有限オートマトンと Presburger 算術

y. (@waidotto)

http://iso.2022.jp/math/tsudoi/11/slide.pdf

2018 年 10 月 28 日 @ 第 11 回関西すうがく徒のつどい 2 日目

目次

目次

記号

定義

- $\mathbb{N} = \{0, 1, 2, \dots\}$ (#0 は自然数)
- $\mathbb{N}_+ = \{1, 2, 3, \dots\}$

数学を数学する: 数学のモデル化

- 数学的な主張の真偽を，数学を使って分析したい
- そのためには「数学的な主張」を数学的に明確に定義する必要がある
- そのために，論理結合子 $\wedge, \vee, \neg, \to, \forall, \exists$ や等号 $=$，変数記号 $x, y, z, \ldots$ などの記号を使って，数学的主張を形式化するという手法が使われる
- 「全数学」を形式化するのはひとまず置いておいて，まずは自然数 $\mathbb{N}$ に関する主張を書けるようにしてみる．そしてそれに必要となる語彙 $\mathcal{L}_{\mathrm{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を用意する

Table: 論理結合子

$\neg\varphi$	φ でない
$\varphi \wedge \psi$	φ かつ ψ
$\varphi \vee \psi$	φ または ψ
$\varphi \to \psi$	φ ならば ψ
$\forall x\varphi$	全ての x について φ が成り立つ
$\exists x\varphi$	φ が成り立つような x が存在する

Table: 算術の語彙 $\mathcal{L}_{\mathrm{arith}}$

$\underline{0}$	定数記号 0
$\underline{1}$	定数記号 1
$x + y$	x と y の和
$x \times y$	x と y の積
$x < y$	x より y の方が大きい

数学を数学する: 数学のモデル化

- 数学的な主張の真偽を，数学を使って分析したい
- そのためには「数学的な主張」を数学的に明確に定義する必要がある
- そのために，論理結合子 $\wedge, \vee, \neg, \to, \forall, \exists$ や等号 $=$，変数記号 $x, y, z, \ldots$ などの記号を使って，数学的主張を形式化するという手法が使われる
- 「全数学」を形式化するのはひとまず置いておいて，まずは自然数 $\mathbb{N}$ に関する主張を書けるようにしてみる．そしてそれに必要となる語彙 $\mathcal{L}_{\mathrm{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を用意する

Table: 論理結合子

$\neg\varphi$	φ でない
$\varphi \wedge \psi$	φ かつ ψ
$\varphi \vee \psi$	φ または ψ
$\varphi \to \psi$	φ ならば ψ
$\forall x\varphi$	全ての x について φ が成り立つ
$\exists x\varphi$	φ が成り立つような x が存在する

Table: 算術の語彙 $\mathcal{L}_{\mathrm{arith}}$

$\underline{0}$	定数記号 0
$\underline{1}$	定数記号 1
$x + y$	x と y の和
$x \times y$	x と y の積
$x < y$	x より y の方が大きい

数学を数学する: 数学のモデル化

- 数学的な主張の真偽を，数学を使って分析したい
- そのためには「数学的な主張」を数学的に明確に定義する必要がある
- そのために，論理結合子 $\wedge, \vee, \neg, \to, \forall, \exists$ や等号 $=$，変数記号 $x, y, z, \ldots$ などの記号を使って，数学的主張を形式化するという手法が使われる
- 「全数学」を形式化するのはひとまず置いておいて，まずは自然数 $\mathbb{N}$ に関する主張を書けるようにしてみる．そしてそれに必要となる語彙 $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を用意する

Table: 論理結合子

$\neg\varphi$	φ でない
$\varphi \wedge \psi$	φ かつ ψ
$\varphi \vee \psi$	φ または ψ
$\varphi \to \psi$	φ ならば ψ
$\forall x\varphi$	全ての x について φ が成り立つ
$\exists x\varphi$	φ が成り立つような x が存在する

Table: 算術の語彙 $\mathcal{L}_{\mathsf{arith}}$

$\underline{0}$	定数記号 0
$\underline{1}$	定数記号 1
$x + y$	x と y の和
$x \times y$	x と y の積
$x < y$	x より y の方が大きい

数学を数学する: 数学のモデル化

- 数学的な主張の真偽を，数学を使って分析したい
- そのためには「数学的な主張」を数学的に明確に定義する必要がある
- そのために，論理結合子 $\wedge, \vee, \neg, \to, \forall, \exists$ や等号 $=$，変数記号 $x, y, z, \ldots$ などの記号を使って，数学的主張を形式化するという手法が使われる
- 「全数学」を形式化するのはひとまず置いておいて，まずは自然数 $\mathbb{N}$ に関する主張を書けるようにしてみる．そしてそれに必要となる語彙 $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を用意する

Table: 論理結合子

$\neg\varphi$	φ でない
$\varphi \wedge \psi$	φ かつ ψ
$\varphi \vee \psi$	φ または ψ
$\varphi \to \psi$	φ ならば ψ
$\forall x\varphi$	全ての x について φ が成り立つ
$\exists x\varphi$	φ が成り立つような x が存在する

Table: 算術の語彙 $\mathcal{L}_{\mathsf{arith}}$

$\underline{0}$	定数記号 0
$\underline{1}$	定数記号 1
$x + y$	x と y の和
$x \times y$	x と y の積
$x < y$	x より y の方が大きい

命題の形式化の例

p が素数であるという主張 $\mathrm{Prime}(p)$ を

$$\underline{1} < p \wedge \forall u \forall v[u < p \wedge v < p \to \neg(u \times v = p)]$$

の略として定義する．このとき，例えば次のような命題を論理式として書くことができる．

例: 素数は無限に存在する

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p)]$$

例: 双子素数予想 (未解決)

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p) \wedge \mathrm{Prime}(p + (\underline{1} + \underline{1}))]$$

例: Goldbach 予想 (未解決)

$$\forall n[\underline{1} + \underline{1} < n \wedge \exists m[n = (\underline{1} + \underline{1}) \times m] \to \exists p \exists q[\mathrm{Prime}(p) \wedge \mathrm{Prime}(q) \wedge n = p + q]]$$

命題の形式化の例

p が素数であるという主張 $\mathrm{Prime}(p)$ を

$$\underline{1} < p \wedge \forall u \forall v[u < p \wedge v < p \to \neg(u \times v = p)]$$

の略として定義する．このとき，例えば次のような命題を論理式として書くことができる．

例: 素数は無限に存在する

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p)]$$

例: 双子素数予想 (未解決)

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p) \wedge \mathrm{Prime}(p + (\underline{1} + \underline{1}))]$$

例: Goldbach 予想 (未解決)

$$\forall n[\underline{1} + \underline{1} < n \wedge \exists m[n = (\underline{1} + \underline{1}) \times m] \to \exists p \exists q[\mathrm{Prime}(p) \wedge \mathrm{Prime}(q) \wedge n = p + q]]$$

命題の形式化の例

p が素数であるという主張 $\mathrm{Prime}(p)$ を

$$\underline{1} < p \wedge \forall u \forall v[u < p \wedge v < p \to \neg(u \times v = p)]$$

の略として定義する．このとき，例えば次のような命題を論理式として書くことができる．

例: 素数は無限に存在する

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p)]$$

例: 双子素数予想 (未解決)

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p) \wedge \mathrm{Prime}(p + (\underline{1} + \underline{1}))]$$

例: Goldbach 予想 (未解決)

$$\forall n[\underline{1} + \underline{1} < n \wedge \exists m[n = (\underline{1} + \underline{1}) \times m] \to \exists p \exists q[\mathrm{Prime}(p) \wedge \mathrm{Prime}(q) \wedge n = p + q]]$$

命題の形式化の例

p が素数であるという主張 $\mathrm{Prime}(p)$ を

$$\underline{1} < p \wedge \forall u \forall v[u < p \wedge v < p \to \neg(u \times v = p)]$$

の略として定義する．このとき，例えば次のような命題を論理式として書くことができる．

例: 素数は無限に存在する

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p)]$$

例: 双子素数予想 (未解決)

$$\forall n \exists p[n < p \wedge \mathrm{Prime}(p) \wedge \mathrm{Prime}(p + (\underline{1} + \underline{1}))]$$

例: Goldbach 予想 (未解決)

$$\forall n[\underline{1} + \underline{1} < n \wedge \exists m[n = (\underline{1} + \underline{1}) \times m] \to \exists p \exists q[\mathrm{Prime}(p) \wedge \mathrm{Prime}(q) \wedge n = p + q]]$$

自然数のすべてを知りたい

定義

- $\mathrm{Prime}(p)$ における p のように，$\forall, \exists$ で指定されていない変数を自由変数という．
- 自由変数のない論理式を文という．
- 論理結合子と語彙 $\mathcal{L}$ (と等号 $=$) を用いて書かれた論理式を $\mathcal{L}$ 論理式といい，文であるような $\mathcal{L}$ 論理式を $\mathcal{L}$ 文という．

- 前頁にあるような $\mathcal{L}_{\mathrm{arith}}$ 文たちの真偽が知りたい
- しかも，個々の文の真偽ではなく，あらゆる文の真偽が全部わかるような，統一的な方法があれば便利である
- そのような方法はあるだろうか？
- 実は，そのような手続きは存在しない！

自然数のすべてを知りたい

定義

- $\mathrm{Prime}(p)$ における p のように，$\forall, \exists$ で指定されていない変数を自由変数という．
- 自由変数のない論理式を文という．
- 論理結合子と語彙 $\mathcal{L}$ (と等号 $=$) を用いて書かれた論理式を $\mathcal{L}$ 論理式といい，文であるような $\mathcal{L}$ 論理式を $\mathcal{L}$ 文という．

- 前頁にあるような $\mathcal{L}_{\mathsf{arith}}$ 文たちの真偽が知りたい
- しかも，個々の文の真偽ではなく，あらゆる文の真偽が全部わかるような，統一的な方法があれば便利である
- そのような方法はあるだろうか？
- 実は，そのような手続きは存在しない！

自然数のすべてを知りたい

定義

- $\mathrm{Prime}(p)$ における p のように，$\forall, \exists$ で指定されていない変数を自由変数という．
- 自由変数のない論理式を文という．
- 論理結合子と語彙 $\mathcal{L}$ (と等号 $=$) を用いて書かれた論理式を $\mathcal{L}$ 論理式といい，文であるような $\mathcal{L}$ 論理式を $\mathcal{L}$ 文という．

