A Coq Library for Finite Types

2^{nd} bachelor seminar talk

Jan Menz

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Advisor: Prof. Dr. rer. nat. Gert Smolka

17. Juni 2016

Contents

Finite types

What is a finite type?

Finite types

What is a finite type?

- Type

Finite types

What is a finite type?

- Type
- Finite number of inhabitants

Finite types

What is a finite type?

- Type
- Finite number of inhabitants

Finite types

What is a finite type?

- Type
- Finite number of inhabitants

What do we have formally?

Finite types

What is a finite type?

- Type
- Finite number of inhabitants

What do we have formally?

- Type

Finite types

What is a finite type?

- Type
- Finite number of inhabitants

What do we have formally?

- Type
- List of inhabitants

Finite types

What is a finite type?

- Type
- Finite number of inhabitants

What do we have formally?

- Type
- List of inhabitants
- Completeness proof for list

FINITE TYPES

What is a finite type?

- Type
- Finite number of inhabitants

What do we have formally?

- Type
- List of inhabitants
- Completeness proof for list
- Decidability of equality

in Coq

```
Class finTypeC (type:eqType): Type := FinTypeC {
enum: list type;
enum_ok: ∀ x: type, count enum x = 1
}.
```

IN COQ

```
Class finTypeC (type:eqType): Type := FinTypeC {
enum: list type;
enum_ok: ∀ x: type, count enum x = 1
}.

 Structure finType: Type := FinType {
type :> eqType;
class: FinTypeC type
}.
```

Equivalence Principles

Finite Types satisfy important equivalences:

About: elem

elem is a projection from a finType to its list of elements

Equivalence Principles

Finite Types satisfy important equivalences:

$$(\forall\ (x : F), p\ x) \leftrightarrow \forall\ x \in (elem\ F), p\ x$$

About: elem

elem is a projection from a finType to its list of elements

Equivalence Principles

Finite Types satisfy important equivalences:

$$(\forall\ (x : F), p\ x) \leftrightarrow \forall\ x \in (elem\ F), p\ x$$
$$(\exists\ (x : F), p\ x) \leftrightarrow \exists\ x \in (elem\ F), p\ x$$

About: elem

elem is a projection from a finType to its list of elements

Equivalence Principles

Finite Types satisfy important equivalences:

$$(\forall\ (x : F), p\ x) \leftrightarrow \forall\ x \in (elem\ F), p\ x$$
$$(\exists\ (x : F), p\ x) \leftrightarrow \exists\ x \in (elem\ F), p\ x$$
$$(\exists\ (x : F), p\ x) \leftrightarrow \exists\ x, x \in (elem\ F) \rightarrow p\ x$$

About: elem

elem is a projection from a finType to its list of elements

Decidability

Finiteness $\to$ decidability of quantifiers.

Decidability

Finiteness → decidability of quantifiers.

```
Instance finType_forall_dec (F:finType) (p:F → ℙ):
(∀ x, dec(p x)) → dec (∀ x, p x).
```

DECIDABILITY

Finiteness → decidability of quantifiers.

```
Instance finType_forall_dec (F:finType) (p:F → ℙ):
(∀ x, dec(p x)) → dec (∀ x, p x).

Instance finType_forall_dec (F:finType) (p:F → ℙ):
(∀ x, dec(p x)) → dec (∃ x, p x).
```

Decidability

Finiteness → decidability of quantifiers.

```
Instance finType_forall_dec (F:finType) (p:F → ℙ):
(∀ x, dec(p x)) → dec (∀ x, p x).

Instance finType_forall_dec (F:finType) (p:F → ℙ):
(∀ x, dec(p x)) → dec (∃ x, p x).
```

Proof: equivalence properties + corresponding instances for lists

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$

DE MORGAN

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
- $\neg\ (P \rightarrow Q) \rightarrow P \wedge \neg\ Q$

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
- $\neg\ (P \rightarrow Q) \rightarrow P \wedge \neg\ Q$

de Morgan

de Morgan's laws:

- $(\forall x : p\ x) \leftrightarrow \neg\ \exists x\ (\neg\ p\ x)$

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
- $\neg\ (P \rightarrow Q) \rightarrow P \wedge \neg\ Q$

