2012年7月26日
12:06

Chap. 3

Categorical SOS and Bialgebras

Plan

- Concurrency, process theory
- SOS
- compositionality: opr. sem. w/ den. 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 program, ← finite string
  (imperative)
  and its execution ← possibly non-term
- a DFA ← finite
  and its accepted language ← infinite, w/ arbitrary long words

- A Büchi autom. and
  the ω-language it recognizes

  φ

  $\boxed{A \subseteq L^{\omega}}$

- 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?

($M =_\beta N$, $[\![M]\!] = [\![N]\!]$, $M \cong_{ctxt} N$, $M \simeq N$ (bisim), ...)

[Foundation 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 $(\mathbb{G}\neg a)$
- after a occurs, b occurs eventually $(\mathbb{G}(a \supset \mathbb{F} b))$

specification in modal (temporal) logic

2012年7月27日
9:17

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. ($\doteqdot$ den.)
  final coalg. sem. ($\doteqdot$ opr.)
  coincide in lucky situations

[Hoare, in early years]

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

Structural Operational Semantics

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

- [Plotkin '81] First appearance
- Used e.g. for def. of the (opr.) 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 :

$$p ::= 0 \quad | \quad a.p \quad | \quad p+p \quad | \quad p \parallel p$$

Termination

action prefix ($a \in L$) "do a and then do p"

non deterministic choice "choose one and do it"

parallel composition "do both in a concurrent manner"

One can also include recursive definitions ⇒ infinitary behaviors

For simplicity we don't do that now

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

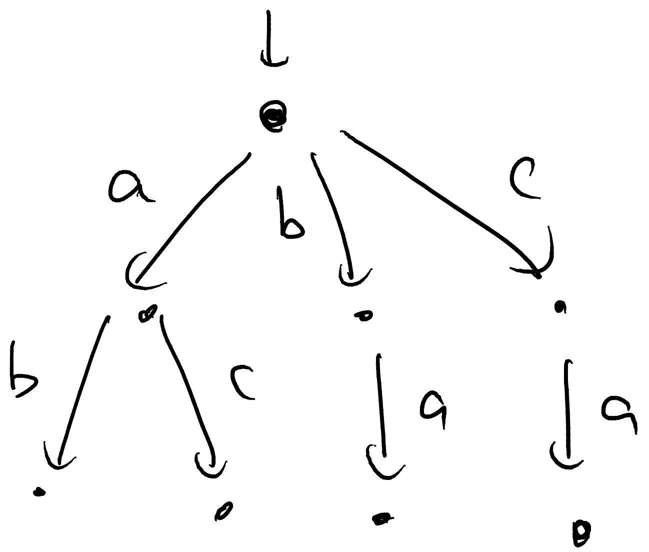
LTS ~ $\left[\begin{array}{l}\text{labeled transition sys.,} \\ \mathcal{P}_{fin}(L \times X) \\ \qquad\uparrow \\ \qquad X\end{array}\right]$.

e.g.

$$[\![\, a.0 \parallel (b.0 + c.0) \,]\!]$$

= ~ "the meaning of —"

:=



2012年7月27日
9:46

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

The SOS answer is as follows.

$\boxed{1}$ You specify the 'meaning' of process operators $(a.\_, +, \parallel, \ldots)$ by means of <u>SOS rules</u>