- 前頁にあるような $\mathcal{L}_{\mathsf{arith}}$ 文たちの真偽が知りたい
- しかも，個々の文の真偽ではなく，あらゆる文の真偽が全部わかるような，統一的な方法があれば便利である
- そのような方法はあるだろうか？
- 実は，そのような手続きは存在しない！

自然数のすべてを知りたい

定義

- $\mathrm{Prime}(p)$ における p のように，$\forall, \exists$ で指定されていない変数を自由変数という．
- 自由変数のない論理式を文という．
- 論理結合子と語彙 $\mathcal{L}$ (と等号 $=$) を用いて書かれた論理式を $\mathcal{L}$ 論理式といい，文であるような $\mathcal{L}$ 論理式を $\mathcal{L}$ 文という．

- 前頁にあるような $\mathcal{L}_{\mathsf{arith}}$ 文たちの真偽が知りたい
- しかも，個々の文の真偽ではなく，あらゆる文の真偽が全部わかるような，統一的な方法があれば便利である
- そのような方法はあるだろうか？
- 実は，そのような手続きは存在しない！

Gödel の不完全性定理

定理 (Gödel の第一不完全性定理, 1931)

自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, \times, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{arith}} \text{ 文} \,\}$$

は決定不能である．すなわち，$\mathcal{L}_{\mathsf{arith}}$ 文 φ が与えられたとき，$\varphi \in \mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, \times, <)$ かどうかを正しく判定するようなひとつの手続き (アルゴリズム) は存在しない．

言い換え:

- Gödel の定理の結論は，たとえこのように数学的命題の厳密に定義された本体 [形式体系のこと] を固定したとしても，数学的思考は本質的に創造的であり続けることから逃れられない，ということである (E. L. Post, 1944)
- 数学者の仕事はなくならなさそう

Gödel の不完全性定理

定理 (Gödel の第一不完全性定理, 1931)

自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, \times, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{arith}} \text{ 文} \,\}$$

は決定不能である．すなわち，$\mathcal{L}_{\mathsf{arith}}$ 文 φ が与えられたとき，$\varphi \in \mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, \times, <)$ かどうかを正しく判定するようなひとつの手続き (アルゴリズム) は存在しない．

言い換え:

- Gödel の定理の結論は，たとえこのように数学的命題の厳密に定義された本体 [形式体系のこと] を固定したとしても，数学的思考は本質的に創造的であり続けることから逃れられない，ということである (E. L. Post, 1944)
- 数学者の仕事はなくならなさそう

Gödel の不完全性定理

定理 (Gödel の第一不完全性定理, 1931)

自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, \times, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{arith}} \text{ 文} \,\}$$

は決定不能である．すなわち，$\mathcal{L}_{\mathsf{arith}}$ 文 φ が与えられたとき，$\varphi \in \mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, \times, <)$ かどうかを正しく判定するようなひとつの手続き (アルゴリズム) は存在しない．

言い換え:

- Gödel の定理の結論は，たとえこのように数学的命題の厳密に定義された本体 [形式体系のこと] を固定したとしても，数学的思考は本質的に創造的であり続けることから逃れられない，ということである (E. L. Post, 1944)
- 数学者の仕事はなくならなさそう

反省会

- 何がいけなかったのか
- 実は $+$ と $\times$ があると，任意の計算が表現できてしまう (表現定理) ことが知られている
- $\mathcal{L}_{\mathsf{arith}}$ 文の真偽の判定は (少なくとも)Turing 機械の停止問題より難しい
- 次善の策を考える
- 語彙として $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を使う代わりに，乗算記号 $\times$ を削除した $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を使うことにしたらどうだろうか？
- $\mathcal{L}_{\mathsf{Pres}}$ では，最初の例のような主張を (少なくとも直接的には) 書くことはできない
- しかし，$\times$ は $+$ の繰り返しで定義できるのだし，表現力は変わらないような気もする
- 実は，Presburger 算術

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{Pres}} \text{ 文} \,\}$$

  は決定可能になる！
- 証明には有限オートマトンを用いる

反省会

- 何がいけなかったのか
- 実は $+$ と $\times$ があると，任意の計算が表現できてしまう (表現定理) ことが知られている
- $\mathcal{L}_{\mathsf{arith}}$ 文の真偽の判定は (少なくとも)Turing 機械の停止問題より難しい
- 次善の策を考える
- 語彙として $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を使う代わりに，乗算記号 $\times$ を削除した $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を使うことにしたらどうだろうか？
- $\mathcal{L}_{\mathsf{Pres}}$ では，最初の例のような主張を (少なくとも直接的には) 書くことはできない
- しかし，$\times$ は $+$ の繰り返しで定義できるのだし，表現力は変わらないような気もする
- 実は，Presburger 算術

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{Pres}} \text{ 文} \,\}$$

  は決定可能になる！
- 証明には有限オートマトンを用いる

反省会

- 何がいけなかったのか
- 実は $+$ と $\times$ があると，任意の計算が表現できてしまう (表現定理) ことが知られている
- $\mathcal{L}_{\mathsf{arith}}$ 文の真偽の判定は (少なくとも)Turing 機械の停止問題より難しい
- 次善の策を考える
- 語彙として $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を使う代わりに，乗算記号 $\times$ を削除した $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を使うことにしたらどうだろうか？
- $\mathcal{L}_{\mathsf{Pres}}$ では，最初の例のような主張を (少なくとも直接的には) 書くことはできない
- しかし，$\times$ は $+$ の繰り返しで定義できるのだし，表現力は変わらないような気もする
- 実は，Presburger 算術

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{Pres}} \text{ 文} \,\}$$

  は決定可能になる！
- 証明には有限オートマトンを用いる

反省会

- 何がいけなかったのか
- 実は $+$ と $\times$ があると，任意の計算が表現できてしまう (表現定理) ことが知られている
- $\mathcal{L}_{\mathsf{arith}}$ 文の真偽の判定は (少なくとも)Turing 機械の停止問題より難しい
- 次善の策を考える
- 語彙として $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を使う代わりに，乗算記号 $\times$ を削除した $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を使うことにしたらどうだろうか？
- $\mathcal{L}_{\mathsf{Pres}}$ では，最初の例のような主張を (少なくとも直接的には) 書くことはできない
- しかし，$\times$ は $+$ の繰り返しで定義できるのだし，表現力は変わらないような気もする
- 実は，Presburger 算術

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{Pres}} \text{ 文} \,\}$$

  は決定可能になる！
- 証明には有限オートマトンを用いる

反省会

- 何がいけなかったのか
- 実は $+$ と $\times$ があると，任意の計算が表現できてしまう (表現定理) ことが知られている
- $\mathcal{L}_{\mathsf{arith}}$ 文の真偽の判定は (少なくとも)Turing 機械の停止問題より難しい
- 次善の策を考える
- 語彙として $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を使う代わりに，乗算記号 $\times$ を削除した $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を使うことにしたらどうだろうか？
- $\mathcal{L}_{\mathsf{Pres}}$ では，最初の例のような主張を (少なくとも直接的には) 書くことはできない
- しかし，$\times$ は $+$ の繰り返しで定義できるのだし，表現力は変わらないような気もする
- 実は，Presburger 算術

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{Pres}} \text{ 文} \,\}$$

  は決定可能になる！
- 証明には有限オートマトンを用いる

反省会

- 何がいけなかったのか
- 実は $+$ と $\times$ があると，任意の計算が表現できてしまう (表現定理) ことが知られている
- $\mathcal{L}_{\mathsf{arith}}$ 文の真偽の判定は (少なくとも)Turing 機械の停止問題より難しい
- 次善の策を考える
- 語彙として $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を使う代わりに，乗算記号 $\times$ を削除した $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を使うことにしたらどうだろうか？
- $\mathcal{L}_{\mathsf{Pres}}$ では，最初の例のような主張を (少なくとも直接的には) 書くことはできない
- しかし，$\times$ は $+$ の繰り返しで定義できるのだし，表現力は変わらないような気もする
- 実は，Presburger 算術

$$\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <) = \{\, \varphi \mid \varphi \text{ は } \mathbb{N} \text{ で正しい } \mathcal{L}_{\mathsf{Pres}} \text{ 文} \,\}$$

  は決定可能になる！
- 証明には有限オートマトンを用いる

目次

文字列

定義

- 有限集合 Σ に対し，Σ 上の文字列全体の集合を Σ^* で表す
- 例えば，$\Sigma = \{\mathsf{a}, \mathsf{b}\}$ のとき，

$$\Sigma^* = \{\varepsilon, \mathsf{a}, \mathsf{b}, \mathsf{aa}, \mathsf{ab}, \mathsf{ba}, \mathsf{bb}, \mathsf{aaa}, \dots\}$$

- ε は空文字列 (長さ 0 の文字列)

有限オートマトンの概要

有限オートマトンとは:

- 計算のモデルのひとつ
- 通常の計算機と異なり，計算能力が非常に制限されている (有限の情報しか記憶できない)

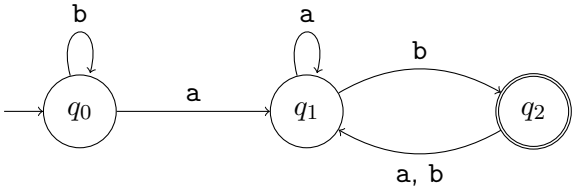


Figure: 有限オートマトンの例 M_0

有限オートマトンによる計算のしかた

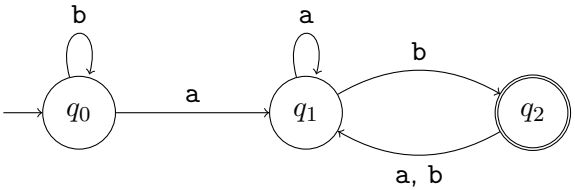
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列

babaab



有限オートマトンによる計算のしかた

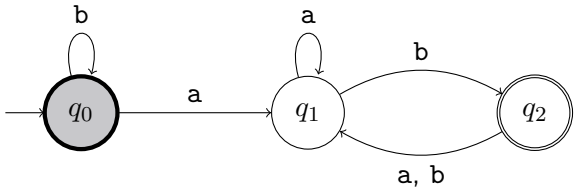
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列

babaab



有限オートマトンによる計算のしかた

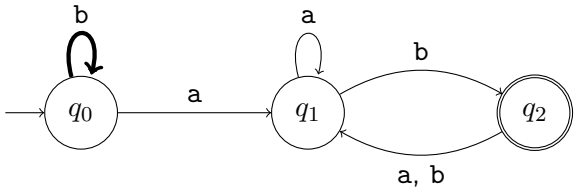
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列

babaab



有限オートマトンによる計算のしかた

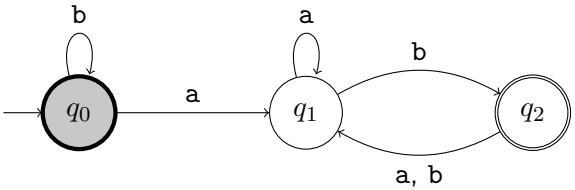
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列

babaab



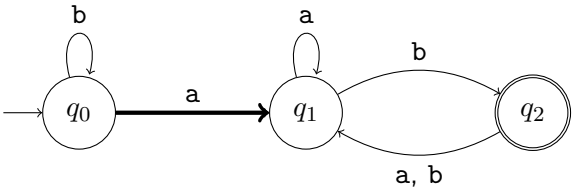
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，◯ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



