

Up-to Techniques using Sized Types

Nils Anders Danielsson

POPL 2018, Los Angeles, 2018-01-11

Introduction

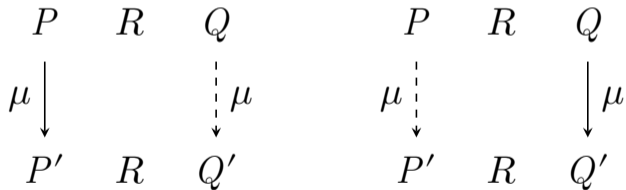
What is an up-to technique?

- ▶ Often used to make it easier to define bisimulations.

Introduction

What is an up-to technique?

- ▶ R is a bisimulation (for CCS):



Introduction

What is an up-to technique?

- ▶ Up to bisimilarity:

$$\begin{array}{ccccc} P & & R & & Q \\ \mu \downarrow & & & & \text{---} \mu \\ P' & \sim & R & \sim & Q' \end{array}$$

$$\begin{array}{ccccc} P & & R & & Q \\ \text{---} \mu \downarrow & & & & \downarrow \mu \\ P' & \sim & R & \sim & Q' \end{array}$$

Introduction

What is an up-to technique?

- ▶ More generally:

$$\begin{array}{ccccc} P & & R & & Q \\ \mu \downarrow & & & & \text{---} \mu \\ P' & & F R & & Q' \end{array}$$

$$\begin{array}{ccccc} P & & R & & Q \\ \text{---} \mu \downarrow & & & & \downarrow \mu \\ P' & & F R & & Q' \end{array}$$

Introduction

- ▶ In dependently typed languages:
Bisimilarity as indexed coinductive data type.
- ▶ If sized types are used:
A class of up-to techniques falls out naturally.

Coinductive data types

Coinduction without sized types

Potentially infinite lists, roughly $\nu X. 1 + A \times X$:

```
data Colist (A : Set) : Set where
  []       : Colist A
  _::_    : A → Colist' A → Colist A

record Colist' (A : Set) : Set where
  coinductive
  field force : Colist A
```


Corecursion using copatterns

A map function for colists:

$$\text{map} : (A \rightarrow B) \rightarrow \text{Colist } A \rightarrow \text{Colist } B$$
$$\text{map } f [] = []$$
$$\text{map } f (x :: xs) = f x :: \text{map}' f xs$$
$$\text{map}' : (A \rightarrow B) \rightarrow \text{Colist}' A \rightarrow \text{Colist}' B$$
$$\text{force } (\text{map}' f xs) = \text{map } f (\text{force } xs)$$

Corecursion using copatterns

A map function for colists:

$$\text{map} : (A \rightarrow B) \rightarrow \text{Colist } A \rightarrow \text{Colist } B$$
$$\text{map } f [] = []$$
$$\text{map } f (x :: xs) = f x :: \lambda \{ .\text{force} \rightarrow \text{map } f (\text{force } xs) \}$$

Corecursion using copatterns

A map function for colists:

$$\text{map} : (A \rightarrow B) \rightarrow \text{Colist } A \rightarrow \text{Colist } B$$
$$\text{map } f [] = []$$
$$\text{map } f (x :: xs) = f x :: \lambda \{ .\text{force} \rightarrow \text{map } f (\text{force } xs) \}$$

Guarded, productive.

Guardedness

0, 1, 2, ...:

nats : Colist \mathbb{N}

nats = 0 :: $\lambda \{ .\text{force} \rightarrow \text{map } (1 + _) \text{ nats} \}$

Not guarded, rejected.

Sized types

Sized types

Previous definition:

```
data Colist (A : Set) : Set where
  []      : Colist A
  _::_    : A → Colist' A → Colist A
```

```
record Colist' (A : Set) : Set where
  coinductive
  field force : Colist A
```

Sized types

With sized types:

```
data Colist (i : Size) (A : Set) : Set where
  []       : Colist i A
  _::__    : A → Colist' i A → Colist i A
```

```
record Colist' (i : Size) (A : Set) : Set where
  coinductive
  field force : {j : Size < i} → Colist j A
```

Sized types

With sized types:

```
data Colist (i : Size) (A : Set) : Set where
  []       : Colist i A
  _::__    : A → Colist' i A → Colist i A
```

```
record Colist' (i : Size) (A : Set) : Set where
  coinductive
  field force : {j : Size < i} → Colist j A
```

$\text{Colist}' i A$: Partially defined colists of depth at least i .

Sized types

With sized types:

```
data Colist (i : Size) (A : Set) : Set where
  []       : Colist i A
  _::__    : A → Colist' i A → Colist i A
```

```
record Colist' (i : Size) (A : Set) : Set where
  coinductive
  field force : {j : Size < i} → Colist j A
```

$\text{Colist}' \infty A$: Fully defined colists.