DE MORGAN

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
- $\neg\,(P \to Q) \to P \wedge \neg\, Q$
- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\ (\neg\, p\ x)$
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x)$

DE MORGAN

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
- $\neg\,(P \rightarrow Q) \rightarrow P \wedge \neg\, Q$

- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\ (\neg\, p\ x)$
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x)$
- $\neg\, (\exists x : p\ x) \leftrightarrow \forall x : \neg\, p\ x$

DE MORGAN

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
- $\neg\,(P \to Q) \to P \wedge \neg\, Q$

- $(\forall x : p\ x) \leftrightarrow \neg\ \exists x\ (\neg\ p\ x)$
- $(\exists x : p\ x) \leftrightarrow \neg\ \forall x\ (\neg\ p\ x)$
- $\neg\ (\exists x : p\ x) \leftrightarrow \forall x : \neg\ p\ x$
- $\neg\ (\forall x : p\ x) \leftrightarrow \exists x : \neg\ p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \land Q) \leftrightarrow \neg P \lor \neg Q$
  - true if P or Q decidable
- $\neg(P \lor Q) \leftrightarrow \neg P \land \neg Q$
- $\neg\,(P \rightarrow Q) \rightarrow P \land \neg\, Q$

- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\ (\neg\, p\ x)$
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x)$
- $\neg\, (\exists x : p\ x) \leftrightarrow \forall x : \neg\, p\ x$
- $\neg\, (\forall x : p\ x) \leftrightarrow \exists x : \neg\, p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \land Q) \leftrightarrow \neg P \lor \neg Q$
  - true if P or Q decidable
- $\neg(P \lor Q) \leftrightarrow \neg P \land \neg Q$
  - true in Coq
- $\neg\,(P \to Q) \to P \land \neg\,Q$

- $(\forall x : p\ x) \leftrightarrow \neg\,\exists x\,(\neg\,p\ x)$
- $(\exists x : p\ x) \leftrightarrow \neg\,\forall x\,(\neg\,p\ x)$
- $\neg\,(\exists x : p\ x) \leftrightarrow \forall x : \neg\,p\ x$
- $\neg\,(\forall x : p\ x) \leftrightarrow \exists x : \neg\,p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
  - true if P or Q decidable
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
  - true in Coq
- $\neg\ (P \rightarrow Q) \rightarrow P \wedge \neg\ Q$
  - true if P and Q decidable

- $(\forall x : p\ x) \leftrightarrow \neg\ \exists x\ (\neg\ p\ x)$
- $(\exists x : p\ x) \leftrightarrow \neg\ \forall x\ (\neg\ p\ x)$
- $\neg\ (\exists x : p\ x) \leftrightarrow \forall x : \neg\ p\ x$
- $\neg\ (\forall x : p\ x) \leftrightarrow \exists x : \neg\ p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
  - true if P or Q decidable
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
  - true in Coq
- $\neg\,(P \to Q) \to P \wedge \neg\, Q$
  - true if P and Q decidable

- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\ (\neg\, p\ x)$
  - true for decidable predicates
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x)$
- $\neg\, (\exists x : p\ x) \leftrightarrow \forall x : \neg\, p\ x$
- $\neg\, (\forall x : p\ x) \leftrightarrow \exists x : \neg\, p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
  - true if P or Q decidable
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
  - true in Coq
- $\neg\,(P \rightarrow Q) \rightarrow P \wedge \neg\, Q$
  - true if P and Q decidable

- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\ (\neg\, p\ x)$
  - true for decidable predicates
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x)$
  - true for decidable predicates on finTypes
- $\neg\, (\exists x : p\ x) \leftrightarrow \forall x : \neg\, p\ x$
- $\neg\, (\forall x : p\ x) \leftrightarrow \exists x : \neg\, p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \land Q) \leftrightarrow \neg P \lor \neg Q$
  - true if P or Q decidable
- $\neg(P \lor Q) \leftrightarrow \neg P \land \neg Q$
  - true in Coq
- $\neg\,(P \to Q) \to P \land \neg\, Q$
  - true if P and Q decidable

- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\ (\neg\, p\ x)$
  - true for decidable predicates
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x)$
  - true for decidable predicates on finTypes
- $\neg\, (\exists x : p\ x) \leftrightarrow \forall x : \neg\, p\ x$
  - true in Coq
- $\neg\, (\forall x : p\ x) \leftrightarrow \exists x : \neg\, p\ x$

de Morgan

de Morgan's laws:

- $\neg(P \wedge Q) \leftrightarrow \neg P \vee \neg Q$
  - true if P or Q decidable
- $\neg(P \vee Q) \leftrightarrow \neg P \wedge \neg Q$
  - true in Coq
- $\neg\,(P \rightarrow Q) \rightarrow P \wedge \neg\, Q$
  - true if P and Q decidable
- $(\forall x : p\ x) \leftrightarrow \neg\, \exists x\, (\neg\, p\ x)$
  - true for decidable predicates
- $(\exists x : p\ x) \leftrightarrow \neg\, \forall x\, (\neg\, p\ x)$
  - true for decidable predicates on finTypes
- $\neg\, (\exists x : p\ x) \leftrightarrow \forall x : \neg\, p\ x$
  - true in Coq
- $\neg\, (\forall x : p\ x) \leftrightarrow \exists x : \neg\, p\ x$
  - true for decidable predicates on finTypes

de Morgan

$$(\exists x : p\ x) \leftrightarrow \neg\, \forall x\ (\neg\, p\ x) \qquad \neg\, (\forall x : p\ x) \leftrightarrow \exists x : \neg\, p\ x$$

DE MORGAN

$(\exists x : p\ x) \leftrightarrow \neg \forall x\ (\neg p\ x)$ $\qquad \neg (\forall x : p\ x) \leftrightarrow \exists x : \neg p\ x$

Problem: instantiation of existential quantifiers

DE MORGAN

$(\exists x : p\ x) \leftrightarrow \neg \forall x\ (\neg\ p\ x)$

$\neg\ (\forall x : p\ x) \leftrightarrow \exists x : \neg\ p\ x$

Problem: instantiation of existential quantifiers

finTypes allow to decide $\exists x : p\ x$ and $\exists x : \neg\ p\ x$

CONSTRUCTIVE CHOICE

Normally: Elim Restriction for $\exists$

CONSTRUCTIVE CHOICE

Normally: Elim Restriction for $\exists$

But: One can construct function $\exists x,\ p\ x \to \{x \mid p\ x\}$

CONSTRUCTIVE CHOICE

Normally: Elim Restriction for $\exists$

But: One can construct function $\exists x,\ p\ x \to \{x \mid p\ x\}$

Try every element of the type.

CONSTRUCTIVE CHOICE

Normally: Elim Restriction for $\exists$

But: One can construct function $\exists x,\ p\ x \rightarrow \{x \mid p\ x\}$

Try every element of the type.

One needs to fulfil p

Cardinality

Cardinality X: Number of inhabitants of X

Cardinality

Cardinality X: Number of inhabitants of X

```
Definition Cardinality (F: finType) := |elem F|
```

Cardinality

Cardinality X: Number of inhabitants of X

```
Definition Cardinality (F: finType) := |elem F|
```

Cardinality Lemmas:

$$\begin{aligned}
Cardinality\ (F_1 \times F_2) &= Cardinality\ F_1 * Cardinality\ F_2 \\
Cardinality\ (option\ F) &= S(Cardinality\ F) \\
Cardinality\ (F_1 + F_2) &= Cardinality\ F_1 + Cardinality\ F_2 \\
Cardinality\ (F_1 \longrightarrow F_2) &= (Cardinality\ F_2)^{(Cardinality\ F_1)}
\end{aligned}$$

Cardinality

Cardinality X: Number of inhabitants of X

```
Definition Cardinality (F: finType) := |elem F|
```

