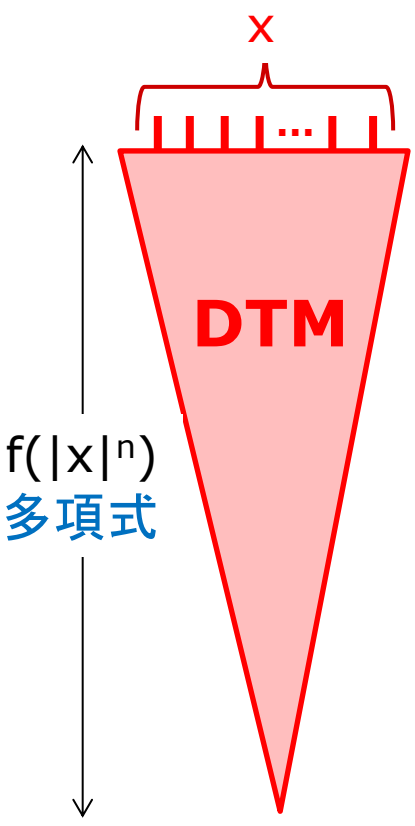
確率性
チューリングマシン



$x \in L$
$\rightarrow Prob(PTM(x, r)=1) \geq 2/3$
$x \notin L$
$\rightarrow Prob(PTM(x, r)=0) \leq 1/3$

$L \in BPP$

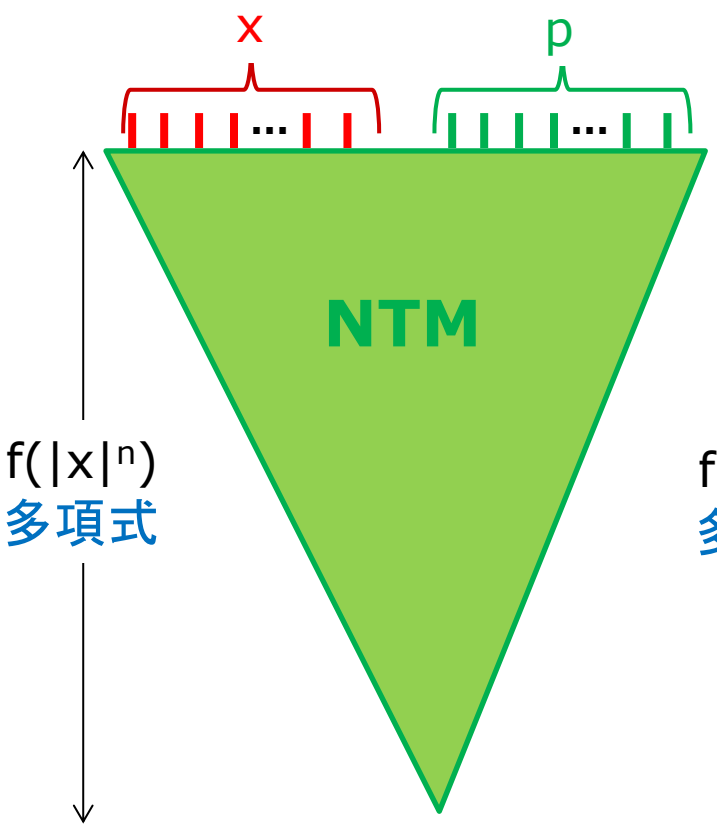
決定性
チューリングマシン



$x \in L$
$\rightarrow DTM(x)=1$
$x \notin L$
$\rightarrow DTM(x)=0$

$L \in P$

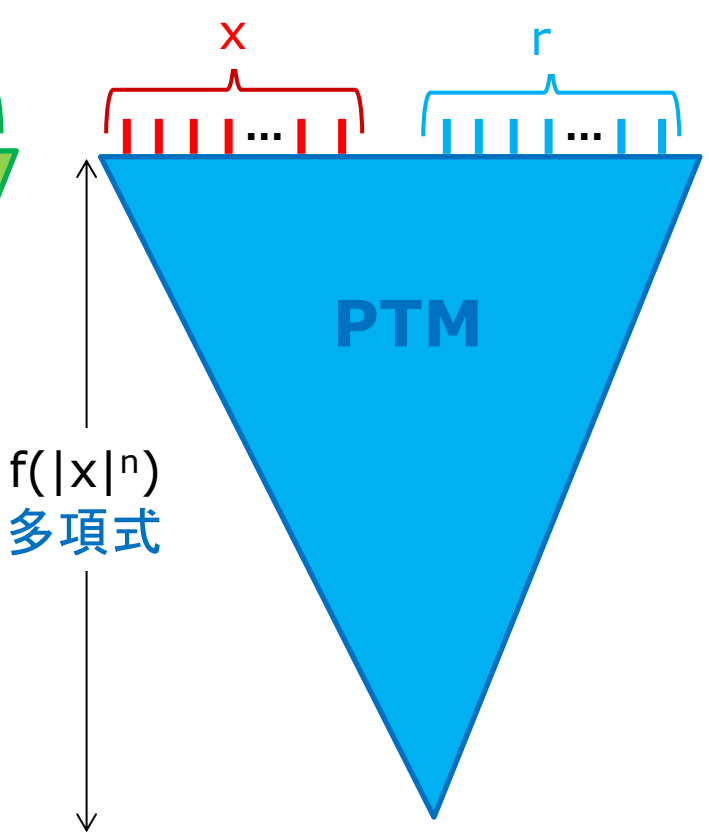
非決定性
チューリングマシン



$x \in L$
$\rightarrow \exists p\{NTM(x, p)=1\}$
$x \notin L$
$\rightarrow \forall p\{NTM(x, p)=0\}$

$L \in NP$

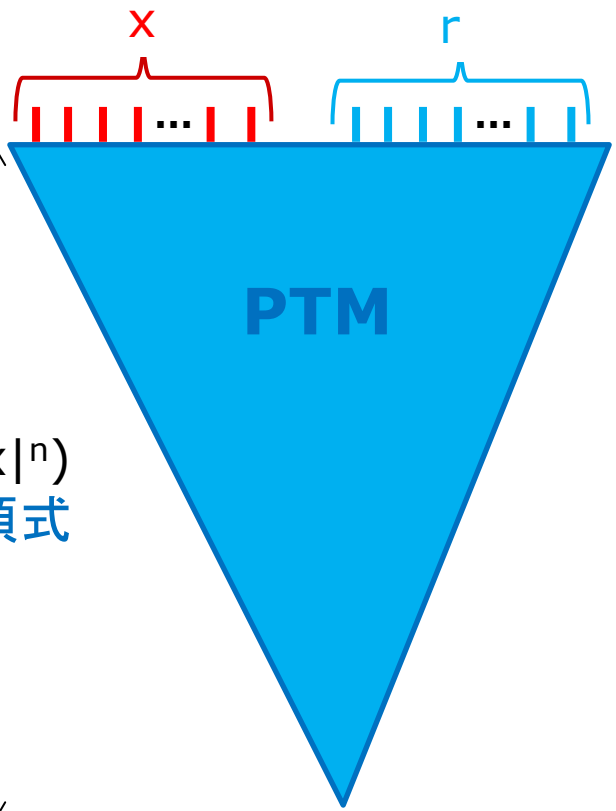
確率性
チューリングマシン



$x \in L$
$\rightarrow Prob(PTM(x, r)=1) \geq 2/3$
$x \notin L$
$\rightarrow Prob(PTM(x, r)=0) \leq 1/3$

$L \in BPP$

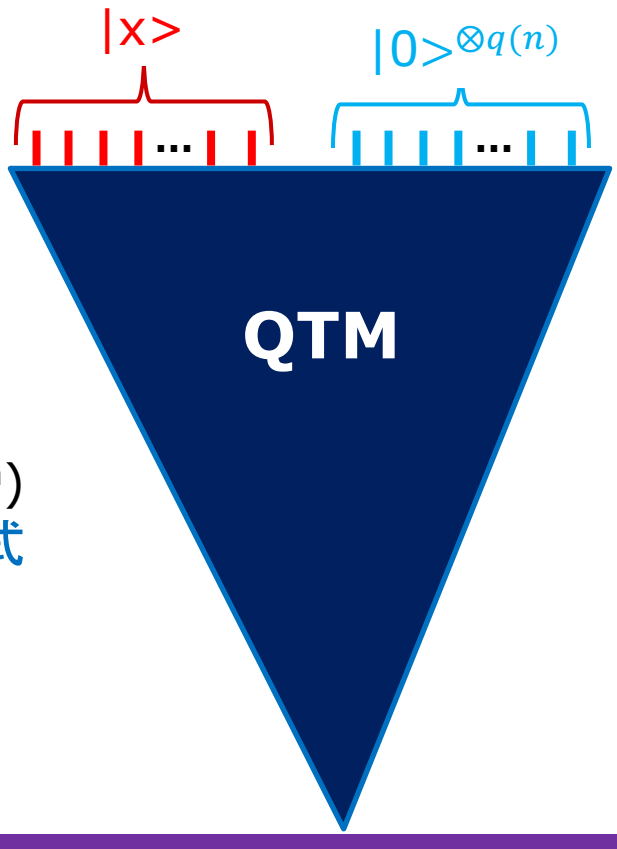
確率性
チューリングマシン



$x \in L$
→ $\mathrm{Prob}(\mathrm{PTM}(x, r)=1) \geq 2/3$
$x \notin L$
→ $\mathrm{Prob}(\mathrm{PTM}(x, r)=0) \leq 1/3$

$L \in BPP$

量子チューリングマシン

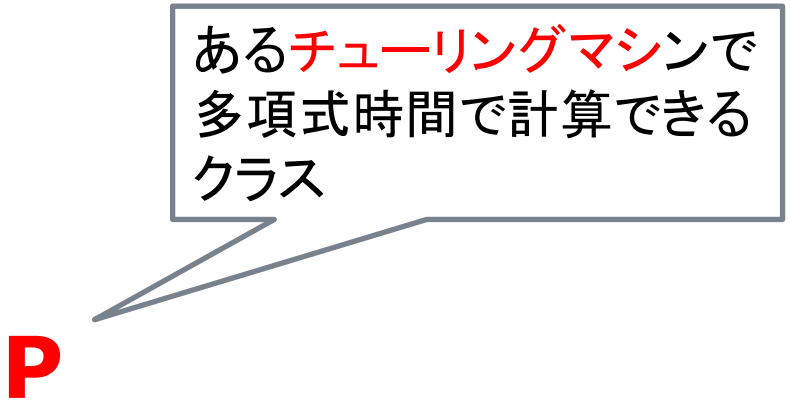


$x \in L$
→ $\mathrm{Prob}(\mathrm{QTM}(|x\rangle|0\rangle^{\otimes q(n)})=1) \geq 2/3$
$x \notin L$
→ $\mathrm{Prob}(\mathrm{QTM}(x\rangle|0\rangle^{\otimes q(n)})=0) \leq 1/3$

