

Type Theory

Coinduction in Type Theory

Andreas Abel

Department of Computer Science and Engineering
Chalmers and Gothenburg University

ESLLI 2016

28th European Summer School in Logic, Language, and Information
unibz, Bozen/Bolzano, Italy
15-19 August 2016

Coinduction

- Coinduction is a technique to, e. g.:
 - Define infinitely running processes.
 - Define infinitely deep derivations.
 - Prove properties about processes and infinite derivations.
- A coinductive definition must be *productive*, i. e., always produce a new piece of the output after finite time.
- Agda recently supports coinduction via copatterns and sized types.
- Agda's termination checker also checks productivity.
- This talk: coinduction for the example of formal languages.

Contents

- 1 Formal Languages
- 2 Coinductive Types and Copatterns
- 3 Bisimilarity
- 4 Sized Coinductive Types
- 5 Conclusions

Formal Languages

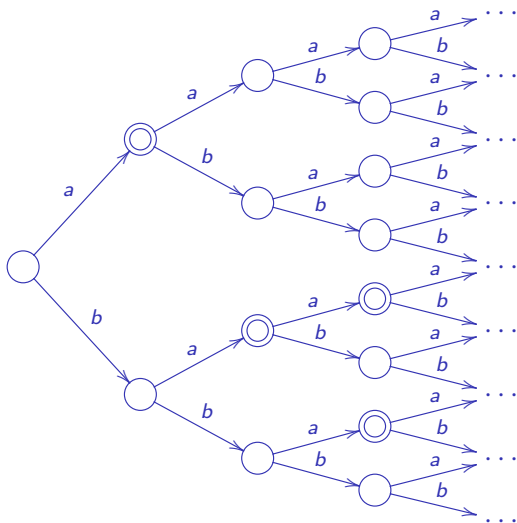
- A **language** is a set of strings over some **alphabet** A .
- Real life examples:
 - Orthographically and grammatically correct English texts (infinite set).
 - Orthographically correct English texts (even bigger set).
 - List of university employees plus their phone extension.
AbelAndreas1731, CoquandThierry1030, DybjerPeter1035, ...
- Programming language examples:
 - The set of grammatically correct JAVA programs.
 - The set of decimal numbers.
 - The set of well-formed string literals.
- Languages can describe protocols, e.g. file access.
 - $A = \{o, r, w, c\}$ (open, read, write, close)
 - Read-only access: *orc*, *oc*, *orrrc*, *orcorrrcoc*, ...
 - Illegal sequences: *c*, *rr*, *orr*, *oco*, ...

Running Example: Even binary numbers

- Even binary numbers: 0, 10, 100, 110, 1000, 1010, ...
- Excluded: 00, 010 (non-canonical); 1, 11 (odd) ...
- Alphabet $A = \{a, b\}$ where a is zero and b is one.
- So $E = \{a, ba, baa, bba, baaa, baba, \dots\}$.

Tries

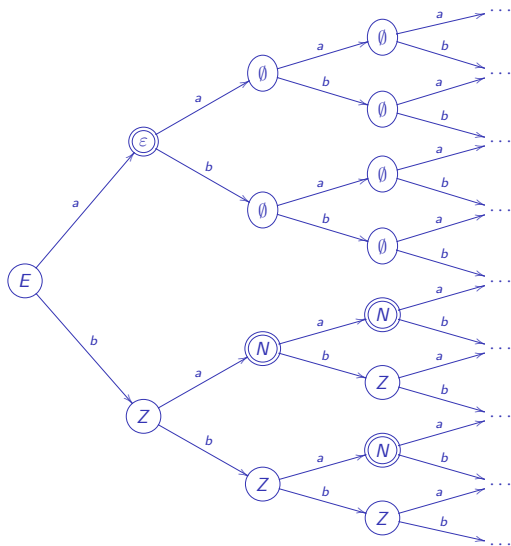
- An infinite trie is a node-labeled A -branching tree.
- I.e., each node has one branch for each letter $a \in A$.
- A language can be represented by an infinite trie.
- To check whether word $a_1 \cdots a_n$ is in the language:
 - We start at the root.
 - At step i , we choose branch a_i .
 - At the final node, the label tells us whether the word is in the language or not.