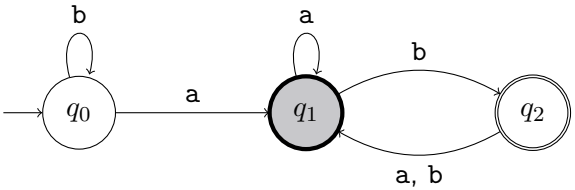
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



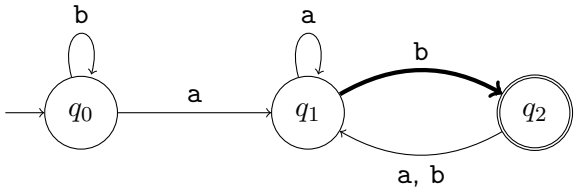
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



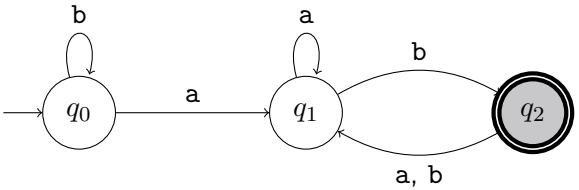
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，◯ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



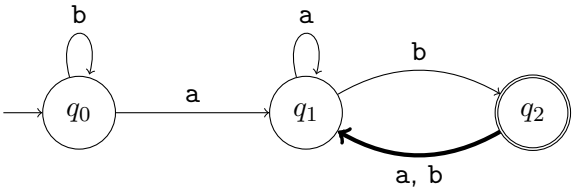
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



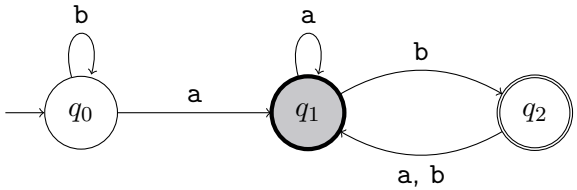
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



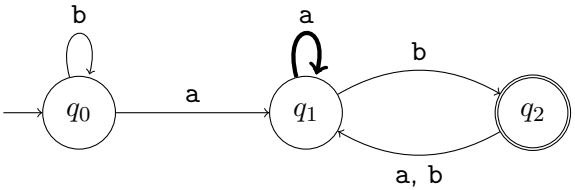
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



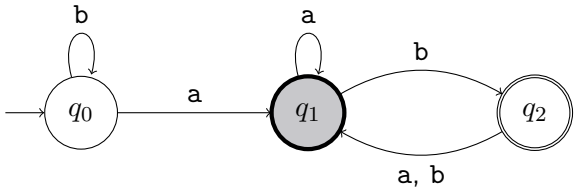
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，◯ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



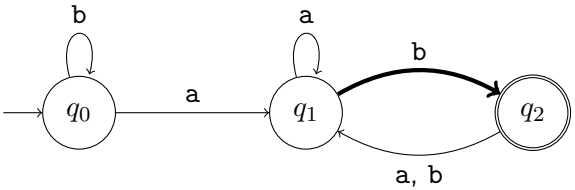
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab



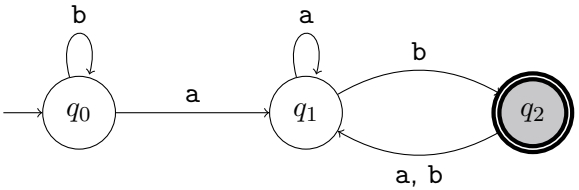
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，◯ の状態にいたら拒否となる

例えば，先程の M_0 に babaab を入力すると:

入力文字列
babaab
→ 受理



有限オートマトンによる計算のしかた

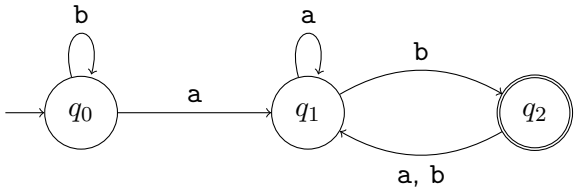
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列

abba



有限オートマトンによる計算のしかた

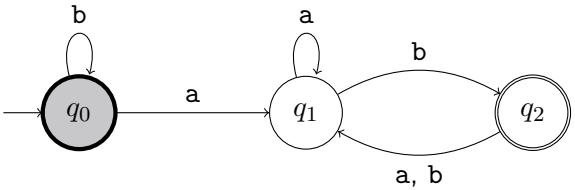
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列

abba



有限オートマトンによる計算のしかた

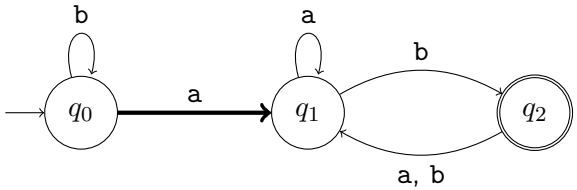
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，◯ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列

abba



有限オートマトンによる計算のしかた

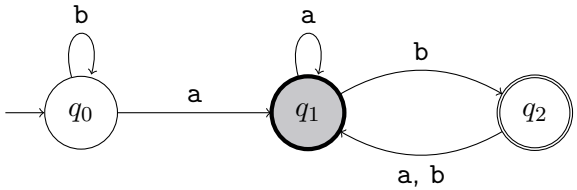
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列

abba



有限オートマトンによる計算のしかた

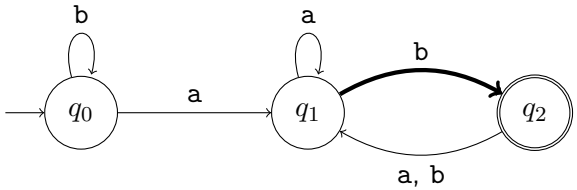
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列

abba



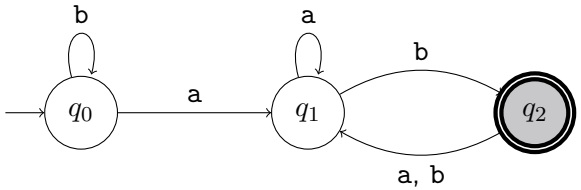
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列
abba



有限オートマトンによる計算のしかた

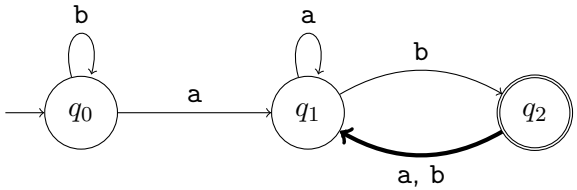
有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列

abba



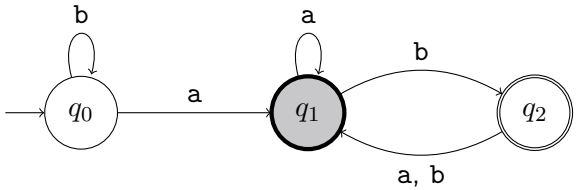
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列
abba



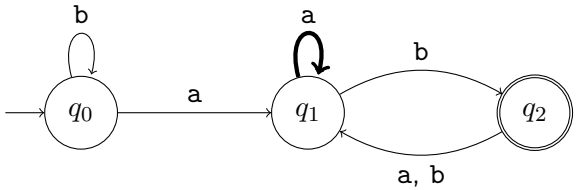
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，$\bigcirc$ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列
abba



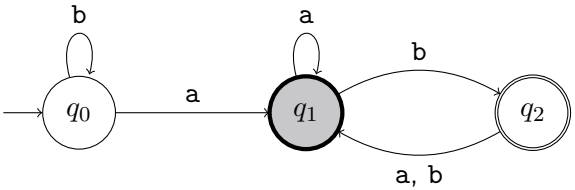
有限オートマトンによる計算のしかた

有限オートマトンの計算は次のように進む:

- 入力は文字列の形で与えられる
- $\rightarrow\bigcirc$ の状態からスタート
- 入力を 1 文字読み込むごとに，内部状態が変化する
- 入力を読み終えた時点で ◎ の状態にいたら計算結果は受理，◯ の状態にいたら拒否となる

例えば，先程の M_0 に abba を入力すると:

入力文字列
abba
→ 拒否



有限オートマトンの正式な定義

(決定性) 有限オートマトン (DFA と略す) M とは以下のデータからなる 5 個組 $(Q, \Sigma, \delta, q_0, F)$ である:

1. 状態の有限集合 Q
2. アルファベットと呼ばれる有限集合 Σ
3. 遷移関数 $\delta\colon Q \times \Sigma \to Q$
4. 開始状態 $q_0 \in Q$
5. 受理状態の集合 $F \subseteq Q$

- 文字列 $w = w_1 w_2 \cdots w_n \in \Sigma^*$ (ただし $w_i \in \Sigma$) が M に受理されるのは，以下を満たす状態の有限列 $r_0, r_1, \ldots, r_n \in Q$ が存在するときである:
  1. $r_0 = q_0$
  2. 各 $i = 0, 1, \ldots, n-1$ に対し $\delta(r_i, w_{i+1}) = r_{i+1}$
  3. $r_n \in F$
- DFA M が受理する Σ^* 上の文字列全体の集合を M が認識する言語と呼び，$L(M)$ で表す:

$$L(M) = \{\, w \in \Sigma^* \mid w \text{ は } M \text{ に受理される} \,\}.$$

有限オートマトンの正式な定義

(決定性) 有限オートマトン (DFA と略す) M とは以下のデータからなる 5 個組 $(Q, \Sigma, \delta, q_0, F)$ である:

1. 状態の有限集合 Q
2. アルファベットと呼ばれる有限集合 Σ
3. 遷移関数 $\delta\colon Q \times \Sigma \to Q$
4. 開始状態 $q_0 \in Q$
5. 受理状態の集合 $F \subseteq Q$

- 文字列 $w = w_1 w_2 \cdots w_n \in \Sigma^*$ (ただし $w_i \in \Sigma$) が M に受理されるのは，以下を満たす状態の有限列 $r_0, r_1, \ldots, r_n \in Q$ が存在するときである:
  1. $r_0 = q_0$
  2. 各 $i = 0, 1, \ldots, n-1$ に対し $\delta(r_i, w_{i+1}) = r_{i+1}$
  3. $r_n \in F$
- DFA M が受理する Σ^* 上の文字列全体の集合を M が認識する言語と呼び，$L(M)$ で表す:

$$L(M) = \{\, w \in \Sigma^* \mid w \text{ は } M \text{ に受理される} \,\}.$$

有限オートマトンの正式な定義

(決定性) 有限オートマトン (DFA と略す) M とは以下のデータからなる 5 個組 $(Q, \Sigma, \delta, q_0, F)$ である:

1. 状態の有限集合 Q
2. アルファベットと呼ばれる有限集合 Σ
3. 遷移関数 $\delta\colon Q \times \Sigma \to Q$
4. 開始状態 $q_0 \in Q$
5. 受理状態の集合 $F \subseteq Q$

- 文字列 $w = w_1 w_2 \cdots w_n \in \Sigma^*$ (ただし $w_i \in \Sigma$) が M に受理されるのは，以下を満たす状態の有限列 $r_0, r_1, \ldots, r_n \in Q$ が存在するときである:
  1. $r_0 = q_0$
  2. 各 $i = 0, 1, \ldots, n-1$ に対し $\delta(r_i, w_{i+1}) = r_{i+1}$
  3. $r_n \in F$
- DFA M が受理する Σ^* 上の文字列全体の集合を M が認識する言語と呼び，$L(M)$ で表す:

$$L(M) = \{\, w \in \Sigma^* \mid w \text{ は } M \text{ に受理される} \,\}.$$

DFA の例 1

DFA $M_1 = (Q, \Sigma, \delta, q_0, F)$ を

- $Q = \{q_0, q_1\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\delta(q_0, \mathtt{a}) = q_0, \delta(q_0, \mathtt{b}) = q_1$
- $\delta(q_1, \mathtt{a}) = q_0, \delta(q_1, \mathtt{b}) = q_1$
- $F = \{q_1\}$

と定義すると右図のようになる:
このとき

Figure: DFA M_1

$$L(M_1) = \{\, w \in \Sigma^* \mid w \text{ は } \mathtt{b} \text{ で終わる文字列} \,\}.$$

DFA の例 1

DFA $M_1 = (Q, \Sigma, \delta, q_0, F)$ を

- $Q = \{q_0, q_1\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\delta(q_0, \mathtt{a}) = q_0, \delta(q_0, \mathtt{b}) = q_1$
- $\delta(q_1, \mathtt{a}) = q_0, \delta(q_1, \mathtt{b}) = q_1$
- $F = \{q_1\}$

と定義すると右図のようになる:

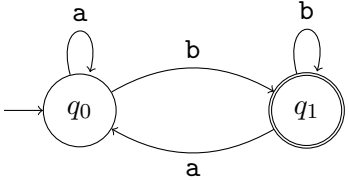


Figure: DFA M_1

このとき

$$L(M_1) = \{\, w \in \Sigma^* \mid w \text{ は } \mathtt{b} \text{ で終わる文字列} \,\}.$$

DFA の例 1

DFA $M_1 = (Q, \Sigma, \delta, q_0, F)$ を

- $Q = \{q_0, q_1\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\delta(q_0, \mathtt{a}) = q_0, \delta(q_0, \mathtt{b}) = q_1$
- $\delta(q_1, \mathtt{a}) = q_0, \delta(q_1, \mathtt{b}) = q_1$
- $F = \{q_1\}$

と定義すると右図のようになる:
このとき

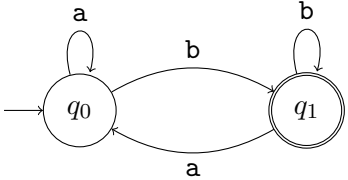


Figure: DFA M_1

$$L(M_1) = \{\, w \in \Sigma^* \mid w \text{ は } \mathtt{b} \text{ で終わる文字列} \,\}.$$

DFA の例 2

$$L(M_2) = \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ に含まれる } \mathtt{a} \text{ の個数は } 3 \text{ の倍数ではない} \,\}$$

となるような DFA M_2 を作る.

DFA $M_2 = (Q, \Sigma, \delta, q_0, F)$ を

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\delta(q_0, \mathtt{a}) = q_1, \delta(q_0, \mathtt{b}) = q_0$
- $\delta(q_1, \mathtt{a}) = q_2, \delta(q_1, \mathtt{b}) = q_1$
- $\delta(q_2, \mathtt{a}) = q_0, \delta(q_2, \mathtt{b}) = q_2$
- $F = \{q_1, q_2\}$

と定義すれば右図のようになる:

Figure: DFA M_2

DFA の例 2

$$L(M_2) = \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ に含まれる } \mathtt{a} \text{ の個数は } 3 \text{ の倍数ではない} \,\}$$

となるような DFA M_2 を作る.

DFA $M_2 = (Q, \Sigma, \delta, q_0, F)$ を

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\delta(q_0, \mathtt{a}) = q_1, \delta(q_0, \mathtt{b}) = q_0$
- $\delta(q_1, \mathtt{a}) = q_2, \delta(q_1, \mathtt{b}) = q_1$
- $\delta(q_2, \mathtt{a}) = q_0, \delta(q_2, \mathtt{b}) = q_2$
- $F = \{q_1, q_2\}$

と定義すれば右図のようになる:

Figure: DFA M_2

DFA の例 2

$$L(M_2) = \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ に含まれる } \mathtt{a} \text{ の個数は } 3 \text{ の倍数ではない} \,\}$$

となるような DFA M_2 を作る.

DFA $M_2 = (Q, \Sigma, \delta, q_0, F)$ を

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\delta(q_0, \mathtt{a}) = q_1, \delta(q_0, \mathtt{b}) = q_0$
- $\delta(q_1, \mathtt{a}) = q_2, \delta(q_1, \mathtt{b}) = q_1$
- $\delta(q_2, \mathtt{a}) = q_0, \delta(q_2, \mathtt{b}) = q_2$
- $F = \{q_1, q_2\}$

と定義すれば右図のようになる:

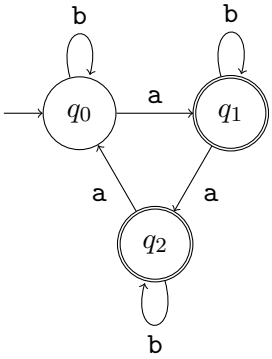


Figure: DFA M_2

正規言語の閉包性

定義

- アルファベット Σ に対し，部分集合 $L \subseteq \Sigma^*$ を Σ 上の言語と呼ぶ
- 言語 L がある DFA M によって $L = L(M)$ と書けるとき，L を正規言語と呼ぶ

正規言語全体は種々の演算で閉じている:

定理

正規言語 $L_1, L_2 \subseteq \Sigma^*$ に対し，以下は全て正規言語である:

補集合 $\Sigma^* \setminus L_1$

和集合 $L_1 \cup L_2$

共通部分 $L_1 \cap L_2$

正規言語の閉包性

定義

- アルファベット Σ に対し，部分集合 $L \subseteq \Sigma^*$ を Σ 上の言語と呼ぶ
- 言語 L がある DFA M によって $L = L(M)$ と書けるとき，L を正規言語と呼ぶ

正規言語全体は種々の演算で閉じている:

定理

正規言語 $L_1, L_2 \subseteq \Sigma^*$ に対し，以下は全て正規言語である:

補集合 $\Sigma^* \setminus L_1$

和集合 $L_1 \cup L_2$

共通部分 $L_1 \cap L_2$

正規言語の閉包性: 証明 (1/3)

補集合について:

証明.

$M = (Q, \Sigma, \delta, q_0, F)$ を，$L_1 = L(M)$ となる DFA とする．
このとき，DFA $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ が認識する言語は $L(M') = \Sigma^* \setminus L_1$ となる． □

例えば先程の M_1 に対して M_1' を作ると

となり，$L(M_1') = \Sigma^* \setminus L(M_1) = \{ w \in \Sigma^* \mid w$ は a で終わるか，または $w = \varepsilon \}$ となる

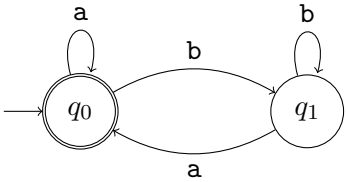
正規言語の閉包性: 証明 (1/3)

補集合について:

証明.

$M = (Q, \Sigma, \delta, q_0, F)$ を，$L_1 = L(M)$ となる DFA とする．
このとき，DFA $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ が認識する言語は $L(M') = \Sigma^* \setminus L_1$ となる． □

例えば先程の M_1 に対して M_1' を作ると



となり，$L(M_1') = \Sigma^* \setminus L(M_1) = \{\, w \in \Sigma^* \mid w$ は a で終わるか，または $w = \varepsilon \,\}$ となる

正規言語の閉包性: 証明 (2/3)

和集合について:

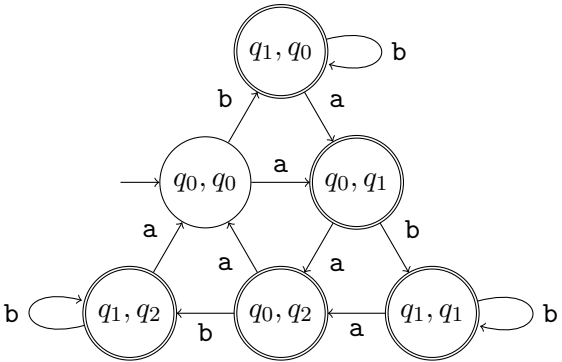
証明.