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

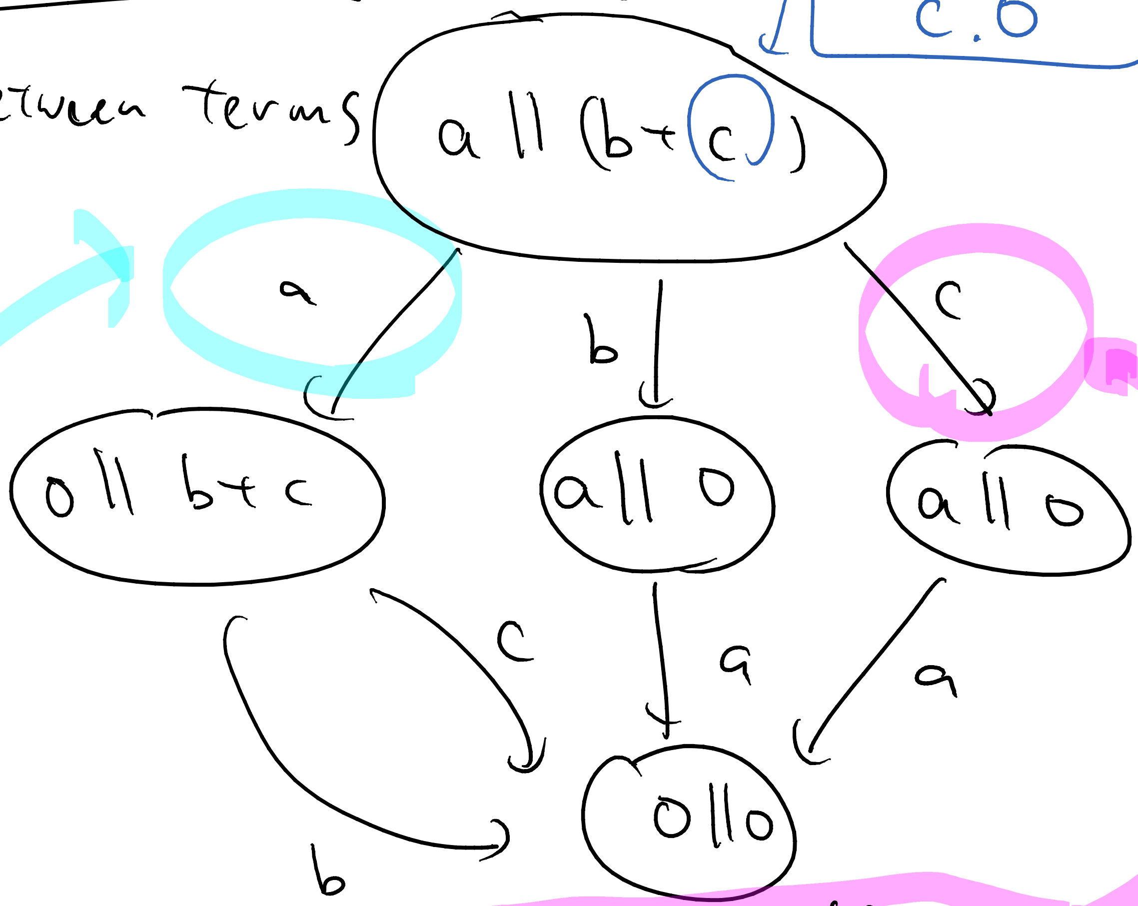
$$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}\ (+\text{-}L) \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}\ (+\text{-}R)$$

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

2012年7月27日
9:54

2 Using these rules, you derive transitions between terms

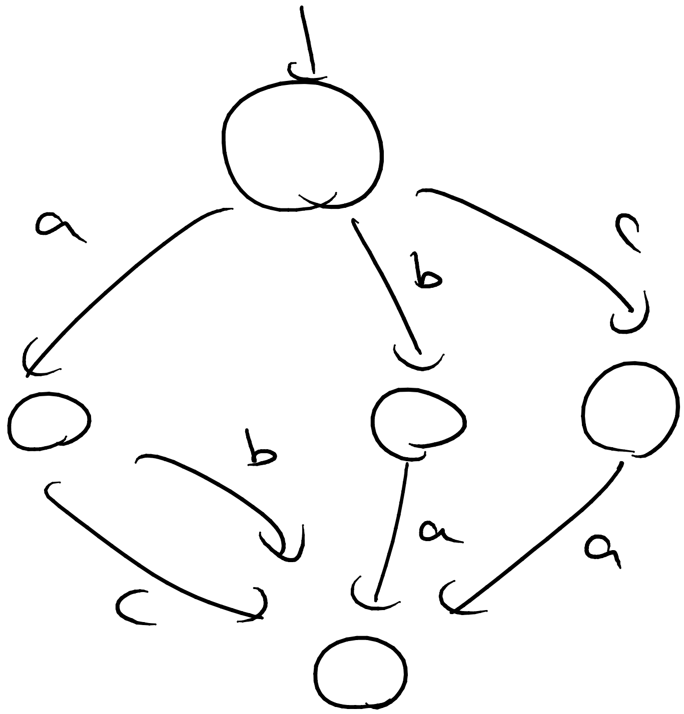
short for c.0



$$\dfrac{\dfrac{\dfrac{}{c \xrightarrow{c} 0}\,(Act)}{b+c \xrightarrow{c} 0}\,(+\text{-}R)}{a \,\|\, (b+c) \xrightarrow{c} a \,\|\, 0}\,(\|\text{-}R)$$