Cardinality Lemmas:

$$
\begin{aligned}
\mathit{Cardinality}\ (F_1 \times F_2) &= \mathit{Cardinality}\ F_1 * \mathit{Cardinality}\ F_2 \\
\mathit{Cardinality}\ (\mathit{option}\ F) &= S(\mathit{Cardinality}\ F) \\
\mathit{Cardinality}\ (F_1 + F_2) &= \mathit{Cardinality}\ F_1 + \mathit{Cardinality}\ F_2 \\
\mathit{Cardinality}\ (F_1 \longrightarrow F_2) &= (\mathit{Cardinality}\ F_2)^{(\mathit{Cardinality}\ F_1)}
\end{aligned}
$$

$$\mathit{Cardinality}\ F = \mathit{card}\ (\mathit{elem}\ F)$$

PIDGEONHOLE PRICIPLES

The following holds on finite Types:

$(\exists \text{ injection } f : F_1 \to F_2) \to \mathit{Cardinality}\ F_1 \leq \mathit{Cardinality}\ F_2$

PIDGEONHOLE PRICIPLES

The following holds on finite Types:

$$(\exists \text{ injection } f : F_1 \to F_2) \to \mathit{Cardinality}\ F_1 \leq \mathit{Cardinality}\ F_2$$

$$(\exists \text{ surjection } f : F_1 \to F_2) \to \mathit{Cardinality}\ F_1 \geq \mathit{Cardinality}\ F_2$$

PIDGEONHOLE PRICIPLES

The following holds on finite Types:

$(\exists \text{ injection } f : F_1 \to F_2) \to \mathit{Cardinality}\ F_1 \leq \mathit{Cardinality}\ F_2$

$(\exists \text{ surjection } f : F_1 \to F_2) \to \mathit{Cardinality}\ F_1 \geq \mathit{Cardinality}\ F_2$

$(\exists \text{ bijection } f : F_1 \to F_2) \to \mathit{Cardinality}\ F_1 = \mathit{Cardinality}\ F_2$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \rightarrow F_2) : |image\ f| = Cardinality\ F_1$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

PIDGEONHOLE PRINCIPLES: IMPORTANT LEMMAS

Reminder: card

```
Fixpoint card A: ℕ :=
match A with | nil ⇒ 0
| x::A′⇒ if decision x el A then card A else 1 + card A
end.
```

