A categorical model for 2-PDAs with states

Jürgen Koslowski

Department of Theoretical Computer Science
Technical University Braunschweig

cmat14, Coimbra

(2014-01-25)

http://www.iti.cs.tu-bs.de/~koslowj/RESEARCH

Content

The context: unification of machine models

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.
- 2PDAs (with two stacks), long known to be Turing complete, could rectify this, but never received much attention.

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.
- 2PDAs (with two stacks), long known to be Turing complete, could rectify this, but never received much attention.
- But even deterministic 2PDAs with a single state suffice [Koslowski 2013],

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.
- 2PDAs (with two stacks), long known to be Turing complete, could rectify this, but never received much attention.
- But even deterministic 2PDAs with a single state suffice [Koslowski 2013], hence states and storage can be disentagled (impossible for TMs).

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.
- 2PDAs (with two stacks), long known to be Turing complete, could rectify this, but never received much attention.
- But even deterministic 2PDAs with a single state suffice [Koslowski 2013], hence states and storage can be disentagled (impossible for TMs). This allows a natural refinement of the Chomsky hierarchy, but also raises questions about the true nature of states.

The context: unification of machine models

- ▷ The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.
- ▷ 2PDAs (with two stacks), long known to be Turing complete, could rectify this, but never received much attention.
- ▷ But even deterministic 2PDAs with a single state suffice [Koslowski 2013], hence states and storage can be disentagled (impossible for TMs). This allows a natural refinement of the Chomsky hierarchy, but also raises questions about the true nature of states.
- ▷ We provide an elegant categorification of ss2PDAs that allows states to be incorporated orthogonally to storage.

The context: unification of machine models

- The familiar non-deterministic machine models for language recognition over an alphabet Σ in the Chomsky-hierarchy
  - finite automata (FAs) for regular languages ($\boldsymbol{REG}$),
  - push-down automata (PDAs) for context-free languages ($\boldsymbol{CF}$),
  - Turing machines (TMs) for semi-decidable languages ($\boldsymbol{SD}$),

  display less of a family resemblance than the defining grammars.
- 2PDAs (with two stacks), long known to be Turing complete, could rectify this, but never received much attention.
- But even deterministic 2PDAs with a single state suffice [Koslowski 2013], hence states and storage can be disentagled (impossible for TMs). This allows a natural refinement of the Chomsky hierarchy, but also raises questions about the true nature of states.
- We provide an elegant categorification of ss2PDAs that allows states to be incorporated orthogonally to storage. The result bears strong resemblance to the tile model of Gadducci and Montanari [2000] for rewriting and abstract concurrent semantics.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$$ (jointly mono)

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet,

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$$ (faithful graph morphism)

$$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$$ (jointly mono)

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \underset{!}{\overset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$$ (faithful graph morphism)

$$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$$ (jointly mono)

$$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$$ (jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \underset{!}{\overset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$$ (faithful graph morphism)

$$\overline{\overline{\qquad\qquad}}$$

$$G_0 \overset{s}{\underset{t}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$$ (jointly mono)

$$\overline{\overline{\qquad\qquad}}$$

$$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$$ (jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)

$$\overline{\overline{\qquad\qquad}}$$

$$G_0 \xrightarrow{L} \Sigma \times G_0$$ (non-obvious relation)

$$\overline{\overline{\qquad\qquad}}$$

$$\overline{\overline{\qquad\qquad}}$$

$$\overline{\overline{\qquad\qquad}}$$

where $\boldsymbol{G} = (G_1 \overset{s}{\underset{t}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \overset{!}{\underset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ .

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$	(faithful graph morphism)
$G_0 \overset{s}{\underset{t}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono)
$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)
$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$	(coalgebra, Aczel and Mendler [1989])

where $\boldsymbol{G} = (G_1 \overset{s}{\underset{t}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \overset{!}{\underset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$$

(faithful graph morphism)

$$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$$

(jointly mono)

$$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$$

(jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)

$$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$$

(coalgebra, Aczel and Mendler [1989])

$$G_0 \times \Sigma \xrightarrow{L} G_0$$

(non-obvious relation, textbook LTS?)

