2012年7月26日
12:06

Chap. 3

Categorical SOS and Bialgebras

Plan

- Concurrency, process theory
- SOS
- Compositionality : opr. sem. w/ dem. reasoning

References

Bartek Klin, TCS 2011

2012年7月26日
12:08

§3.1 Introduction.

Computer Science

finitary formalism,
representing
(possibly) infinitary behaviors

E.g.

- a while (imperative) program, ← finite string
  and its execution ← possibly non-term
- a DFA ← finite
  and its accepted language ← infinite, w/ arbitrary long words

2012年7月26日
12:12

- A Büchi autom. and
  the ω-language it recognizes
  ↑
  $A \subseteq L^\omega$
- A higher-order recursion scheme and the tree language it produces
- (... and of course)
  a λ-term and its reduction sequence

2012年7月26日
12:14

A central question

- Given a finitary representation (a program, an autom., ...),
  what is its (infinitary) behavior?

"Now notice that this question is ill-formulated ... We mortal humans are incapable of writing down infinite behaviors ...

The central question, refined and often asked

1 Given two presentations M, N, do they exhibit the same behavior?

($M =_D N$, $[\![M]\!] = [\![N]\!]$, $M \cong_{CTX} N$, $M \overset{\cdot}{\sim} N$ (bisim), ...)

[Fondation of program transformation, automata minimization, etc.

° same job, more efficiently

2 Given a presentation M, does its behavior satisfy a given specification P?

[e.g. P is
- strongly normalizing (termination)
- never produces a label a ($G(\neg a)$)
- after a occurs, b occurs eventually ($G(a \supset Fb)$)

specification in modal (temporal) logic

Anyway To answer such questions, we need a mathematical def. of the behavior of a presentation M.

This is what SEMANTICS (in CS) is about.

"What is its 'meaning'?"

Two common styles of semantics:
- denotational sem.
  mathematical / algebraic, abstract, easy to reason with
- operational sem.
  concrete, akin to actual implem.

---

cf Winskel, Formal Semantics of Prog. Languages (MIT Press)

The distinction is not clear-cut...

[e.g.
- Is game semantics den. or opr.?
- In what follows, we see
  initial alg. sem. (≒ den.)
  final coalg. sem. (≒ opr.)
  coincide in lucky situations

[Hoare, in early years]

"Once denotational model is available, (nasty) operational models should immediately be thrown away"

2012年7月27日
9:25

Structural Operational Semantics

aims at bringing a mathematical order to operational semantics. (SOS)

- [Plotkin '81] First appearance
- Used e.g. for def. of the (oppr.) sem. of ML, but
  ↳ i.e. language specification
- is much more widely used for process calculi
  ↳ CCS, CSP, ACP, π-cal., ... simple prog. lang. for Concurrent systems/processes
- [Turi, Plotkin '97] Categorical formulation via alg. & coalg.

goal of this part

2012年7月27日
9:34

SOS The first example

- The process algebra:

$$p ::= 0 \mid a.p \mid p + p \mid p \parallel p$$

- 0 ← Termination
- $a.p$ ← action prefix ($a \in L$) "do a and then do p"
- $p + p$ ← nondeterministic choice "choose one and do it"
- $p \parallel p$ ← parallel composition "do both in a concurrent manner"

One can also include recursive definitions ⇒ infinitary behaviors

For simplicity we don't do that now

2012年7月27日
9:38

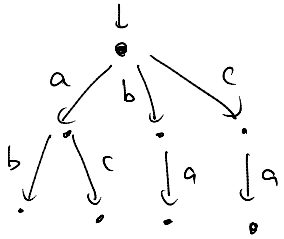
We define the (operational) sem. for this process alg. using

LTS $\subseteq a \left[\text{labeled transition sys.}, \ \mathcal{P}_{fin}(L \times X)^X \right]$.

e.g.

$[\![a.0 \parallel (b.0 + c.0)]\!]$ = "the meaning of —"

:=



2012年7月27日
9:46

Q How to define $[\![-]\!]$ in a math. rigorous manner?

The SOS answer is as follows.

