定理証明器Coqの効率的な有限ドメイン関数ライブラリ

Efficient Finite-Domain Function Library for the Coq Proof Assistant

坂口 和彦　　亀山 幸義

筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻

2016/6/10 IPSJ PRO @ 浜松

概要

- Coq の有限集合・有限ドメイン関数ライブラリを改良
- 証明から抽出されるプログラムの**実行効率を大幅に改善**
- 既存ライブラリに基く証明をそのまま [*1] 使える（モジュール性）

[*1] 極一部の例外を除く

有限ドメイン関数

- **定義域が有限集合**の関数
- 様々なデータ構造の表現に有用
  - 有限グラフ，有限オートマトン，行列，有限集合の冪集合，
    . . .
- **配列**と一対一に対応

対話的定理証明器 Coq

- 計算機上で数学的証明を記述するためのソフトウェア
  - プログラムの正当性（仕様を満たすこと）の証明も書ける
- 証明から**関数型プログラムを自動的に抽出** [Paulin-Mohring 1989] できる
  - OCaml, Haskell, Scheme 等に対応

本研究で改良するライブラリ

- SSReflect [Gonthier et al. 2015] の有限集合・有限ドメイン関数ライブラリ
  - 四色定理・奇数位数定理などの証明のために開発された
  - 証明の手間の削減に大きく貢献している
- 問題点: 効率的なプログラムを抽出できない

本研究で改良するライブラリ

- SSReflect [Gonthier et al. 2015] の有限集合・有限ドメイン関数ライブラリ
  - 四色定理・奇数位数定理などの証明のために開発された
  - 証明の手間の削減に大きく貢献している
- 問題点: 効率的なプログラムを抽出できない
- 本研究では**有限性の特徴付けを変更**することで性能向上に貢献した

有限性の特徴付けと有限ドメイン関数

変更前

T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と ${}'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {}'I_{|T|}, \\ \mathsf{dec}_T : {}'I_{|T|} \to T \end{cases}$$

有限性の特徴付けと有限ドメイン関数

変更前

T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と ${}'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {}'I_{|T|}, \\ \mathsf{dec}_T : {}'I_{|T|} \to T \end{cases}$$

$$\begin{array}{rcl} \mathsf{enum}_T & = & [\ x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{|T|-1}\ ] \\ & & \quad\downarrow\quad\downarrow\qquad\downarrow\qquad\quad\downarrow \\ f & = & [\ y_0,\ y_1,\ \ldots,\ y_i,\ \ldots,\ y_{|T|-1}\ ] \end{array}$$

$$f(x) = ?$$

有限性の特徴付けと有限ドメイン関数

変更前

T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と ${}'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {}'I_{|T|}, \\ \mathsf{dec}_T : {}'I_{|T|} \to T \end{cases}$$

$x \neq x_0, x_1, \ldots$ $\quad x = x_i$

$$\begin{array}{rcl} \mathsf{enum}_T & = & [\ x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{|T|-1}\ ] \\ & & \downarrow\ \ \downarrow\ \ \ \downarrow\ \ \ \downarrow \\ f & = & [\ y_0,\ y_1,\ \ldots,\ y_i,\ \ldots,\ y_{|T|-1}\ ] \end{array}$$