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \underset{!}{\overset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$	(faithful graph morphism)
$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono)
$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)
$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$	(coalgebra, Aczel and Mendler [1989])
$G_0 \times \Sigma \xrightarrow{L} G_0\mathcal{P}$	(textbook LTS!)

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \underset{!}{\overset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma} \qquad \text{(faithful graph morphism)}$$

$$\overline{\overline{}}$$

$$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma \qquad \text{(jointly mono)}$$

$$\overline{\overline{}}$$

$$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma \qquad \text{(jointly mono, relation } G_0 \times G_0 \longrightarrow\!\!\!\!\triangleright\; \Sigma)$$

$$\overline{\overline{}}$$

$$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P} \qquad \text{(coalgebra, Aczel and Mendler [1989])}$$

$$\overline{\overline{}}$$

$$G_0 \times \Sigma \xrightarrow{L} G_0\mathcal{P} \qquad \text{(textbook LTS!)}$$

$$\overline{\overline{}}$$

$$\Sigma \xrightarrow{L}\!\!\!\!\triangleright\; G_0 \times G_0 \qquad \text{(reversed obvious relation)}$$

$$\overline{\overline{}}$$

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \underset{!}{\overset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$	(faithful graph morphism)
$G_0 \underset{t}{\overset{s}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono)
$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)
$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$	(coalgebra, Aczel and Mendler [1989])
$G_0 \times \Sigma \xrightarrow{L} G_0\mathcal{P}$	(textbook LTS!)
$\Sigma \xrightarrow{L} (G_0 \times G_0)\mathcal{P}$	(this looks promising)

where $\boldsymbol{G} = (G_1 \underset{t}{\overset{s}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \underset{!}{\overset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$	(faithful graph morphism)
$G_0 \overset{s}{\underset{t}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono)
$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)
$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$	(coalgebra, Aczel and Mendler [1989])
$G_0 \times \Sigma \xrightarrow{L} G_0\mathcal{P}$	(textbook LTS!)
$\Sigma \xrightarrow{L} (G_0, G_0)\boldsymbol{rel}$	(hom-component of a graph morphism)

where $\boldsymbol{G} = (G_1 \overset{s}{\underset{t}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \overset{!}{\underset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (0)

Most categorical approaches have focussed on labled transition systems (LTSs), that form the core of FAs (disregarding initial/final states):

$\boldsymbol{G} \xrightarrow{\langle !, \ell \rangle} \boldsymbol{\Sigma}$	(faithful graph morphism)
$G_0 \overset{s}{\underset{t}{\leftleftarrows}} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono)
$G_0 \times G_0 \xleftarrow{\langle s, t \rangle} G_1 \xrightarrow{\ell} \Sigma$	(jointly mono, relation $G_0 \times G_0 \longrightarrow \Sigma$)
$G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$	(coalgebra, Aczel and Mendler [1989])
$G_0 \times \Sigma \xrightarrow{L} G_0\mathcal{P}$	(textbook LTS!)
$\Sigma \xrightarrow{L} (G_0, G_0)\boldsymbol{rel}$	(hom-component of a graph morphism)
$\boldsymbol{\Sigma} \xrightarrow{L} \boldsymbol{rel}$	("finitary" graph morphism)

where $\boldsymbol{G} = (G_1 \overset{s}{\underset{t}{\rightrightarrows}} G_0)$ is a finite graph, Σ is an alphabet, and $\boldsymbol{\Sigma} = (\Sigma \overset{!}{\underset{!}{\rightrightarrows}} 1)$ is a single-node graph with hom-set Σ.