Sized types

The map function is size-preserving:

$$\begin{aligned} \text{map} &: \forall \{i\} \rightarrow (A \rightarrow B) \rightarrow \text{Colist } i \ A \rightarrow \text{Colist } i \ B \\ \text{map } f \ [] &= [] \\ \text{map } f \ (x \ :: \ xs) &= f \ x \ :: \ \lambda \{ \text{.force} \rightarrow \text{map } f \ (\text{force } xs) \} \end{aligned}$$

Sized types

The size is smaller in every corecursive call:

```
nats :  $\forall \{i\} \rightarrow \text{Colist } i \mathbb{N}$   
nats = 0 ::  $\lambda \{ .\text{force} \rightarrow \text{map } (1 + \_) \text{ nats} \}$ 
```

Sized types

The size is smaller in every corecursive call:

`nats` : $\forall i \rightarrow \text{Colist } i \mathbb{N}$

`nats` $i = 0$:: $\lambda \{ .\text{force } \{j\} \rightarrow \text{map } (1 + _) (\text{nats } j) \}$

Bisimilarity

A tiny process calculus

Labels/actions:

```
data Label : Set where
  ● : Label
```

A tiny process calculus

Processes:

data Proc : Set where
 \emptyset : Proc
 $_|_$: Proc \rightarrow Proc \rightarrow Proc
 \bullet : Proc' \rightarrow Proc

record Proc' : Set where
 coinductive
 field force : Proc

Roughly $\nu C. \mu I. 1 + I \times I + C$.

A tiny process calculus

Transition relation:

```
data  $_{[_] \mapsto}$  : Proc  $\rightarrow$  Label  $\rightarrow$  Proc  $\rightarrow$  Set where  
  action    :  $\bullet$  P [  $\bullet$  ]  $\mapsto$  force P  
  par-left  : P [  $\mu$  ]  $\mapsto$  P'  $\rightarrow$  P | Q [  $\mu$  ]  $\mapsto$  P' | Q  
  par-right : Q [  $\mu$  ]  $\mapsto$  Q'  $\rightarrow$  P | Q [  $\mu$  ]  $\mapsto$  P | Q'
```


Bisimilarity

R is a bisimulation:

$$\begin{array}{ccc} P & R & Q \\ \mu \downarrow & & \text{---} \mu \\ P' & R & Q' \end{array}$$

$$\begin{array}{ccc} P & R & Q \\ \text{---} \mu \downarrow & & \downarrow \mu \\ P' & R & Q' \end{array}$$

Bisimilarity

record **Bisimilar** ($i : \text{Size}$) ($P Q : \text{Proc}$) : **Set** where

inductive

field **left-to-right** :

$$P [\mu] \mapsto P' \rightarrow \exists Q' \rightarrow Q [\mu] \mapsto Q' \times \text{Bisimilar}' i P' Q'$$

right-to-left :

$$Q [\mu] \mapsto Q' \rightarrow \exists P' \rightarrow P [\mu] \mapsto P' \times \text{Bisimilar}' i P' Q'$$

record **Bisimilar'** ($i : \text{Size}$) ($P Q : \text{Proc}$) : **Set** where

coinductive

field **force** : $\{j : \text{Size} < i\} \rightarrow \text{Bisimilar } j P Q$

Examples

Bisimilarity is transitive:

$$\begin{aligned} \text{trans} : & \text{Bisimilar } i \ P \ Q \rightarrow \\ & \text{Bisimilar } i \ Q \ R \rightarrow \\ & \text{Bisimilar } i \ P \ R \end{aligned}$$

Note that the proof is size-preserving.

Examples

Parallel composition preserves bisimilarity:

$$\begin{aligned} \text{-cong} : & \text{Bisimilar } i \ P \ P' \rightarrow \\ & \text{Bisimilar } i \ Q \ Q' \rightarrow \\ & \text{Bisimilar } i \ (P \mid Q) \ (P' \mid Q') \end{aligned}$$

Note that the proof is size-preserving.

Examples

Prefixing preserves bisimilarity:

$$\bullet\text{-cong} : \text{Bisimilar}' \ i \ (\text{force } P) \ (\text{force } Q) \rightarrow \\ \text{Bisimilar} \ i \ (\bullet P) \ (\bullet Q)$$

Note that the proof is size-preserving.