$$f(x) = \ ?$$

有限性の特徴付けと有限ドメイン関数

変更前

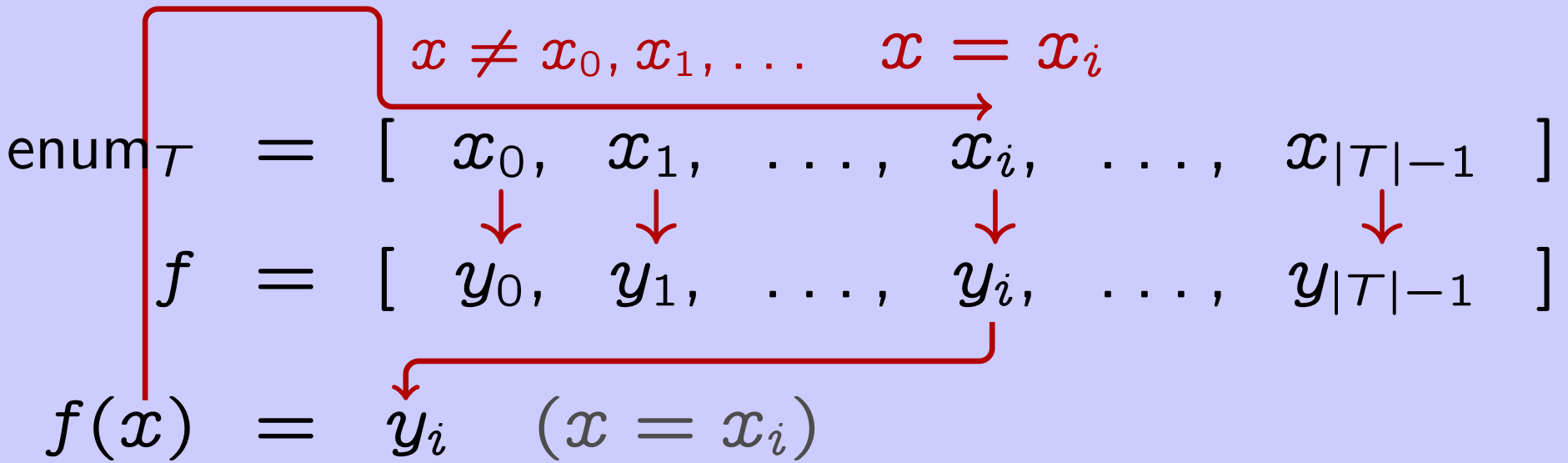
T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と ${}'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {}'I_{|T|}, \\ \mathsf{dec}_T : {}'I_{|T|} \to T \end{cases}$$



有限性の特徴付けと有限ドメイン関数

変更前

T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と $'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {'I}_{|T|}, \\ \mathsf{dec}_T : {'I}_{|T|} \to T \end{cases}$$

$$\begin{aligned} \mathsf{enum}_T &= [\ x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{|T|-1}\ ] \\ f &= [\ y_0,\ y_1,\ \ldots,\ y_i,\ \ldots,\ y_{|T|-1}\ ] \\ f(x) &= \ ? \end{aligned}$$

有限性の特徴付けと有限ドメイン関数

変更前

T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と ${}'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {}'I_{|T|}, \\ \mathsf{dec}_T : {}'I_{|T|} \to T \end{cases}$$

$\mathsf{enc}_T(x)$-th element

$$\mathsf{enum}_T = [\ x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{|T|-1}\ ]$$

$$f = [\ y_0,\ y_1,\ \ldots,\ y_i,\ \ldots,\ y_{|T|-1}\ ]$$

$$f(x) = ?$$

有限性の特徴付けと有限ドメイン関数

変更前

T の要素を重複無く列挙:

$$\begin{cases} \mathsf{enum}_T : \mathsf{seq}(T), \\ \forall x : T.\ |\mathsf{enum}_T|_x = 1 \end{cases}$$

変更後

T と ${}'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間に全単射が存在:

$$\begin{cases} \mathsf{enc}_T : T \to {}'I_{|T|}, \\ \mathsf{dec}_T : {}'I_{|T|} \to T \end{cases}$$

$\mathsf{enc}_T(x)$-th element

$$\mathsf{enum}_T = [\ x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{|T|-1}\ ]$$

$$f = [\ y_0,\ y_1,\ \ldots,\ y_i,\ \ldots,\ y_{|T|-1}\ ]$$

$$f(x) = y_{\mathsf{enc}_T(x)}$$

有限集合型に対する操作

- 値の列挙
- 濃度の計算
- 自然数への符号化・復号化
- 計算可能な量化

値の列挙: enum A

- 有限集合型 T の部分集合 A[*2] に含まれる値を列挙
- 有限集合型に対する最も基本的な操作

[*2] $T \to$ bool の関数で表される

値の列挙: enum A

- 有限集合型 T の部分集合 A[*2] に含まれる値を列挙
- 有限集合型に対する最も基本的な操作
- 変更前
  - 有限性を値の列挙によって特長付けている
  - $A : T \to$ bool で filter すれば良い