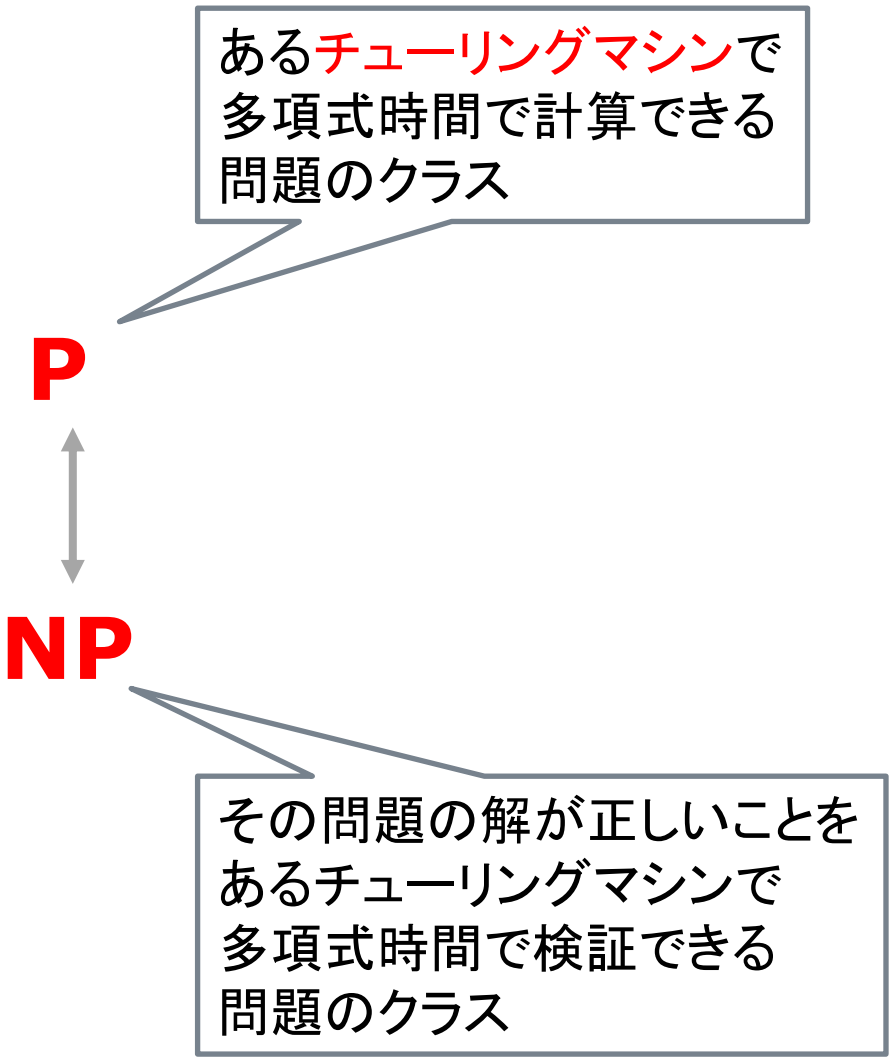
$L \in BQP$

決定論と確率論
古典論と量子論

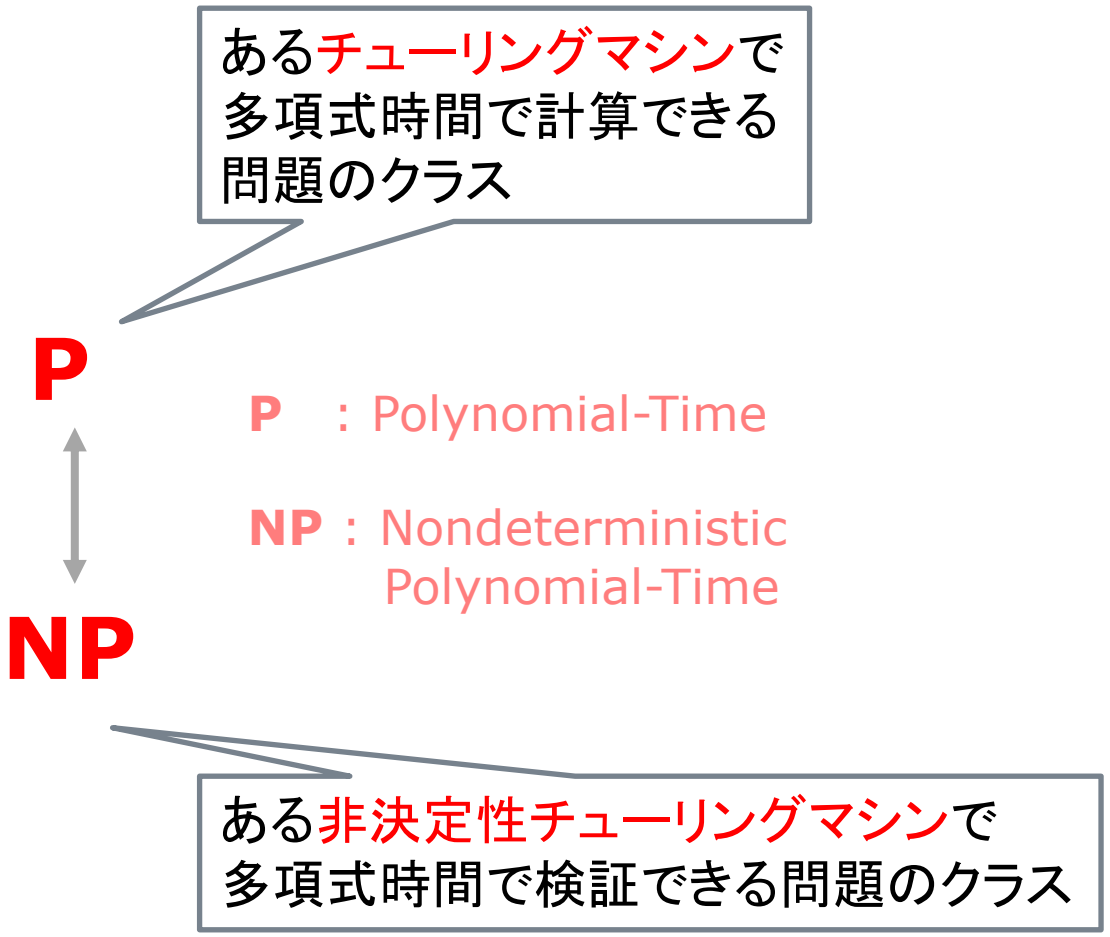
Pクラス



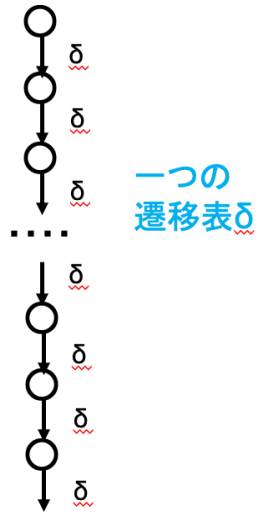
PクラスとNPクラス


あるチューリングマシンで
多項式時間で計算できる
問題のクラス
P
NP
その問題の解が正しいことを
あるチューリングマシンで
多項式時間で検証できる
問題のクラス

NPクラスの別の定義


あるチューリングマシンで
多項式時間で計算できる
問題のクラス
P
P ：Polynomial-Time
NP：Nondeterministic
Polynomial-Time
NP
ある非決定性チューリングマシンで
多項式時間で検証できる問題のクラス