1 You specify the 'meaning' of process operators $(a.-, +, \parallel, \ldots)$ by means of SOS rules

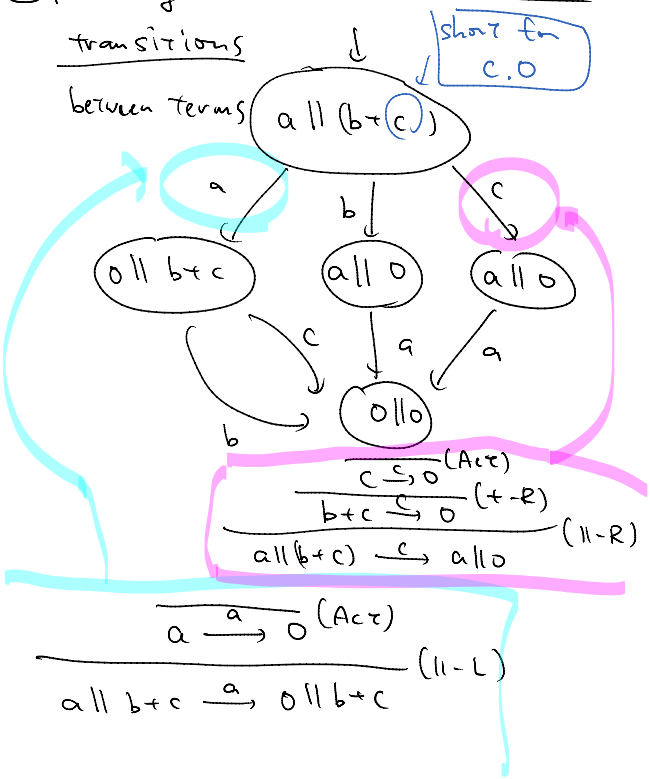
$$\frac{}{a.x \xrightarrow{a} x} \text{(Act)}$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} (+\text{-L}) \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} (+\text{-R})$$

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} (\parallel\text{-L}) \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} (\parallel\text{-R})$$

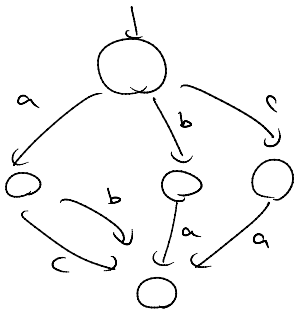
2012年7月27日
9:54

[2] Using these rules, you derive transitions between terms

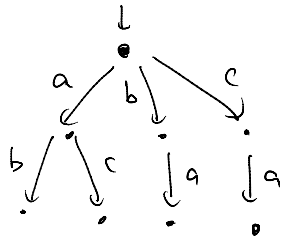


2012年7月27日
9:58

[3] the resulting LTS



is bisimilar to the one 3 pages ago:



2012年7月28日
18:38

Moreover, a desirable property:

Compositionality
(Modularity, 「要素還元性」
"Bisimilarity is a congruence")
For each process opr. σ,
$[\![t_i]\!]$ and $[\![s_i]\!]$ are bisimilar
("equivalence of LTSs")

$\Longrightarrow$ $[\![\sigma(t_1,\dots,t_n)]\!]$ and $[\![\sigma(s_1,\dots,s_n)]\!]$ are bisimilar.

- Enables algebraic reasoning:

$$\frac{t_1 \sim s_1 \quad \dots \quad t_n \sim s_n}{\sigma(t_1,\dots,t_n) \sim \sigma(s_1,\dots,s_n)}$$

($\sim$ denotes bisim. LTSs)

- Replaceability, maintainability, ...
- Typical property of denotational semantics

2012年7月28日
18:42

Fact There are so-called syntactic formats (GSOS, tyft, De Simone, ...) w/ following (meta) results:

(templates for SOS rules)

- If all the SOS rules adhere to the format,
- Then the induced $[\![-]\!]$ always is compositional.

Such a metaresult (and discovery of such a syntactic format) will be the goal of the categorical/bialgebraic development.

2012年7月27日
9:59

We notice strong (co)algebraic flavor in SOS:

- {process terms} → a ∥ (b+c), ...
  is an initial algebra
- An LTS is a coalgebra
- "... is bisimilar" → coinduction
- $\dfrac{}{a \xrightarrow{a} 0}\,(\mathrm{Act}) \qquad \dfrac{a \parallel b+c \xrightarrow{a} 0 \parallel b+c}{a \parallel b+c \xrightarrow{a} 0 \parallel b+c}\,(\parallel\text{-L})$: inductive flavor

⇒ Bialgebraic modeling!

2012年7月27日
9:40

BTW. Process alg., concurrency

Their significance:

- Nowadays few computational tasks are sequential; most are parallel
  * the Internet
  * a multicore processor
  * HPC
- Parallelism / concurrency results in vast complexity
  * Nondeterminism is inevitable: "who goes first?"
  * n computing units ⇒ exp(n) complexity

2012年7月28日
18:26

§3.2 Bialgebraic Modeling: The simple setting

Here we present an (even simpler) example of SOS via bialgebras.