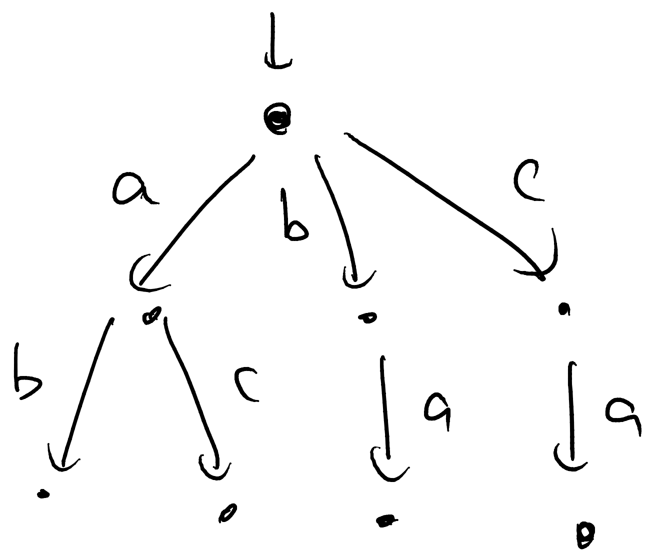
$$\dfrac{\dfrac{}{a \xrightarrow{a} 0}\,(Act)}{a \,\|\, b+c \xrightarrow{a} 0 \,\|\, b+c}\,(\|\text{-}L)$$

$\boxed{3}$ The resulting LTS



is bisimilar to the one

3 pages ago:



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:

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

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, ...)

templates for SOS rules

w/ following (meta) results:

- 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 <u>(co)algebraic flavor</u> in SOS:

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

$\Rightarrow$ <u>Bialgebraic modeling</u>!

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 $\Rightarrow \exp(n)$ complexity

§3.2 Bialgebraic Modeling : The Simple Setting

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

(Following [Klin, TCS 2011])

<u>We fix :</u>

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

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

<u>Notice that</u>

- $T_\Sigma 0 = \{\Sigma\text{-terms with no variables}\}$

  carries an initial algebra

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

- An $(L\times\_)$-coalgebra $\begin{array}{c} L\times X \\ c\uparrow \\ X \end{array}$ is a <u>stream automaton</u>; and its state $x\in X$ induces an L-stream $a_0a_1\ldots\in L^\omega$ <u>by coinduction</u>;

$$\begin{tikzcd} L\times X \arrow[r, dashed] & L\times L^\omega \\ X \arrow[u, "c"] \arrow[r, dashed, "beh(c)"'] & L^\omega \arrow[u, "\cong"', "final"] \end{tikzcd}$$

$$x \longmapsto (beh(c))(x)$$

2012年7月28日
18:35

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

$$\begin{tikzcd}
T_\Sigma 0 \arrow[r, "[\![-]\!]"] & L^\omega \\
(\text{process term}) \arrow[u, phantom, "\rotatebox{90}{$\in$}"] \arrow[r, mapsto] & (\text{stream}) \arrow[u, phantom, "\rotatebox{90}{$\in$}"]
\end{tikzcd}$$

We could use either

$$\begin{tikzcd}
\Sigma(T_\Sigma 0) \arrow[d, "\text{init.}", "\cong"'] & \\
T_\Sigma 0 \arrow[r] & L^\omega \arrow[d, "\text{final}", "\cong"'] \\
& L \times L^\omega
\end{tikzcd}$$

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

Def. A simple stream SOS rule is

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

where

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

More precisely: a simple str. SOS rule is

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

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

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

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

$$\frac{\bigcirc \xrightarrow{a_1} \bigcirc \quad \cdots \quad \bigcirc \xrightarrow{a_n} \bigcirc}{f(\bigcirc, \dots, \bigcirc) \longrightarrow \bigcirc}$$

Example · $L = \{a, b\}$

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

$\underline{c_a}$: "constantly a"

$\Sigma_1 = \Sigma_3 = \Sigma_4 = \cdots = \emptyset$

- Λ consists of

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

$$\frac{x_1 \xrightarrow{l_1} x_1' \qquad x_2 \xrightarrow{l_2} x_2'}{alt(x_1, x_2) \xrightarrow{l_1} alt(x_2', x_1')} \quad \left(\begin{array}{l}\text{for each} \\ l_1, l_2 \\ \in L\end{array}\right)$$

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

The current 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.

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

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

"map of functors"

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

(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)

2012年7月28日
19:05

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

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

A <u>natural transformation</u>

$$\begin{tikzcd} \mathbb{C} \arrow[r, bend left=50, "F"] \arrow[r, bend right=50, "G"'] \arrow[r, phantom, "\Downarrow\alpha"] & \mathbb{D} \end{tikzcd}$$

In the old age it was sometimes written

$\alpha : F \dot{\to} G : \mathbb{C} \to \mathbb{D}$

is a family

$$\left\{ FX \xrightarrow{\alpha_X} \underset{}{\overset{\mathbb{D}}{GX}} \right\}_{X \in \mathbb{C}}$$

of $\mathbb{D}$-arrows.