NPクラスの別の定義



一つの
遷移表δ

決定性チューリングマシン

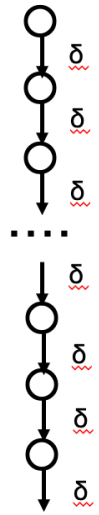
ある決定性チューリングマシンで
多項式時間で計算できる問題のクラス

P

NP

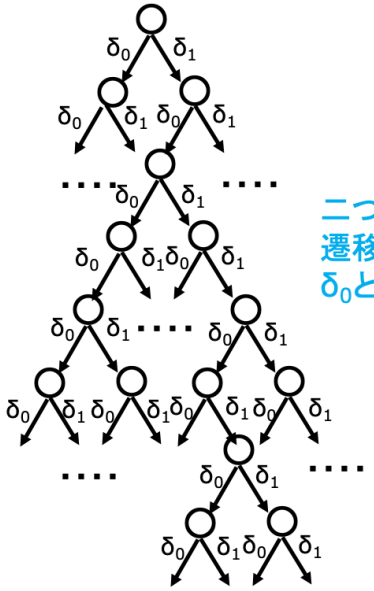
ある非決定性チューリングマシンで
多項式時間で検証できる問題のクラス

NPクラスの別の定義



一つの
遷移表δ

決定性チューリングマシン



二つの
遷移表
δ_0とδ_1

非決定性チューリングマシン

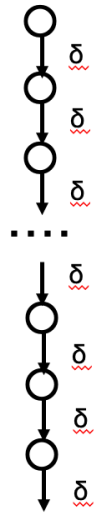
あるチューリングマシンで
多項式時間で計算できる
問題のクラス

P

NP

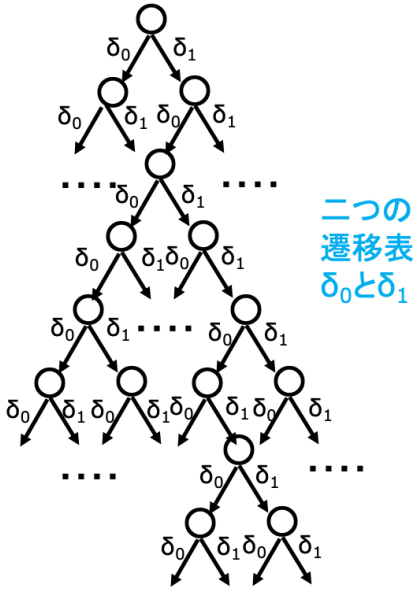
ある非決定性チューリングマシンで
多項式時間で検証できる問題のクラス

BPPクラス



一つの
遷移表δ

決定性チューリングマシン



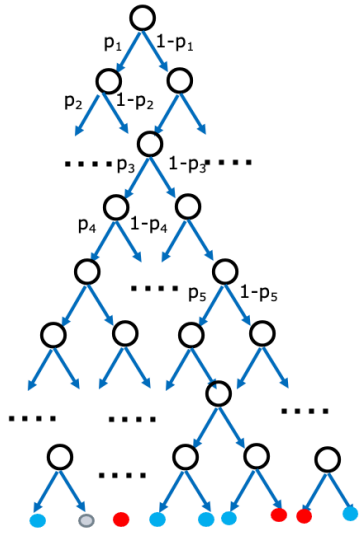
二つの
遷移表
δ_0とδ_1

非決定性チューリングマシン

P → **BPP**

NP

あるチューリングマシンで
多項式時間で計算できる
問題のクラス

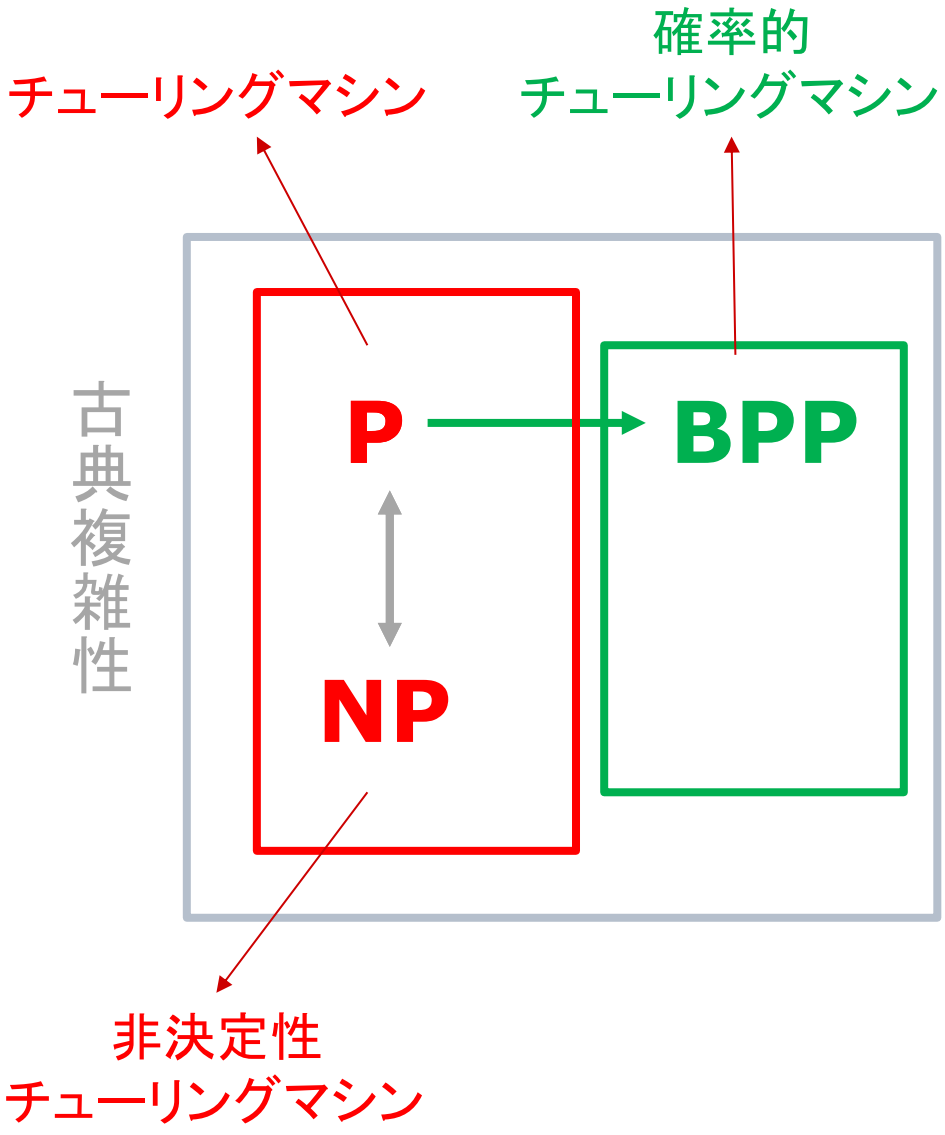


確率性チューリングマシン

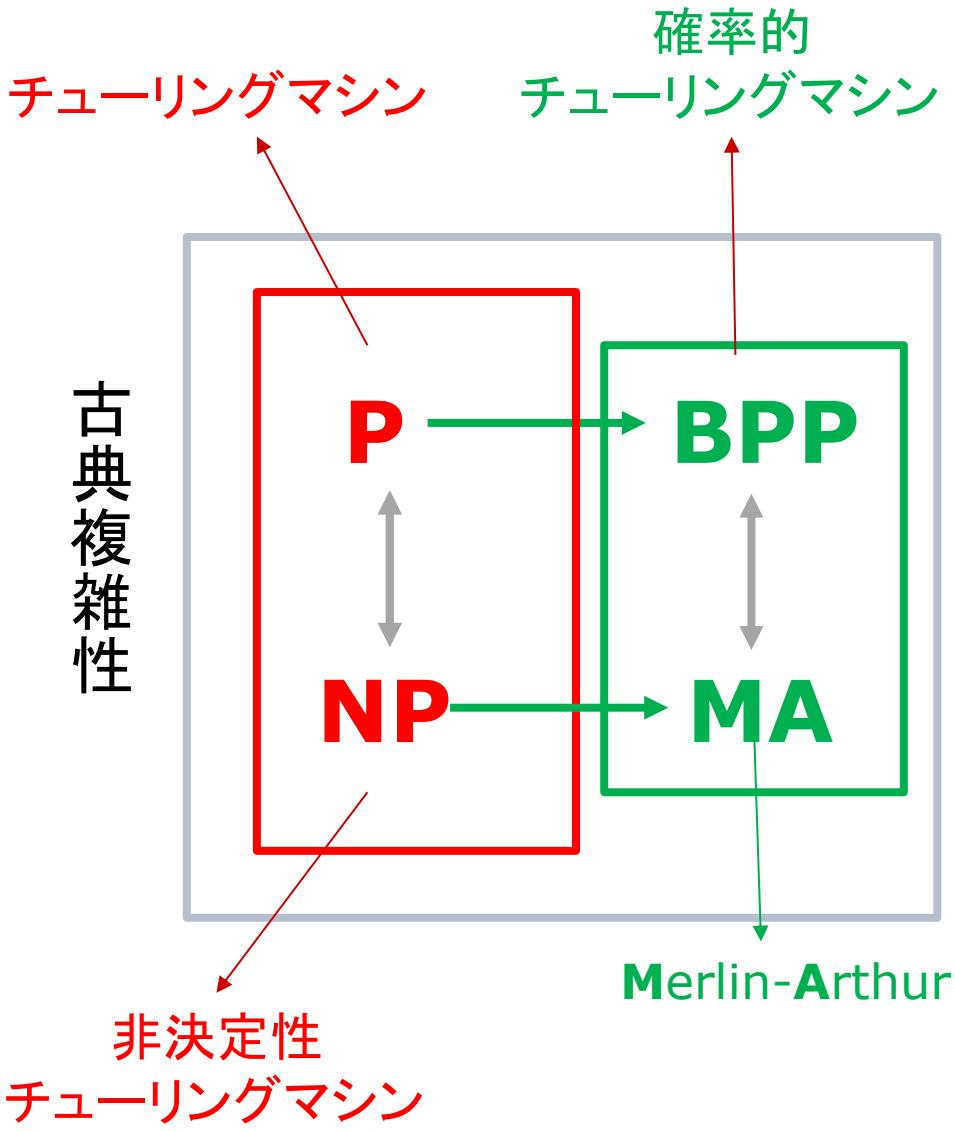
ある確率性チューリングマシンで
多項式時間で検証できる問題のクラス

ある非決定性チューリングマシンで
多項式時間で検証できる問題のクラス

古典複雑性



古典複雑性

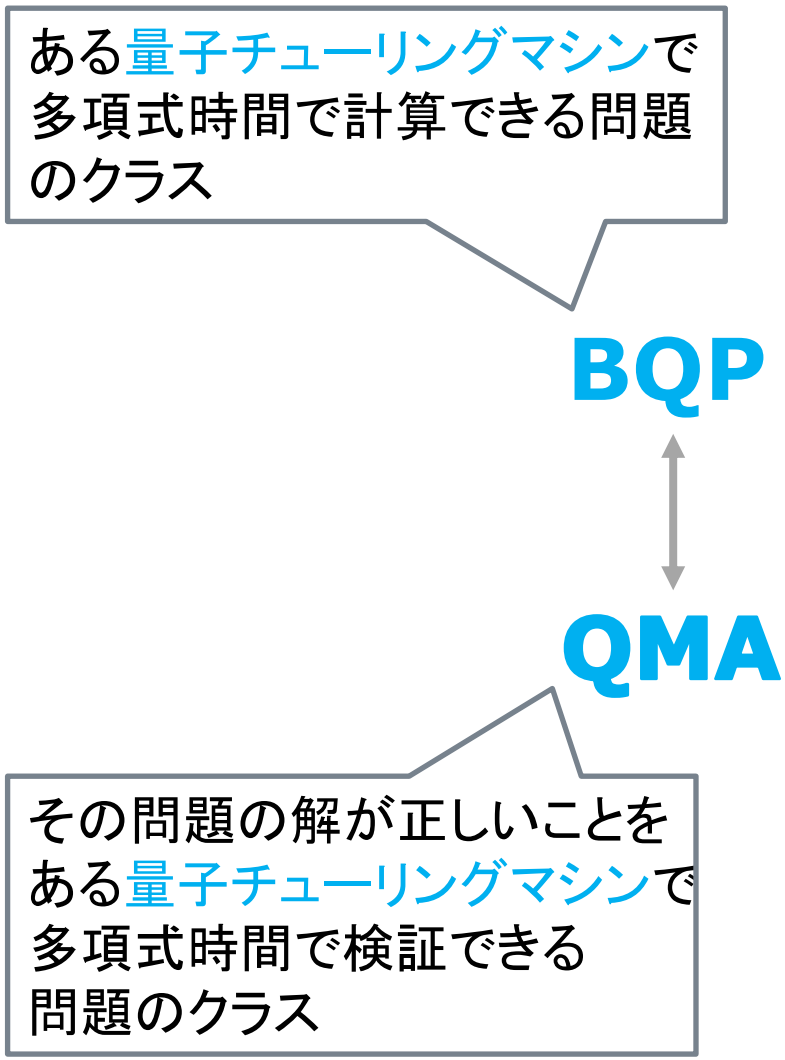

チューリングマシン
確率的
チューリングマシン
古典
複雑
性
P
BPP
NP
MA
Merlin-Arthur
非決定性
チューリングマシン

量子複雑性BQPクラス

ある量子チューリングマシンで多項式時間で計算できる問題のクラス

BQP

BQPクラスとQMAクラス



BQPクラスとQMAクラス

ある量子チューリングマシンで
多項式時間で計算できる問題
のクラス

BQP

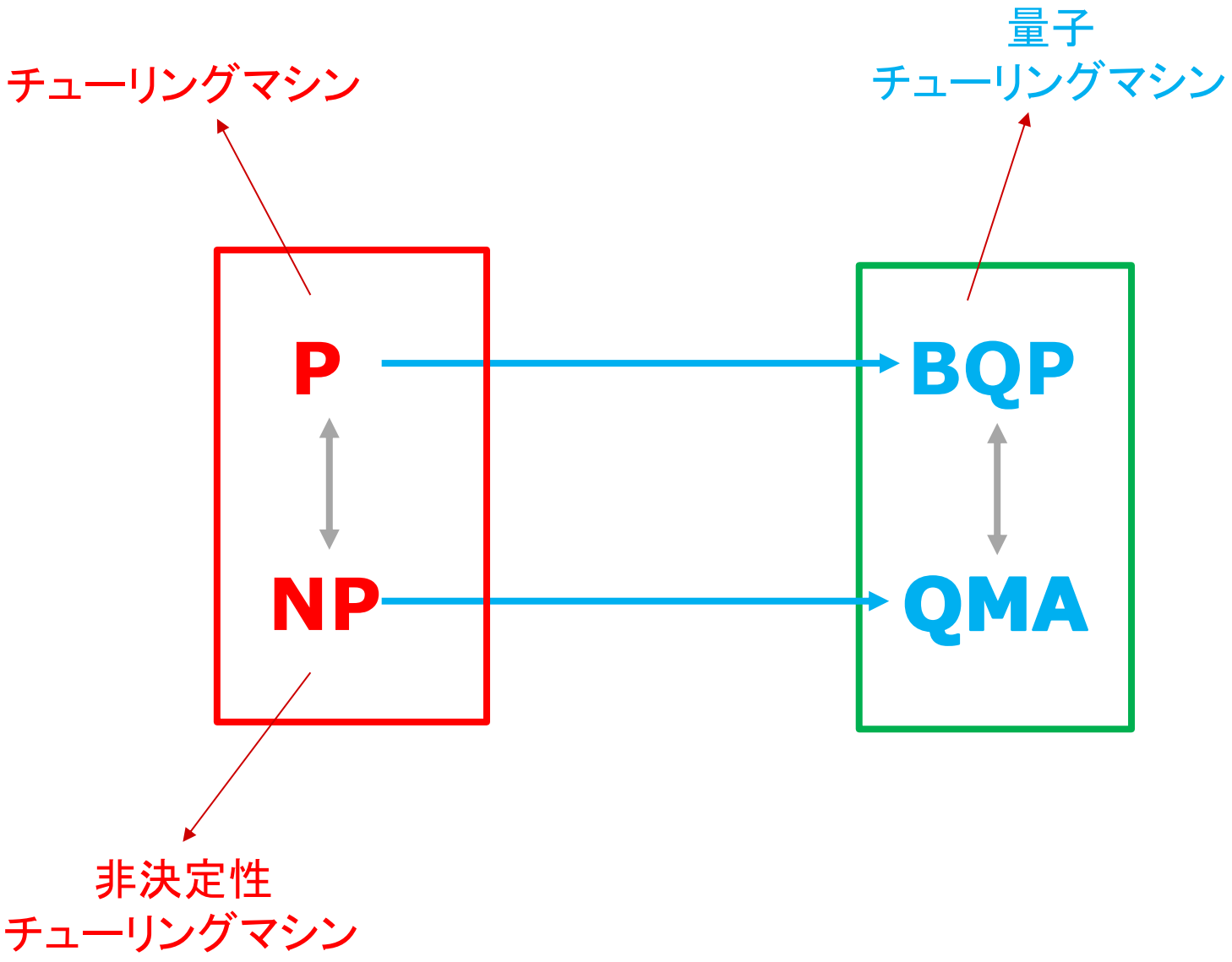
BQP：Bounded-Error Quantum
Polynomial-Time