[*2] $T \to$ bool の関数で表される

値の列挙: enum A

- 有限集合型 T の部分集合 A[*2] に含まれる値を列挙
- 有限集合型に対する最も基本的な操作
- 変更前
  - 有限性を値の列挙によって特長付けている
  - $A : T \to \mathsf{bool}$ で filter すれば良い
- 変更後
  - $'I_{|T|}$ の値を列挙したリスト $[0, \ldots, |T| - 1]$ に対して符号化関数 enc_T を map することで T の値を列挙

[*2] $T \to \mathsf{bool}$ の関数で表される

濃度の計算: $\#|A|$

- 有限集合型 T の部分集合 A の濃度を計算

濃度の計算: $\#|A|$

- 有限集合型 T の部分集合 A の濃度を計算
- 変更前
  - enum A の長さを返す
- 変更後
  - リストを介さずに直接的に再定義（高速化のため）

濃度の計算: #|A|

- 有限集合型 T の部分集合 A の濃度を計算
- 変更前
  - enum A の長さを返す
- 変更後
  - リストを介さずに直接的に再定義（高速化のため）
- 有限集合型に対する濃度の計算 $|T| を別途定義
  - 有限集合型のインスタンスから値を取り出すだけなので **速い**

自然数への符号化・復号化

- T と $'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間の全単射

自然数への符号化・復号化

- T と $'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間の全単射
- 有限集合型の**部分集合**に対する符号化・復号化関数が元から定義されている
  - `enum_rank_in`, `enum_val`
  - 先頭から探す方式で定義されているので**遅い**

自然数への符号化・復号化

- T と $'I_{|T|} = \{0, \ldots, |T| - 1\}$ の間の全単射
- 有限集合型の**部分集合**に対する符号化・復号化関数が元から定義されている
  - `enum_rank_in`, `enum_val`
  - 先頭から探す方式で定義されているので**遅い**
- **有限集合型**に対する符号化・復号化関数を別途定義
  - `fin_encode`, `fin_decode`
  - 有限集合型のインスタンスに含まれる全単射を使うだけなので**速い**

タプル・有限ドメイン関数

- 有限ドメイン関数はタプルを使って表現する
- タプル: 長さ固定のリスト

```
Structure tuple_of (n : nat) (T : Type) : Type :=
  Tuple {tval :> seq T; _ : size tval == n}.
```

タプル・有限ドメイン関数

- 有限ドメイン関数はタプルを使って表現する
- タプル: 長さ固定のリスト

```
Structure tuple_of (n : nat) (T : Type) : Type :=
  Tuple {tval :> seq T; _ : size tval == n}.
```

- 関数 $f : T \to U$ から有限ドメイン関数へ
  - T の値を列挙したタプルに f を map する
- 有限ドメイン関数 $f : T \to U$ の関数適用 $f(x)$
  - f を表現するタプルの $\mathrm{enc}_T(x)$ 番目を返す
  - 元の定義では `enum_rank_in` が使われている

プログラム抽出

- Coq のプログラム抽出では
  - 帰納的データ型 (`Inductive`, `Record`)
  - 関数や値の定義 (`Definition`)

  に対して抽出先言語の別の定義を割り当てられる

プログラム抽出

- Coq のプログラム抽出では
  - 帰納的データ型 (`Inductive`, `Record`)
  - 関数や値の定義 (`Definition`)

  に対して抽出先言語の別の定義を割り当てられる
- タプルを配列として抽出

```
Extract Inductive tuple_of => "array"
  ["Array.of_list"] "(fun f t -> f (Array.to_list t))".
```

- `Array.of_list`, `Array.to_list` が出現する定義をより効率的な定義で置き換え
  - `codom_tuple`, `tnth` 等

証明のモジュール性 - 1

- 過去に書かれた証明が新しい証明器やライブラリでもチェックできるよう互換性を維持するのは**非常に困難**
  - 同じ値を返すよう関数を再定義するだけで以前の証明が通らなくなる
  - 証明器のバージョンが上がると以前の証明が通らなくなる

証明のモジュール性 - 1

- 過去に書かれた証明が新しい証明器やライブラリでもチェックできるよう互換性を維持するのは**非常に困難**
  - 同じ値を返すよう関数を再定義するだけで以前の証明が通らなくなる
  - 証明器のバージョンが上がると以前の証明が通らなくなる
- **変更した定義の中身を触らせない**ことで互換性を維持
  - 定義を隠蔽して用意された補題による証明を強制
  - 定義の隠蔽には Coq のモジュールを使用