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

PIDGEONHOLE PRINCIPLES

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

---

injection $(f : F_1 \to F_2)$

$$\text{Cardinality } F_1 \leq \text{Cardinality } F_2$$

PIDGEONHOLE PRINCIPLES

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

---

injection $(f : F_1 \to F_2)$

$$|\text{image f}| \leq \text{Cardinality}\ F_2$$

PIDGEONHOLE PRINCIPLES

$$\textit{card } A \leq |A|$$

$$(f : F_1 \to F_2) : |\textit{image } f| = \textit{Cardinality } F_1$$

$$\textit{dupfree } A \to \textit{card } A = |A|$$

$$(A : \textit{list } F) : \textit{card } A \leq \textit{Cardinality } F$$

$$\textit{injective } f \to \textit{dupfree } (\textit{image } f)$$

$$\textit{dupfree } (\textit{elem } F)$$

$$\textit{surjective } f \to (\textit{elem } F_2) \subseteq \textit{image } f$$

$$A \subseteq B \to \textit{card } A \leq \textit{card } B$$

---

injection $(f : F_1 \to F_2)$

$$\text{card (image f)} \leq \text{Cardinality } F_2$$

PIDGEONHOLE PRINCIPLES

$$\mathit{card}\ A \leq |A|$$

$$(f : F_1 \to F_2) : |\mathit{image}\ f| = \mathit{Cardinality}\ F_1$$

$$\mathit{dupfree}\ A \to \mathit{card}\ A = |A|$$

$$(A : \mathit{list}\ F) : \mathit{card}\ A \leq \mathit{Cardinality}\ F$$

$$\mathit{injective}\ f \to \mathit{dupfree}\ (\mathit{image}\ f)$$

$$\mathit{dupfree}\ (\mathit{elem}\ F)$$

$$\mathit{surjective}\ f \to (\mathit{elem}\ F_2) \subseteq \mathit{image}\ f$$

$$A \subseteq B \to \mathit{card}\ A \leq \mathit{card}\ B$$

---

injection $(f : F_1 \to F_2)$

$$\text{card (image f)} \leq \text{Cardinality } F_2\ \checkmark$$

PIDGEONHOLE PRINCIPLES

$$\frac{\begin{array}{ll} \textit{card } A \leq |A| & \textit{injective } f \to \textit{dupfree } (\textit{image } f) \\ (f : F_1 \to F_2) : |\textit{image } f| = \textit{Cardinality } F_1 & \textit{dupfree } (\textit{elem } F) \\ \textit{dupfree } A \to \textit{card } A = |A| & \textit{surjective } f \to (\textit{elem } F_2) \subseteq \textit{image } f \\ (A : \textit{list } F) : \textit{card } A \leq \textit{Cardinality } F & A \subseteq B \to \textit{card } A \leq \textit{card } B \end{array}}{\text{surjection } (f : F_1 \to F_2)}$$

$$\text{Cardinality } F_1 \geq \text{Cardinality } F_2$$

PIDGEONHOLE PRINCIPLES

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

---

surjection $(f : F_1 \to F_2)$

$$|\text{image f}| \geq \text{Cardinality}\ F_2$$

PIDGEONHOLE PRINCIPLES

$$\frac{\begin{array}{ll} \mathit{card}\ A \leq |A| & \mathit{injective}\ f \to \mathit{dupfree}\ (\mathit{image}\ f) \\ (f : F_1 \to F_2) : |\mathit{image}\ f| = \mathit{Cardinality}\ F_1 & \mathit{dupfree}\ (\mathit{elem}\ F) \\ \mathit{dupfree}\ A \to \mathit{card}\ A = |A| & \mathit{surjective}\ f \to (\mathit{elem}\ F_2) \subseteq \mathit{image}\ f \\ (A : \mathit{list}\ F) : \mathit{card}\ A \leq \mathit{Cardinality}\ F & A \subseteq B \to \mathit{card}\ A \leq \mathit{card}\ B \end{array}}{\text{surjection } (f : F_1 \to F_2)}$$

$$|\text{image f}| \geq |\text{elem } F_2|$$

PIDGEONHOLE PRINCIPLES

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

---

surjection $(f : F_1 \to F_2)$

$$|\text{image f}| \geq \text{card } |\text{elem } F_2|$$

PIDGEONHOLE PRINCIPLES

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

---

surjection $(f : F_1 \to F_2)$

$$\text{card } |\text{image f}| \geq \text{card } |\text{elem } F_2|$$

PIDGEONHOLE PRINCIPLES

$card\ A \leq |A|$

$(f : F_1 \to F_2) : |image\ f| = Cardinality\ F_1$

$dupfree\ A \to card\ A = |A|$

$(A : list\ F) : card\ A \leq Cardinality\ F$

$injective\ f \to dupfree\ (image\ f)$

$dupfree\ (elem\ F)$

$surjective\ f \to (elem\ F_2) \subseteq image\ f$

$A \subseteq B \to card\ A \leq card\ B$

---

surjection $(f : F_1 \to F_2)$

$$card\ |image\ f| \geq card\ |elem\ F_2|\ \checkmark$$

FIXED POINT ALGORITHMS

Definition (fixed point)

An argument x is a *fixed point* of a function *f* if $f\ x = x$.

FIXED POINT ALGORITHMS

Definition (fixed point)

An argument x is a *fixed point* of a function *f* if $f\ x = x$.

Fixed point algorithm: Apply f to it's result until you get a fixed point, then terminate.

FIXED POINT ALGORITHMS

Definition (fixed point)

An argument x is a *fixed point* of a function f if $f\ x = x$.

Fixed point algorithm: Apply f to it's result until you get a fixed point, then terminate.

Problem: not structurally recursive

FIXED POINT ALGORITHMS

Definition (fixed point)

An argument x is a *fixed point* of a function f if $f\ x = x$.

Fixed point algorithm: Apply f to it's result until you get a fixed point, then terminate.