Trie of E 

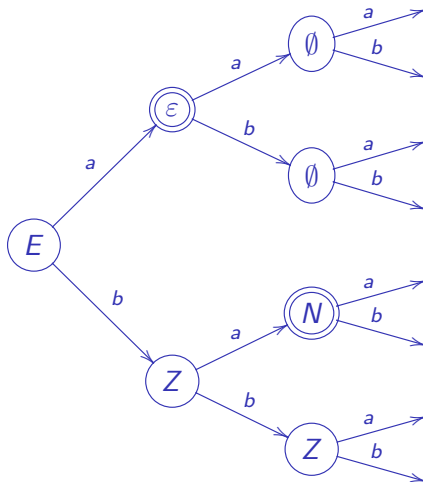
Regular Languages

- A trie is regular if it has only *finitely* many different *subtrees*.
- Each node of the trie corresponds to one of these languages:

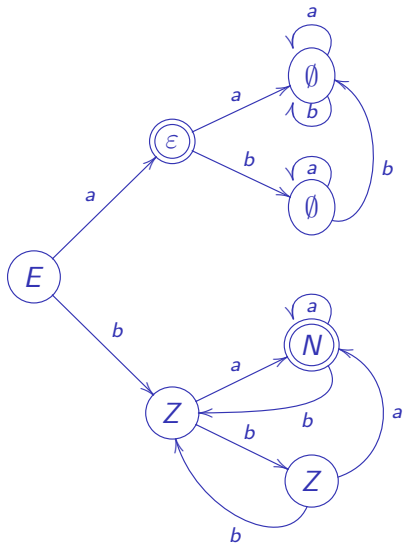
E	even binary numbers
Z	strings ending in a
N	strings not ending in b
ε	the empty string
\emptyset	nothing (empty language)



Cutting duplications at depth 3



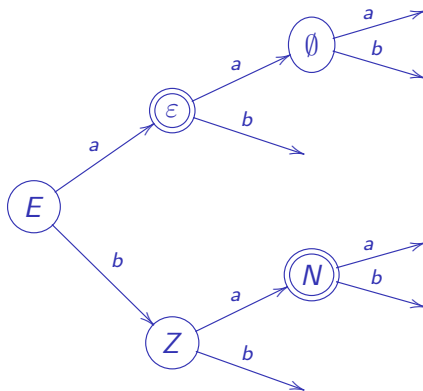
Bending branches ...



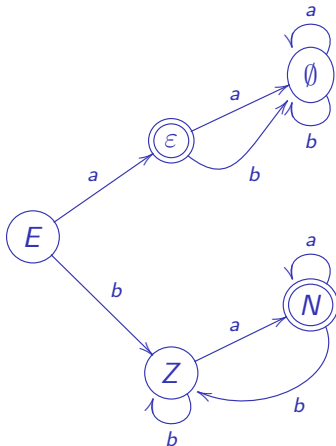
Finite Automata

- We have arrived at a familiar object: a finite automaton.
- Depending on what we cut, we get different automata for E .
- If we cut *all* duplicate subtrees, we get the *minimal* automaton.

Removing duplicate subtrees II...



Bending branches II ...



Extensional Equality of Automata

- All automata for E unfold to the same trie.
- This gives a extensional notion of automata *equality*:
 - 1 Recognizing the same language.
 - 2 I.e., unfold to the same trie.

Automata, Formally

- An automaton consists of
 - A set of **states** S .
 - A function $\nu : S \rightarrow \text{Bool}$ singling out the **accepting** states.
 - A **transition** function $\delta : S \rightarrow A \rightarrow S$.

$s \in S$	νs	$\delta s a$	$\delta s b$
E	\times	ε	Z
ε	\checkmark	\emptyset	\emptyset
\emptyset	\times	\emptyset	\emptyset
Z	\times	N	Z
N	\checkmark	N	Z

- Language automaton**
 - State = language l accepted when starting from that state.
 - νl : Language l is **nullable** (accepts the empty word)?
 - $\delta l a = \{w \mid aw \in l\}$: **Brzowski derivative**.