α_X : α's <u>component</u> at $X \in \mathbb{C}$

subject to the naturality condition

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

$$\begin{tikzcd} X \arrow[d, "f"] \\ Y \end{tikzcd} \qquad \begin{tikzcd} FX \arrow[r, "\alpha_X"] \arrow[d, "Ff"'] & GX \arrow[d, "Gf"] \\ FY \arrow[r, "\alpha_Y"'] & GY \end{tikzcd}$$

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

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

Then:

- $$\begin{array}{c} \Sigma(T_\Sigma 0) \\ \cong \downarrow \text{init.} \\ T_\Sigma 0 \end{array}$$ has an F-coalg. structure, by

$$\begin{tikzcd}
\Sigma(T_\Sigma 0) \arrow[r, dashed] \arrow[dd, "\text{init.}"] & \Sigma(F(T_\Sigma 0)) \arrow[d, "\lambda T_\Sigma 0"] \\
& F(\Sigma(T_\Sigma 0)) \arrow[d, "F(\text{init.})"] \\
T_\Sigma 0 \arrow[r, dashed] & F(T_\Sigma 0)
\end{tikzcd}$$

!!

- Which can be used in

$$\begin{tikzcd}
F(T_\Sigma 0) \arrow[r, dashed] & F(L^\omega) \\
T_\Sigma 0 \arrow[u] \arrow[r, dashed, "⟦_⟧"] & L^\omega \arrow[u, "\text{final}"]
\end{tikzcd}$$

2012年7月28日
19:14

(more on this is coming later)

Prop.

$$\frac{\text{a simple str. SOS specification } \Lambda}{\frac{}{\text{a natural transformation } \Sigma F \Rightarrow F\Sigma}}$$

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

Proof.

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

$$\underbrace{(F(-))^{|f|}}_{= (L \times (-))^{|f|}} \Rightarrow \underbrace{F\Sigma}_{= L \times (\coprod_{h \in \Sigma} (-)^{|h|})}$$

2012年7月28日
19:17

We define such functions by $\left(\begin{array}{l}\text{Let} \\ f \in \Sigma_n\end{array}\right)$

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

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

where the rule in Δ corresponding to

$$\boxed{\dfrac{\circ \xrightarrow{a_1} \circ \quad \cdots \quad \circ \xrightarrow{a_n} \circ}{f(\circ, \dots, \circ) \longrightarrow \bigcirc}}$$

is

$$\boxed{\dfrac{x_1 \xrightarrow{a_1} x_1' \quad \cdots \quad x_n \xrightarrow{a_n} x_n'}{f(x_1, \dots, x_n) \xrightarrow{a} g(x'_{i_1}, \dots, x'_{i_m})}}$$

2012年7月28日
19:24

We need to check naturality:

$$\begin{tikzcd}
S \arrow[d, "f"] & (L\times S)^n \arrow[r] \arrow[d, "(L\times f)^n"] & L\times\left(\coprod_{h\in\Sigma} S^{|h|}\right) \arrow[d] \\
S' & (L\times S')^n \arrow[r] & L\times\left(\coprod_{h\in\Sigma} (S')^{|h|}\right)
\end{tikzcd}$$

which is easy.

[1] Given a natural transf.

$$\frac{\dfrac{\Sigma F \Rightarrow F\Sigma}{\coprod_{f\in\Sigma}(F(-))^{|f|} \Rightarrow F\Sigma}}{\underbrace{(F(-))^{|f|}}_{= (L\times(-))^{|f|}} \Rightarrow \underbrace{F\Sigma}_{= L\times\left(\coprod_{h\in\Sigma}(-)^{|h|}\right)} \text{ for each } f\in\Sigma}$$