Examples

\emptyset is a left identity of parallel composition:

\emptyset -left-identity : Bisimilar $i (\emptyset \mid P) P$

Examples

Two processes:

$P \ Q : \text{Proc}$

$P = \emptyset \mid (\bullet P' \mid \bullet P')$

$Q = \bullet Q' \mid \bullet Q'$

$P' \ Q' : \text{Proc}'$

$\text{force } P' = P$

$\text{force } Q' = Q$

Examples

P and Q are bisimilar:

$\text{sim} : \forall \{i\} \rightarrow \text{Bisimilar } i \text{ P Q}$

$\text{sim} = \text{trans } \emptyset\text{-left-identity}$

$(|- \text{cong } (\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$
 $(\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$

$\text{P Q} : \text{Proc}$

$\text{P} = \emptyset \mid (\bullet \text{P}' \mid \bullet \text{P}')$

$\text{Q} = \bullet \text{Q}' \mid \bullet \text{Q}'$

$\text{P}' \text{Q}' : \text{Proc}'$

$\text{force P}' = \text{P}$

$\text{force Q}' = \text{Q}$

Examples

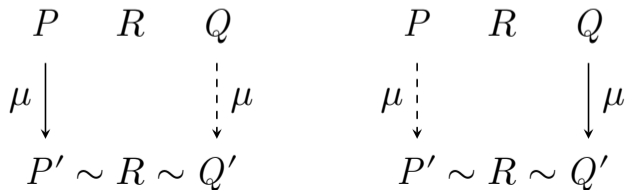
P and Q are bisimilar:

$\text{sim} : \forall \{i\} \rightarrow \text{Bisimilar } i \text{ P Q}$

$\text{sim} = \text{trans } \emptyset\text{-left-identity}$

$(|-\text{cong } (\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$
 $(\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$

Compare trans and up to bisimilarity:



Examples

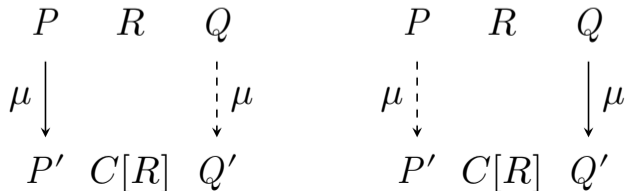
P and Q are bisimilar:

$\text{sim} : \forall \{i\} \rightarrow \text{Bisimilar } i \text{ P Q}$

$\text{sim} = \text{trans } \emptyset\text{-left-identity}$

$(|-\text{cong } (\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$
 $(\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$

Compare $|-\text{cong}/\bullet\text{-cong}$ and up to context:



Examples

P and Q are bisimilar:

$\text{sim} : \forall \{i\} \rightarrow \text{Bisimilar } i \text{ P Q}$

$\text{sim} = \text{trans } \emptyset\text{-left-identity}$

$(|-\text{cong } (\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$
 $(\bullet\text{-cong } (\lambda \{ .\text{force} \rightarrow \text{sim} \})))$

- ▶ The proofs are size-preserving, so they can be combined freely.
- ▶ No extra work is required to show that the proofs are size-preserving.

Weak bisimilarity

- ▶ For full CCS: *Weak* bisimilarity.
- ▶ Size-preserving preservation lemmas.
- ▶ Transitivity:

Weakly-bisimilar $\infty P Q \rightarrow$

Weakly-bisimilar $\infty Q R \rightarrow$

Weakly-bisimilar $\infty P R$

Cannot be proved in a size-preserving way.

- ▶ The problem of “weak bisimulation up to”.

Generalisation

- ▶ A general notion of (sound) up-to technique can be defined.
- ▶ Every size-preserving predicate transformer is an up-to technique.
- ▶ Closed under composition.
- ▶ Closely related to a class of up-to techniques identified by Pous: Functions below the “companion”.

Conclusion

When using a type theory with sized types to define bisimilarity a useful class of up-to techniques falls out naturally.

Extra material

Containers

- ▶ Containers (well-behaved functors):

$$\begin{aligned} \text{Container} &: \text{Set} \rightarrow \text{Set}_1 \\ \llbracket _ \rrbracket &: \text{Container } X \rightarrow (X \rightarrow \text{Set}) \rightarrow (X \rightarrow \text{Set}) \end{aligned}$$

- ▶ Greatest fixpoints of containers:

$$\nu : \text{Container } X \rightarrow \text{Size} \rightarrow (X \rightarrow \text{Set})$$

Up-to techniques

Up-to techniques (sound):

Up-to-technique :

Container $X \rightarrow ((X \rightarrow \text{Set}) \rightarrow (X \rightarrow \text{Set})) \rightarrow \text{Set}_1$

Up-to-technique $C \ F =$

$\forall R \rightarrow R \subseteq \llbracket C \rrbracket (F \ R) \rightarrow R \subseteq \nu \ C \ \infty$

Size-preserving

- ▶ Size-preserving predicate transformers:

Size-preserving :

Container $X \rightarrow ((X \rightarrow \text{Set}) \rightarrow (X \rightarrow \text{Set})) \rightarrow \text{Set}_1$

Size-preserving $C \ F =$

$\forall R \ i \rightarrow R \subseteq \nu \ C \ i \rightarrow F \ R \subseteq \nu \ C \ i$

- ▶ Every size-preserving predicate transformer is an up-to technique.