Formal Languages, Coinductively Formalized

Andreas Abel

Department of Computer Science and Engineering
Chalmers and Gothenburg University

Termination and Circular Proofs
Department of Mathematics (LAMA)
Université Savoie Mont Blanc, Chambery, France
19 July 2017

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, ooc, ...

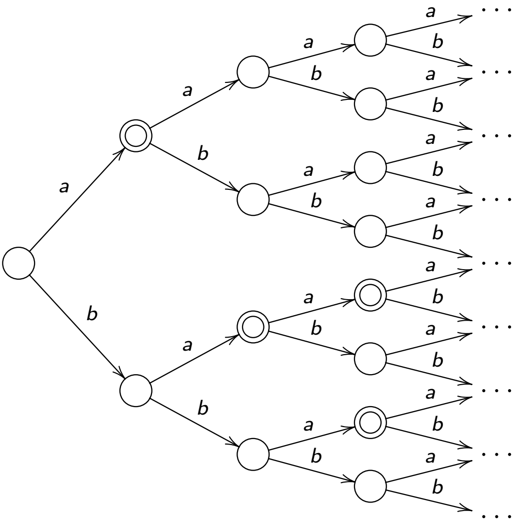
Running Example: Even binary numbers

- 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 $a \in A$.
- Languages: representable by infinite Bool-labelled tries.
- 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.
- A trie memoizes a function $f : \mathrm{List}\, A \to \mathsf{Bool}$.

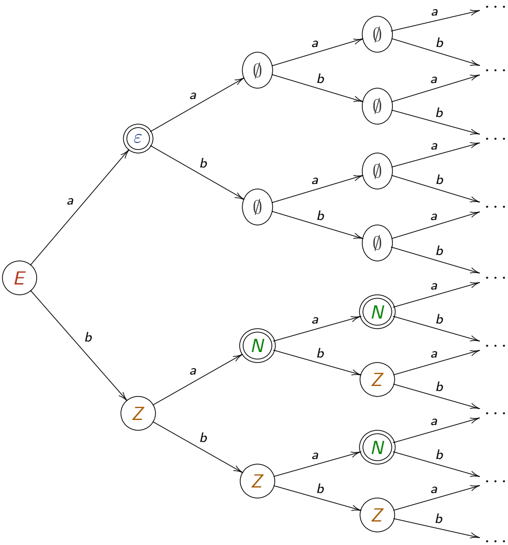
Trie of E



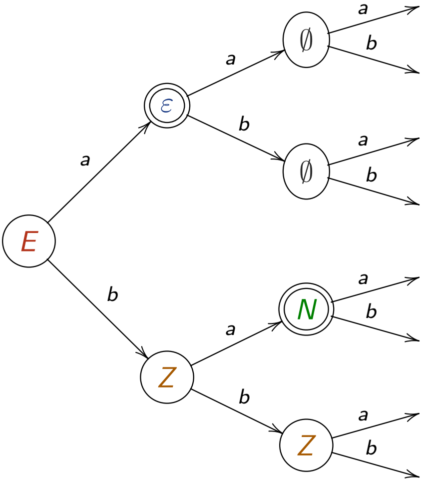
Regular Languages

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

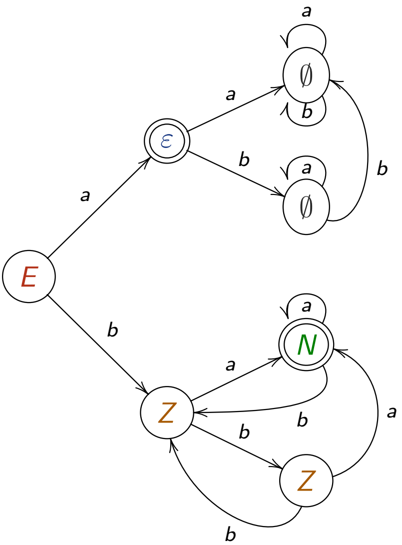
E	even binary numbers
Z	strings ending in a
N	strings not ending in b
ε	the empty string
$\emptyset$	nothing (empty language)


E
a
b
ε
Z
∅
N
Z
a
b
...

Cutting duplications at depth 3



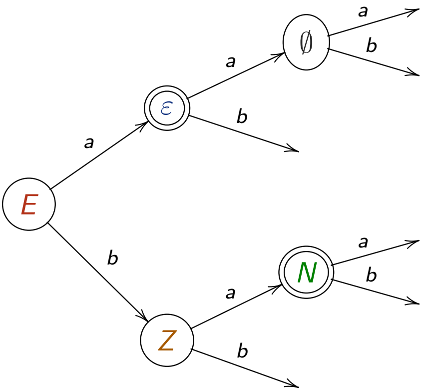
Bending branches . . .