(Following [Klin, TCS 2011])

We fix:

- $\mathbb{V}\mathrm{ar}$, a countable set of metavariables
- Σ: an algebraic signature (identified with $\Sigma : \mathbf{Sets} \to \mathbf{Sets}, \quad X \mapsto \coprod_{\sigma \in \Sigma} X^{|\sigma|}$)
- L, a set of labels

2012年7月28日
18:30

det. by Σ

We consider the process alg. (i.e. a simple progr. lang.) that is for expressing L-streams

$a_0 a_1 a_2 \ldots \in L^\omega$.

Notice that

- $T_\Sigma 0 = \{\Sigma\text{-terms with no variables}\}$ carries an initial algebra

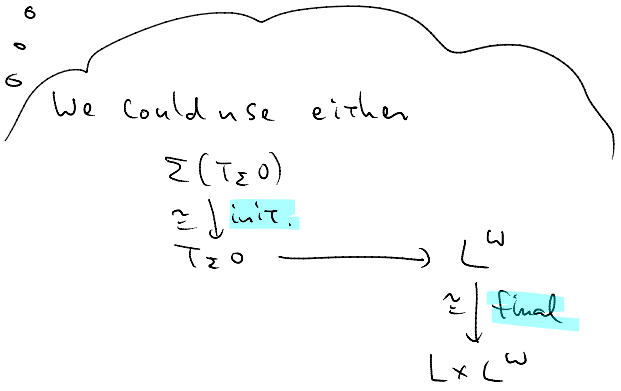
$$\begin{array}{c} \Sigma(T_\Sigma 0) \\ \cong \downarrow \text{init.} \\ T_\Sigma 0 \end{array}$$

- An $(L \times \_)$-coalgebra $\begin{array}{c} L \times X \\ c \uparrow \\ X \end{array}$ is a stream automaton; and its state $x \in X$ induces an L-stream $a_0 a_1 \ldots \in L^\omega$ by coinduction:

$$\begin{array}{ccc} L \times X & \dashrightarrow & L \times L^\omega \\ c \uparrow & & \cong \uparrow \text{final} \\ X & \xrightarrow[\text{beh}(c)]{} & L^\omega \\ x & \longmapsto & (\text{beh}(c))(x) \end{array}$$

Our goal Operational semantics of this process alg., that is (prog. lang.)

$$\begin{array}{ccc} T_\Sigma 0 & \xrightarrow{[\![-]\!]} & L^\omega \\ \cup & & \cup \\ (\text{process term}) & \longmapsto & (\text{stream}) \end{array}$$



As before, we start with SOS rules, now subject to a certain syntactic format:

Def. A simple stream SOS rule is

$$\frac{x_1 \xrightarrow{a_1} x_1' \quad \ldots \quad x_n \xrightarrow{a_n} x_n'}{f(x_1, \ldots, x_n) \xrightarrow{b} g(y_1, \ldots, y_m)}$$

where

- $f \in \Sigma_n$, $g \in \Sigma_m$ (operations)
- $x_1, \ldots, x_n, x_1', \ldots, x_n' \in \mathbb{V}\text{ar}$
- $y_j \in \{x_1', \ldots, x_n'\}$ for $j \in [1, m]$
- $b, a_1, \ldots, a_n \in L$.

More precisely: a simple str. SOS rule is

$$R = (f, g, (a_1, \ldots, a_n), b, \theta)$$

where $\theta : m \to n$ is a function.

Def. A simple stream SOS specification for Σ is a set Λ of stream SOS rules s.t.

for each $f \in \Sigma_n$ and each $a_1, \ldots, a_n \in L$, there is exactly one rule in Λ of the form

$$\frac{0 \xrightarrow{a_1} 0 \quad \ldots \quad 0 \xrightarrow{a_n} 0}{f(0, \ldots, 0) \to \textcircled{\scriptsize ///}}$$

Example - $L = \{a, b\}$

- $\Sigma_0 = \{c_a, c_b\}$ $\quad \Sigma_2 = \{alt\}$

"constantly a"

$\Sigma_1 = \Sigma_3 = \Sigma_4 = \cdots = 0$

- Λ consists of

$$\frac{}{c_a \xrightarrow{a} c_a} \qquad \frac{}{c_b \xrightarrow{b} c_b}$$

$$\frac{x_1 \xrightarrow{l_1} x_1' \qquad x_2 \xrightarrow{l_2} x_2'}{alt(x_1, x_2) \xrightarrow{l_1} alt(x_2', x_1')} \quad (\text{for each } l_1, l_2 \in L)$$

Then Λ is a simple str. SOS specif. for Σ.