QMA：Quantum MA

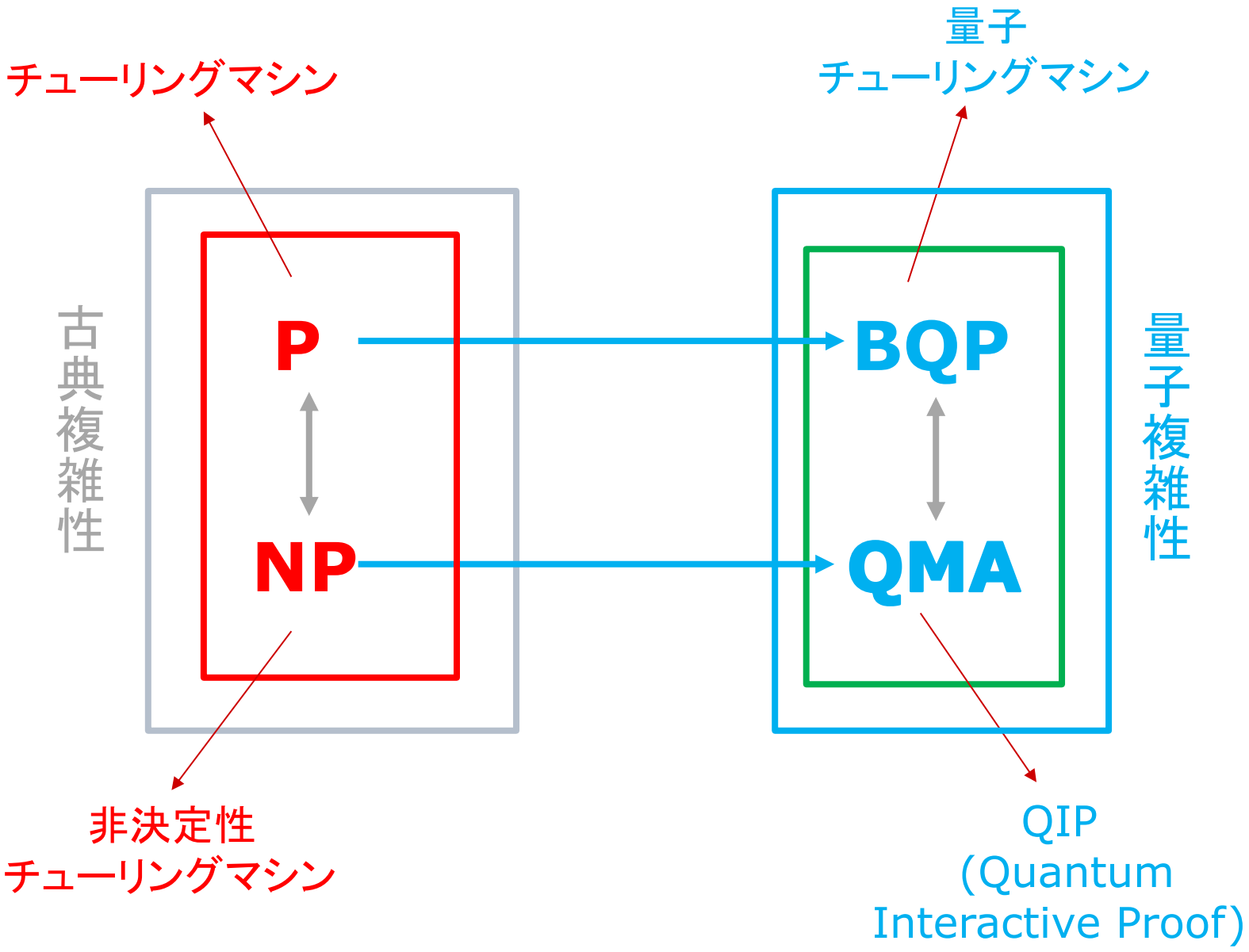
QMA

その問題の解が正しいことを
ある量子チューリングマシンで
多項式時間で検証できる
問題のクラス

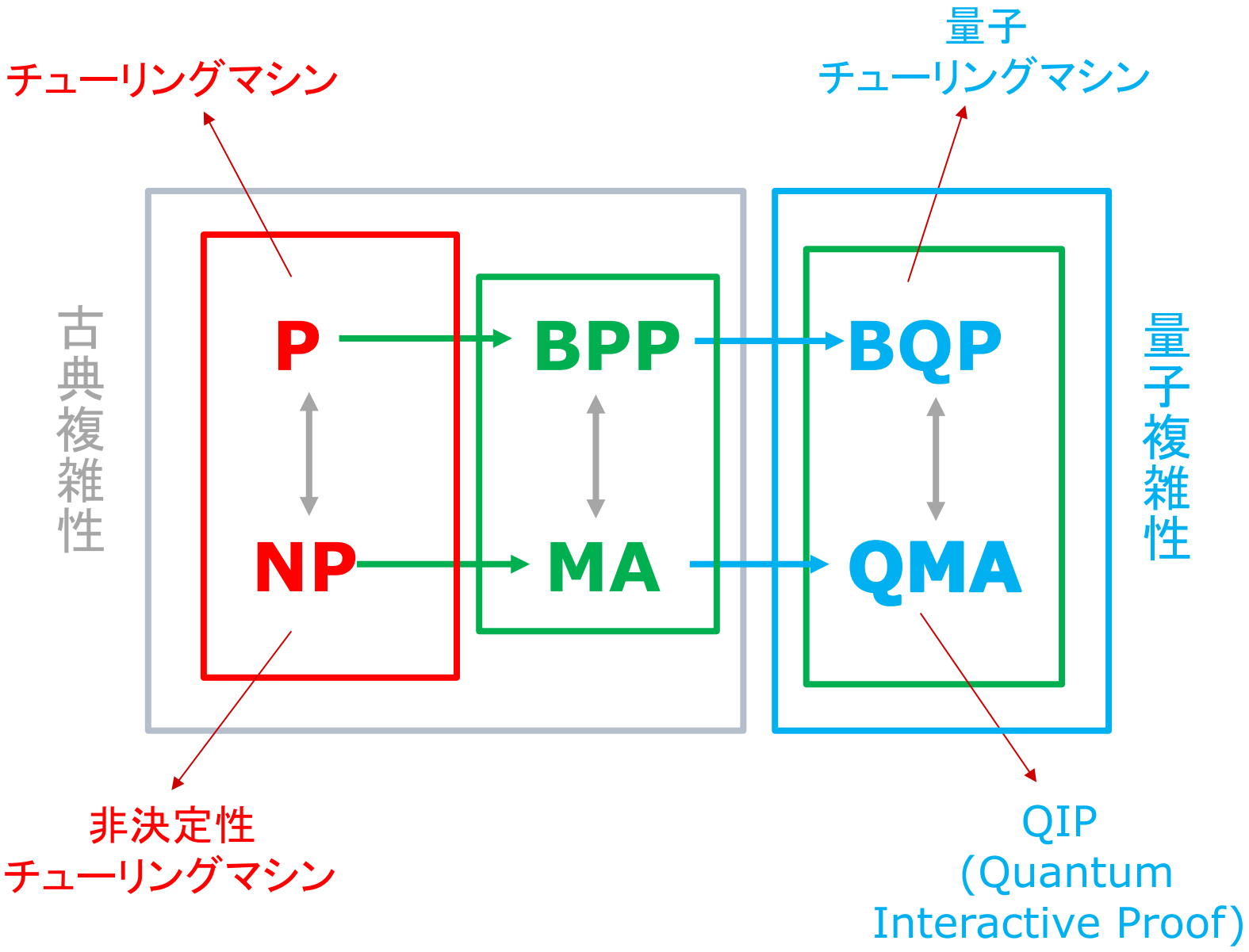
チューリングマシンと量子チューリングマシン



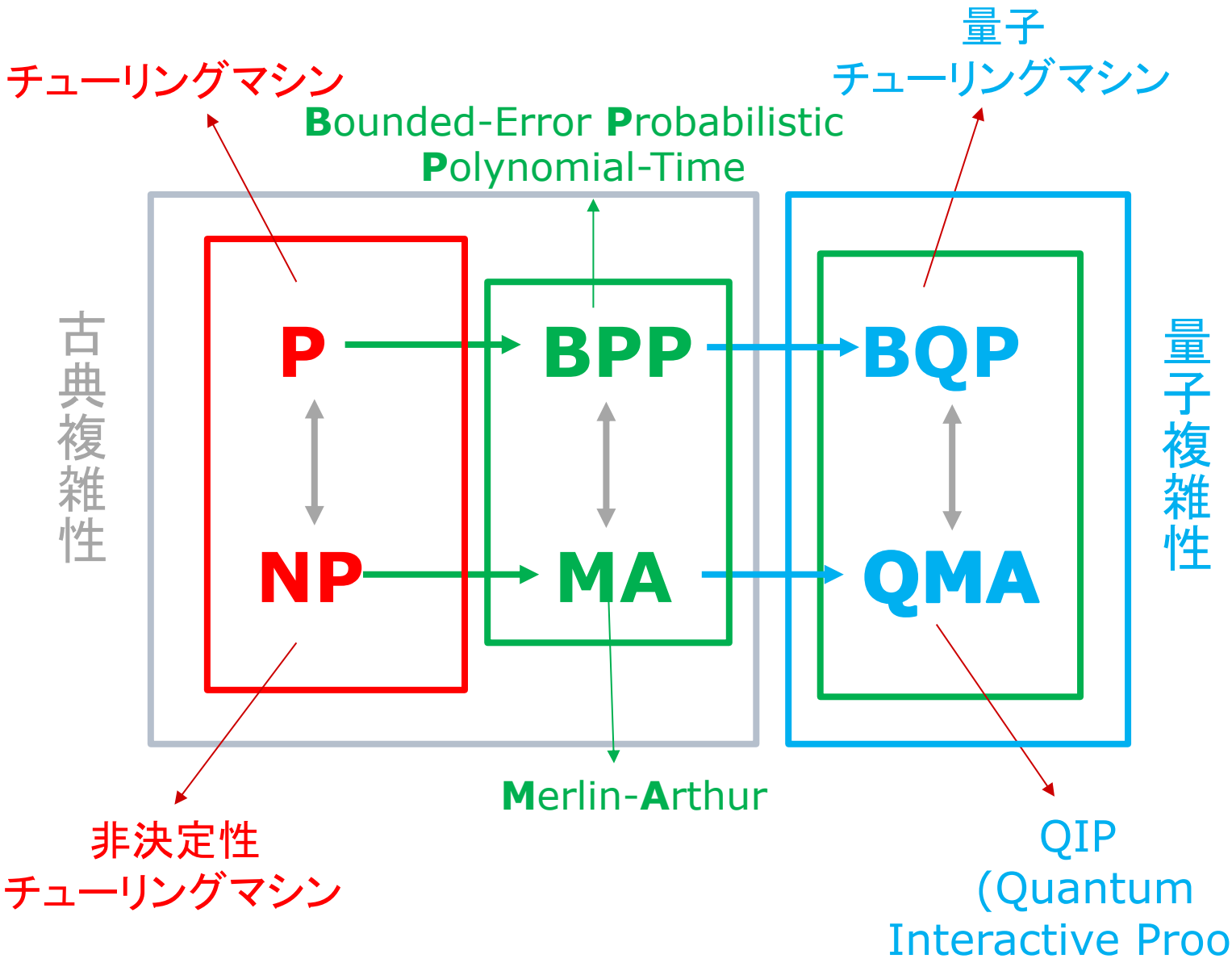
古典複雑性と量子複雑性



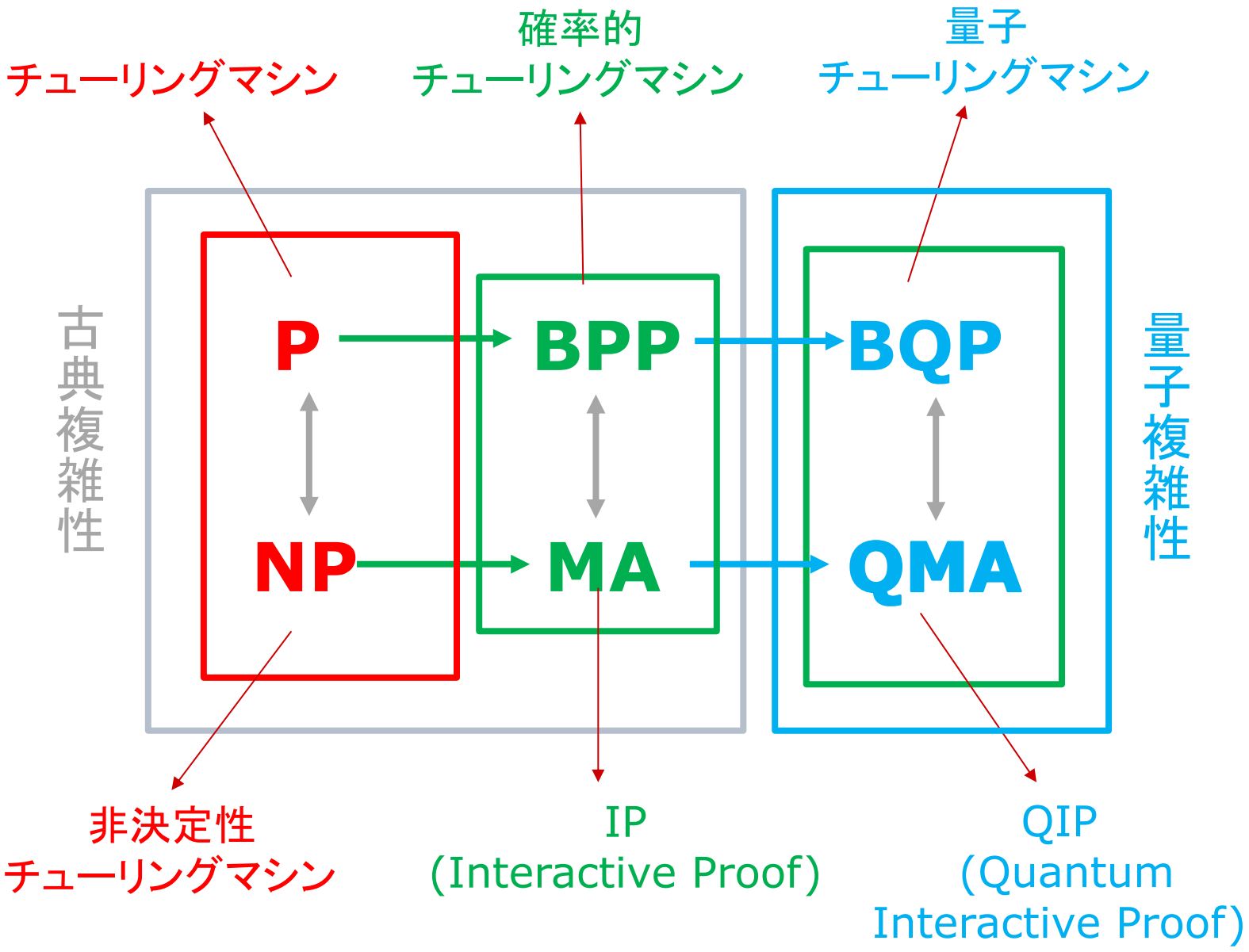
BPPクラスとMAクラス



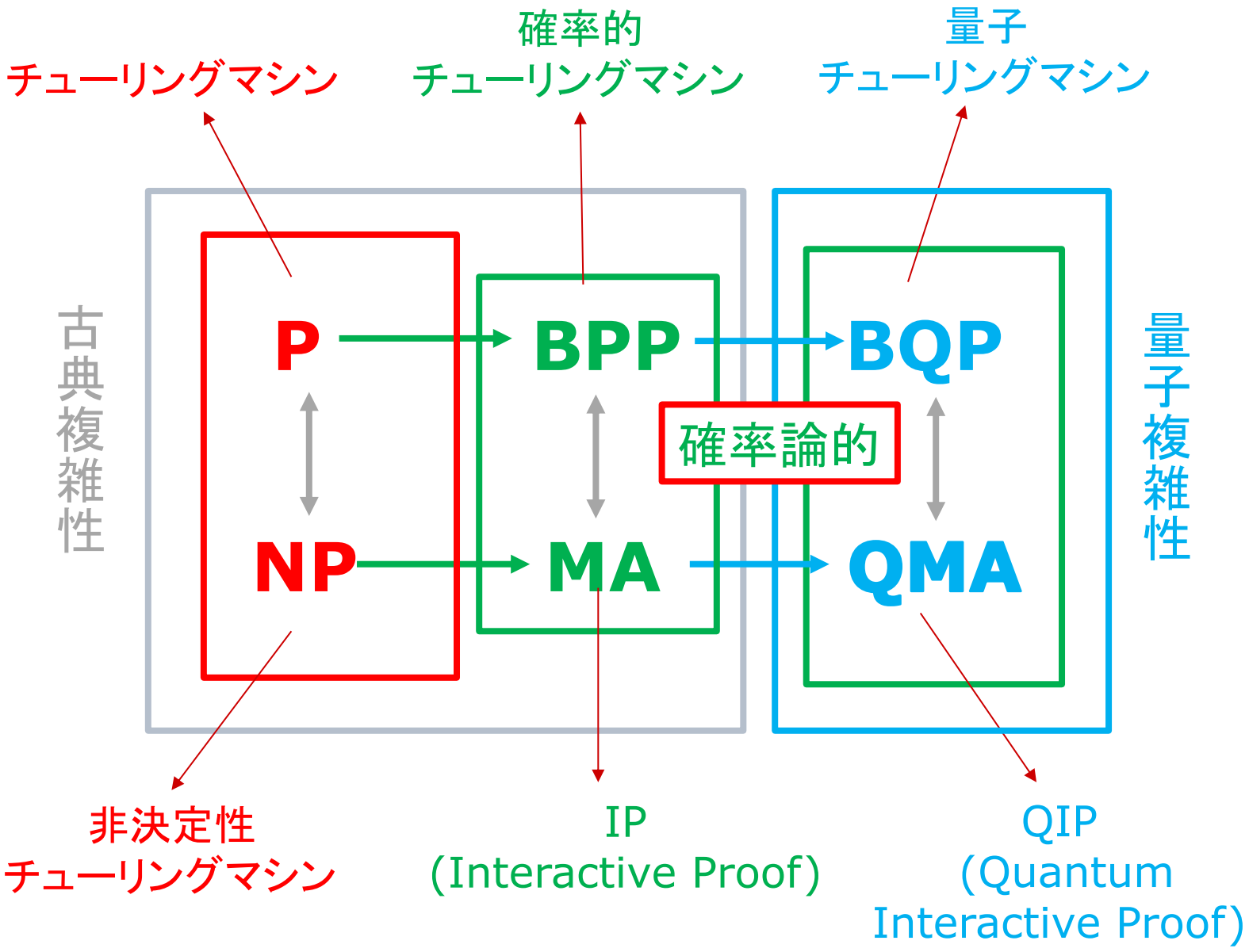
BPPクラスとMAクラス



確率性チューリングマシンとInteractive Proof



確率と証明の結びつき


チューリングマシン
確率的
チューリングマシン
量子
チューリングマシン
古典複雑性
P
BPP
BQP
確率論的
NP
MA
QMA
量子複雑性
非決定性
チューリングマシン
IP
(Interactive Proof)
QIP
(Quantum
Interactive Proof)