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ をそれぞれ $L_1 = L(M_1), L_2 = L(M_2)$ となる DFA とする.
このとき，DFA $M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), (F_1 \times Q_2) \cup (Q_1 \times F_2))$ (ただし，$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$) が認識する言語は $L(M) = L_1 \cup L_2$ となる. □

- 共通部分 $L_1 \cap L_2$ も同様にして証明できる (受理状態を $F_1 \times F_2$ とすればよい)

正規言語の閉包性: 証明 (3/3)

例えば先程の M_1, M_2 に対して M を作ると



となり,
$L(M) = L(M_1) \cup L(M_2) = \{ w \in \Sigma^* \mid w$ は b で終わるか a の個数が 3 の倍数でない $\}$ となる

正規言語の閉包性: 連結

定理

正規言語は連結演算についても閉じている．すなわち，$L_1, L_2 \subseteq \Sigma^*$ が正規言語ならば

$$L_1 \circ L_2 = \{\, uv \mid u \in L_1, v \in L_2 \,\}$$

も正規言語である．

- 先程と同様にして証明しようとしてみる
- $L_1 = L(M_1), L_2 = L(M_2)$ として，$L_1 \circ L_2 = L(M)$ なる M を作りたい
- $uv \in L_1 \circ L_2$ に対して，M_1 を u に対して動作させ，受理したら M_2 を v に対して動作させるようにしたい
- ところが，$w \in L_1 \circ L_2$ が与えられたとき，どこで区切ればよいのかわからない！
- 「非決定性」と呼ばれる概念を導入する

正規言語の閉包性: 連結

定理

正規言語は連結演算についても閉じている．すなわち，$L_1, L_2 \subseteq \Sigma^*$ が正規言語ならば

$$L_1 \circ L_2 = \{\, uv \mid u \in L_1, v \in L_2 \,\}$$

も正規言語である．

- 先程と同様にして証明しようとしてみる
- $L_1 = L(M_1), L_2 = L(M_2)$ として，$L_1 \circ L_2 = L(M)$ なる M を作りたい
- $uv \in L_1 \circ L_2$ に対して，M_1 を u に対して動作させ，受理したら M_2 を v に対して動作させるようにしたい
- ところが，$w \in L_1 \circ L_2$ が与えられたとき，どこで区切ればよいのかわからない！
- 「非決定性」と呼ばれる概念を導入する

正規言語の閉包性: 連結

定理

正規言語は連結演算についても閉じている．すなわち，$L_1, L_2 \subseteq \Sigma^*$ が正規言語ならば

$$L_1 \circ L_2 = \{\, uv \mid u \in L_1, v \in L_2 \,\}$$

も正規言語である．

- 先程と同様にして証明しようとしてみる
- $L_1 = L(M_1), L_2 = L(M_2)$ として，$L_1 \circ L_2 = L(M)$ なる M を作りたい
- $uv \in L_1 \circ L_2$ に対して，M_1 を u に対して動作させ，受理したら M_2 を v に対して動作させるようにしたい
- ところが，$w \in L_1 \circ L_2$ が与えられたとき，どこで区切ればよいのかわからない！
- 「非決定性」と呼ばれる概念を導入する

正規言語の閉包性: 連結

定理

正規言語は連結演算についても閉じている．すなわち，$L_1, L_2 \subseteq \Sigma^*$ が正規言語ならば

$$L_1 \circ L_2 = \{\, uv \mid u \in L_1, v \in L_2 \,\}$$

も正規言語である．

- 先程と同様にして証明しようとしてみる
- $L_1 = L(M_1), L_2 = L(M_2)$ として，$L_1 \circ L_2 = L(M)$ なる M を作りたい
- $uv \in L_1 \circ L_2$ に対して，M_1 を u に対して動作させ，受理したら M_2 を v に対して動作させるようにしたい
- ところが，$w \in L_1 \circ L_2$ が与えられたとき，どこで区切ればよいのかわからない！
- 「非決定性」と呼ばれる概念を導入する

非決定性有限オートマトンの正式な定義

非決定性有限オートマトン (NFA と略す) N とは以下のデータからなる 5 個組 $(Q, \Sigma, \delta, q_0, F)$ である:

1. 状態の有限集合 Q
2. アルファベットと呼ばれる有限集合 Σ
3. 遷移関数 $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$
4. 開始状態 $q_0 \in Q$
5. 受理状態の集合 $F \subseteq Q$

- 文字列 $w \in \Sigma^*$ が N に受理されるのは，w のある分割 $w = w_1 w_2 \cdots w_n$ (ただし $w_i \in \Sigma \cup \{\varepsilon\}$) と，以下を満たす状態の有限列 $r_0, r_1, \ldots, r_n \in Q$ が存在するときである:
  1. $r_0 = q_0$
  2. 各 $i = 0, 1, \ldots, n-1$ に対し $\delta(r_i, w_{i+1}) \ni r_{i+1}$
  3. $r_n \in F$

非決定性有限オートマトンの正式な定義

非決定性有限オートマトン (NFA と略す) N とは以下のデータからなる 5 個組 $(Q, \Sigma, \delta, q_0, F)$ である:

1. 状態の有限集合 Q
2. アルファベットと呼ばれる有限集合 Σ
3. 遷移関数 $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$
4. 開始状態 $q_0 \in Q$
5. 受理状態の集合 $F \subseteq Q$

- 文字列 $w \in \Sigma^*$ が N に受理されるのは，w のある分割 $w = w_1 w_2 \cdots w_n$ (ただし $w_i \in \Sigma \cup \{\varepsilon\}$) と，以下を満たす状態の有限列 $r_0, r_1, \ldots, r_n \in Q$ が存在するときである:
  1. $r_0 = q_0$
  2. 各 $i = 0, 1, \ldots, n-1$ に対し $\delta(r_i, w_{i+1}) \ni r_{i+1}$
  3. $r_n \in F$

非決定性有限オートマトンによる計算

- NFA は，入力を読み終えた時点で受理状態にいることができるような，うまい経路を選ぶことができるとき受理する
- 別の見方: 入力を 1 文字読むごとに，遷移可能な全ての可能性を同時にシミュレートして，入力を読み終えたときにどれかひとつでも受理状態にいれば受理する

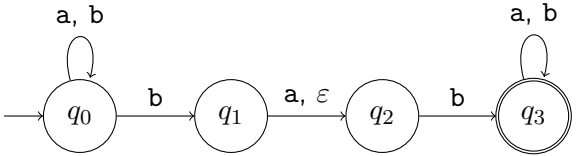


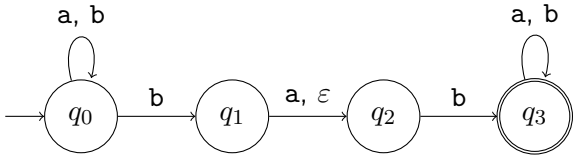
Figure: NFA の例 N_1

非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列

ababba

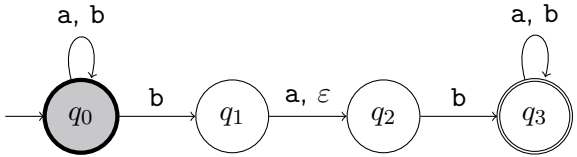


非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列

ababba

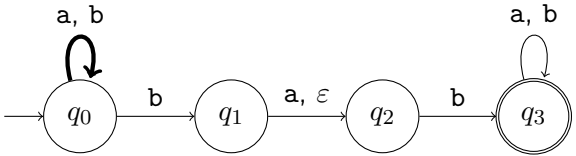


非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列

ababba

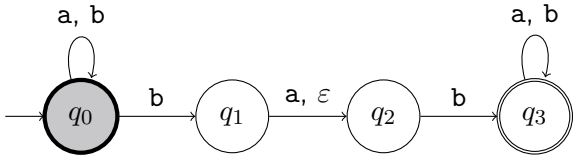


非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列

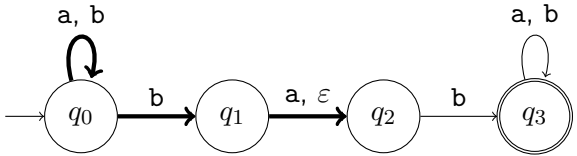
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

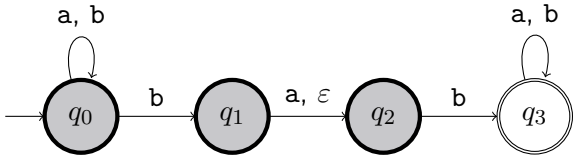
入力文字列
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

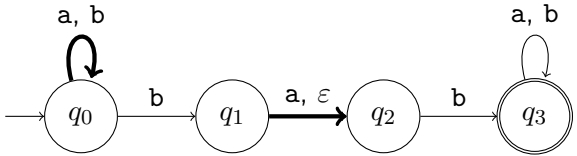
入力文字列
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

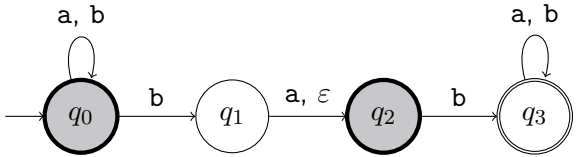
入力文字列
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

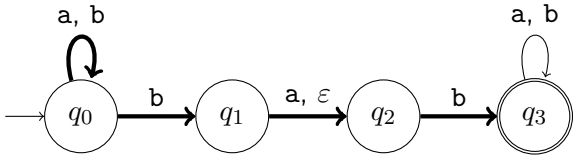
入力文字列
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

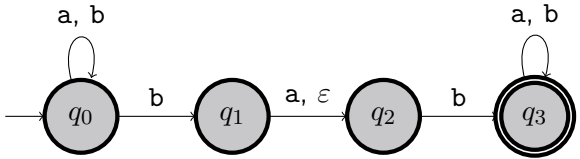
入力文字列
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列
ababba

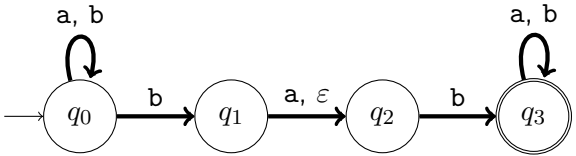


非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列

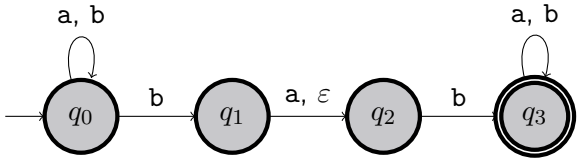
ababb**a**



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

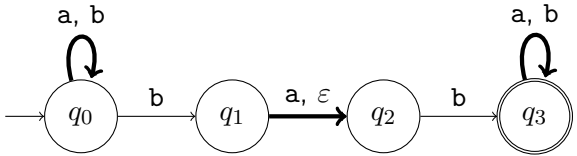
入力文字列
ababb**a**



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

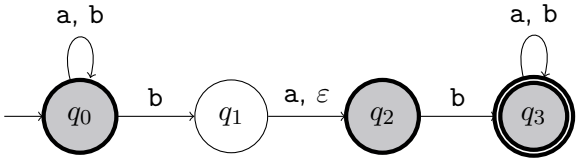
入力文字列
ababba



非決定性有限オートマトンによる計算の例

例えば，前頁の NFA N_1 に ababba を入力すると:

入力文字列
ababba
$\rightarrow$ 受理



正規言語の連結を認識する NFA

NFA を使えば連結が書ける:

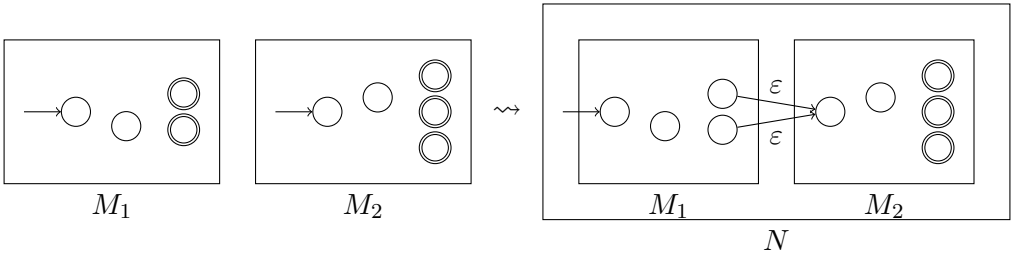


Figure: $L(M_1) \circ L(M_2)$ を認識する NFA N

- 正規言語が連結で閉じていることの証明を完成させるには，NFA が認識する言語が DFA にも認識できることを示せばよい

正規言語の連結を認識する NFA

NFA を使えば連結が書ける:

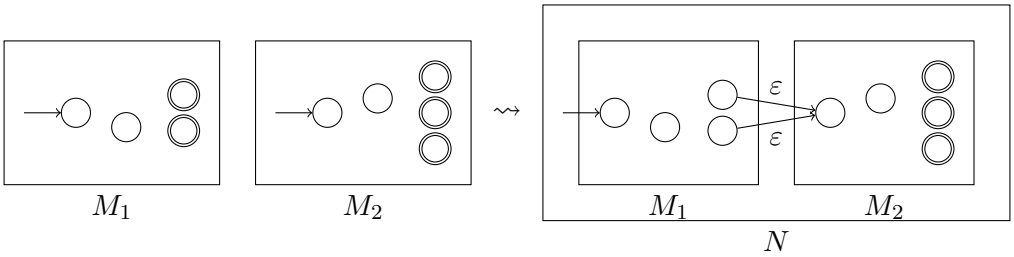


Figure: $L(M_1) \circ L(M_2)$ を認識する NFA N

- 正規言語が連結で閉じていることの証明を完成させるには，NFA が認識する言語が DFA にも認識できることを示せばよい

NFA を DFA に変換する: サブセット構成

定理

任意の NFA N に対し，$L(N) = L(M)$ となるような DFA M が存在する．

証明.

$N = (Q, \Sigma, \delta, q_0, F)$ とし，$R \subseteq Q$ に対し

$$E(R) = \{\, r \mid r \text{ はある } R \text{ の元から } 0 \text{ 回以上の } \varepsilon \text{ 遷移で到達できる} \,\}$$

と定める．このとき $M = (Q', \Sigma, \delta', q_0', F')$ を以下のように構成すればよい:

- $Q' = \mathcal{P}(Q)$
- $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$
- $q_0' = E(\{q_0\})$
- $F' = \{\, R \in Q' \mid R \cap F \neq \emptyset \,\}$

□

NFA を DFA に変換する: サブセット構成

定理

任意の NFA N に対し，$L(N) = L(M)$ となるような DFA M が存在する．

証明.

$N = (Q, \Sigma, \delta, q_0, F)$ とし，$R \subseteq Q$ に対し

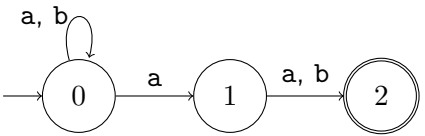
$$E(R) = \{\, r \mid r \text{ はある } R \text{ の元から } 0 \text{ 回以上の } \varepsilon \text{ 遷移で到達できる} \,\}$$

と定める．このとき $M = (Q', \Sigma, \delta', q_0', F')$ を以下のように構成すればよい:

- $Q' = \mathcal{P}(Q)$
- $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$
- $q_0' = E(\{q_0\})$
- $F' = \{\, R \in Q' \mid R \cap F \neq \emptyset \,\}$

□

サブセット構成の例

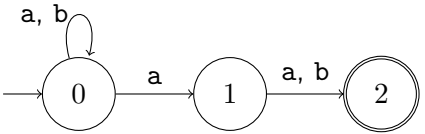
例えば NFA N_2 を 

とすると

$L(N_2) = \{\, w \in \{\mathsf{a}, \mathsf{b}\}^* \mid w$ は後ろから 2 文字目が $\mathsf{a}\,\}$ であり，これを DFA に変換すると

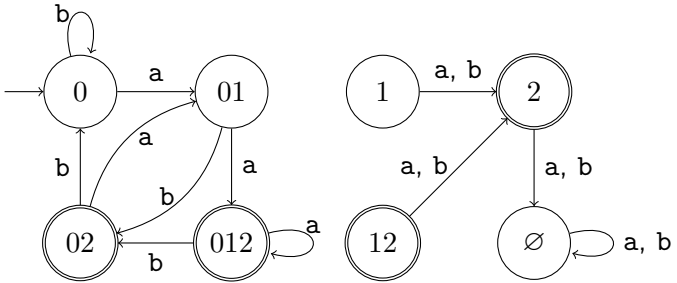
となる (実際には右半分は不要)

サブセット構成の例

例えば NFA N_2 を 

とすると

$L(N_2) = \{\, w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w$ は後ろから 2 文字目が $\mathtt{a}\,\}$ であり，これを DFA に変換すると



となる (実際には右半分は不要)

目次

何がやりたかったのか (復習)

- 算術の語彙 $\mathcal{L}_{\mathsf{arith}} = \{\underline{0}, \underline{1}, +, \times, <\}$ を用いて書ける $\mathcal{L}_{\mathsf{arith}}$ 文の真偽を教えてくれるような機械は存在しない (不完全性定理)
- 強すぎる $\mathcal{L}_{\mathsf{arith}}$ の表現力を下げるために，乗算記号 $\times$ を取り除いて新たな語彙 $\mathcal{L}_{\mathsf{Pres}} = \{\underline{0}, \underline{1}, +, <\}$ を作った
- 自然数の $\mathcal{L}_{\mathsf{Pres}}$ 理論 $\mathrm{Th}(\mathbb{N}; \underline{0}, \underline{1}, +, <)$ は決定可能だろうか？ $\rightarrow$ Yes!
- その証明に有限オートマトンが必要なので，導入をした $\leftarrow$ いまここ

証明

- φ を与えられた $\mathcal{L}_{\mathsf{Pres}}$ 文とする
- φ の真偽を判定したい
- 一般に φ は複雑な形をしているため，まずは φ を簡単な形に変形・分解していく

証明

- φ を与えられた $\mathcal{L}_{\mathsf{Pres}}$ 文とする
- φ の真偽を判定したい
- 一般に φ は複雑な形をしているため，まずは φ を簡単な形に変形・分解していく

証明: ステップ 1: $<$ の除去

- $<$ は加算記号 $+$ があれば定義できるのであってもなくても同じ
- 実際,

$$x < y \iff \exists z[y = x + z + \underline{1}]$$

なので，全ての $\mathcal{L}_{\mathsf{Pres}}$ 論理式は $\{\underline{0}, \underline{1}, +\}$ のみを使った論理式に (真偽を変化させずに) 書き換えられる

- よって，φ はいくつかの等式を論理結合子で組み合わせたものであると仮定してよい

証明: ステップ 1: $<$ の除去

- $<$ は加算記号 $+$ があれば定義できるのであってもなくても同じ
- 実際,

$$x < y \iff \exists z[y = x + z + \underline{1}]$$

なので，全ての $\mathcal{L}_{\mathsf{Pres}}$ 論理式は $\{\underline{0}, \underline{1}, +\}$ のみを使った論理式に (真偽を変化させずに) 書き換えられる

- よって，φ はいくつかの等式を論理結合子で組み合わせたものであると仮定してよい

証明: ステップ 2: 等式の分解

- φ に現れる等式は全て $\alpha + \beta = \gamma$ という形に書き直せる (ここで α, β, γ は変数または $\underline{0}, \underline{1}$)
- 例えば，等式

$$((x + y) + \underline{1}) + x = z + z$$

は新しい変数 u, v, w を使って

$$\exists u \exists v \exists w [x + y = u \wedge u + \underline{1} = v \wedge v + x = w \wedge z + z = w]$$

と書き換えられる

- 一般の場合も同様 (等式に現れる $+$ の個数から 1 を引いた数だけ新しい変数を用意する)

証明: ステップ 3: 冠頭標準形への変形

- 任意の論理式は $\forall, \exists$ を手前にくくり出すことで同値な冠頭標準形に変形できる
- 例えば

$$\exists x[\exists y[x = y + y] \wedge \forall y[\neg(x = y + \underline{1})]]$$

という論理式は変数名を付け替えることで

$$\exists x \exists y \forall z[x = y + y \wedge \neg(x = z + \underline{1})]$$

と同値であることがわかる

- よって，φ に含まれる変数を $x_1, x_2, \ldots, x_n$ とすると，φ は $\forall, \exists$ を含まないある論理式 $\psi(x_1, x_2, \ldots, x_n)$ について

$$\mathrm{Q}_1 x_1 \mathrm{Q}_2 x_2 \cdots \mathrm{Q}_\mathrm{n} x_n[\psi(x_1, x_2, \ldots, x_n)]$$

という形をしているとしてよい (ここで各 $\mathrm{Q_i}$ は $\forall$ または $\exists$)

証明: ステップ 3: 冠頭標準形への変形

- 任意の論理式は $\forall, \exists$ を手前にくくり出すことで同値な冠頭標準形に変形できる
- 例えば

$$\exists x[\exists y[x = y + y] \wedge \forall y[\neg(x = y + \underline{1})]]$$