The syntactic format is very much restrictive, e.g.

$$\frac{x_1 \xrightarrow{l_1} x_1'}{zip(x_1, x_2) \xrightarrow{l_1} (x_2, x_1')}$$

does not satisfy the (zip) restriction.

$zip(a_0 a_1 \ldots, b_0 b_1 \ldots) = a_0 b_0 a_1 b_1 \ldots$

Exercise What is the intention of the operator zip ?

Goals - Derive $[\![-]\!] : T_\Sigma 0 \to L^\omega$

- Show $[\![-]\!]$ is compositional

Crucial observation :

a simple str. SOS specification Λ

a natural transformation

$\Sigma F \Rightarrow F\Sigma$

("map of functors")

(where $F = L \times \_$)

[More generally :

A spec. subj. to a certain syntactic format

a natural transformation

$\Sigma F \Rightarrow F\Sigma$

- for many different F
- this can be more complex (later)

... finally we need to introduce nat. trans.!

Def. Let $\mathbb{C} \underset{G}{\overset{F}{\rightrightarrows}} \mathbb{D}$ be functors.

A natural transformation

$$\mathbb{C} \xrightarrow[G]{F} \mathbb{D} \quad (\Downarrow \alpha)$$

is a family

$$\{FX \xrightarrow{\alpha_X} GX\}_{X \in \mathbb{C}}$$

of $\mathbb{D}$-arrows.

In the old age it was sometimes written $\alpha : F \Rightarrow G : \mathbb{C} \to \mathbb{D}$

α_X : α's component at $X \in \mathbb{C}$

2012年7月28日
19:07

subject to the naturality condition

$\mathbb{C}$ | $\mathbb{D}$

$$\begin{array}{ccc} X & & FX \xrightarrow{\alpha_X} GX \\ \downarrow f & & Ff \downarrow \quad // \quad \downarrow Gf \\ Y & & FY \xrightarrow[\alpha_Y]{} GY \end{array}$$

Before exhibiting

a simple str. SOS specification Λ

a natural transformation

$$\Sigma F \Rightarrow F\Sigma$$

(where $F = L \times \_$)

we see why $\Sigma F \Rightarrow F\Sigma$ is useful in the current setting.

2012年7月28日
19:09

Assume we have obtained

$$\Sigma F \xrightarrow{\lambda} F\Sigma. \qquad (F = L \times \_)$$

Then:

- $\Sigma(T_\Sigma 0)$ ($\xi \downarrow$ init., $T_\Sigma 0$) has an F-coalg. structure, by

$$\begin{array}{ccc} \Sigma(T_\Sigma 0) & \dashrightarrow & \Sigma(F(T_\Sigma 0)) \\ & & \downarrow \lambda_{T_\Sigma 0} \\ \downarrow \text{init.} & & F(\Sigma(T_\Sigma 0)) \\ & & \downarrow F(\text{init.}) \\ T_\Sigma 0 & \dashrightarrow & F(T_\Sigma 0) \end{array}$$

- Which can be used in

$$\begin{array}{ccc} F(T_\Sigma 0) & \dashrightarrow & F(L^\omega) \\ \uparrow & & \uparrow \text{final} \\ T_\Sigma 0 & \xrightarrow{[\![-]\!]} & L^\omega \end{array}$$

2012年7月28日
19:14

(more on this is coming later)

Prop.

a simple str. SOS specification Λ

a natural transformation

$$\Sigma F \Rightarrow F\Sigma$$

(where $F = L \times \_$)

Proof.