複雑性理論の転換

Agenda Part III
複雑性理論の転換

- Interactive Proof
  - Interactive Proofと証明概念の転換
  - Interactive Proofの基本的想定とパターン
  - PCP定理
- MIP*=RE定理
  - グラフの三色塗り分け問題の複雑性
  - N**Pクラスの階層とMIP*=RE定理
  - P,NPの関係の一般化とMIP*=RE定理
- 有限の中の無限、無限の中の有限

Interactive Proof

Interactive Proofの登場

1990年代に入って、計算複雑性理論は、大きく発展する。

その転機となったのは、László Babai, Shafi Goldwasser, Silvio Micali, Shlomo Moran, Charles Rackoff らによるInteractive Proof Systemである。Interactive Proofの登場は、計算複雑性理論を一変させた。

彼ら（彼女）らは、1993年、新設された「ゲーデル賞」の最初の受賞者となる。この手法とその拡張は、MIP*＝RE定理の証明で本質的な役割を果たす。

こうした方向での研究の重要な達成としての、1996-98年のPCP Theoremも、MIP*＝RE定理の証明で重要な役割を果たすことになる。

Interactive Proofと証明概念の転換

「証明者」と「検証者」の分離と両者の「対話」としての証明

- これまでも、数学的証明の特質については、様々な議論があった。証明が従うべき演繹規則、その出発点としての公理群、証明体系の無矛盾性 … 等々、ただ、それは、主要に証明を行う側から証明を考えたものだった。

- Interactive Proofの特徴は、「証明者」と「検証者」を分離し、証明を「証明者」と「検証者」の両者の「対話」の過程として捉え返すことである。

証明を、検証の側から捉え直す
検証されたものが正しい証明である

- それはまた、証明を証明の世界に閉じたものと考えることをやめて、証明を検証との関係で、検証の側から捉え返すことでもあった。

- 証明は、もともとが正しい推論によって導かれているから、正しいと検証されるのだと考えることと、正しいと検証されるから正しい証明だと考えることは別のことである。

- Interactive Proofのアプローチは、証明の正しさについて後者の考え方をとる。正しいと検証されたものが正しい証明なのである。

「確率的に検証可能な証明」というコンセプト

- 「正しいものから正しいものを演繹する」という証明観では、証明の世界と確率の世界を結びつけるのは難しい。演繹のルールは確率論的性質を持たず、厳密に決定論的に振る舞うからである。
- ただ、検証の世界は、確率論的な性質を持つ。証明者が、正しい証明を持っていない場合には、検証者に返す答がランダムなものになるのは明らかだし、正しい証明を持っている場合には、証明者が提示するサンプルの全てを検証する必要はない。
- 検証から証明を捉え返した時、証明の世界と確率の世界は結びつき、「確率的に検証可能な証明」という重要なコンセプトが現れてくる。

証明概念の転換としての Interactive Proof

- こうした、「証明観」の大きな転換は、証明が可能とするものの領域を大きく広げることとなった。

- Interactive Proofの基本的な想定と、それによる証明の射程の拡大を見る前に、我々が、これまで考えてきた、いわば「古典」的な「証明論」が、複雑性クラスのNP-完全のクラスにすっぽりと含まれることを見ておこう。

古典的な証明のイメージ

命題：T

証明：$\pi_1, \pi_2, \pi_3, \ldots, \pi_n = T$

- π_iは、公理であるか、予め定義された論理的な演繹規則から導かれた命題である。（ここでは、ZFCの公理系を取ろう）
- 論理的な演繹規則は、一階の述語論理を利用することにしよう。
- π_iが、$\pi_1, \pi_2, \pi_3, \ldots, \pi_{i-1}$から導かれることは、簡単にチェックされなければならない。それが、我々が証明を理解できるということである。

チェックのアルゴリズムと数学的導出

命題：T

証明：$\pi_1, \pi_2, \pi_3, \ldots, \pi_n = T$

- 数学的証明のシステムを形式化するには、$\pi_1, \pi_2, \pi_3, \ldots, \pi_{i-1} \rightarrow \pi_i$ なる数学的導出の各ステップを形式化すれば十分である。それは、チェックのアルゴリズムを形式化することに等しい。
- 数学的導出のルール、すなわち、それぞれのチェック・アルゴリズムが、証明の中に現れる命題と証明全体を規定している。
- また、証明が有限の時間のうちに検証可能であるなら、それを構成する、それぞれのチェック・アルゴリズム、すなわち、数学的導出のルールは、時間的な制約条件を満たさなければならない。

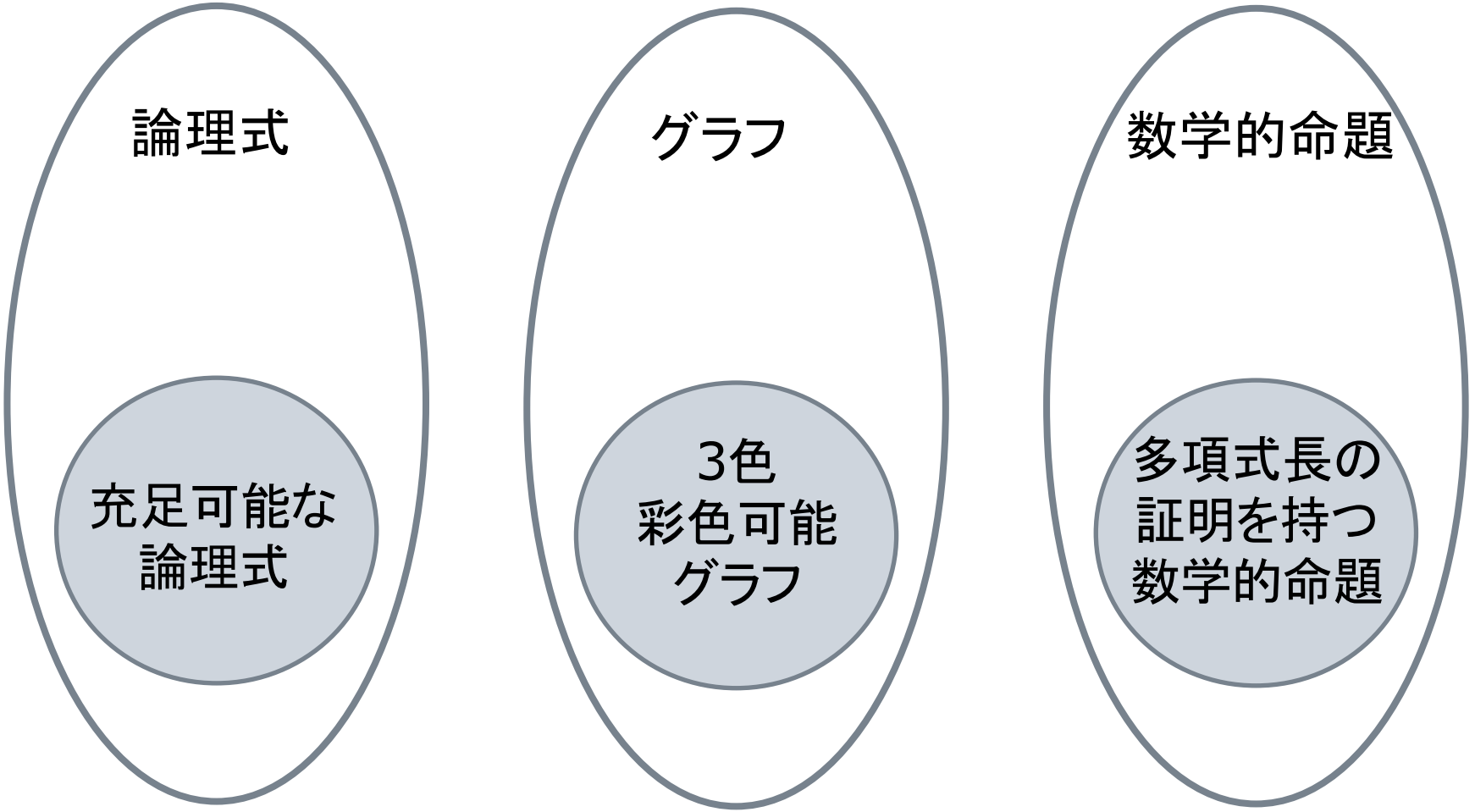
数学的証明問題はNP完全問題である

数学の問題を解くことは難しいが、その証明を理解することは証明を行うより容易である。かつその証明の理解は、多項式時間で行われる。数学的証明は、NP問題である。

証明可能な数学的な命題が、NP-困難のクラスに属することは自明である。なぜなら、数学的に記述された全てのNP問題は、証明可能な数学的命題のクラスに還元できるからである。

よって、(実効的に)証明可能な数学的な命題のクラスは、NP完全である。

代表的なNP完全のクラス

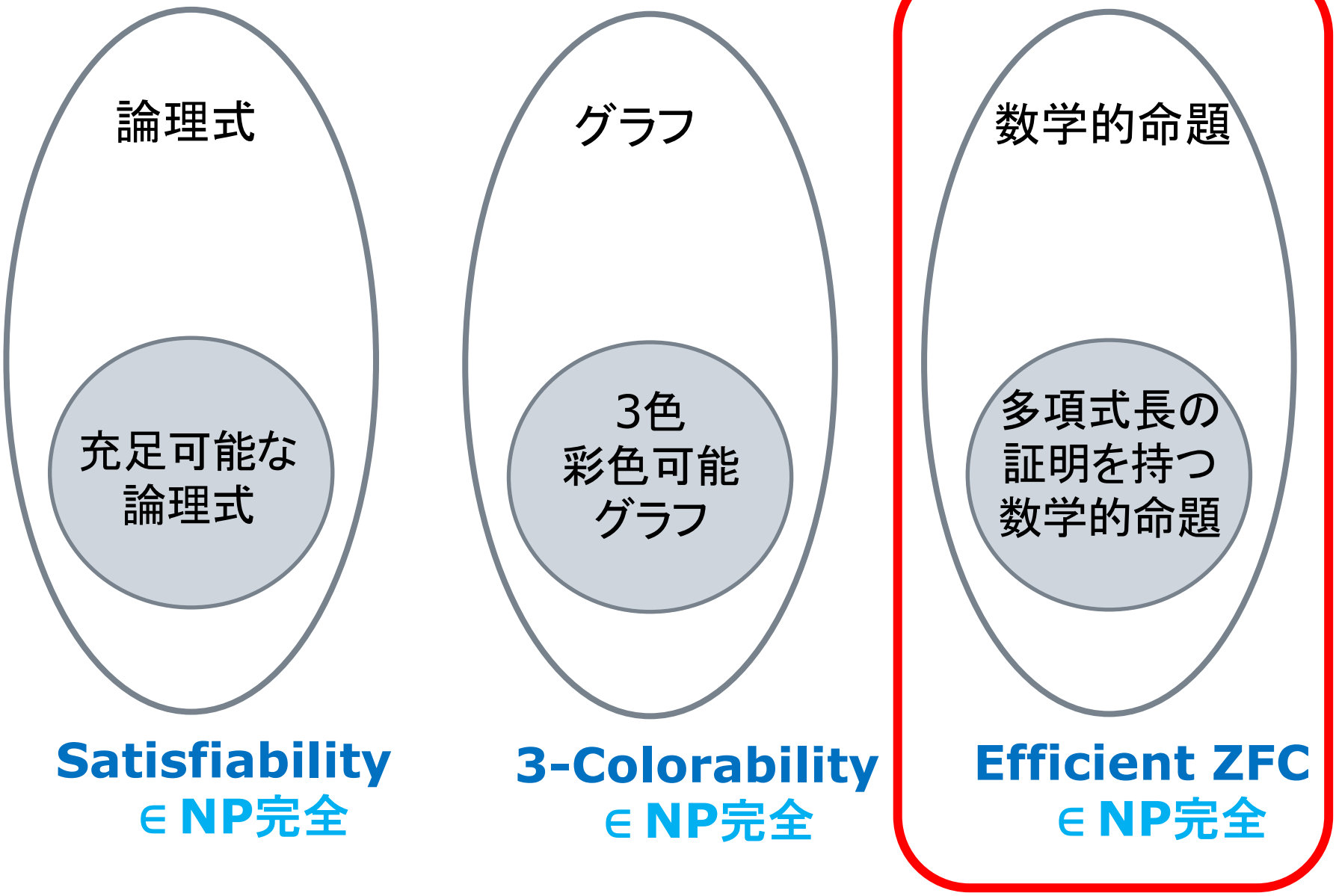


Satisfiability
∈ NP完全

3-Colorability
∈ NP完全

Efficient ZFC
∈ NP完全

代表的なNP完全のクラス

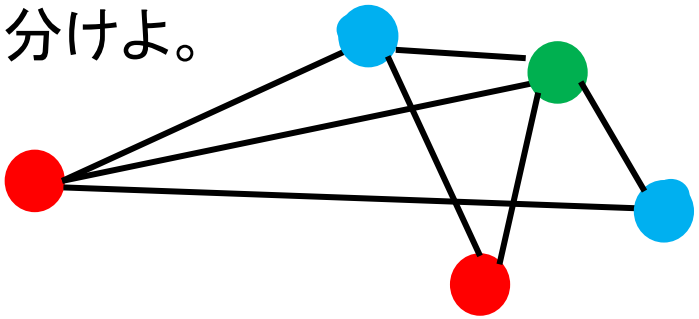


NP問題 3-SATの証明と検証の例

- 問題:
  論理式 $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_4} \vee x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4})$
  を充足する x_1, x_2, x_3, x_4 を求めよ。
- 証明:
  証明とは、論理式を充足する変数の割り当てを見つけることである。この例の場合、次のように解ける。
  x_1=Tなら、第一項と第二項はTとなる。x_2=T なら第三項もTになる。x_1= x_2= Tの時、x_3, x_4の値にかかわらず、この論理式は充足される。(一般には、変数の数がnの時、2^n乗個の場合を考える必要がある。)
- 検証:
  x_1= x_2= T を元の論理式に代入して、確かめる。
  (この検証は、n個のチェックで終わる)