Problem: not structurally recursive

FIXED POINT ALGORITHMS

Definition (fixed point)

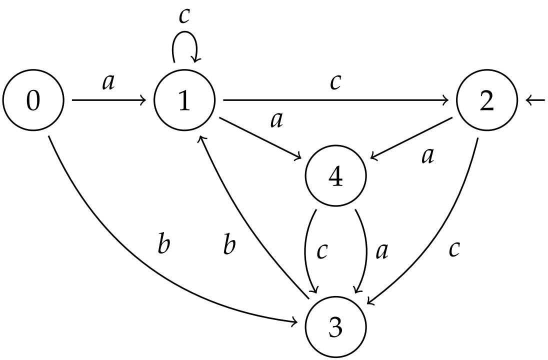
An argument x is a *fixed point* of a function f if $f\ x = x$.

Fixed point algorithm: Apply f to it's result until you get a fixed point, then terminate.

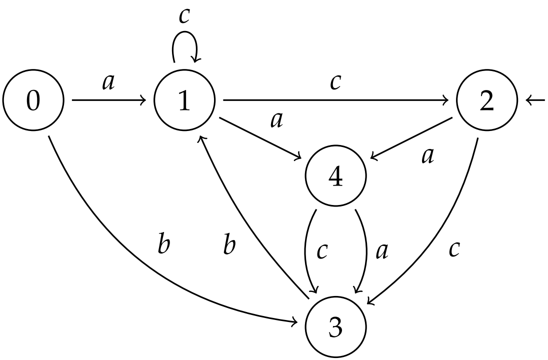
Problem: not structurally recursive

Special case: Compute subset of some set

Example: reachable states in NFAs

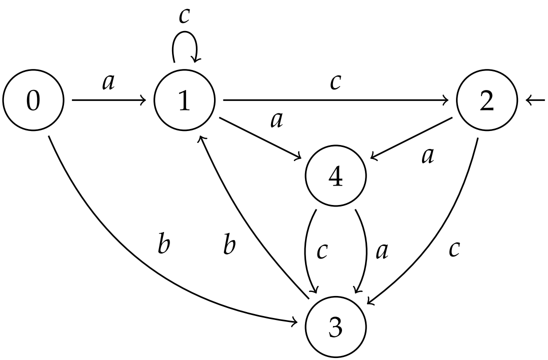


Example: reachable states in NFAs



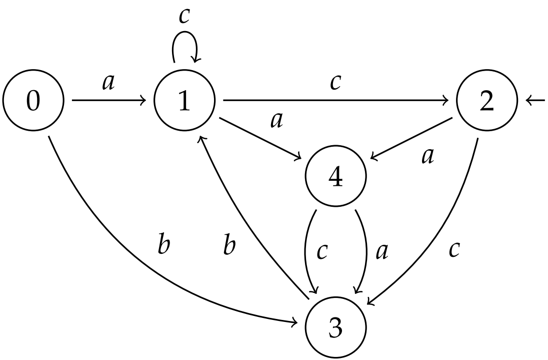
1. {}

Example: reachable states in NFAs



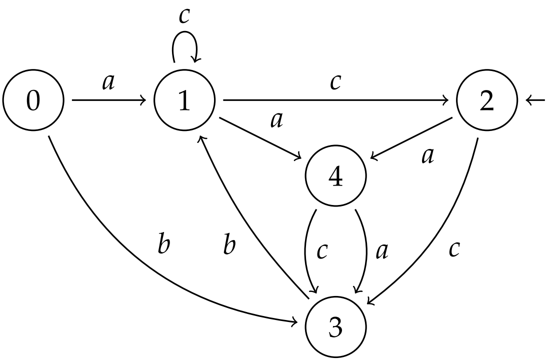
1. {}
2. {2}

Example: reachable states in NFAs



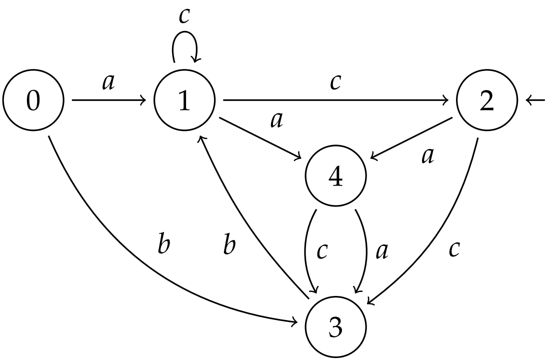
1. {}
2. {2}
3. {2,3}

Example: reachable states in NFAs



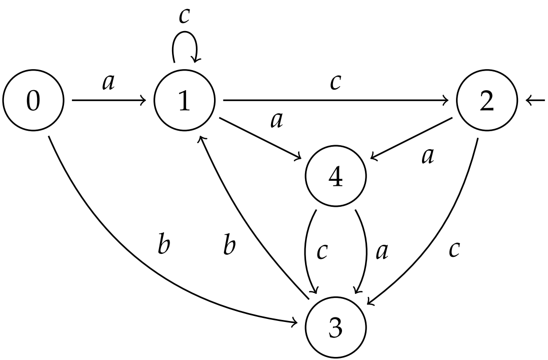
1. {}
2. {2}
3. {2,3}
4. {2,3,4}

Example: reachable states in NFAs



1. {}
2. {2}
3. {2,3}
4. {2,3,4}
5. {2,3,4,1}

Example: reachable states in NFAs



1. {}
2. {2}
3. {2,3}
4. {2,3,4}
5. {2,3,4,1}
6. {2,3,4,1} ← fixed point

Finite closure iteration

Compute subset of finite type F

Finite closure iteration

Compute subset of finite type F

- step: list $\mathrm{F} \to F \to \mathbb{P}$

Finite closure iteration

Compute subset of finite type F

- step: list F $\to F \to \mathbb{P}$
- function pick A : {x | step A x $\land \neg (x \in A)$} + $\forall x$, step A x $\to\ x \in A$.

Finite closure iteration

Compute subset of finite type F

- step: list F $\rightarrow F \rightarrow \mathbb{P}$
- function pick A : {x | step A x $\wedge \neg (x \in A)$} + $\forall x$, step A x $\rightarrow x \in A$.
  - decide ($\forall x, step\ A\ x \rightarrow x \in A$.)

Finite closure iteration