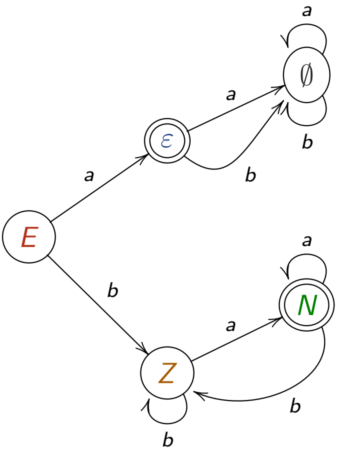
Final Automata

- We have arrived at a familiar object: a final 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

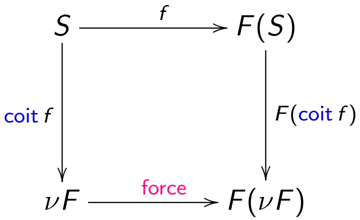
- Language E and friends can be specified by *differential equations*:
- ν gives the *initial value*.

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

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

Final Coalgebras

- (Weakly) final coalgebra.



- Coiteration = finality witness.

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

- Copattern matching *defines* coit by corecursion:

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

Automata as Coalgebra

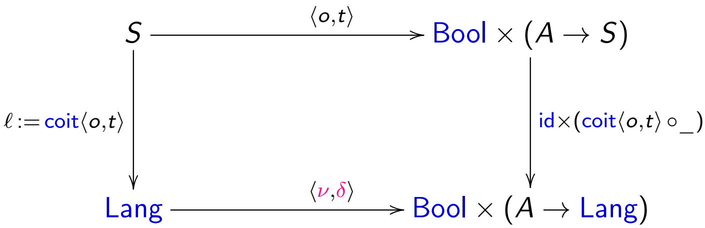
- Arbib & Manes (1986), Rutten (1998), Traytel (2016).
- Automaton structure over set of states S:

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

- 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



$$\begin{aligned} \nu \circ \ell &= o \\ \nu\ (\ell\, s) &= o\, s \end{aligned}$$

"nullable"

$$\begin{aligned} \delta \circ \ell &= (\ell \circ \_) \circ t \\ \delta\ (\ell\, s) &= \ell \circ (t\, s) \\ \delta\ (\ell\, s)\ a &= \ell\,(t\, s\, a) \end{aligned}$$

(Brzozowski) derivative

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.

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

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

- Deflationary iteration.

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

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:

$$\begin{array}{lcll} \nu\,(\emptyset\,\{i\}) & = & \mathsf{false} & : \mathsf{Bool} \\ \delta\,(\emptyset\,\{i\})\,\{j\}\,a & = & \emptyset\,\{j\} & : \mathsf{Lang}\,j \end{array}$$

- 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:

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

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

(Not Necessarily Finite) Automata

- Recapitulate automata à la Rutten (1998):

S	:	Set	state set (could be infinite)
ν	:	$S \to$ Bool	accepting state?
δ	:	$S \times A \to S$	transition function

- Automaton is record/object.

```
record DA (S : Set) : Set where
  field ν : (s : S) → Bool
        δ : (s : S) (a : A) → S

  νs : ∀{i} (ss : List i S) → Bool
  νs ss = List.any ν ss

  δs : ∀{i} (ss : List i S) (a : A) → List i S
  δs ss a = List.map (λ s → δ s a) ss
```

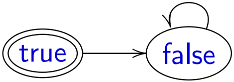
Constructing Automata

- Automaton for the empty language $\emptyset$:

$$
\begin{aligned}
&\emptyset A : DA\ \top \\
&\nu\ \ \emptyset A\ \ s\ \ \ \ \ = \text{false} \\
&\delta\ \ \emptyset A\ \ s\ \ a = s
\end{aligned}
$$

- Automaton for the empty word ε:

$$
\begin{aligned}
&\varepsilon A : DA\ \text{Bool} \\
&\nu\ \ \varepsilon A\ \ b\ \ \ \ \ = b \\
&\delta\ \ \varepsilon A\ \ b\ \ a = \text{false}
\end{aligned}
$$

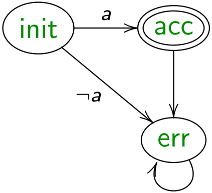


Constructing Automata

Accepting a specific character a.

```
data 3States : Set where
  init acc err : 3States

charA : (a : A) → DA 3States
ν (charA a) init   = false
ν (charA a) acc    = true
ν (charA a) err    = false
δ (charA a) init x =
  if ⌊ a ≟ x ⌋ then acc else err
δ (charA a) acc x  = err
δ (charA a) err x  = err
```



Unioning Automata

Union automaton.

- Transition in lock-step in $S_1 \times S_2$.
- Accept if s_1 or s_2 is accepting.