証明のモジュール性 - 2

ライブラリの移行に際して証明の変更がほぼ不要であることを以下の例で確認:

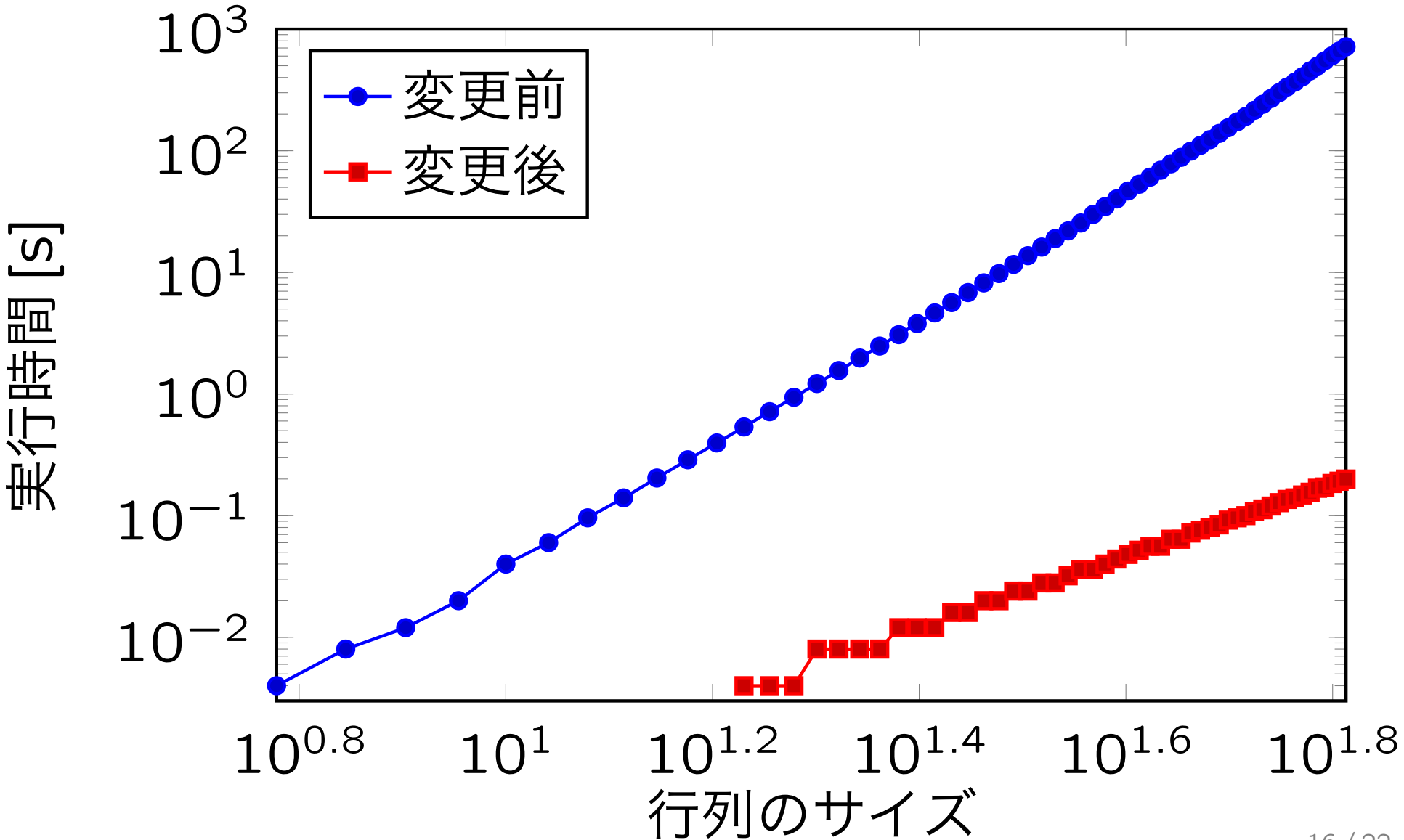
- 奇数位数定理の形式証明 [Gonthier et al. 2013]
  - 変更を要するのは**約 17 万行**の証明のうち **10 行以下**
- 適用例
  - 行列積
  - プレスバーガー算術の決定手続き