という論理式は変数名を付け替えることで

$$\exists x \exists y \forall z[x = y + y \wedge \neg(x = z + \underline{1})]$$

と同値であることがわかる

- よって，φ に含まれる変数を $x_1, x_2, \ldots, x_n$ とすると，φ は $\forall, \exists$ を含まないある論理式 $\psi(x_1, x_2, \ldots, x_n)$ について

$$\mathsf{Q}_1 x_1 \mathsf{Q}_2 x_2 \cdots \mathsf{Q}_\mathsf{n} x_n[\psi(x_1, x_2, \ldots, x_n)]$$

という形をしているとしてよい (ここで各 Q_i は $\forall$ または $\exists$)

証明: 証明の方針

- φ の部分論理式 ψ_i $(i = 0, 1, \ldots, n)$ を

$$
\begin{aligned}
\psi_0 &\equiv \mathsf{Q_1} x_1 \mathsf{Q_2} x_2 \cdots \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)] \ (\equiv \varphi), \\
\psi_1(x_1) &\equiv \mathsf{Q_2} x_2 \cdots \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)], \\
&\cdots, \\
\psi_i(x_1, \ldots, x_i) &\equiv \mathsf{Q_{i+1}} x_{i+1} \cdots \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)], \\
&\cdots, \\
\psi_{n-1}(x_1, \ldots, x_{n-1}) &\equiv \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)], \\
\psi_n(x_1, \ldots, x_{n-1}, x_n) &\equiv \psi(x_1, \ldots, x_n)
\end{aligned}
$$

  と定める

- 各 $i = n, n-1, \ldots, 1, 0$ に対し，$\psi_i(a_1, \ldots, a_i)$ が真となるような値 $(a_1, \ldots, a_i) \in \mathbb{N}^i$ を受理する有限オートマトン M_i を構成する
- M_0 が ε を受理するかどうかで φ の真偽を判定できる！

証明: 証明の方針

- φ の部分論理式 ψ_i $(i = 0, 1, \ldots, n)$ を

$$
\begin{aligned}
\psi_0 &\equiv \mathsf{Q_1} x_1 \mathsf{Q_2} x_2 \cdots \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)] \ (\equiv \varphi), \\
\psi_1(x_1) &\equiv \mathsf{Q_2} x_2 \cdots \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)], \\
&\cdots, \\
\psi_i(x_1, \ldots, x_i) &\equiv \mathsf{Q_{i+1}} x_{i+1} \cdots \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)], \\
&\cdots, \\
\psi_{n-1}(x_1, \ldots, x_{n-1}) &\equiv \mathsf{Q_n} x_n [\psi(x_1, \ldots, x_n)], \\
\psi_n(x_1, \ldots, x_{n-1}, x_n) &\equiv \psi(x_1, \ldots, x_n)
\end{aligned}
$$

  と定める
- 各 $i = n, n-1, \ldots, 1, 0$ に対し，$\psi_i(a_1, \ldots, a_i)$ が真となるような値 $(a_1, \ldots, a_i) \in \mathbb{N}^i$ を受理する有限オートマトン M_i を構成する
- M_0 が ε を受理するかどうかで φ の真偽を判定できる！

証明: 有限オートマトンに数値を入力する方法

- M_i の入力アルファベットは

$$\Sigma_i = \left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 1 \end{bmatrix}, \cdots \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \right\}$$

  という 2^i 個の元からなる集合とする (ただし，$\Sigma_0 = \emptyset, \Sigma_0^* = \{\varepsilon\}$ とする)
- $\psi_i(a_1, \ldots, a_i)$ の真偽を知りたければ，各 $a_1, \ldots, a_i$ の 2 進展開のビットを下から順に M_i に入力する
- 例えば，$\psi_3(3, 4, 13)$ なら $3 = (11)_2, 4 = (100)_2, 13 = (1101)_2$ だから M_3 に

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

  を入力すればよい

証明: 有限オートマトンに数値を入力する方法

- M_i の入力アルファベットは

$$\Sigma_i = \left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 1 \end{bmatrix}, \dots \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \right\}$$

  という 2^i 個の元からなる集合とする (ただし，$\Sigma_0 = \emptyset, \Sigma_0^* = \{\varepsilon\}$ とする)
- $\psi_i(a_1, \dots, a_i)$ の真偽を知りたければ，各 $a_1, \dots, a_i$ の 2 進展開のビットを下から順に M_i に入力する
- 例えば，$\psi_3(3, 4, 13)$ なら $3 = (11)_2, 4 = (100)_2, 13 = (1101)_2$ だから M_3 に

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

  を入力すればよい

証明: DFA で足し算をする (1/2)

- まず $\psi_n(a_1, \ldots, a_n)$ の真偽を判定する M_n を作る
- 正規言語の閉包性から，
  $\alpha + \beta = \gamma$ の形の論理式の真偽が判定できれば十分
- α, β, γ が何かによって場合分けする:
  - $x_i + x_j = x_k$ $(i \neq j \neq k \neq i)$ のときが最も重要
    例えば $i = 1, j = 2, k = 3$ のときは右図のようになる

Figure: $a_1 + a_2 = a_3$ を検査する DFA

証明: DFA で足し算をする (1/2)

- まず $\psi_n(a_1, \ldots, a_n)$ の真偽を判定する M_n を作る
- 正規言語の閉包性から,
  $\alpha + \beta = \gamma$ の形の論理式の真偽が判定できれば十分
- α, β, γ が何かによって場合分けする:
  - $x_i + x_j = x_k$ $(i \neq j \neq k \neq i)$ のときが最も重要
    例えば $i = 1, j = 2, k = 3$ のときは右図のようになる

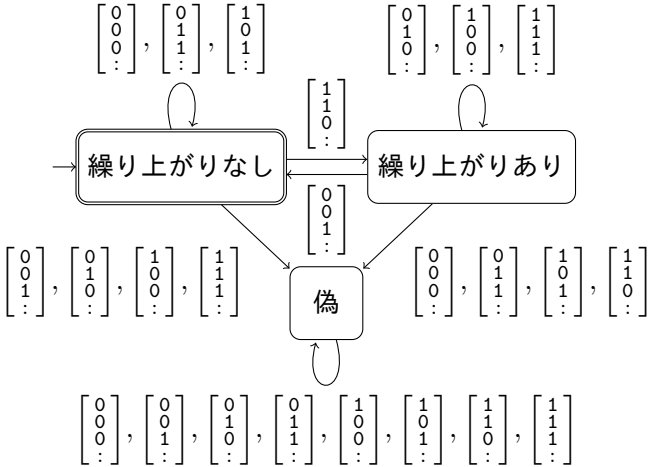


Figure: $a_1 + a_2 = a_3$ を検査する DFA

証明: DFA で足し算をする (2/2)

- α, β, γ が何かによって場合分けする (続き):
  - α, β, γ が全て $\underline{0}$ または $\underline{1}$ のとき: 真偽によって全部受理 or 全部拒否
  - $x_i + \underline{0} = x_i$ のとき: 全部受理する
  - $x_i + x_i = \underline{1}$ または $x_i + \underline{1} = x_i$ のとき: 全部拒否する
  - $\underline{0} + \underline{0} = x_i$ または $x_i + x_i = \underline{0}$ または $x_i + x_i = x_i$ または $x_i + x_j = x_j$ $(i \neq j)$ のとき: $a_i = 0$ かどうかを検査
  - $x_i + \underline{0} = x_j$ $(i \neq j)$ のとき: $a_i = a_j$ かどうかを検査
  - $x_i + x_j = \underline{0}$ $(i \neq j)$ のとき: $a_i = a_j = 0$ かどうかを検査
  - $\underline{0} + \underline{1} = x_i$ のとき: $a_i = 1$ かどうかを検査
  - $\underline{1} + \underline{1} = x_i$ のとき: $a_i = 2$ かどうかを検査
  - $x_i + x_i = x_j$ $(i \neq j)$ のとき: 前頁と同様のアイデアで計算する
  - $x_i + x_j = \underline{1}$ $(i \neq j)$ または $x_i + \underline{1} = x_j$ $(i \neq j)$ のとき: 同様に計算する
- 正規言語の閉包性を利用して DFA M_n を作る

証明: ∃ の処理

- 次に，M_n をもとに M_{n-1} を作る
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \exists x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合を考える
- ψ_{n-1} の真偽を判定するために，x_n の値を非決定的に「推測」する
- DFA M_n の各遷移の n 段目を「潰して」NFA N_{n-1} を作る
- 例えば $\psi_3(x_1, x_2, x_3) \equiv x_1 + x_3 = x_2$, $\psi_2(x_1, x_2) \equiv \exists x_3[x_1 + x_3 = x_2]$ のときは右図のようになる

Figure: $a_1 + a_3 = a_2$ を検査する DFA M_3

証明: ∃ の処理

- 次に，M_n をもとに M_{n-1} を作る
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \exists x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合を考える
- ψ_{n-1} の真偽を判定するために，x_n の値を非決定的に「推測」する
- DFA M_n の各遷移の n 段目を「潰して」NFA N_{n-1} を作る
- 例えば
  $\psi_3(x_1, x_2, x_3) \equiv x_1 + x_3 = x_2$,
  $\psi_2(x_1, x_2) \equiv \exists x_3[x_1 + x_3 = x_2]$ のときは右図のようになる

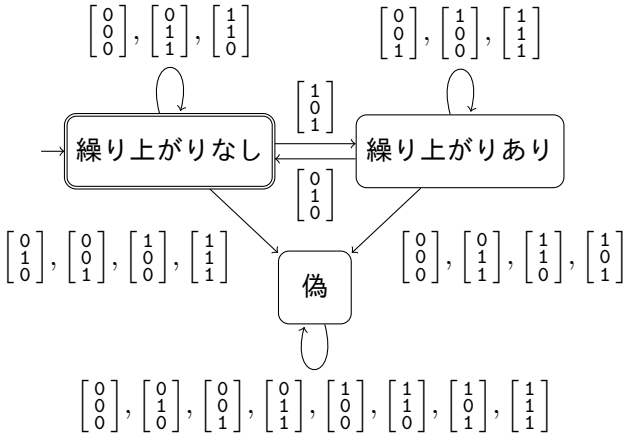


Figure: $a_1 + a_3 = a_2$ を検査する DFA M_3

証明: ∃ の処理

- 次に，M_n をもとに M_{n-1} を作る
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \exists x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合を考える
- ψ_{n-1} の真偽を判定するために，x_n の値を非決定的に「推測」する
- DFA M_n の各遷移の n 段目を「潰して」NFA N_{n-1} を作る
- 例えば
  $\psi_3(x_1, x_2, x_3) \equiv x_1 + x_3 = x_2$,
  $\psi_2(x_1, x_2) \equiv \exists x_3[x_1 + x_3 = x_2]$ のときは右図のようになる