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

$$\frac{\boldsymbol{K} \longrightarrow \boldsymbol{\Sigma}^\star}{}$$ (fibre-small faithful functor)

$$\frac{}{\boldsymbol{\Sigma}^\star \longrightarrow \boldsymbol{rel}}$$ (lax functor, Rosenthal [1996])

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

$$\frac{\boldsymbol{K} \longrightarrow \boldsymbol{\Sigma}^\star}{\overline{}}$$

(fibre-small faithful functor)

$$\frac{\Sigma^\star \longrightarrow (K_0 \times K_0)\mathcal{P}}{\boldsymbol{\Sigma}^\star \longrightarrow \boldsymbol{rel}}$$

(lax homomorphism)

(lax functor, Rosenthal [1996])

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

$$\frac{\dfrac{\dfrac{\boldsymbol{K} \longrightarrow \boldsymbol{\Sigma}^\star}{(K_0 \times K_0)\mathcal{P} \longrightarrow \Sigma^\star\mathcal{P}}}{\Sigma^\star \longrightarrow (K_0 \times K_0)\mathcal{P}}}{\boldsymbol{\Sigma}^\star \longrightarrow \boldsymbol{rel}}$$

(fibre-small faithful functor)

(quantale-enriched category, Betti [1980])

(lax homomorphism)

(lax functor, Rosenthal [1996])

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

$$\frac{\dfrac{\dfrac{\boldsymbol{K} \longrightarrow \boldsymbol{\Sigma}^\star}{(K_0 \times K_0)\mathcal{P} \longrightarrow \Sigma^\star\mathcal{P}}}{\Sigma^\star \longrightarrow (K_0 \times K_0)\mathcal{P}}}{\boldsymbol{\Sigma}^\star \longrightarrow \boldsymbol{rel}}$$

(fibre-small faithful functor)

(quantale-enriched category, Betti [1980])

(lax homomorphism)

(lax functor, Rosenthal [1996])

- ▷ The bottom lines would seem to place our subject squarely into the realm of categorical relational algebra.

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

$\boldsymbol{K} \longrightarrow \boldsymbol{\Sigma}^\star$	(fibre-small faithful functor)
$(K_0 \times K_0)\mathcal{P} \longrightarrow \Sigma^\star\mathcal{P}$	(quantale-enriched category, Betti [1980])
$\Sigma^\star \longrightarrow (K_0 \times K_0)\mathcal{P}$	(lax homomorphism)
$\boldsymbol{\Sigma}^\star \longrightarrow \boldsymbol{rel}$	(lax functor, Rosenthal [1996])

- ▷ The bottom lines would seem to place our subject squarely into the realm of categorical relational algebra.
- ▷ Morphisms of coalgebras $G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$ turn out to be functional bisimulations, while spans are needed to model general bisimulations.

Categorical approaches to LTSs (1)

Using the free monoid $\Sigma^\star$ and categories $\boldsymbol{K}$ instead one obtains

$$\frac{\dfrac{\dfrac{\boldsymbol{K} \longrightarrow \boldsymbol{\Sigma}^\star}{(K_0 \times K_0)\mathcal{P} \longrightarrow \Sigma^\star\mathcal{P}}}{\Sigma^\star \longrightarrow (K_0 \times K_0)\mathcal{P}}}{\boldsymbol{\Sigma}^\star \longrightarrow \boldsymbol{rel}}$$

(fibre-small faithful functor)

(quantale-enriched category, Betti [1980])

(lax homomorphism)

(lax functor, Rosenthal [1996])

- The bottom lines would seem to place our subject squarely into the realm of categorical relational algebra.
- Morphisms of coalgebras $G_0 \xrightarrow{L} (\Sigma \times G_0)\mathcal{P}$ turn out to be functional bisimulations, while spans are needed to model general bisimulations.
- Joyal, Winskel and Nielsen [1994] as well as Cockett and Spooner [1997] approach bisimulations synthetically; in an enriched context this has been done by Schmitt and Worytkiewicz [2006].