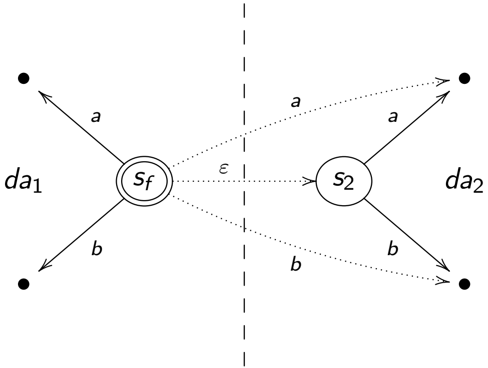
$$
\begin{array}{l}
\_\oplus\_ : \forall\{S_1\ S_2\}\ (da_1 : \mathsf{DA}\ S_1)\ (da_2 : \mathsf{DA}\ S_2) \to \mathsf{DA}\ (S_1 \times S_2) \\
\nu\ \ (da_1 \oplus da_2)\ \ (s_1\ ,\ s_2)\ \ \ \ = \nu\ da_1\ s_1\ \ \vee\ \ \nu\ da_2\ s_2 \\
\delta\ \ (da_1 \oplus da_2)\ \ (s_1\ ,\ s_2)\ \ a\ = \delta\ da_1\ s_1\ a\ \ ,\ \ \delta\ da_2\ s_2\ a
\end{array}
$$

Power automaton: being in several states at the same time.

$$
\begin{array}{l}
\mathsf{powA} : \forall\{S\}\ (da : \mathsf{DA}\ S) \to \mathsf{DA}\ (\mathsf{List}\ \infty\ S) \\
\nu\ \ (\mathsf{powA}\ da)\ \ ss\ \ \ \ = \nu\mathsf{s}\ da\ ss \\
\delta\ \ (\mathsf{powA}\ da)\ \ ss\ \ a\ = \delta\mathsf{s}\ da\ ss\ a
\end{array}
$$

Automaton for Language Composition

Compose two automata, picking initial state s_2 of da_2.



Automaton for Language Composition

A composed state is one $s_1 : S_1$ and possibly several $ss_2 \subset S_2$.

$$\text{composeA} : \forall \{S_1\ S_2\}$$
$$(da_1 : \text{DA}\ S_1)\ (s_2 : S_2)\ (da_2 : \text{DA}\ S_2) \to \text{DA}\ (S_1 \times \text{List}\ \infty\ S_2)$$

We accept if in a final state in S_2
or the final state S_1 if the initial state in S_2 is accepting.

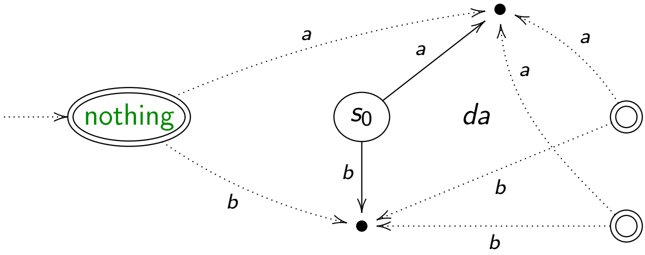
$$\nu\ \ (\text{composeA}\ da_1\ s_2\ da_2)\ \ (s_1\ ,\ ss_2)\ \ =$$
$$(\nu\ da_1\ s_1 \wedge \nu\ da_2\ s_2) \vee \nu s\ da_2\ ss_2$$

If in final state in S_1 we also transition from initial state in S_2.

$$\delta\ \ (\text{composeA}\ da_1\ s_2\ da_2)\ \ (s_1\ ,\ ss_2)\ \ a\ \ =$$
$$\delta\ da_1\ s_1\ a\ ,\ \delta s\ da_2\ (\text{if}\ \nu\ da_1\ s_1\ \text{then}\ s_2 :: ss_2\ \text{else}\ ss_2)\ a$$

Automaton for Language Iteration

- Kleene star of automaton with initial state s_0.
- New initial (and final state) nothing.



- Additionally transit from final states to successors of s_0.

Automata: Taking Stock

- We now can translate regular expressions to deterministic automata.
- Model implementations of automata very direct.
- All constructions preserve finiteness.
- TODO: connect to efficient implementation.
- All constructions have been formally verified in Agda.

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.

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

- Formal proof attempt.

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

- Not coiterative / guarded by constructors!

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.

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

- Coinductive proof of dist accepted.

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

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
- Abel/Pientka (ICFP 13): Well-founded recursion with copatterns
- Abel (CMCS 16): Equational Reasoning about Formal Languages in Coalgebraic Style

Related work

- Hagino (1987): Coalgebraic types
- Cockett et al.: Charity
- Dmitriy Traytel (PhD TU Munich, 2015): Languages coinductively in Isabelle
- Kozen, Silva (2016): Practical coinduction
- Hughes, Pareto, Sabry (POPL 1996)
- Papers on sized types (1998–2015): e.g. Sacchini (LICS 2013)