Differential equations

- Language E and friends can be specified by *differential equations*:
- ν gives the *initial value*.

$$\nu \emptyset = \text{false}$$

$$\delta \emptyset x = \emptyset$$

$$\nu \varepsilon = \text{true}$$

$$\delta \varepsilon x = \emptyset$$

$$\nu E = \text{false}$$

$$\delta E a = \varepsilon$$

$$\delta E b = Z$$

$$\nu N = \text{true}$$

$$\delta N a = N$$

$$\delta N b = Z$$

$$\nu Z = \text{false}$$

$$\delta Z a = N$$

$$\delta Z b = Z$$

- For these simple forms, solutions exist always.
What is the general story?

Final Coalgebras

- (Weakly) final coalgebra.

$$\begin{array}{ccc}
 S & \xrightarrow{f} & F(S) \\
 \text{coit } f \downarrow & & \downarrow F(\text{coit } f) \\
 \nu F & \xrightarrow{\text{force}} & F(\nu F)
 \end{array}$$

- Coiteration = finality witness.

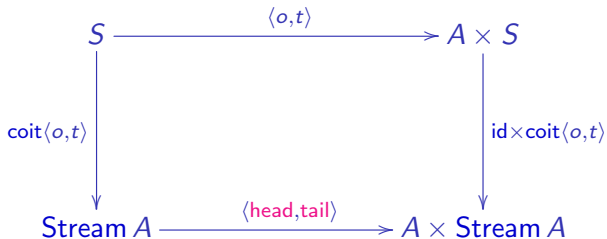
$$\text{force} \circ \text{coit } f = F(\text{coit } f) \circ f$$

- Copattern matching *defines* `coit` by corecursion:

$$\text{force}(\text{coit } f \ s) = F(\text{coit } f)(f \ s)$$

Streams as Final Coalgebra

- Output automaton is coalgebra $\langle o, t \rangle : S \rightarrow A \times S$.
- Final coalgebra = automaton unrolling = stream: $\nu S. A \times S$.



- Termination by induction on observation depth:

$$\text{head} (\text{coit} \langle o, t \rangle s) = o s$$

$$\text{tail} (\text{coit} \langle o, t \rangle s) = \text{coit} \langle o, t \rangle (t s)$$

Automata as Coalgebra

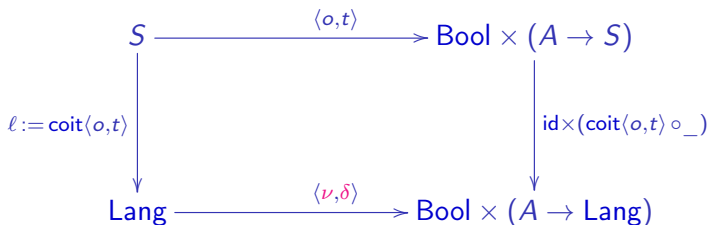
- Arbib & Manes (1986), Rutten (1998), Traytel (2016).
- Automaton structure over set of states S :

$$\begin{array}{ll}
 o & : S \rightarrow \text{Bool} & \text{“output”}: \text{acceptance} \\
 t & : S \rightarrow (A \rightarrow S) & \text{transition}
 \end{array}$$

- Automaton is coalgebra with $F(S) = \text{Bool} \times (A \rightarrow S)$.

$$\langle o, t \rangle : S \longrightarrow \text{Bool} \times (A \rightarrow S)$$

Formal Languages as Final Coalgebra



$$\nu \circ \ell = o \quad \text{“nullable”}$$

$$\nu (\ell s) = o s$$

$$\delta \circ \ell = (\ell \circ _) \circ t \quad \text{(Brzozowski) derivative}$$

$$\delta (\ell s) = \ell \circ (t s)$$

$$\delta (\ell s) a = \ell (t s a)$$

Languages – Rule-Based

- Coinductive tries Lang defined via observations/projections ν and δ :
- Lang is the greatest type consistent with these rules:

$$\frac{l : \text{Lang}}{\nu l : \text{Bool}} \qquad \frac{l : \text{Lang} \quad a : A}{\delta l a : \text{Lang}}$$

- Empty language $\emptyset : \text{Lang}$.
- Language of the empty word $\varepsilon : \text{Lang}$ defined by copattern matching:

$$\begin{aligned} \nu \varepsilon &= \text{true} && : \text{Bool} \\ \delta \varepsilon a &= \emptyset && : \text{Lang} \end{aligned}$$