Compute subset of finite type F

- step: list F $\rightarrow F \rightarrow \mathbb{P}$
- function pick A : {x | step A x $\wedge \neg (x \in A)$} + $\forall x$, step A x $\rightarrow\ x \in A$.
  - decide $(\forall x, \mathit{step}\ A\ x \rightarrow\ x \in\ A.)$
  - 2^{nd} case: use de Morgan to get existential

Finite closure iteration

Compute subset of finite type F

- step: list F $\rightarrow F \rightarrow \mathbb{P}$
- function pick A : {x | step A x $\wedge \neg (x \in A)$} + $\forall x$, step A x $\rightarrow\ x \in A$.
  - decide $(\forall x, step\ A\ x \rightarrow\ x \in\ A.)$
  - 2^{nd} case: use de Morgan to get existential
  - Then use constructive choice

Finite closure iteration

Compute subset of finite type F

- step: list F $\rightarrow F \rightarrow \mathbb{P}$
- function pick A : {x | step A x $\wedge \neg (x \in A)$} + $\forall x$, step A x $\rightarrow\ x \in A$.
  - decide $(\forall x, step\ A\ x \rightarrow\ x \in\ A.)$
  - 2^{nd} case: use de Morgan to get existential
  - Then use constructive choice

FCStep

```
Definition FCStep A :=
match (pick A) with
|inl L ⇒ match L with
         |exists _ x _ ⇒ x::A end
|inr _ ⇒ A end.
```

Finite closure iteration

Claim: for all lists A

```
iter (Cardinality F) FCStep A
```

is a fixed point of FCStep.

Finite closure iteration

Claim: for all lists A

```
iter (Cardinality F) FCStep A
```

is a fixed point of FCStep.

Definition (monotone)

A function (f: list X → list X) is called *monotone* if a given list A is either a fixed-point of f or $card\ (f\ A) > card\ A$

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

MONOTONICITY

Claim: for all monotone functions f and lists A

`iter (Cardinality F) f A`

is a fixed point of f.