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

2012年7月28日
19:30

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

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

and consider

$$k\big((a_1, x_1'), \dots, (a_{|f|}, x_{|f|}')\big)$$

$$=: \big(a,\ \kappa_g(x_{i_1}', \dots, x_{i_{|g|}}')\big)$$

From this we define a rule

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

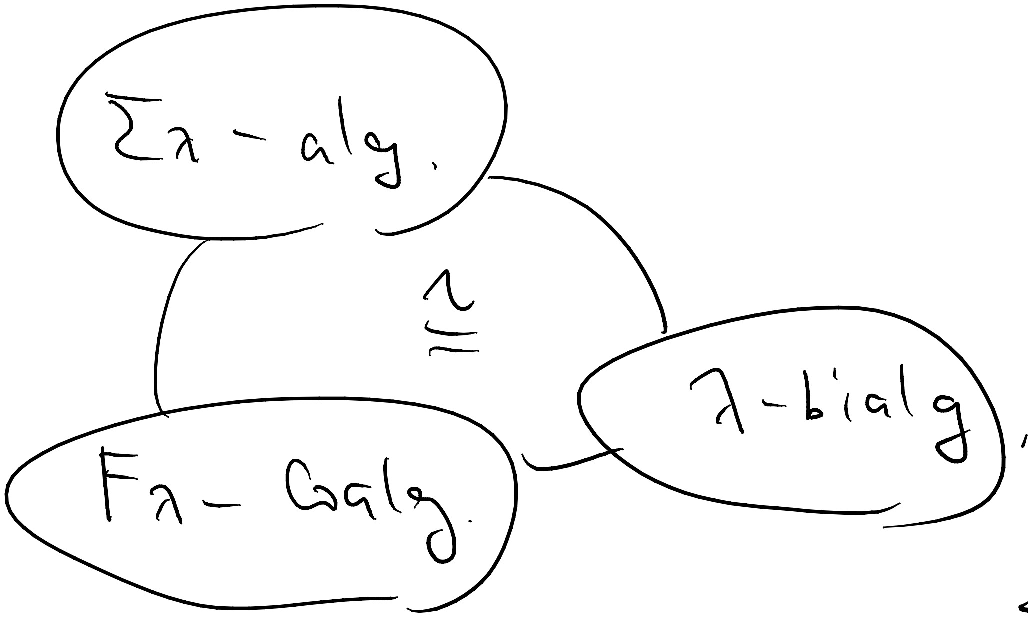
We do this for each f, $a_1, \dots, a_{|f|}$
and define a simple str. SOS spec.

2012年7月28日
19:35

It is not hard to see that [I] and [J] are converse to each other $\boxed{}$

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

We shall now fully exploit this ...



7/31

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

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

that is,

$$\begin{tikzcd}
\mathbf{Coalg}_F \arrow[r, "\Sigma_\lambda"] \arrow[d, "\text{forgetful}"'] \arrow[dr, phantom, "="] & \mathbf{Coalg}_F \arrow[d] \\
\mathbf{Sets} \arrow[r, "\Sigma"'] & \mathbf{Sets}
\end{tikzcd}$$

Concretely:

$$\mathbf{Coalg}_F \xrightarrow{\ \Sigma_\lambda\ } \mathbf{Coalg}_F$$

$$\begin{pmatrix} FX \\ \uparrow c \\ X \end{pmatrix} \longmapsto \begin{pmatrix} F\Sigma X \\ \uparrow \lambda_X \\ \Sigma F X \\ \uparrow \Sigma c \\ \Sigma X \end{pmatrix}$$

2012年7月28日
19:46

Exercise Write down Σ_λ's action on arrows (Use naturality of λ)

Proof.) Straightforward. $\blacksquare$

A 2-categorical view:

- $$\begin{tikzcd}
 & \mathbb{C} \arrow[dr, "\mathrm{id}"] \arrow[dd, phantom, "\Downarrow"] & \\
\mathbf{Coalg}_F \arrow[ur, "\mathrm{forg.}"] \arrow[dr, "\mathrm{forget.}"'] & & \mathbb{C} \\
 & \mathbb{C} \arrow[ur, "F"'] &
\end{tikzcd}$$ in $\mathbf{Cat}$ is an inserter.

- $$\begin{tikzcd}
 & \mathbb{C} \arrow[r, "\Sigma"] \arrow[dr, "\mathrm{id}"] \arrow[dd, phantom, "\Downarrow"] & \mathbb{C} \arrow[dr, "\mathrm{id}"] \arrow[d, phantom, "="] & \\
 & & \mathbb{C} \arrow[r, "\Sigma"] \arrow[d, phantom, "\Downarrow\lambda"] & \mathbb{C} \\
\mathbf{Coalg}_F \arrow[uur, "\mathrm{forg.}"] \arrow[r, "\mathrm{forg.}"'] & \mathbb{C} \arrow[ur, "F"] \arrow[r, "\Sigma"'] & \mathbb{C} \arrow[ur, "F"'] &
\end{tikzcd}$$

induces

$$\mathbf{Coalg}_F \overset{\Sigma_\lambda}{\dashrightarrow} \mathbf{Coalg}_F .$$