[I] $\Sigma F \Rightarrow F\Sigma$

$$\frac{\coprod_{f \in \Sigma} (F(-))^{|f|} \Rightarrow F\Sigma}{(F(-))^{|f|} \Rightarrow F\Sigma \quad \text{for each } f \in \Sigma}$$

$(F(-))^{|f|} = (L \times (-))^{|f|}$ $\qquad F\Sigma = L \times (\coprod_{f \in \Sigma} (-)^{|f|})$

2012年7月28日
19:17

We define such functions by (Let $f \in \Sigma_n$)

$$(L \times S)^n \longrightarrow L \times \left(\coprod_{k \in \Sigma} S^{|k|}\right)$$

$$(a_1, s_1, \ldots, a_n, s_n) \longmapsto \left(a,\ k_g(s_{i_1}, \ldots, s_{i_m})\right)$$

where the rule in Λ corresponding to

$$\frac{o \xrightarrow{a_1} o' \quad \ldots \quad o \xrightarrow{a_n} o''}{f(o, \ldots, o) \longrightarrow \bigcirc}$$

is

$$\frac{x_1 \xrightarrow{a_1} x_1' \quad \ldots \quad x_n \xrightarrow{a_n} x_n'}{f(x_1, \ldots, x_n) \xrightarrow{a} g(x_{i_1}', \ldots, x_{i_m}')}$$

2012年7月28日
19:24

We need to check naturality:

$$\begin{array}{ccc} S & (L\times S)^n & \longrightarrow & L\times\left(\coprod_{h\in\Sigma} S^{|h|}\right) \\ \downarrow f & \downarrow (L\times f)^n & & \downarrow \\ S' & (L\times S')^n & \longrightarrow & L\times\left(\coprod_{h\in\Sigma}(S')^{|h|}\right) \end{array}$$

which is easy.

[↑] Given a natural transf.

$$\cfrac{\cfrac{\Sigma F \Rightarrow F\Sigma}{\coprod_{f\in\Sigma}(F(-))^{|f|}\Rightarrow F\Sigma}}{(F(-))^{|f|}\Rightarrow F\Sigma \quad \text{for each } f\in\Sigma}$$

$(F(-))^{|f|}$ = $(L\times(-))^{|f|}$; $F\Sigma$ = $L\times\left(\coprod_{h\in\Sigma}(-)^{|h|}\right)$

2012年7月28日
19:30

We fix $f\in\Sigma$, $a_1,\dots,a_{|f|}\in L$.

Take its component at $X':=\{x_1',\dots,x_{|f|}'\}$:

$$(L\times X')^{|f|} \xrightarrow[\text{this by } k]{\text{we denote}} L\times\coprod_{h\in\Sigma}(X')^{|h|}$$

and consider

$$k\left((a_1,x_1'),\dots,(a_{|f|},x_{|f|}')\right) =: \left(a,\ kg\left(x_{i_1}',\dots,x_{i_{|g|}}'\right)\right)$$

From this we define a rule

$$\frac{x_1\xrightarrow{a_1}x_1'\ \dots\ x_n\xrightarrow{a_n}x_n'}{f(x_1,\dots,x_n)\xrightarrow{a} g(x_{i_1}',\dots,x_{i_{|g|}}')}$$

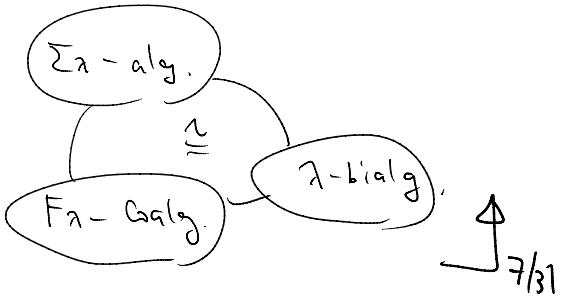
We do this for each f, $a_1,\dots,a_{|f|}$ and define a simple str. SOS spec.

2012年7月28日
19:35

It is not hard to see that [I] and [↑] are converse to each other ∎

Therefore we transformed a set of rules into an abstract SOS rule $\lambda:\Sigma F\Rightarrow F\Sigma$.

We shall now fully exploit this...



2012年7月28日
19:37

Prop. λ lifts $\Sigma:\mathbf{Sets}\to\mathbf{Sets}$ to

$$\Sigma\lambda:\mathbf{Coalg}_F\longrightarrow\mathbf{Coalg}_F,$$

that is,

$$\begin{array}{ccc} \mathbf{Coalg}_F & \xrightarrow{\Sigma\lambda} & \mathbf{Coalg}_F \\ \text{forget}\downarrow & = & \downarrow \\ \mathbf{Sets} & \xrightarrow[\Sigma]{} & \mathbf{Sets} \end{array}$$

Concretely:

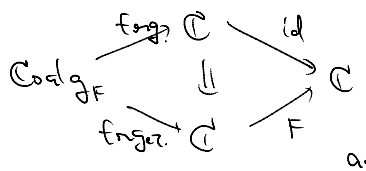
$$\begin{array}{ccc} \mathbf{Coalg}_F & \xrightarrow{\Sigma\lambda} & \mathbf{Coalg}_F \\ FX & \longmapsto & \begin{pmatrix} F\Sigma X \\ \uparrow\lambda_X \\ \Sigma FX \\ \uparrow\Sigma c \\ \Sigma X \end{pmatrix} \\ \uparrow c & & \\ X & & \end{array}$$

2012年7月28日
19:46

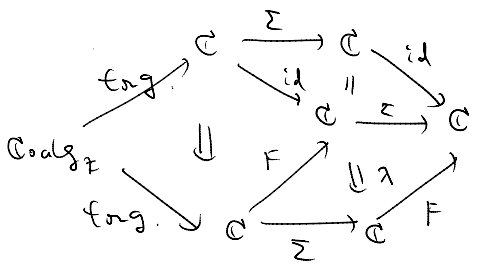
Exercise Write down $\Sigma\lambda$'s action on arrows (use naturality of λ)

Proof. Straightforward. □

A 2-categorical view:

- 

$\mathbb{C}$ is an inserter.

- 

induces

$$\mathrm{Coalg}_F \xrightarrow{\Sigma\lambda} \mathrm{Coalg}_F.$$

2012年7月28日
19:50

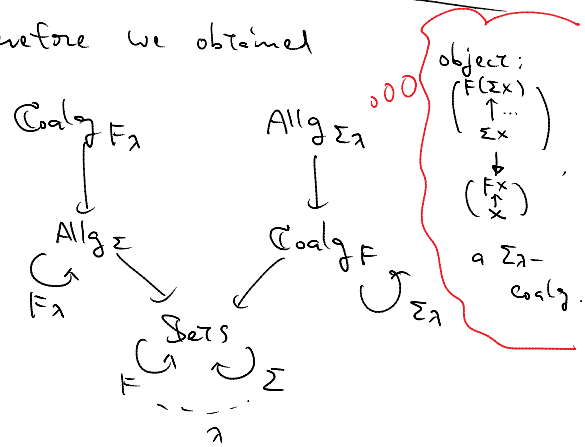
Dually:

Prop. λ lifts $F: \mathbf{Sets} \to \mathbf{Sets}$

to $F\lambda : \mathrm{Alg}_\Sigma \to \mathrm{Alg}_\Sigma$

$$\begin{pmatrix} \Sigma X \\ \downarrow a \\ X \end{pmatrix} \mapsto \begin{pmatrix} \Sigma FX \\ \downarrow \lambda_X \\ F\Sigma X \\ \downarrow Fa \\ FX \end{pmatrix}$$

Therefore we obtained



2012年7月28日
19:55

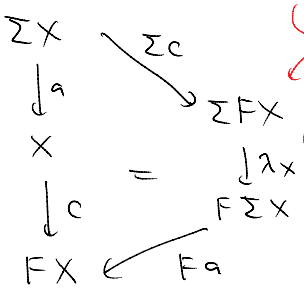
Moreover:

Def. A **λ-bialgebra** is a pair

$$\begin{pmatrix} \Sigma X \\ \downarrow a \\ X \end{pmatrix} \text{ and } \begin{pmatrix} FX \\ \uparrow c \\ X \end{pmatrix}$$

(alg. & coalg. with the same carrier)

s.t.



"The pentagon diagram"

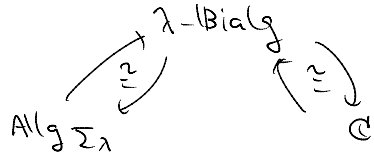
A map of λ-bialg. is f s.t.

$$\begin{array}{ccc} \Sigma X & \xrightarrow{\Sigma f} & \Sigma Y \\ \downarrow & & \downarrow \\ X & \xrightarrow{f} & Y \\ \downarrow & & \downarrow \\ FX & \xrightarrow{Ff} & FY \end{array}$$

$\Rightarrow$ λ-Bialg., the cat. of λ-bialg.

2012年7月28日
19:58

Prop. We have isomorphisms between categories:



Proof. Not hard. for example,

$$\left[\begin{array}{ccc} (\Sigma X & \xrightarrow{\Sigma c} & \Sigma FX \xrightarrow{\lambda_X} F\Sigma X) \\ a \downarrow & & \uparrow Fa \\ (X & \xrightarrow[c]{} & FX) \end{array} \right] \in \mathrm{Alg}_{\Sigma\lambda}$$

$$\mapsto \begin{pmatrix} \Sigma X \\ \downarrow a \\ X \\ \downarrow c \\ FX \end{pmatrix}$$

(pentagon is obvious)

□

2012年7月28日
20:01

We have seen this:
"What is an initial coalgebra?"

