Type Theory

Coinduction in Type Theory

Andreas Abel

Department of Computer Science and Engineering
Chalmers and Gothenburg University

Type Theory – Course CM0859 (2017-1)
Universidad EAFIT, Medellin, Colombia
6-10 March 2017

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

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, ...

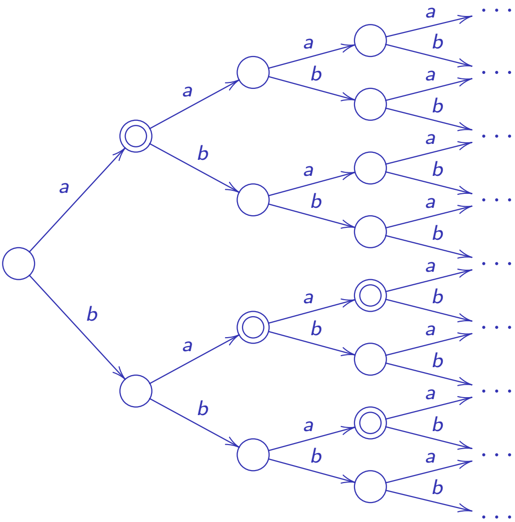
Running Example: Even binary numbers

- 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 $E = \{a, ba, baa, bba, baaa, baba, \ldots\}$.

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.

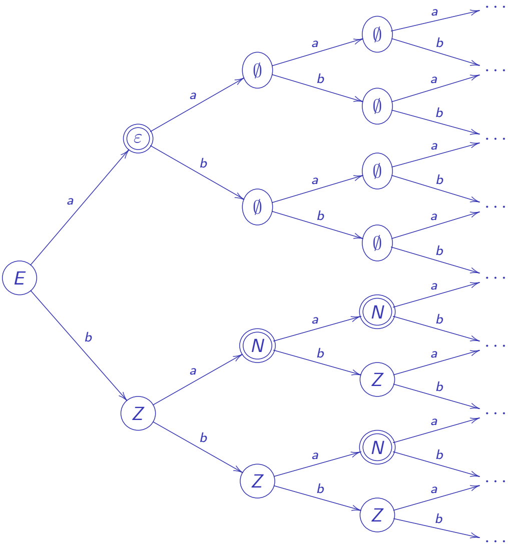
Trie of E



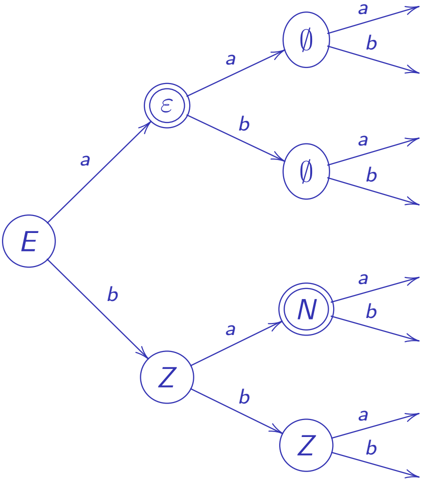
Regular Languages

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

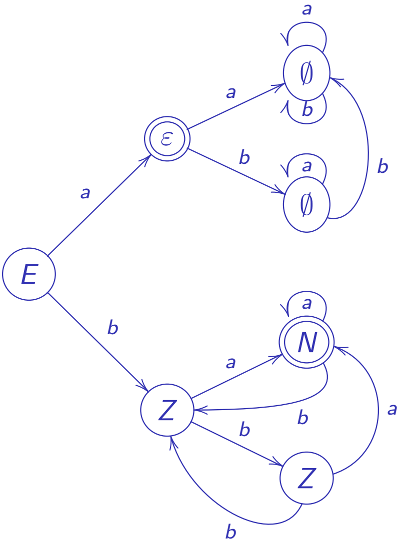
E	even binary numbers
Z	strings ending in a
N	strings not ending in b
ε	the empty string
$\emptyset$	nothing (empty language)


E
ε
Z
∅
N
Z
a
b
. . .

Cutting duplications at depth 3



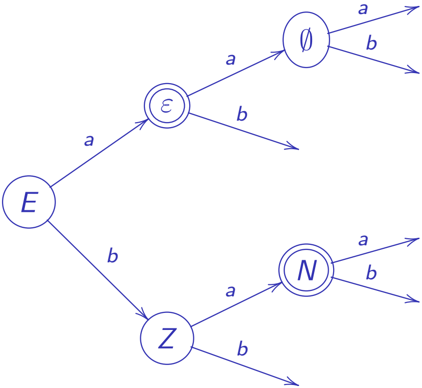
Bending branches . . .



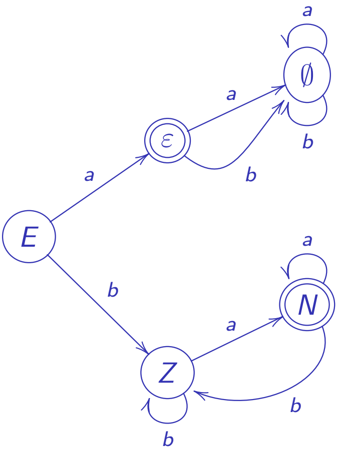
Finite Automata

- 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
  1. A set of states S.
  2. A function $\nu : S \to \mathsf{Bool}$ singling out the accepting states.
  3. A transition function $\delta : S \to A \to S$.