2012年7月28日
19:50

Dually:

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

to $F_\lambda : \mathbf{Alg}_\Sigma \longrightarrow \mathbf{Alg}_\Sigma$

$$\left(\begin{array}{c} \Sigma X \\ \downarrow a \\ X \end{array}\right) \longmapsto \left(\begin{array}{c} \Sigma F X \\ \downarrow \lambda_X \\ F\Sigma X \\ \downarrow Fa \\ FX \end{array}\right)$$

Therefore we obtained

$$\begin{tikzcd}
\mathbf{Coalg}_{F_\lambda} \arrow[d] & & \mathbf{Alg}_{\Sigma_\lambda} \arrow[d] \\
\mathbf{Alg}_\Sigma \arrow[loop left, "F_\lambda"] \arrow[dr] & & \mathbf{Coalg}_F \arrow[loop right, "\Sigma_\lambda"] \arrow[dl] \\
& \mathbf{Sets} \arrow[loop left, "F"] \arrow[loop right, "\Sigma"] &
\end{tikzcd}$$

λ

ooO

object:

$$\begin{tikzcd}
F(\Sigma X) \arrow[d] & FX \\
\Sigma X \arrow[u, "\cdots"] & X \arrow[u]
\end{tikzcd}
\qquad
\left(\begin{array}{c} F(\Sigma X) \\ \uparrow \cdots \\ \Sigma X \end{array}\right) \downarrow \left(\begin{array}{c} FX \\ \uparrow \\ X \end{array}\right)$$

a Σ_λ-coalg.

2012年7月28日
19:55

Moreover:

Def. A <u>λ-bialgebra</u> is a pair

$$\begin{tikzcd} \Sigma X \arrow[d, "a"] \\ X \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} FX \\ X \arrow[u, "c"] \end{tikzcd}$$

(alg. & coalg. with the same carrier)

s.t.

$$\begin{tikzcd} \Sigma X \arrow[d, "a"] \arrow[r, "\Sigma c"] & \Sigma F X \arrow[d, "\lambda_X"] \\ X \arrow[d, "c"] & F \Sigma X \arrow[dl, "Fa"] \\ FX & \end{tikzcd}$$

"The pentagon diagram"

A <u>map</u> of λ-bialg. is f s.t.

$$\begin{tikzcd} \Sigma X \arrow[r, "\Sigma f"] \arrow[d] & \Sigma Y \arrow[d] \\ X \arrow[r, "f"] \arrow[d] & Y \arrow[d] \\ FX \arrow[r, "Ff"] & FY \end{tikzcd}$$

$\Rightarrow$ λ-$\mathbb{B}$ialg., the cat. of λ-bialg.

Prop. We have isomorphisms between categories:

$$
\begin{tikzcd}
 & \lambda\text{-}\mathbb{B}\mathrm{ialg} \arrow[dl, bend left, "\cong"] \arrow[dr, bend left, "\cong"] & \\
\mathrm{Alg}\,\Sigma_\lambda \arrow[ur, bend left] & & \mathbb{C}\mathrm{oalg}\,F_\lambda \arrow[ul, bend left]
\end{tikzcd}
$$

Proof. Not hard. for example,

$$
\left[
\begin{tikzcd}
\Sigma X \arrow[r, "\Sigma c"] \arrow[d, "a"'] & \Sigma F X \arrow[r, "\lambda_X"] & F\Sigma X \arrow[dl, "Fa"] \\
X \arrow[r, "c"'] & F X &
\end{tikzcd}
\right] \in \mathrm{Alg}_{\Sigma\lambda}
$$

$$
\longmapsto
\left(
\begin{tikzcd}
\Sigma X \arrow[d, "a"] \\
X \arrow[d, "c"] \\
F X
\end{tikzcd}
\right)
$$

(pentagon is obvious)

1

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

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

is an initial F-coalg.

Proof.

$$\begin{tikzcd} F0 \arrow[r] & FX \\ 0 \arrow[u, "!"] \arrow[r, dashed, "!"'] \arrow[ur, phantom, "\checkmark"] & X \arrow[u, "c"'] \end{tikzcd}$$

trivial due to 0: initial.