NP問題 3-Colorabilityの証明と検証の例

- 問題：
  あるグラフG=(V,E)が与えられた時、このグラフの頂点Vに、{赤, 緑, 青}の三色の色を塗り、かつグラフの全ての辺Eの両端の色を異なるように塗り分けよ。



- 証明：
  証明は、条件を満たす頂点への色の割り当て$\{赤, 緑, 青\}^V$を見つけることである。
- 検証：
  全ての辺Eについて、両端の頂点Vの色が異なっていることを確かめる。

Interactive Proofの基本的想定とパターン

ProverとVerifierの役割

Interactive Proofでは、Prover(「証明者」)、Verifier(「検証者」)について、次のような役割を想定をする。

1. 「証明者」は、「検証者」に「証明」を提示し、「検証者」は、その「証明」を「検証」する。
2. ただし、「証明者」は、常に正しいことをいうとは限らない。
3. 「検証者」は、何度でも「証明者」に質問をすることができる。「証明者」は、必ずそれに答えなければならない。「検証者」は、その答えを「検証」する
4. 「証明者」が「検証者」に、その「証明」が正しいと確信させるのに成功した時、その証明は終わる。

ProverとVerifierの計算能力

Interactive Proofでは、Prover(「証明者」)、Verifier(「検証者」)の計算能力について、次のような想定をする。

1. 「証明者」の方が、「計算者」より高い計算能力を持つ。
2. 「証明者」は無限の計算能力を持つと想定する。複雑性理論での「オラクル」と似た役割を果たす。
3. 「検証者」は、人間がモデルと思っていい。その計算能力は多項式時間で制限され有限である。複雑性のクラスとしては、PまたはBPPのクラスで特徴付けられる。

Interactive Proofの基本的なパターン

Interactive Proofには、次のようなパターンがある。

1. **IP（Interactive Proof）:**
   証明者一人と検証者との対話型証明。
2. **MIP(Multi Prover Interactive Proof):**
   複数の証明者と検証者との対話型証明。
3. **MIP*(Multi Prover Interactive Proof with Entanglement):**
   エンタングルした量子を共有する複数の証明者と検証者との対話型証明。

Answer

Question

IP

Interactive Proof

Answer a

Answer b

Question x

Question y

MIP

Multi Prover
Interactive Proof

2004年、コンピュータ・サイエンティストのRichard Cleve, Peter Hoyer, Ben Toner, John Watrousらは、Bellの思考実験がInteractive Proofと極めて似ていることに気づく。

彼らは、量子的なMulti-Prover Interactive Proof MIP*を研究することを提案する。彼らが提起した問題は、次のようなものだった。

$\omega_q(G)$を近似的に求めるアルゴリズムは存在するだろうか？

Entanglement

Answer a

Answer b

Question x

Question y

MIP^*

Multi Prover
Interactive Proof
with Entanglement

Entanglement

複数のエンタングルした量子の共有を許す

Answer a

Answer b

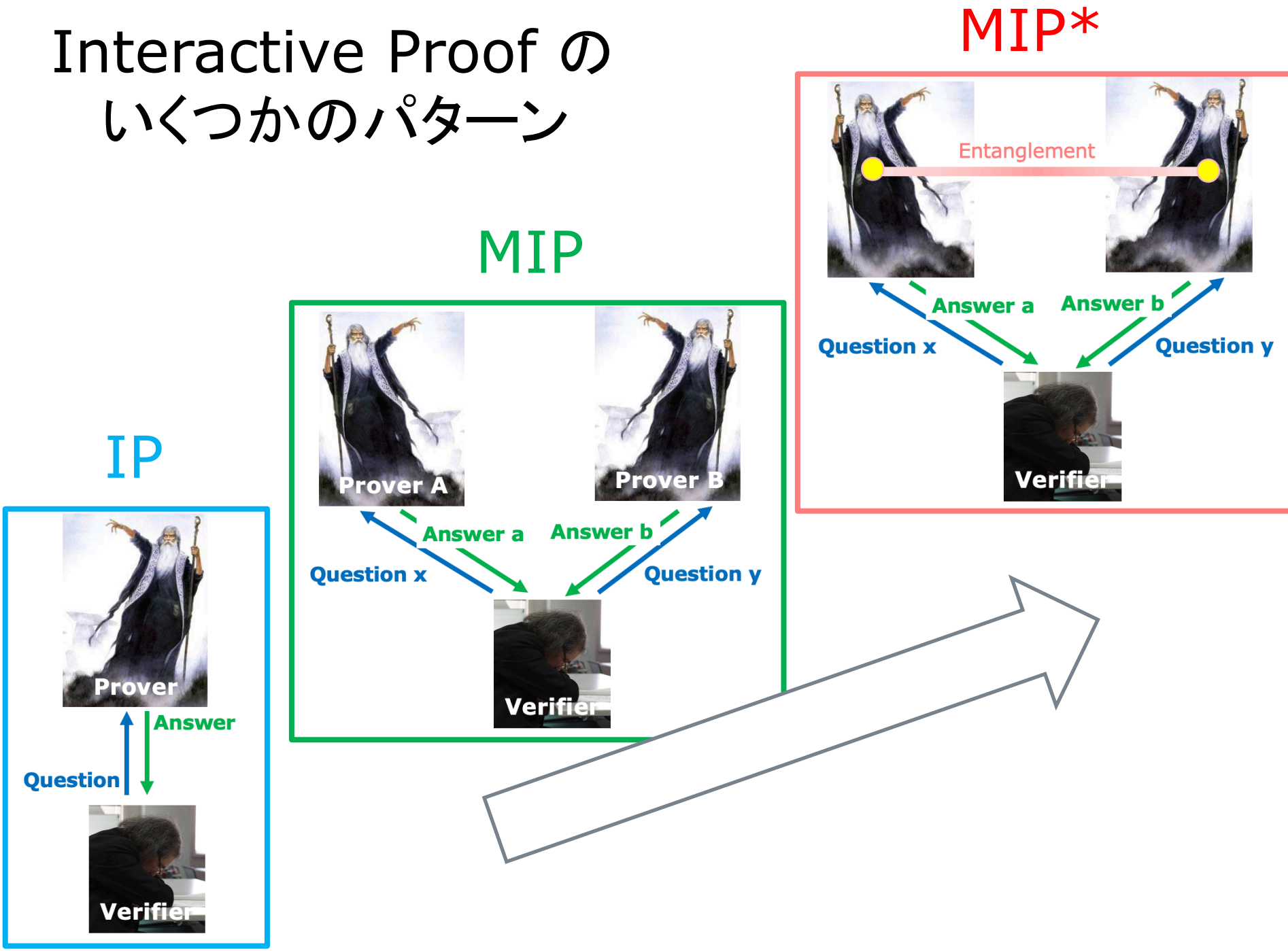
Question x

Question y

MIP^*

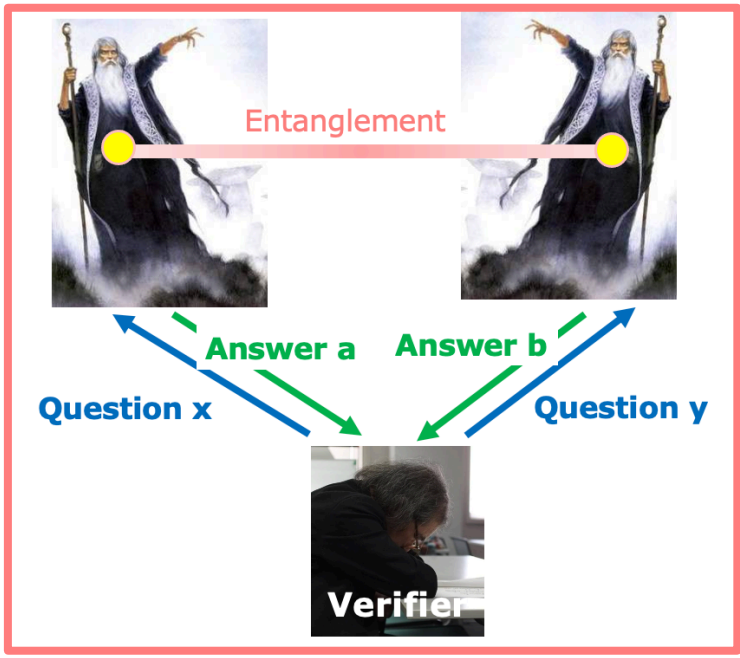
Multi Prover
Interactive Proof
with Entanglement

Interactive Proof の いくつかのパターン



Interactive Proof の基本的達成

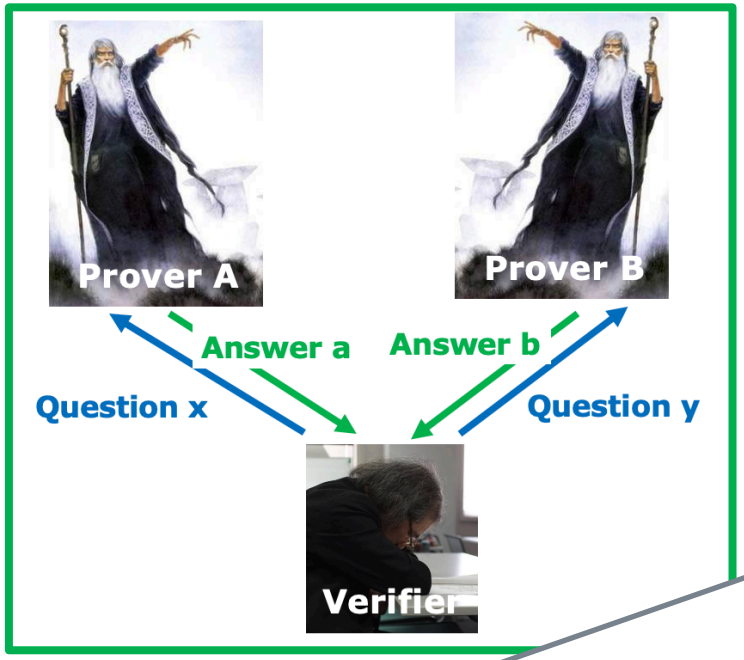
MIP*= RE



2020年 JNVWY

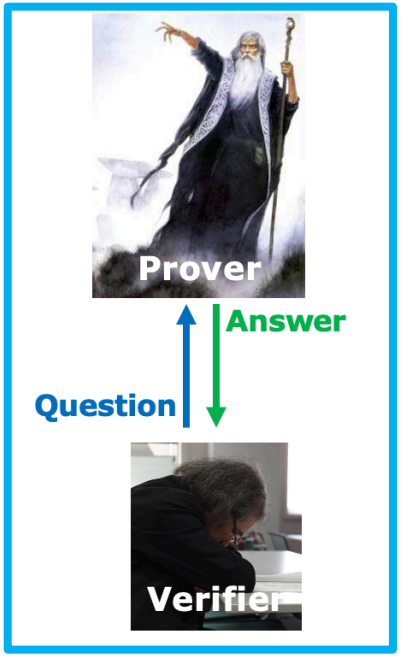
1991年 Babai, Fortnow, Lund

MIP= NEXP



1990年 Shamir

IP = PSPACE



対応する複雑性のクラス

PCP定理

PCP定理

- PCPは、Probabilistically checkable proof の意である。この定理は、NP問題は、「確率的に検証可能な証明」を持つことを主張している。もちろん、Interactive Proofのスタイルを使う。

- さまざまな定式化がある。1992年の、Arora-Lund-Motwani-Sudan-Szegedyの定式化では、
  NP ⊆ PCP[log n, 1] の形で述べられている。
- すなわち、log n ビットの情報のやり取りで、固定数の箇所の証明をチェックするだけで証明の検証が可能だという、驚くべき内容の定理である。

- MIP*=RE の証明でも、この定理は大活躍する。

NP Verifier と PCP Verifier

NP Verifier

- 入力を読む
- 証明の全体を読む
- 受理/拒否？

PCP Verifier

- 入力を読む
- ランダムにコインを投げる (O(log n))
- 証明の数ビットを読む
- 受理/拒否？

Completeness: $s \in L$ なら、証明は、高い確率で受理される

Soundness : $s \notin L$ なら、証明は、高い確率で拒否される

PCP定理とその応用

PCP Theorem

全てのNP集合は、PCP Verifierを持つ

ランダムさは、不確実性を加えるが、新しい力を与える

数量化できる： 2^{-k}のエラーには、3k ビットを読めばいい

理論的応用： 近似の難しさ、計算量の下限、
局所的にテスト可能なコード 等々

MIP* = RE定理

20/11/27 マルレク 「コンピュータ・サイエンスの現在 -- 「MIP*＝RE定理」とは何か？ --」はじめに から

- 「MIP*＝RE定理」というのは、MIP*という複雑性のクラスが、「決定不能」だということを述べている定理である。それは、ゲーデルの「不完全性定理」に端を発する、一連の「決定不能」型定理の一つと考えることが出来る。
- ゲーデルの結果は、人間の認識可能性の一つの限界を示すものとして、多くの人にいわば認識論的・哲学的なレベルで衝撃を与えた。
- ただ、その方法の適用は、数学の基礎の理論の内側にとどまっていて、科学の方法論そのものに影響を与えることはなかったし、ゲーデルの定理が、他の数理科学の分野の問題解決に応用されたことはなかったように思う。

20/11/27 マルレク 「コンピュータ・サイエンスの現在 -- 「MIP*＝RE定理」とは何か？ --」はじめに から

- 21世紀の「不完全性定理」とも呼ぶべき「MIP*＝RE定理」は、この点では、明らかに様相が異なる。
- それは、新興科学としての「コンピュータ・サイエンス」の「決定不能」型定理として登場したのだが、その定理は、狭い意味での 「コンピュータ・サイエンス」の枠を超え、純粋数学や物理学の基礎理論に、重要な応用を持つのである。
- エンタングルしたnonlocalゲームの値の「決定不能」性は、量子力学の基礎の「ティレルソンの問題」に対する否定的な答えを導く。また、この結果は、数学の作用素環論の長年の難問だった「コンヌの埋め込み予想」否定的に解決する。