Proof Sketch.

Let f: *list* $F \to$ *list* F be monotone. Then by induction on $n \in \mathbb{N}$

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

Proof Sketch.

Let f: *list* $F \rightarrow$ *list* F be monotone. Then by induction on $n \in \mathbb{N}$ every A: *list* F is either a fixed point of f

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

Proof Sketch.

Let *f: list F* $\rightarrow$ *list F* be monotone. Then by induction on $n \in \mathbb{N}$ every *A: list F* is either a fixed point of *f* or *card (iter n f A)* $\geq n$.

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

Proof Sketch.

Let f: *list F* $\rightarrow$ *list F* be monotone. Then by induction on $n \in \mathbb{N}$ every A: *list F* is either a fixed point of f or *card (iter n f A)* $\geq n$.

Now choose n = *Cardinality F*.

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

Proof Sketch.

Let f: *list F* $\rightarrow$ *list F* be monotone. Then by induction on $n \in \mathbb{N}$ every A: *list F* is either a fixed point of f or *card (iter n f A)* $\geq n$.

Now choose n = *Cardinality F*.
Assume *iter n f a* is no fixed point of f.

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

Proof Sketch.

Let f: *list* $F \rightarrow$ *list* F be monotone. Then by induction on $n \in \mathbb{N}$ every A: *list* F is either a fixed point of f
or *card (iter* n f A*)* $\geq n$.

Now choose n = *Cardinality* F.
Assume *iter* n f a is no fixed point of f.
Then *card*(f (*iter* n f a)) > *Cardinality* F because f is monotone.

Monotonicity

Claim: for all monotone functions f and lists A

```
iter (Cardinality F) f A
```

is a fixed point of f.

Proof Sketch.

Let *f: list F* $\rightarrow$ *list F* be monotone. Then by induction on $n \in \mathbb{N}$ every *A: list F* is either a fixed point of *f* or *card (iter n f A)* $\geq n$.

Now choose n = *Cardinality F*.
Assume *iter n f a* is no fixed point of f.
Then *card(f (iter n f a))* > *Cardinality F* because f is monotone.
But *Cardinality F* = *card (elem F)* and *elem F* contains all possible elements of type F. ∎

FINITE CLOSURE ITERATION: AGAIN

Claim: for all lists A

```
iter (Cardinality F) FCStep A
```

is a fixed point of FCStep.

Proof Sketch.

By case analysis on *pick A*: FCStep is monotone. ■

Finite Closure iteration: Again

Claim: for all lists A

`iter (Cardinality F) FCStep A`

is a fixed point of FCStep.

Proof Sketch.

By case analysis on *pick A*: FCStep is monotone. ■

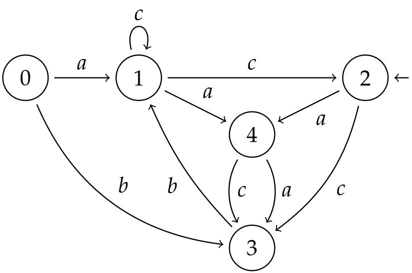
⇒ `iter (Cardinality F) FCStep A`
is the fixed point we want.

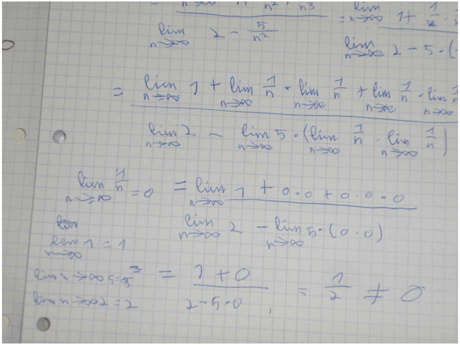
A glimpse into the crystal ball

What will the future bring?

A glimpse into the crystal ball

What will the future bring?





Sources and Inspiration

- Smolka, Gert and Brown, Chad E.
  Introduction to Computational Logic
  Lecture Notes SS 2014
- Smolka, Gert
  Base library for ICL
  Version: February 15^{th} 2016

The End

Thank you for your attention

Any questions? Ask away!