$s \in S$	νs	$\delta\, s\, a$	$\delta\, s\, b$
E	✗	ε	Z
ε	✓	$\emptyset$	$\emptyset$
$\emptyset$	✗	$\emptyset$	$\emptyset$
Z	✗	N	Z
N	✓	N	Z

- Language automaton
  1. State = language ℓ accepted when starting from that state.
  2. $\nu\ell$: Language ℓ is nullable (accepts the empty word)?
  3. $\delta \ell a = \{w \mid aw \in \ell\}$: Brzozowski derivative.

Differential equations

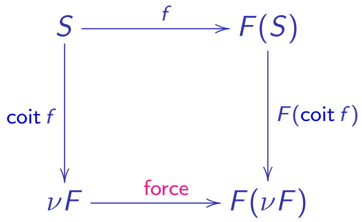
- Language E and friends can be specified by *differential equations*:
- ν gives the *initial value*.

$$
\begin{array}{lcl}
\nu\,\emptyset & = & \mathsf{false} \\
\delta\,\emptyset\,x & = & \emptyset \\
\\
\nu\,\varepsilon & = & \mathsf{true} \\
\delta\,\varepsilon\,x & = & \emptyset \\
\\
\nu\,E & = & \mathsf{false} \\
\delta\,E\,a & = & \varepsilon \\
\delta\,E\,b & = & Z
\end{array}
\qquad
\begin{array}{lcl}
\nu\,N & = & \mathsf{true} \\
\delta\,N\,a & = & N \\
\delta\,N\,b & = & Z \\
\\
\nu\,Z & = & \mathsf{false} \\
\delta\,Z\,a & = & N \\
\delta\,Z\,b & = & Z
\end{array}
$$

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

Final Coalgebras

- (Weakly) final coalgebra.



- Coiteration = finality witness.

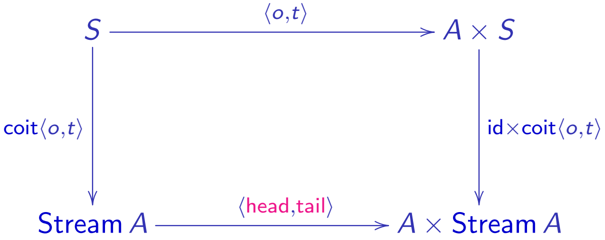
$$\mathsf{force} \circ \mathsf{coit}\ f = F\,(\mathsf{coit}\ f) \circ f$$

- Copattern matching *defines* coit by corecursion:

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

Streams as Final Coalgebra

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



- Termination by induction on observation depth:

$$\begin{array}{lcl} \mathsf{head}\ (\mathsf{coit}\ \langle o, t\rangle\ s) & = & o\ s \\ \mathsf{tail}\ \ (\mathsf{coit}\ \langle o, t\rangle\ s) & = & \mathsf{coit}\ \langle o, t\rangle\ (t\ s) \end{array}$$

Automata as Coalgebra

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

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

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

$$\langle o, t \rangle \quad : \quad S \longrightarrow \mathsf{Bool} \times (A \to S)$$

Formal Languages as Final Coalgebra

$$
\begin{array}{ccc}
S & \xrightarrow{\quad\langle o,t\rangle\quad} & \mathsf{Bool} \times (A \to S) \\
{\scriptstyle \ell := \mathsf{coit}\langle o,t\rangle}\big\downarrow & & \big\downarrow{\scriptstyle \mathsf{id}\times(\mathsf{coit}\langle o,t\rangle \circ \_)} \\
\mathsf{Lang} & \xrightarrow{\quad\langle \nu,\delta\rangle\quad} & \mathsf{Bool} \times (A \to \mathsf{Lang})
\end{array}
$$

$\nu \circ \ell$	$=$	o	"nullable"
$\nu\ (\ell\, s)$	$=$	$o\, s$	
$\delta \circ \ell$	$=$	$(\ell \circ \_) \circ t$	(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:

$$\frac{l : \mathsf{Lang}}{\nu\, l : \mathsf{Bool}} \qquad \frac{l : \mathsf{Lang} \qquad a : A}{\delta\, l\, a : \mathsf{Lang}}$$

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

$$\begin{array}{lcll} \nu\,\varepsilon & = & \mathsf{true} & : \mathsf{Bool} \\ \delta\,\varepsilon\, a & = & \emptyset & : \mathsf{Lang} \end{array}$$

Corecursion

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

$$\begin{array}{lcl} \nu\,\emptyset & = & \mathsf{false} \\ \delta\,\emptyset\,a & = & \emptyset \end{array}$$

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

$$\begin{array}{lcl} \nu\,(k \cup l) & = & \nu\,k \vee \nu\,l \\ \delta\,(k \cup l)\,a & = & \delta\,k\,a \cup \delta\,l\,a \end{array}$$

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

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

- Not accepted because $\cup$ is not a constructor.