Remarks

Remarks

- The outer bijection persists, when Σ / $\Sigma^\star$ is replaced by an arbitrary graph/category X, giving rise to a Grothendieck-type construction.

Remarks

- The outer bijection persists, when Σ / $\Sigma^{\star}$ is replaced by an arbitrary graph/category X, giving rise to a Grothendieck-type construction. But not all intermediate stages admit a similar generalization, in particular not coalgebra.

Remarks

- The outer bijection persists, when Σ / $\Sigma^\star$ is replaced by an arbitrary graph/category X, giving rise to a Grothendieck-type construction. But not all intermediate stages admit a similar generalization, in particular not coalgebra.
- Oplax transformations as morphisms between lax functors into rel translate into simulations between LTSs (JK, several talks since 2003, Sobociński [2012]).

Remarks

- The outer bijection persists, when Σ / $\Sigma^\star$ is replaced by an arbitrary graph/category X , giving rise to a Grothendieck-type construction. But not all intermediate stages admit a similar generalization, in particular not coalgebra.
- Oplax transformations as morphisms between lax functors into rel translate into simulations between LTSs (JK, several talks since 2003, Sobociński [2012]).
- So far we have ignored initial/final states.

Remarks

- The outer bijection persists, when $\Sigma \,/\, \Sigma^\star$ is replaced by an arbitrary graph/category X, giving rise to a Grothendieck-type construction. But not all intermediate stages admit a similar generalization, in particular not coalgebra.
- Oplax transformations as morphisms between lax functors into $\boldsymbol{rel}$ translate into simulations between LTSs (JK, several talks since 2003, Sobociński [2012]).
- So far we have ignored initial/final states. We'd prefer a categorical interpretation rather than selecting arbitrary subsets of states. But the attempt to use simulations from, resp., into a special LTS fails.

Remarks

- The outer bijection persists, when Σ / $\Sigma^\star$ is replaced by an arbitrary graph/category X, giving rise to a Grothendieck-type construction. But not all intermediate stages admit a similar generalization, in particular not coalgebra.
- Oplax transformations as morphisms between lax functors into rel translate into simulations between LTSs (JK, several talks since 2003, Sobociński [2012]).
- So far we have ignored initial/final states. We'd prefer a categorical interpretation rather than selecting arbitrary subsets of states. But the attempt to use simulations from, resp., into a special LTS fails.
- Instead, one has to use modules rather than oplax natural transformations, from, resp., into the discrete lax functor $\Sigma^\star \xrightarrow{D} \mathit{rel}$.

Remarks

- The outer bijection persists, when $\Sigma \,/\, \Sigma^\star$ is replaced by an arbitrary graph/category X, giving rise to a Grothendieck-type construction. But not all intermediate stages admit a similar generalization, in particular not coalgebra.
- Oplax transformations as morphisms between lax functors into rel translate into simulations between LTSs (JK, several talks since 2003, Sobociński [2012]).
- So far we have ignored initial/final states. We'd prefer a categorical interpretation rather than selecting arbitrary subsets of states. But the attempt to use simulations from, resp., into a special LTS fails.
- Instead, one has to use modules rather than oplax natural transformations, from, resp., into the discrete lax functor $\Sigma^\star \xrightarrow{D} rel$. In the context of graphs this means that instead of Σ we need to consider the reflexive graph Σ^ϵ with hom-set $\Sigma + \{\epsilon\}$.

Moving up the Chomsky hierarchy: Walters' approach

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.
- Instead, we slightly extend Walters' [1989] categorification of a certain type of context-free grammars (CFGs), which functionally separates terminals (= elements of Σ) from variables,

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.
- Instead, we slightly extend Walters' [1989] categorification of a certain type of context-free grammars (CFGs), which functionally separates terminals (= elements of Σ) from variables, rather than (classically) lumping them together and forming a free monoid.