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

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

Thm. Asm

Σ : Sets → Sets has an initial algebra

F : Sets → Sets has a final coalgebra

Then

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

$$\begin{tikzcd}
\mathbf{Coalg}_{F_\lambda} \arrow[d] & & \mathbf{Alg}_{\Sigma_\lambda} \arrow[d] \\
\mathbf{Alg}_{\Sigma} \arrow[loop left, "F_\lambda"] \arrow[dr] & & \mathbf{Coalg}_{F} \arrow[loop below, "\Sigma_\lambda"] \arrow[dl] \\
& \mathbf{Sets} \arrow[loop left, "F"] \arrow[loop right, "\Sigma"] &
\end{tikzcd}$$

λ

$\exists$ initial obj.

$\exists$ final obj.

Therefore there are

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

Now use.

$$\begin{tikzcd}
& \lambda\text{-}\mathbf{Bialg} \arrow[dl, bend left, "\cong"] \arrow[dr, bend left] & \\
\mathbf{Alg}_{\Sigma_\lambda} \arrow[ur] & & \mathbf{Coalg}_{F_\lambda} \arrow[ul, "\cong"]
\end{tikzcd}$$

$\boxed{\text{z}}$

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

- $\Sigma(T_\Sigma 0)$ has an F-Coalg. structure, by
$$\Sigma(T_\Sigma 0) \longrightarrow T_\Sigma 0$$

$$\begin{tikzcd}
\Sigma(T_\Sigma 0) \arrow[r, dashed] \arrow[dd, "\mathrm{init.}"] & \Sigma(F(T_\Sigma 0)) \arrow[d, "\lambda_{T_\Sigma 0}"] \\
 & F(\Sigma(T_\Sigma 0)) \arrow[d, "F(\mathrm{init.})"] \\
T_\Sigma 0 \arrow[r, dashed] & F(T_\Sigma 0)
\end{tikzcd}$$

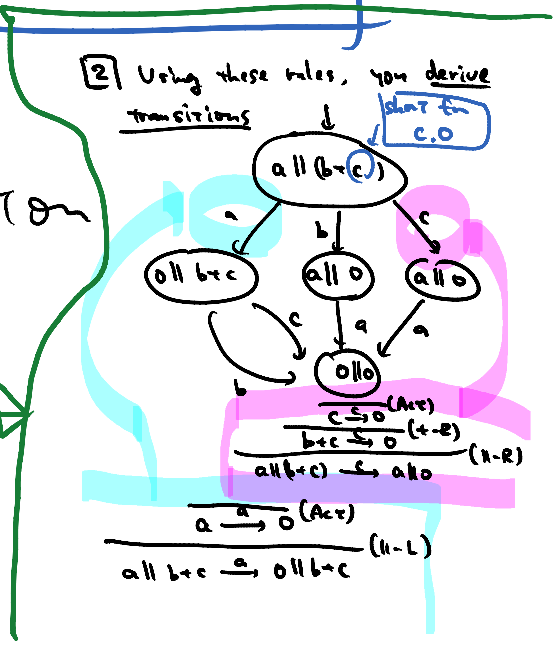
Trivial $F\lambda$-Coalg. str.

!!

- Which can be used in
$$\begin{tikzcd}
F(T_\Sigma 0) \arrow[r, dashed] & F(L^\omega) \\
T_\Sigma 0 \arrow[u] \arrow[r, dashed, "{[\![-]\!]}"] & L^\omega \arrow[u, "\mathrm{final}"]
\end{tikzcd}$$

Even More concretely

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



However, we now see more:

- By the dual scheme we can
  * equip $\begin{matrix} F(L^\omega) \\ \uparrow \\ L^\omega \end{matrix}$ with an alg.

    str. $\begin{matrix} \Sigma(L^\omega) \\ \downarrow \\ L^\omega \end{matrix}$

  * and

$$\begin{tikzcd} \Sigma(T_\Sigma 0) \arrow[r, dashed] \arrow[d, "init"] & \Sigma(L^\omega) \arrow[d] \\ T_\Sigma 0 \arrow[r, dashed] & L^\omega \end{tikzcd}$$

Note

On the last page: final coalg. semantics (operational)

on this page: initial alg. semantics (denotational)

Point These two coincide !!

By

$$\begin{tikzcd}
\Sigma(T_\Sigma 0) \arrow[r, dashed] \arrow[d, "\cong"', "\mathrm{init}"] & \Sigma(L^\omega) \arrow[d] \\
T_\Sigma 0 \arrow[r, dashed, "[\![-]\!]"] \arrow[d] & L^\omega \arrow[d, "\cong"', "\mathrm{final}"] \\
F(T_\Sigma 0) \arrow[r, dashed] & F(L^\omega)
\end{tikzcd}$$

initial λ-bialg.

final λ-bialg.