適用例 1: 行列の積

- SSReflect の線形代数ライブラリ [Gonthier 2011] で行列の積を計算する単純な例
  - $m \times n$ 行列は $'I_m \times' I_n$ を定義域とする有限ドメイン関数
- 整数の $n \times n$ 行列同士の積の計算時間を計測
- 計算時間をおよそ $\mathcal{O}(n^5)$ **から** $\mathcal{O}(n^3)$ **へ改善**

適用例 1: 行列の積

性能評価



適用例2: プレスバーガー算術

- 自然数と加算に関する一階の理論
- 論理式の充足可能性・妥当性が**決定可能**
  - Coq 上での自動証明にも使われる:
    `omega`, `lia` [Besson 2007] など

適用例2: プレスバーガー算術

- 自然数と加算に関する一階の理論
- 論理式の充足可能性・妥当性が**決定可能**
  - Coq 上での自動証明にも使われる:
    `omega`, `lia` [Besson 2007] など
- Coq 上でプレスバーガー算術の決定可能性を証明・決定手続きを抽出
  - 本研究の手法でどの程度高速化できるか実験

適用例2: プレスバーガー算術

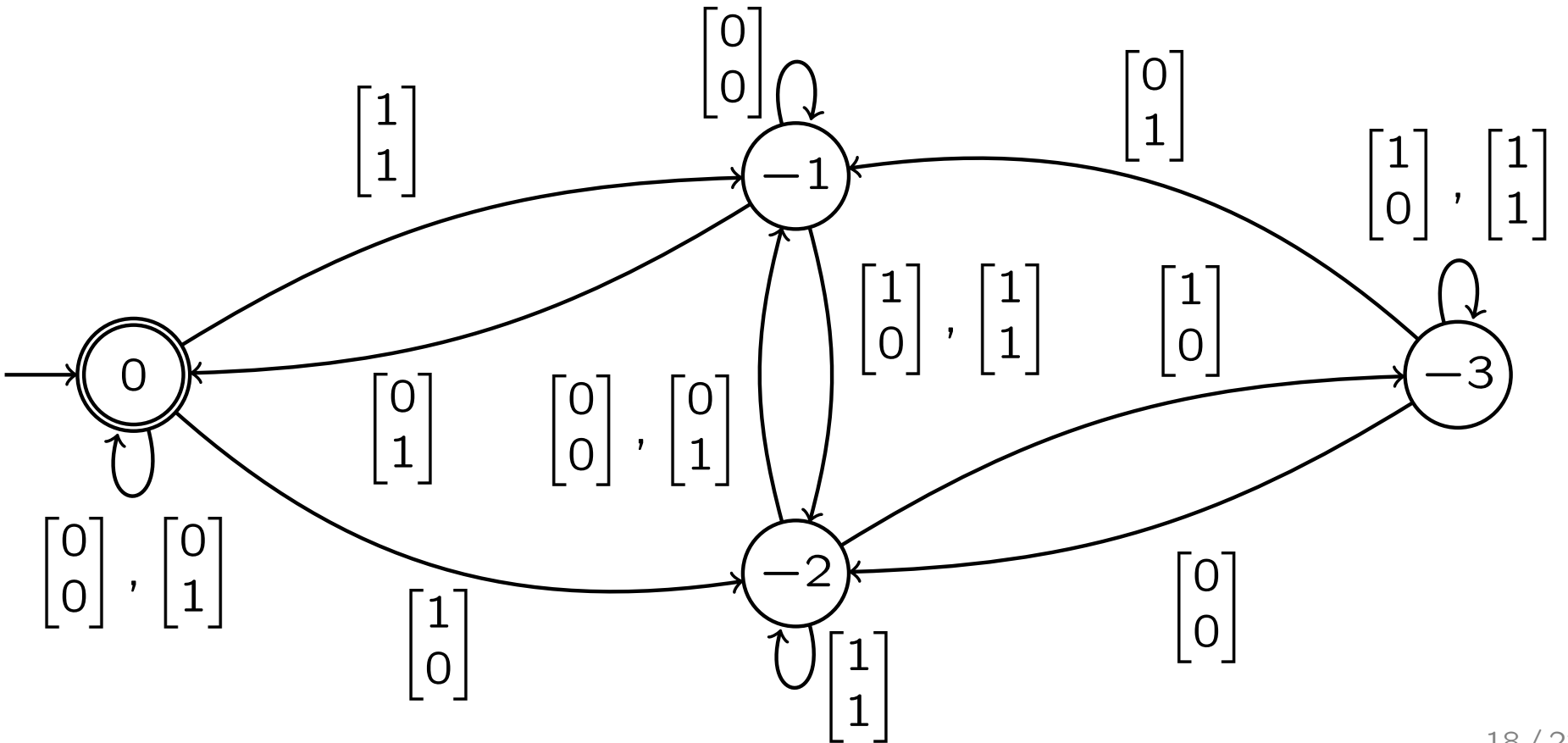
決定手続き

- 論理式を有限オートマトンに変換・各種判定問題を言語の問題に帰着 [Boudet et al. 1996]

適用例2: プレスバーガー算術

決定手続き

- 論理式を有限オートマトンに変換・各種判定問題を言語の問題に帰着 [Boudet et al. 1996]
- 例: $3x_1 - x_2 \leq 0$ に対応するDFA



適用例2: プレスバーガー算術

決定手続き

- 論理式を有限オートマトンに変換・各種判定問題を言語の問題に帰着 [Boudet et al., 1996]
- 例: $3x_1 - x_2 \leq 0$ に対応

Coq による定理証明

2015 12

この決定手続きの形式化の詳細:

http://tcug.jp/books/2015-12/

坂口和彦

Tsukuba Coq Users' Group

適用例2: プレスバーガー算術

性能評価

Table: プレスバーガー算術の決定手続きの実行時間 [*3]

#	決定手続き	状態数 [*4]	計算結果	実行時間 [s] [*5]	
				変更前	変更後
1	SAT	35 (640)	UNSAT	0.016	0.008
2	SAT	70 (660)	UNSAT	0.008	0.004
3	SAT	156 (4000)	UNSAT	0.068	0.028
4	SAT	277 (10560)	SAT	0.116	0.056
5	SAT	6 (2^8)	SAT	0.024	0.008
6	SAT	8 (2^{13})	SAT	0.124	0.076
7	VALID	8 (2^{40})	VALID	N/A	0.036
8	VALID	262 (2^{36})	VALID	N/A	0.168

[*3] 計算対象の命題は発表資料の表 2 を参照

[*4] カッコ内は到達不能な状態を含む状態数である