Lem. If $\mathbb{C}$ has an initial obj. 0, for any functor $F : \mathbb{C} \to \mathbb{C}$

the coalgebra

$$\begin{array}{c} \mathbb{C} \\ \downarrow ! \\ 0 \\ \text{(initial)} \end{array} \quad \begin{array}{c} F0 \\ \uparrow ! \\ 0 \end{array}$$

is an initial F-coalg.

Proof.

$$\begin{array}{ccc} F0 & \longrightarrow & FX \\ !\uparrow & \searrow & \uparrow c \\ 0 & \xrightarrow{!} & X \end{array}$$

trivial due to 0: initial.

Lem. $\mathbb{C}$ has a final obj. 1

$$\Rightarrow \quad \begin{array}{c} F1 \\ \downarrow ! \\ 1 \end{array} \text{ is a final algebra.}$$

2012年7月28日
20:06

Thm. Assm

$\Sigma : \mathbf{Sets} \to \mathbf{Sets}$ has an initial algebra

$F : \mathbf{Sets} \to \mathbf{Sets}$ has a final coalgebra

Then

- $\begin{array}{c} \Sigma A \\ \downarrow \text{init} \\ A \end{array}$ is canonically a λ-bialgebra.
- It is moreover an initial bialg.
- $\begin{array}{c} FZ \\ \uparrow \text{final} \\ Z \end{array}$ is canonically a λ-bialg.
- It is moreover a final λ-bialg.

2012年7月28日
20:08

Proof. We apply the lemmas (2 pages ago) To

$$\begin{array}{ccc} \mathbf{Coalg}\, F_\lambda & & \mathbf{Alg}\, \Sigma_\lambda \\ \downarrow & & \downarrow \\ \mathbf{Alg}\, \Sigma & & \mathbf{Coalg}\, F \\ F_\lambda \circlearrowright & & \Sigma_\lambda \circlearrowright \\ & \mathbf{Sets} & \\ & F \circlearrowright \quad \Sigma \circlearrowright & \\ & \lambda \text{ - - - -} & \end{array}$$

∃ initial obj. ∃ final obj.

Therefore there are

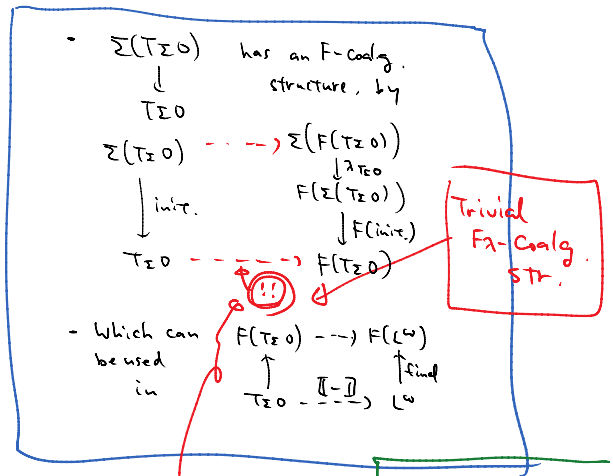
- an init. obj. in $\mathbf{Alg}\, \Sigma_\lambda$, and
- a final obj. in $\mathbf{Coalg}\, F_\lambda$.

Now use.

$$\begin{array}{ccc} & \lambda\text{-}\mathbf{Bialg} & \\ \overset{\cong}{\swarrow} & & \overset{\cong}{\searrow} \\ \mathbf{Alg}\, \Sigma_\lambda & & \mathbf{Coalg}\, F_\lambda \end{array}$$

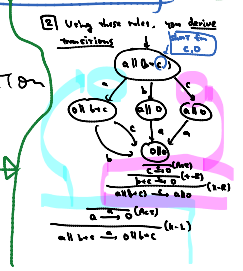
2012年7月28日
20:10

Concretely this is what we did on some 13 pages ago:



Even more concretely

This "stream automaton structure" on terms is what we did in (Rule-based deriv. of transitions)



2012年7月28日
20:13

However, we now see more:

- By the dual scheme we can
  - * equip $F(L^\omega) \uparrow L^\omega$ with an alg. str. $\Sigma(L^\omega) \to L^\omega$
  - * and

$$\begin{array}{ccc} \Sigma(T_\Sigma 0) & \dashrightarrow & \Sigma(L^\omega) \\ \downarrow \text{init} & & \downarrow \\ T_\Sigma 0 & \dashrightarrow & L^\omega \end{array}$$

Note
On the last page: final coalg. semantics (operational)
on this page: initial alg. semantics (denotational)

2012年7月28日
20:19

Point These two coincide!!