Bisimilarity

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

$$\frac{l \cong k}{\nu\, l \equiv \nu\, k}\ \cong\nu \qquad \frac{l \cong k \qquad a : A}{\delta\, l\, a \cong \delta\, k\, a}\ \cong\delta$$

- Equivalence relation via provable ≅refl, ≅sym, and ≅trans.

$$\begin{array}{lcll}
\cong\text{trans} & : & (p : l \cong k) \to (q : k \cong m) \to l \cong m & \\
\cong\nu\,(\cong\text{trans}\, p\, q) & = & \equiv\text{trans}\,(\cong\nu\, p)\,(\cong\nu\, q) & : \ \nu\, l \equiv \nu\, k \\
\cong\delta\,(\cong\text{trans}\, p\, q)\, a & = & \cong\text{trans}\,(\cong\delta\, p\, a)\,(\cong\delta\, q\, a) & : \ \delta\, l\, a \cong \delta\, m\, a
\end{array}$$

- Congruence for language constructions.

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

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.

$$
\begin{array}{lllr}
\multicolumn{3}{l}{\delta\,(k \cdot (l \cup m))\,a} & \\
= & \boxed{\delta\,k\,a \cdot (l \cup m)} & \cup\ \delta\,(l \cup m)\,a & \text{by definition} \\
\cong & \boxed{(\delta\,k\,a \cdot l \cup \delta\,k\,a \cdot m)} & \cup\ (\delta\,l\,a \cup \delta\,m\,a) & \text{by coind. hyp. (wish)} \\
\cong & \multicolumn{2}{l}{(\delta\,k\,a \cdot l \cup \delta\,l\,a) \cup (\delta\,k\,a \cdot m \cup \delta\,m\,a)} & \text{by union laws} \\
= & \multicolumn{2}{l}{\delta\,((k \cdot l) \cup (k \cdot m))\,a} & \text{by definition}
\end{array}
$$

- Formal proof attempt.

$$\cong\!\delta\ \text{dist}\ a \;=\; \cong\!\text{trans}\ (\cong\!\cup\ \boxed{\text{dist}}\ \ldots)\ \ldots$$

- Not coiterative / guarded by constructors!

Construction of greatest fixed-points

- Iteration to greatest fixed-point.

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

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

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

- Deflationary iteration.

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

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 $\forall j{<}i.\, A = (j : \mathsf{Size}{<}\, i) \to A$.
- Well-founded recursion on ordinals, roughly:

$$\frac{f : \forall\, i.\, (\forall\, j{<}i.\, \nu^j F) \to \nu^i F}{\mathsf{fix}\, f : \forall\, i.\, \nu^i F}$$

Sized coinductive type of languages

- $\mathsf{Lang}\, i \cong \mathsf{Bool} \times (\forall j{<}i.\ A \to \mathsf{Lang}\, j)$

$$\frac{l : \mathsf{Lang}\, i}{\nu\, l : \mathsf{Bool}} \qquad \frac{l : \mathsf{Lang}\, i \qquad j < i \qquad a : A}{\delta\, l\, \{j\}\, a : \mathsf{Lang}\, j}$$

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

$$\begin{array}{lcl@{\quad}c@{\quad}l} \nu\, (\emptyset\, \{i\}) & = & \mathsf{false} & : & \mathsf{Bool} \\ \delta\, (\emptyset\, \{i\})\, \{j\}\, a & = & \emptyset\, \{j\} & : & \mathsf{Lang}\, j \end{array}$$

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

Type-based guardedness checking

- Union preserves size/guardeness:

$$\frac{k : \mathsf{Lang}\, i \qquad l : \mathsf{Lang}\, i}{k \cup l : \mathsf{Lang}\, i}$$

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

- Composition is accepted and also guardedness-preserving:

$$\frac{k : \mathsf{Lang}\, i \qquad l : \mathsf{Lang}\, i}{k \cdot l : \mathsf{Lang}\, i}$$

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

Guardedness-preserving bisimilarity proofs

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

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

- Equivalence and congruence rules are guardedness preserving.

$$\begin{array}{lcll}
\cong\mathsf{trans} & : & (p : l \cong^i k) \to (q : k \cong^i m) \to l \cong^i m & \\
\cong\nu\,(\cong\mathsf{trans}\, p\, q) & = & \equiv\mathsf{trans}\,(\cong\nu\, p)\,(\cong\nu\, q) & : \ \nu\, l \equiv \nu\, k \\
\cong\delta\,(\cong\mathsf{trans}\, p\, q)\, j\, a & = & \cong\mathsf{trans}\,(\cong\delta\, p\, j\, a)\,(\cong\delta\, q\, j\, a) & : \ \delta\, l\, a \cong^j \delta\, m\, a
\end{array}$$

- Coinductive proof of dist accepted.

$$\cong\delta\ \mathsf{dist}\ j\ a\ =\ \cong\mathsf{trans}\ j\ (\cong\cup\ \boxed{(\mathsf{dist}\ \ \mathrm{j})}\ (\cong\mathsf{refl}\ j))\ \ldots$$

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.