[*5] N/A はスタックオーバーフローによって計算できなかったことを示す

まとめ

- 有限集合型・有限ドメイン関数ライブラリを高速化
  - アイディアは非常に簡単: $\{0, \ldots, |T| - 1\}$ と T の間の全単射を用いる
- モジュール性
  - SSReflect を用いて書かれた証明をほぼそのまま利用可能
- 実験によって本手法の有効性を検証
  - 行列の積
  - プレスバーガー算術の決定手続き

参考文献 I

Frédéric Besson. "Fast Reflexive Arithmetic Tactics the Linear Case and Beyond". In: *Types for Proofs and Programs*. Vol. 4502. Lecture Notes in Computer Science. Springer, 2007, pp. 48–62.

Alexandre Boudet and Hubert Comon. "Diophantine equations, Presburger arithmetic and finite automata". In: *Trees in Algebra and Programming — CAAP '96*. Vol. 1059. Lecture Notes in Computer Science. Springer, 1996, pp. 30–43.

Georges Gonthier. "Point-Free, Set-Free Concrete Linear Algebra". In: *Interactive Theorem Proving*. Vol. 6898. Lecture Notes in Computer Science. Springer, 2011, pp. 103–118.

参考文献 II

Georges Gonthier et al.
"A Machine-Checked Proof of the Odd Order Theorem".
In: *Interactive Theorem Proving*. Vol. 7998.
Lecture Notes in Computer Science. Springer, 2013,
pp. 163–179.

Georges Gonthier, Assia Mahboubi, and Enrico Tassi.
A Small Scale Reflection Extension for the Coq system.
Research Report. https://hal.inria.fr/inria-00258384v16. INRIA, 2015.

Christine Paulin-Mohring. "Extracting F_ω's programs from proofs in the Calculus of Constructions". In: *Sixteenth Annual ACM Symposium on Principles of Programming Languages*.
Austin: ACM, Jan. 1989.