By

$$\begin{array}{ccc} \Sigma(T_\Sigma 0) & \dashrightarrow & \Sigma(L^\omega) \\ \cong \downarrow \text{init} & & \downarrow \\ T_\Sigma 0 & \xrightarrow{[\![-]\!]} & L^\omega \\ \downarrow & & \cong \downarrow \text{final} \\ F(T_\Sigma 0) & \dashrightarrow & F(L^\omega) \end{array}$$

initial λ-bialg. / final λ-bialg.

∴ If $[\![-]\!]$ makes the above diagram commute, then in particular

$$\begin{array}{ccc} T_\Sigma 0 & \xrightarrow{[\![-]\!]} & L^\omega \\ \downarrow & & \cong \downarrow \text{final} \\ F(T_\Sigma 0) & \dashrightarrow & F(L^\omega) \end{array}$$

thus this $[\![-]\!]$ is the same as $[\![-]\!]$ 2 pages ago.

2012年7月28日
20:20

Compositionality

By $[\![-]\!]$ being an algebra hom., we immediately have

$$[\![f(t_1, \dots, t_n)]\!] = [f]([\![t_1]\!], \dots, [\![t_n]\!])$$

$$\begin{array}{ccc} \Sigma(T_\Sigma 0) & \dashrightarrow & \Sigma(L^\omega) \\ \cong \downarrow \text{init} & = & \downarrow \\ T_\Sigma 0 & \xrightarrow{[\![-]\!]} & L^\omega \\ \downarrow & & \cong \downarrow \text{final} \\ F(T_\Sigma 0) & \dashrightarrow & F(L^\omega) \end{array}$$

[f]: The interpretation of $f \in \Sigma$ in $\Sigma(L^\omega) \to L^\omega$

What is crucial:

$[\![f(t_1, \dots, t_n)]\!]$ is a function on $[\![t_1]\!], \dots, [\![t_n]\!]$

2012年7月30日
10:26

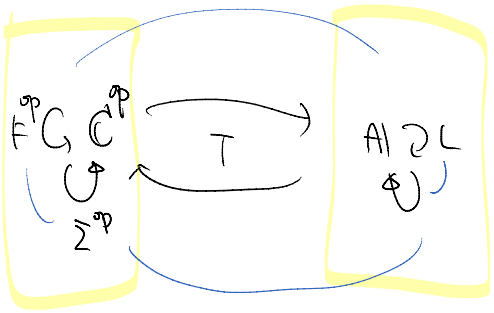
From this we have compositionality.

$$\frac{t_1 \sim s_1 \quad \dots \quad t_n \sim s_n}{f(t_1, \dots, t_n) \sim f(s_1, \dots, s_n)}$$

i.e. bisimilarity is a congruence.

↑ final coalg. sem.

2012年8月1日
14:36



2012年7月30日
15:20

§3.3 Bialgebraic Modeling Beyond the Simple Setting

[1] As we saw, an abstract SOS rule of the form

$$\Sigma F \Rightarrow F \Sigma$$

is very much restricted.

(For example, "zip" of two streams cannot be modeled)

More expressive formats of abstract SOS rules:

- $\Sigma(F \times id) \Rightarrow F\, T_\Sigma$

  $(F \times id)$: the cofree copointed functor over F

  T_Σ: the free monad over Σ

2012年7月30日
15:26

This corresponds to the well-known GSOS format.

- $\Sigma F^\infty \Rightarrow F\, T_\Sigma$

  F^∞: cofree comonad

  T_Σ: free monad

(This canonically induces a distributive law $T_\Sigma F^\infty \Rightarrow F^\infty T_\Sigma$)

2012年7月30日
15:28

[2] For many functors F / base categories $\mathbb{C}$

- For probabilistic systems: take F that involves a distribution functor $\mathcal{D}$ [Bartels]
- Timed systems [Kick et al.]
- Continuous prob. sys. (with $\mathbb{C} = \mathbf{Meas}$) [Bacci, Miculan]
- For value-passing / name-passing calculi [Turi, Fiore, Staton, ...] (with $\mathbb{C}$: a presheaf cat.)