20/11/27 マルレク 「コンピュータ・サイエンスの現在 -- 「MIP*＝RE定理」とは何か？ --」はじめに から

- ここでは、ゲーデルの結果を多くの人が受容したスタイルに倣って、誤解を恐れずに、この定理の認識論的・哲学的含意を考えようと思う。
- こうした反省は、正確さには難があったとしても、ものごとを「総合的・俯瞰的」に理解する上では有効なものだと僕は考えている。もちろん、ここでの解釈は、僕の解釈である。

グラフの三色塗り分け問題の複雑性

グラフの三色塗り分け問題の複雑性

n個の頂点を持つグラフが3色で塗り分けられるかという問題は、代表的なNP-完全問題である。

n個の頂点が3色に塗られたグラフは、3^n個存在する。それらのグラフの中から「塗り分け」の条件を満たすグラフを探すのは、容易ではない。この問題への答えが「YES」であることを示すこと、すなわち、このグラフが3色の塗り分けを持つことを「証明」するのは「むずかしい」。

ただ、その「証明」は、「これがこのグラフの3色の塗り分けだ」という形をしているはずである。こうした「証明」が与えられた時、その「証明」が「正しい」か否かの「検証」は、n個の頂点が「塗り分け」の条件を満たしているかのチェックで、多項式時間で可能である。その「検証」は、元のNP問題より「やさしい」問題で、多項式時間のクラスPに属する。

複雑性クラス NEXP

こうした、PとNPの関係を拡張して、次のようなクラスを考えることができる。
2^n個の頂点を持つグラフが3色で塗り分けられるかという問題を考える。2^n個の頂点が3色に塗られたグラフは、3^{2^n}個存在する。それらのグラフの中から「塗り分け」の条件を満たすグラフを探すのは、もちろん容易ではない。この問題の複雑性クラスをNEXPと呼ぶ。

なぜ、NEXP（それは"N+EXP"だ）という名前がつくかは、次のように考えればいい。この問題が「YES」という解を持つなら、「これがこのグラフの3色の塗り分けだ」という「証明」を持つ。この「証明」が正しいことの「検証」は、2^n個の頂点を一つづつ調べて、それが「塗り分け」の条件を満たしているかチェックすればいい。

それは、3^{2^n}個のグラフを検索することの「むずかしさ」とくらべれば、2^n個の頂点のチェックは、はるかに「やさしい」ことだ。この「検証」の複雑性は、2^n すなわち、EXP(指数関数的時間の複雑性)に属することになる。

もちろん、EXPクラスは、けっして「やさしい」クラスではない。それは、PやNPよりもずっと複雑なクラスである。それにもかかわらず、NEXPクラスの「むずかしさ」と比べると、EXPクラスはずっと「やさしい」クラスなのだ。

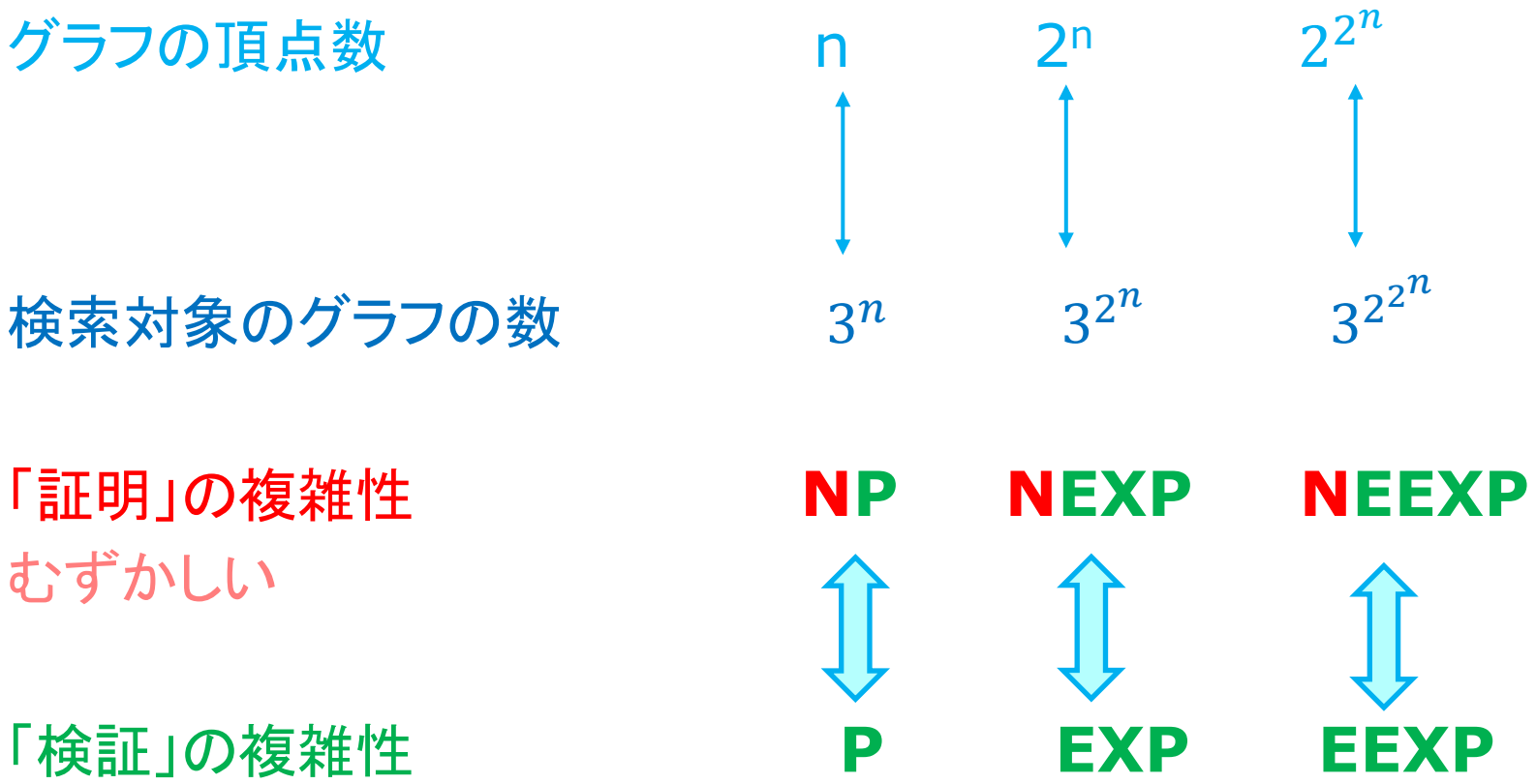
複雑性クラス NEEXP

同様に、2^{2^n}個の頂点を持つグラフが3色で塗り分けられるかという問題を考え、その複雑性NEEXPを考えることができる。このクラスは、その「証明」が与えられた時、それが「正しい」ことの「検証」が、2^{2^n}の複雑性 EEXPで可能なクラスである。

ここでも、証明の複雑性NEEXPは「むずかしい」のだが、その検証のクラスEEXPは、それと比べると「やさしい」ことになる。

こうした PとNPの関係の拡張は、もっと先に進めることができる。それについては、次に取り上げることにしよう。

グラフの三色塗り分け問題の複雑性



「塗り分け」の条件チェック：　辺で結ばれた頂点の色が異なること

N**Pクラスの階層とMIP*=RE定理

N**Pクラスの階層とMIP*=RE定理

先に、グラフの三色塗り分け問題を例に、次のような、二つの対をなす複雑性のクラスを三つ見てきた。

1. NP　　$O(3^n)$ の証明時間
   P　　O(n) の検証時間
2. NEXP　　$O(3^{2^n})$ の証明時間
   EXP　　$O(2^n)$ の検証時間
3. NEEXP　　$O(3^{2^{2^n}})$ の証明時間
   EEXP　　$O(2^{2^n})$ の検証時間

$$NP \subsetneq NEXP \subsetneq NEEXP$$

一つ確認しておきたいのは、2番目の NEXPクラスは、NPクラスではないということである。NPクラスは、証明のの検証が多項式時間で終わるクラスのことだが、このNEXPクラスは、検証に指数関数的時間を要するクラスである。多項式時間では、このクラスの検証は終わらない。

同様に、3番目のNEEXPクラスは、NPクラスでもNEXPクラスでもない。それは、証明の検証に、はるかに長い時間を必要とする、それらよりずっと複雑なクラスである。

形から見れば、NP, NEXP, NEEXP, …という延長上に、NEEEXPなり NEEEEXP … といった複雑性クラスを考えることができそうである。また、それは、大きな認識の飛躍のようにも思えないかもしれない。

NP 1972年、NEXP 1991年、NEEXP 2019年

結果的には、確かにそうなのだが、そうしたアイデアは複雑性理論の中で、最初から「自然」に生まれたものではないことには注意が必要である。そのことは、複雑性理論の歴史を振り返ればわかる。

歴史的には、1番目の NPクラスとNP-完全の概念の定式化は、1971年の CookとLevinに遡る。この間紹介してきた「グラフの3色塗り分け問題」がNP-完全であることを示したのはKarpで1972年のことだ。1970年代初頭は、複雑性理論の創世記である。

2番目のNEXPクラスの特徴づけがなされ、その基本的な性質が解明されたのは、BabaiとFortnowとLundが、NEXP = MIP を証明した時である。MIPはMulti-Prover Interactive Proof である。1991年のことだ。1990年代、Interactive Proofの手法の導入によって、複雑性理論は新しい段階に入る。

3番目のNEEXPクラスの特徴づけをあたえたのは、Anand Natarajan と John Wrightである。彼らは、BabaiとFortnowとLundが、NEXPの特徴付けに利用したMIPを、エンタングルメントで拡張した MIP* というInteractive Proofの新しい枠組みを使って、NEEXP ⊆ MIP* を証明した。2019年、つい一年半前のことである。

要するに、NPからNEXPに至るのに 20年、NEXPからNEEXPに至るのに、約30年の時間を要したということだ。

ただ、NatarajanとWrightのNEEXP ⊆ MIP* の証明の後、その手法を応用して、我々の複雑性の認識は、急展開することになる。

MIP*＝RE定理

2020年、Zhengfeng Ji, Anand Natarajan, Thomas Vidick, John Wright, Henry Yuen [JNVWY]らは、MIP*＝RE定理の証明の中で、次のことを証明する。

NEEEXP ⊆ MIP*
NEEEEXP ⊆ MIP*
NEEEEEXP ⊆ MIP*
NEEEEEEXP ⊆ MIP*
...
...
RE ⊆ MIP*

こうして、MIP*という枠組みの中で、我々は、NP, NEXP, NEEXP, NEEEXP, NEEEEXP, … と無限に続く複雑性クラスの階層をキチンと理解することができるようになった。無限に続くと言ったが、それは正確ではないかもしれない。Eの並びは、2の肩に乗っかる2の冪乗（Conwayのchain表記を使えば、2 → (2 → (2 → 2) …)の並びで表現される巨大だが有限な数に対応している。

NPからNEEXPに進むまでに50年の歴史が必要だったのだが、いまや、我々は、この複雑性の階層を一望できるのだ。

MIP*＝RE定理は、複雑性理論に革命をもたらした。

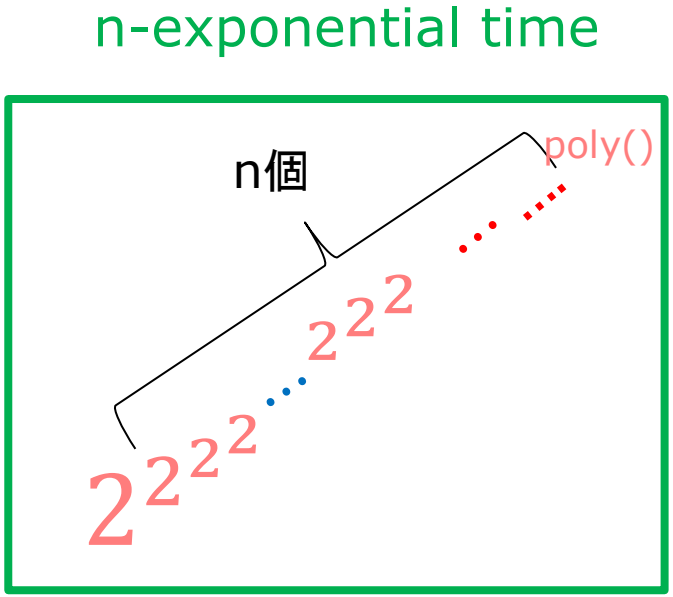
NEEXP ⊆ MIP* から MIP* = RE へ

Introspection とゲームの圧縮(**compression**)の手法を、繰り返し利用すると、次のような関係を証明できる。

RE ⊆ MIP*

NP ⊆ MIP*　　（自明）
NEXP ⊆ MIP*　　(‘91 BFL)
NEEXP ⊆ MIP*　　(’19 AV)
NEEEXP ⊆ MIP*
NEEEEXP ⊆ MIP*
NEEEEEXP ⊆ MIP*
NEEEEEEXP ⊆ MIP*
…

RE ⊆ MIP*　　(’20 JNVWY)