Corecursion

- Empty language $\emptyset : \text{Lang}$ defined by corecursion:

$$\nu \emptyset = \text{false}$$

$$\delta \emptyset a = \emptyset$$

- Language union $k \cup l$ is pointwise disjunction:

$$\nu (k \cup l) = \nu k \vee \nu l$$

$$\delta (k \cup l) a = \delta k a \cup \delta l a$$

- Language composition $k \cdot l$ à la Brzozowski:

$$\nu (k \cdot l) = \nu k \wedge \nu l$$

$$\delta (k \cdot l) a = \begin{cases} (\delta k a \cdot l) \cup \delta l a & \text{if } \nu k \\ (\delta k a \cdot l) & \text{otherwise} \end{cases}$$

- Not accepted because \cup is not a constructor.

Bisimilarity

- Equality of infinite tries is defined coinductively.
- \cong is the greatest relation consistent with

$$\frac{l \cong k}{\nu l \equiv \nu k} \cong_{\nu} \quad \frac{l \cong k \quad a : A}{\delta l a \cong \delta k a} \cong_{\delta}$$

- Equivalence relation via provable \cong_{refl} , \cong_{sym} , and \cong_{trans} .

$$\begin{aligned} \cong_{\text{trans}} & : (p : l \cong k) \rightarrow (q : k \cong m) \rightarrow l \cong m \\ \cong_{\nu} (\cong_{\text{trans}} p q) & = \equiv \text{trans} (\cong_{\nu} p) (\cong_{\nu} q) : \nu l \equiv \nu k \\ \cong_{\delta} (\cong_{\text{trans}} p q) a & = \cong_{\text{trans}} (\cong_{\delta} p a) (\cong_{\delta} q a) : \delta l a \cong \delta m a \end{aligned}$$

- Congruence for language constructions.

$$\frac{k \cong k' \quad l \cong l'}{(k \cup k') \cong (l \cup l')} \cong_{\cup}$$

Proving bisimilarity

- Composition distributes over union.

$$\text{dist} : \forall k \ l \ m. \ k \cdot (l \cup m) \cong (k \cdot l) \cup (k \cdot m)$$

- Proof. Observation $\delta _ a$, case k nullable, l not nullable.

$$\begin{aligned}
 & \delta(k \cdot (l \cup m)) a \\
 &= \boxed{\delta k a \cdot (l \cup m)} \cup \delta(l \cup m) a && \text{by definition} \\
 &\cong \boxed{(\delta k a \cdot l \cup \delta k a \cdot m)} \cup (\delta l a \cup \delta m a) && \text{by coind. hyp. (wish)} \\
 &\cong (\delta k a \cdot l \cup \delta l a) \cup (\delta k a \cdot m \cup \delta m a) && \text{by union laws} \\
 &= \delta((k \cdot l) \cup (k \cdot m)) a && \text{by definition}
 \end{aligned}$$

- Formal proof attempt.

$$\cong \delta \text{ dist } a = \cong_{\text{trans}} (\cong \cup \boxed{\text{dist}} \dots) \dots$$

- Not coiterative / guarded by constructors!

Construction of greatest fixed-points

- Iteration to greatest fixed-point.

$$\top \supseteq F(\top) \supseteq F^2(\top) \supseteq \dots \supseteq F^\omega(\top) = \bigcap_{n < \omega} F^n(\top)$$

- Naming $\nu^i F = F^i(\top)$.

$$\begin{aligned} \nu^0 F &= \top \\ \nu^{n+1} F &= F(\nu^n F) \\ \nu^\omega F &= \bigcap_{n < \omega} \nu^n F \end{aligned}$$

- Deflationary iteration.

$$\nu^i F = \bigcap_{j < i} F(\nu^j F)$$

Sized coinductive types

- Add to syntax of type theory

Size	type of ordinals
i	ordinal variables
$\nu^i F$	sized coinductive type
Size < i	type of ordinals below i

- Bounded quantification $\forall j < i. A = (j : \text{Size} < i) \rightarrow A$.
- Well-founded recursion on ordinals, roughly:

$$\frac{f : \forall i. (\forall j < i. \nu^j F) \rightarrow \nu^i F}{\text{fix } f : \forall i. \nu^i F}$$