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

$$\begin{tikzcd}
T_\Sigma 0 \arrow[r, dashed, "[\![-]\!]"] \arrow[d] & L^\omega \arrow[d, "\cong"', "\mathrm{final}"] \\
F(T_\Sigma 0) \arrow[r, dashed] & F(L^\omega)
\end{tikzcd}$$

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

$$
\begin{tikzcd}
\Sigma(T_\Sigma 0) \arrow[r, dashed] \arrow[d, "\cong"', "\mathrm{init}"] \arrow[dr, phantom, "//"] & \Sigma(L^\omega) \arrow[d] \\
T_\Sigma 0 \arrow[r, dashed, "{[\![-]\!]}"] \arrow[d] & L^\omega \arrow[d, "\cong"', "\mathrm{final}"] \\
F(T_\Sigma 0) \arrow[r, dashed] & F(L^\omega)
\end{tikzcd}
$$

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

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

What is crucial:

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

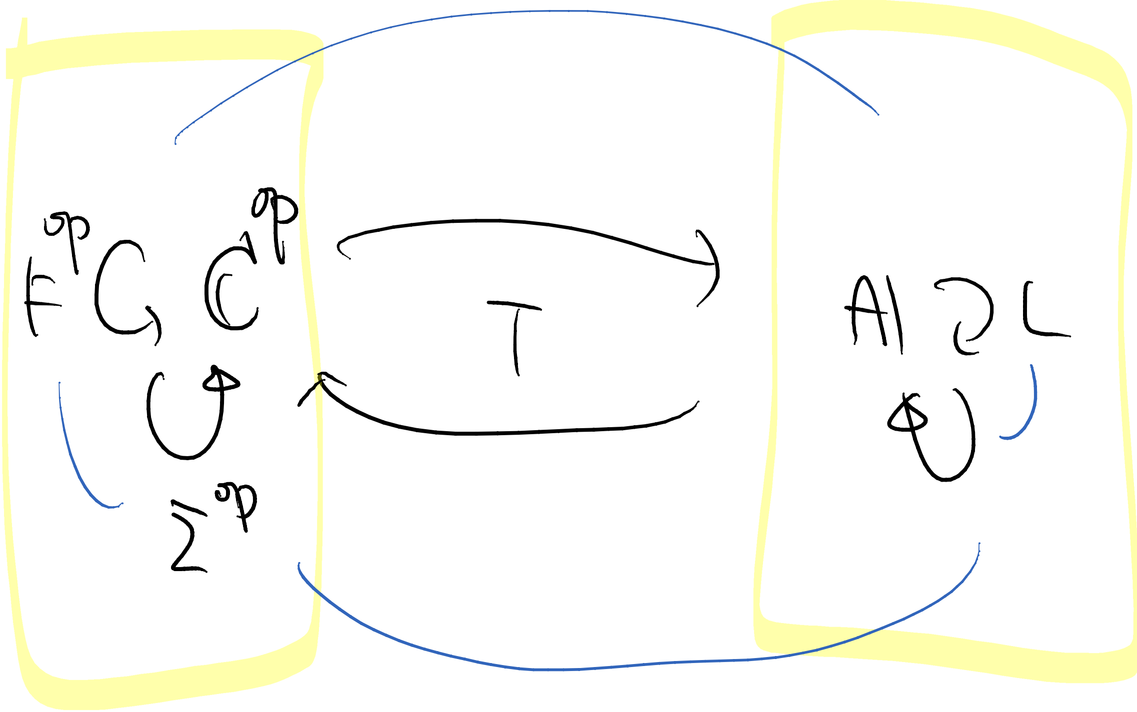
2012年7月30日
10:26

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.


$F^{op}C, C^{op}$
Σ^{op}
T

§3.3 Bialgebraic Modeling Beyond the Simple Setting

$\boxed{1}$ 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:

- $\Sigma(\underbrace{F \times \mathrm{id}}_{\substack{\text{the cofree}\\ \text{copointed functor}\\ \text{over } F}}) \Rightarrow F \underbrace{T_\Sigma}_{\substack{\text{the free monad}\\ \text{over } \Sigma}}$

2012年7月30日
15:26

This corresponds to the well-known GSOS format.

- $\Sigma \underbrace{F^{\infty}}_{\text{cofree comonad}} \Rightarrow F \underbrace{T_\Sigma}_{\text{free monad}}$

[This canonically induces a distributive law

$T_\Sigma F^{\infty} \Rightarrow F^{\infty} T_\Sigma$]

$\boxed{2}$ 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.)