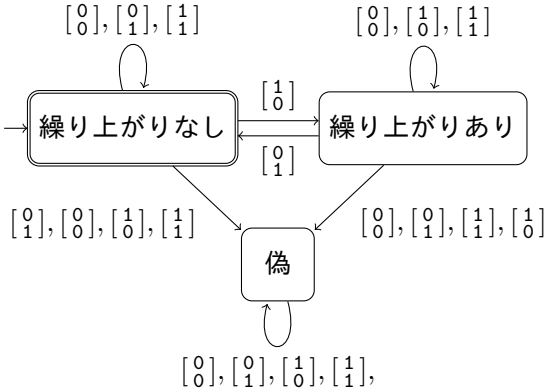


Figure: $\exists x_3[a_1 + x_3 = a_2]$ を検査する NFA N_2

証明

- できあがった NFA N_{n-1} をサブセット構成により DFA M_{n-1} に変換する
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \forall x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合には，$\forall x_n$ が $\neg\exists x_n\neg$ と同じであることを利用して，補集合を取って $\exists$ を除去してからまた補集合をとればよい
- 同様の作業を $n-1, n-2, \ldots, 1$ に対して行う
- 最後に，DFA M_1 の遷移を全て ε 遷移に潰して得られる NFA N_0 を DFA M_0 に変換し，M_0 が ε を受理するかどうかをチェックすれば ψ_0 $(\equiv \varphi)$ の真偽がわかる！
- Q.E.D.

証明

- できあがった NFA N_{n-1} をサブセット構成により DFA M_{n-1} に変換する
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \forall x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合には，$\forall x_n$ が $\neg\exists x_n\neg$ と同じであることを利用して，補集合を取って $\exists$ を除去してからまた補集合をとればよい
- 同様の作業を $n-1, n-2, \ldots, 1$ に対して行う
- 最後に，DFA M_1 の遷移を全て ε 遷移に潰して得られる NFA N_0 を DFA M_0 に変換し，M_0 が ε を受理するかどうかをチェックすれば ψ_0 $(\equiv \varphi)$ の真偽がわかる！
- Q.E.D.

証明

- できあがった NFA N_{n-1} をサブセット構成により DFA M_{n-1} に変換する
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \forall x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合には，$\forall x_n$ が $\neg\exists x_n\neg$ と同じであることを利用して，補集合を取って $\exists$ を除去してからまた補集合をとればよい
- 同様の作業を $n-1, n-2, \ldots, 1$ に対して行う
- 最後に，DFA M_1 の遷移を全て ε 遷移に潰して得られる NFA N_0 を DFA M_0 に変換し，M_0 が ε を受理するかどうかをチェックすれば ψ_0 $(\equiv \varphi)$ の真偽がわかる！
- Q.E.D.

証明

- できあがった NFA N_{n-1} をサブセット構成により DFA M_{n-1} に変換する
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \forall x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合には，$\forall x_n$ が $\neg\exists x_n\neg$ と同じであることを利用して，補集合を取って $\exists$ を除去してからまた補集合をとればよい
- 同様の作業を $n-1, n-2, \ldots, 1$ に対して行う
- 最後に，DFA M_1 の遷移を全て ε 遷移に潰して得られる NFA N_0 を DFA M_0 に変換し，M_0 が ε を受理するかどうかをチェックすれば ψ_0 $(\equiv \varphi)$ の真偽がわかる！
- Q.E.D.

証明

- できあがった NFA N_{n-1} をサブセット構成により DFA M_{n-1} に変換する
- $\psi_{n-1}(a_1, \ldots, a_{n-1}) \equiv \forall x_n[\psi(a_1, \ldots, a_{n-1}, x_n)]$ の場合には，$\forall x_n$ が $\neg\exists x_n\neg$ と同じであることを利用して，補集合を取って $\exists$ を除去してからまた補集合をとればよい
- 同様の作業を $n-1, n-2, \ldots, 1$ に対して行う
- 最後に，DFA M_1 の遷移を全て ε 遷移に潰して得られる NFA N_0 を DFA M_0 に変換し，M_0 が ε を受理するかどうかをチェックすれば ψ_0 $(\equiv \varphi)$ の真偽がわかる！
- Q.E.D.

目次

掛け算が悪いのか

- ここまで，自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論は決定不能だが，$\mathcal{L}_{\mathsf{Pres}}$ 理論は決定可能であることを見た
- 乗算記号 $\times$ を使うことが決定不能性の原因なのだろうか？
- 実は，$\mathcal{L}_{\mathsf{arith}}$ から加算記号 $+$ (と $<$) を取り除いた語彙を $\mathcal{L}_{\mathsf{Tars}} = \{\underline{1}, \times\}$ とするとき，正整数 $\mathbb{N}_+$ の $\mathcal{L}_{\mathsf{Tars}}$ 理論 $\mathrm{Th}(\mathbb{N}_+; \underline{1}, \times)$ (Tarski 算術) は決定可能であることが知られている
  証明のアイデア: 素因数分解によるモノイドの同型 $(\mathbb{N}_+, \times) \cong \bigoplus_{n \geq 0} (\mathbb{N}, +)$ を使う
- つまり，足し算 $+$ と掛け算 $\times$ は単独ではよくわかるが，一緒にするとよくわからなくなる
- 混ぜるな危険

掛け算が悪いのか

- ここまで，自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論は決定不能だが，$\mathcal{L}_{\mathsf{Pres}}$ 理論は決定可能であることを見た
- 乗算記号 $\times$ を使うことが決定不能性の原因なのだろうか？
- 実は，$\mathcal{L}_{\mathsf{arith}}$ から加算記号 $+$ (と $<$) を取り除いた語彙を $\mathcal{L}_{\mathsf{Tars}} = \{\underline{1}, \times\}$ とするとき，正整数 $\mathbb{N}_+$ の $\mathcal{L}_{\mathsf{Tars}}$ 理論 $\mathrm{Th}(\mathbb{N}_+; \underline{1}, \times)$ (Tarski 算術) は決定可能であることが知られている
  証明のアイデア: 素因数分解によるモノイドの同型 $(\mathbb{N}_+, \times) \cong \bigoplus_{n \geq 0} (\mathbb{N}, +)$ を使う
- つまり，足し算 $+$ と掛け算 $\times$ は単独ではよくわかるが，一緒にするとよくわからなくなる
- 混ぜるな危険

掛け算が悪いのか

- ここまで，自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論は決定不能だが，$\mathcal{L}_{\mathsf{Pres}}$ 理論は決定可能であることを見た
- 乗算記号 $\times$ を使うことが決定不能性の原因なのだろうか？
- 実は，$\mathcal{L}_{\mathsf{arith}}$ から加算記号 $+$ (と $<$) を取り除いた語彙を $\mathcal{L}_{\mathsf{Tars}} = \{\underline{1}, \times\}$ とするとき，正整数 $\mathbb{N}_+$ の $\mathcal{L}_{\mathsf{Tars}}$ 理論 $\mathrm{Th}(\mathbb{N}_+; \underline{1}, \times)$ (Tarski 算術) は決定可能であることが知られている
  証明のアイデア: 素因数分解によるモノイドの同型 $(\mathbb{N}_+, \times) \cong \bigoplus_{n \geq 0} (\mathbb{N}, +)$ を使う
- つまり，足し算 $+$ と掛け算 $\times$ は単独ではよくわかるが，一緒にするとよくわからなくなる
- 混ぜるな危険

掛け算が悪いのか

- ここまで，自然数 $\mathbb{N}$ の $\mathcal{L}_{\mathsf{arith}}$ 理論は決定不能だが，$\mathcal{L}_{\mathsf{Pres}}$ 理論は決定可能であることを見た
- 乗算記号 $\times$ を使うことが決定不能性の原因なのだろうか？
- 実は，$\mathcal{L}_{\mathsf{arith}}$ から加算記号 $+$ (と $<$) を取り除いた語彙を $\mathcal{L}_{\mathsf{Tars}} = \{\underline{1}, \times\}$ とするとき，正整数 $\mathbb{N}_+$ の $\mathcal{L}_{\mathsf{Tars}}$ 理論 $\mathrm{Th}(\mathbb{N}_+; \underline{1}, \times)$ (Tarski 算術) は決定可能であることが知られている
  証明のアイデア: 素因数分解によるモノイドの同型 $(\mathbb{N}_+, \times) \cong \bigoplus_{n \geq 0} (\mathbb{N}, +)$ を使う
- つまり，足し算 $+$ と掛け算 $\times$ は単独ではよくわかるが，一緒にするとよくわからなくなる
- 混ぜるな危険

参考文献 (1/2)

有限オートマトンについて:

 M. Sipser, 計算理論の基礎 [原著第 2 版] 1. オートマトンと言語, 共立出版, 2008.

今回の発表の種本 (Presburger 算術の決定可能性のオートマトンを使った証明):

 M. Sipser, 計算理論の基礎 [原著第 2 版] 2. 計算可能性の理論, 共立出版, 2008.

Presburger 算術の決定可能性の QE を使った証明:

 萩谷昌己・西崎真也. 論理と計算のしくみ. 岩波書店, 2007.

 C. Smoryński, Logical Number Theory I, Universitext, Springer-Verlag Berlin Heidelberg, 1991.

参考文献 (2/2)

Tarski 算術の決定可能性について:

- C. Smoryński, Logical Number Theory I, Universitext, Springer-Verlag Berlin Heidelberg, 1991.
- A. Mostowski, On Direct Products of Theories, *J. Symbolic Logic* **17** no. 1 (1952) 1–31.
- P. Cegielski, Theorie elementaire de la multiplication des entiers naturels, in: C. Berline, K. McAloon, J.-P. Ressayre (eds.) *Model Theory and Arithmetic*, Springer-Verlag, New York, 1981.

Gödel の不完全性定理の言い換え:

- E. L. Post, Recursively enumerable sets of positive integers and their decision problems, Bull. Amer. Math. Soc., **50** (1944), 284–316.