Sized coinductive type of languages

- $\text{Lang } i \cong \text{Bool} \times (\forall j < i. A \rightarrow \text{Lang } j)$

$$\frac{l : \text{Lang } i}{\nu l : \text{Bool}} \quad \frac{l : \text{Lang } i \quad j < i \quad a : A}{\delta l \{j\} a : \text{Lang } j}$$

- $\emptyset : \forall i. \text{Lang } i$ by copatterns and induction on i :

$$\begin{aligned} \nu(\emptyset \{i\}) &= \text{false} : \text{Bool} \\ \delta(\emptyset \{i\}) \{j\} a &= \emptyset \{j\} : \text{Lang } j \end{aligned}$$

- Note $j < i$.
- On right hand side, $\emptyset : \forall j < i. \text{Lang } j$ (coinductive hypothesis).

Type-based guardedness checking

- Union preserves size/guardedness:

$$\frac{k : \text{Lang } i \quad l : \text{Lang } i}{k \cup l : \text{Lang } i}$$

$$\begin{aligned} \nu(k \cup l) &= \nu k \vee \nu l \\ \delta(k \cup l) \{j\} a &= \delta k \{j\} a \cup \delta l \{j\} a \end{aligned}$$

- Composition is accepted and also guardedness-preserving:

$$\frac{k : \text{Lang } i \quad l : \text{Lang } i}{k \cdot l : \text{Lang } i}$$

$$\begin{aligned} \nu(k \cdot l) &= \nu k \wedge \nu l \\ \delta(k \cdot l) \{j\} a &= \begin{cases} (\delta k \{j\} a \cdot l) \cup \delta l \{j\} a & \text{if } \nu k \\ (\delta k \{j\} a \cdot l) & \text{otherwise} \end{cases} \end{aligned}$$

Guardedness-preserving bisimilarity proofs

- Sized bisimilarity \cong is greatest family of relations consistent with

$$\frac{l \cong^i k}{\nu l \equiv \nu k} \cong \nu \quad \frac{l \cong^i k \quad j < i \quad a : A}{\delta l a \cong^j \delta k a} \cong \delta$$

- Equivalence and congruence rules are guardedness preserving.

$$\begin{aligned} \cong \text{trans} & : (p : l \cong^i k) \rightarrow (q : k \cong^i m) \rightarrow l \cong^i m \\ \cong \nu (\cong \text{trans } p q) & = \equiv \text{trans } (\cong \nu p) (\cong \nu q) : \nu l \equiv \nu k \\ \cong \delta (\cong \text{trans } p q) j a & = \cong \text{trans } (\cong \delta p j a) (\cong \delta q j a) : \delta l a \cong^j \delta m a \end{aligned}$$

- Coinductive proof of `dist` accepted.

$$\cong \delta \text{ dist } j a = \cong \text{trans } j (\cong \cup \boxed{(\text{dist } j)} (\cong \text{refl } j)) \dots$$

Conclusions

- Tracking guardedness in types allows
 - natural modular corecursive definition
 - natural bisimilarity proof using equation chains
- Implemented in Agda (ongoing)
- Abel et al (POPL 13): Copatterns [2]
- Abel/Pientka (ICFP 13): Well-founded recursion with copatterns [1]

References I



Andreas Abel and Brigitte Pientka.

Wellfounded recursion with copatterns: A unified approach to termination and productivity.

In *ICFP'13*, pages 185–196. ACM, 2013.



Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer.

Copatterns: Programming infinite structures by observations.

In *POPL'13*, pages 27–38. ACM, 2013.






Robin Cockett and Tom Fukushima.

About Charity.

Technical report, Department of Computer Science, The University of Calgary, 1992.

Yellow Series Report No. 92/480/18.

References II

-  Tatsuya Hagino.
A Categorical Programming Language.
PhD thesis, University of Edinburgh, 1987.
-  John Hughes, Lars Pareto, and Amr Sabry.
Proving the correctness of reactive systems using sized types.
In *POPL'96*, pages 410–423. ACM, 1996.
-  Dexter Kozen and Alexandra Silva.
Practical coinduction.
MSCS, FirstView:1–21, 2016.

References III



Jorge Luis Sacchini.

Type-based productivity of stream definitions in the calculus of constructions.

In *LICS'13*, pages 233–242. IEEE CS Press, 2013.



Dmitriy Traytel.

Formal languages, formally and coinductively.

In *FSCD'16*, volume 52 of *LIPICs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.