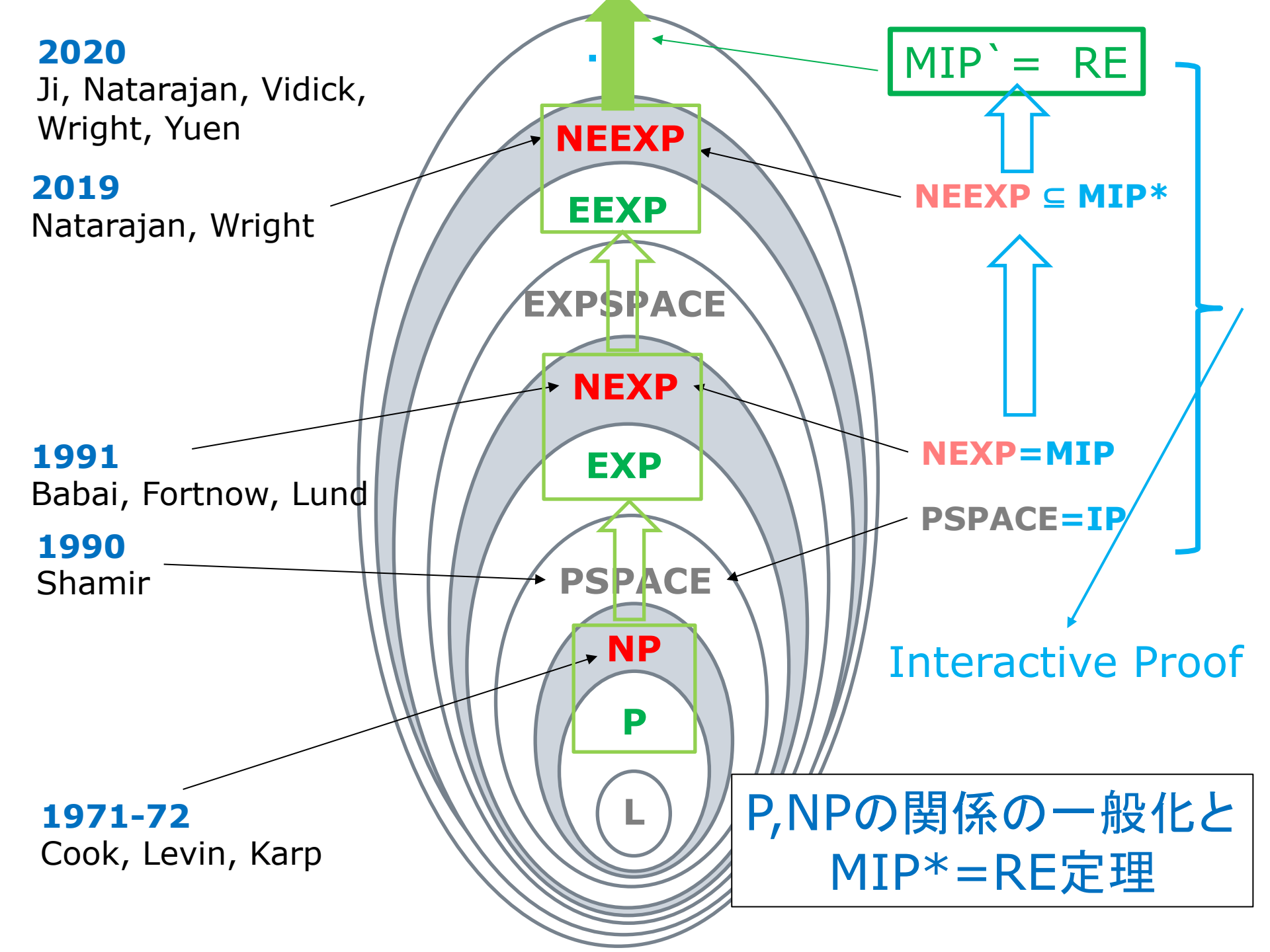


P,NPの関係の一般化と
MIP*=RE定理

P,NPの関係の一般化とMIP*＝RE定理

先に述べたことを図にまとめた。左側に、証明年度と証明者、右側に証明された命題を記入してある。

左側では、証明年代に大きなギャップがあること、すなわち、NP -> NEXP に20年、NEXP -> NEEXPに30年かかっていること、右側では、90年代以降では、証明手法として、Interactive Proofが大きな役割を果たしていることに、注目してほしい。


2020
Ji, Natarajan, Vidick, Wright, Yuen
2019
Natarajan, Wright
1991
Babai, Fortnow, Lund
1990
Shamir
1971-72
Cook, Levin, Karp
NEEXP
EEXP
EXPSPACE
NEXP
EXP
PSPACE
NP
P
L
MIP`= RE
NEEXP ⊆ MIP*
NEXP=MIP
PSPACE=IP
Interactive Proof
P,NPの関係の一般化と
MIP*=RE定理

MIP*＝REが明らかにしたこと

MIP*＝REが明らかにしたことの一つの小さな含意は、これまで「NP-困難」と一括りにされていた領域に、時間や空間のリソースによるある意味自明な階層の他に、NP概念の拡張としての明確な階層が存在することを示したことだと、僕は感じている。

（個人的には、「NP-完全」の定義の際に登場する「NP-困難」という領域が、具体的なイメージが及ばず、なにか正体不明の「暗黒」の領域のように感じていた。）

MIP*＝REの重要な含意は、こうした階層の「上限」として、RE(「帰納的可算」)のクラスがあるということだ。こうしてMIP*＝REは、有限な計算しか扱わなかった複雑性理論を、「帰納的可算」という無限を扱う計算可能性理論と結び付ける。

ただ、2の肩に2の冪乗が n個乗っかった指数関数は、nがどんなに大きかったとしても、それが具体的な数字である限り、帰納的で決定可能である。それが、決定可能ではないもの（「帰納的可算」）を含むためには、その系列が「無限」に続くことが必要になる。

先に見てきたのは、図の左側の n-exponential な時間で「検証可能」な複雑性を考えて、複雑性概念を拡張すると、ついには、「帰納的可算」のクラスに至るという話である。

n-exponential な数は、たとえそれがどんなに巨大なものであっても、有限な自然数であることに変わりはない。ただ、その有限の「無限」な連鎖の果てに、良く定義されたある意味自然な概念が、改めて、たち現れるのだ。

有限の中の無限、無限の中の有限

振り返ってみよう。

我々は、無限のものを認識できない。まず、認識の対象を形式的に有限な手段で定義可能なものに限る。「帰納的可算」というのは、そうした概念である。ただ、そこには、キチンと定義はできても、計算不可能だったり証明できないものが含まれる。

それで、「帰納的可算」の概念に手を入れ、キチンと定義され、かつ、計算可能なもの・証明可能なものを「帰納的」なものとして再定義する。「チューリング・マシンで実効的に計算可能なものが、計算可能である。」という「チャーチ・チューリングのテーゼ」は「計算可能性理論」（それは、「計算不可能性理論」でもあるのだが）の、ある意味、完成形である。

ただ、この「チャーチ・チューリングのテーゼ」の定義に従えば、「計算可能」であるはずなのだが、現実的には、どうしても計算できないものがたくさんあることに気付く。こうした実際には手に負えないものを、現実的に計算可能なものとを区別しようという動きが生まれる。「計算複雑性」の理論誕生の背景は、そうしたものだと思う。

こうして、「多項式時間」で計算できるものと、「多項式時間」では計算できないものとを区別しようという一つの基準が導入される。この基準は、ある意味合理的なものであった。1970年代に登場した複雑性理論は、コンピュータ・サイエンスの中心分野の一つとして発展し、成功する。それはいいことだ。

「拡大されたチャーチ・チューリングのテーゼ」

ただ、悪いニュースもある。いつの間にか、計算を科学する様々な分野に、次のような考えが生まれてくる。それは、もとの「チャーチ・チューリングのテーゼ」を修正した、「チューリング・マシンで多項式時間で計算可能なものが、計算可能である。」という考えが、「計算可能性」の定義として広まりはじめたことだ。それを「拡大されたチャーチ・チューリングのテーゼ」という。

「拡大されたチャーチ・チューリングのテーゼ」が、おそらく「誤り」であるだろうという、「傍証」は、近年の量子コンピュータ技術の発展による「量子優越性」の実験で示された。なぜなら、「量子優越性」というのは、「量子コンピュータで多項式時間で計算可能なものには、古典的コンピュータでは多項式時間で計算できないものが含まれている」ということだからである。例えば、素因数分解がその例だ。

量子コンピュータで「多項式時間で計算可能」なものと、古典的コンピュータで「多項式時間で計算可能」なものとが一致しないのなら、「計算可能性」の定義は「多項式時間」という規定だけでは、一意には決まらないのだ。（ここで「古典的コンピュータ」というのは、我々が日常的に使っている普通のコンピュータのことで、その計算能力は「（古典的）チューリング・マシン」に等しい。）

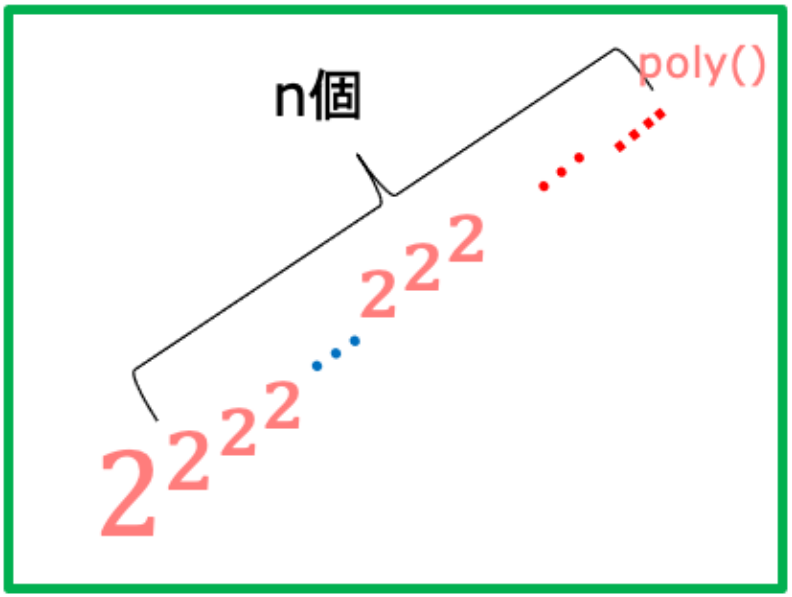
おそらくは誤りである「拡大されたチャーチ・チューリングのテーゼ」への拝跪は、我々の認識の「限界」（もちろん、計算科学の内部での話だ）を「多項式時間」と等置する傾向を生み出してきたのではないかとも感じている。

確かに、PとNPの関係では、NPも「多項式時間で検証可能なクラス」として「多項式時間」で限界づけられている。また、全ての証明可能な数学的命題が、NP-完全のクラスに属するにのだから、「多項式時間」を数学的認識の「限界」とみなすことは、充分、合理的でもあるように見える。

ただ、前回まで見てきた、MIP*＝REに結実するアプローチは、「多項式時間で検証可能なクラス」の束縛をはずしたことで可能となったことに、注意が必要だと僕は考えている。

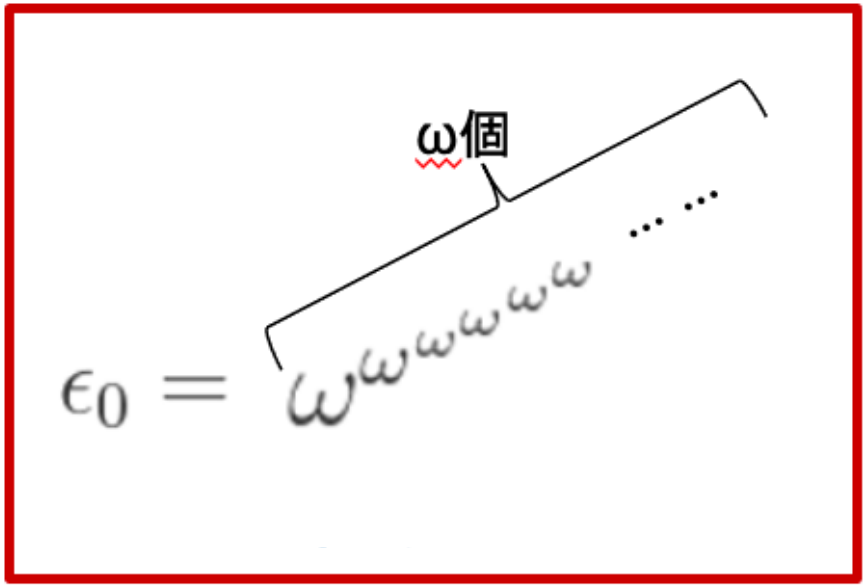
機会があれば、無限について初めて自由に考えた数学者カントールの、超限順序数の構成を紹介しようと思う（図の右側）。ここでのωは、自然数全体を表す。左のべきは、2についてだが、こちらは、もっと巨大な数のべきである。それにも関わらず、両者は、よく似ていると僕は思う。

n-exponential time



MIP*=RE証明に登場する
証明/検証に必要な時間複雑性
n→∞ の時　帰納的可算(RE)
のクラスとなる。帰納的可算集合
の存在は、ゲーデルの不完全性
定理のもう一つの表現である。

イプシロン・ゼロ



カントールが構成した超限順序数
この集合は、可算である！
ゲンツェンは、このε$_0$までの超限帰
納法を用いて自然数論の無矛盾性
を証明した。ゲーデルの不完全性
定理に反して！

驚くべきことに、このイプシロン・ゼロは、可算なのだ。無限の拡大を究極まで進めたとしても、我々は、可算無限を超えることは難しいのだ。そして、それは「証明可能性」について、驚くべき洞察を可能にする。ゲーデルの不完全性定理が禁じた「自然数論の無矛盾性証明」が可能となる。

おそらく、「多項式時間」の拘束に拘泥せず、無限について自由に考えることが、我々の数学的認識を豊かに広げる鍵の一つだと、僕は考えている。

自然数論の無矛盾性証明
ゲンツェン

自然数論の無矛盾性証明の不可能性
ゲーデルの不完全性定理

ω個

$\epsilon_0 = \omega^{\omega^{\omega^{\omega^{\omega^{\omega}}}}} \cdots \cdots$

無限

RE: 帰納的可算

可算

n個

poly()

$2^{2^{2^{2}}}$

有限

有限と無限を結ぶ可算無限