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.
- Instead, we slightly extend Walters' [1989] categorification of a certain type of context-free grammars (CFGs), which functionally separates terminals (= elements of Σ) from variables, rather than (classically) lumping them together and forming a free monoid.
- Walters views morphisms $G \xrightarrow{\gamma} \Sigma^{\epsilon}$ between finite reflexive graphs as regular grammars rather than as LTSs.

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.
- Instead, we slightly extend Walters' [1989] categorification of a certain type of context-free grammars (CFGs), which functionally separates terminals (= elements of Σ) from variables, rather than (classically) lumping them together and forming a free monoid.
- Walters views morphisms $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma}^{\epsilon}$ between finite reflexive graphs as regular grammars rather than as LTSs. Then morphsms between suitable multi-graphs (edges have finitely many inputs and one output; this yields bottom-up parsing) capture a class of CFGs (in Walters Normal Form (WNF)) that generate all context-free languages.

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.
- Instead, we slightly extend Walters' [1989] categorification of a certain type of context-free grammars (CFGs), which functionally separates terminals (= elements of Σ) from variables, rather than (classically) lumping them together and forming a free monoid.
- Walters views morphisms $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma}^{\epsilon}$ between finite reflexive graphs as regular grammars rather than as LTSs. Then morphsms between suitable multi-graphs (edges have finitely many inputs and one output; this yields bottom-up parsing) capture a class of CFGs (in Walters Normal Form (WNF)) that generate all context-free languages.
- Walters wanted to illustrate his construction of the free category with products over a multi-graph.

Moving up the Chomsky hierarchy: Walters' approach

- Some approaches to describe at least real-time PDAs by coalgebraic methods are presently under way, but they seem to be very intricate.
- Instead, we slightly extend Walters' [1989] categorification of a certain type of context-free grammars (CFGs), which functionally separates terminals (= elements of Σ) from variables, rather than (classically) lumping them together and forming a free monoid.
- Walters views morphisms $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma}^{\epsilon}$ between finite reflexive graphs as regular grammars rather than as LTSs. Then morphsms between suitable multi-graphs (edges have finitely many inputs and one output; this yields bottom-up parsing) capture a class of CFGs (in Walters Normal Form (WNF)) that generate all context-free languages.
- Walters wanted to illustrate his construction of the free category with products over a multi-graph. However, a more direct way of extracting the generated language becomes available with top-down parsing, hence we revert to co-multi-graphs or cm-graphs, for short.

Walters' approach slightly generalized

Definition

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph Σ_N with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in N$.

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph $\boldsymbol{\Sigma_N}$ with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in \boldsymbol{N}$. ($\emptyset_{\boldsymbol{N}}$ is terminal.)

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph $\boldsymbol{\Sigma_N}$ with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in \boldsymbol{N}$. ($\boldsymbol{\emptyset_N}$ is terminal.)

(1) A CFG à la Walters (CFW) γ over Σ is a faithful cm-graph morphism $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma_N}$ with $\boldsymbol{G}$ finite.

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph $\boldsymbol{\Sigma_N}$ with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in \boldsymbol{N}$. ($\emptyset_{\boldsymbol{N}}$ is terminal.)

(1) A CFG à la Walters (CFW) γ over Σ is a faithful cm-graph morphism $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma_N}$ with $\boldsymbol{G}$ finite.

- Terminals (= elements of Σ) label the edges of $\boldsymbol{\Sigma_N}$, while the set $\mathcal{B}$ of variables is the set of $\boldsymbol{G}$-nodes.

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph $\boldsymbol{\Sigma_N}$ with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in \boldsymbol{N}$. ($\boldsymbol{\emptyset_N}$ is terminal.)

(1) A CFG à la Walters (CFW) γ over Σ is a faithful cm-graph morphism $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma_N}$ with $\boldsymbol{G}$ finite.

- Terminals (= elements of Σ) label the edges of $\boldsymbol{\Sigma_N}$, while the set $\mathcal{B}$ of variables is the set of $\boldsymbol{G}$-nodes.
- Classical CFG-productions $X \longrightarrow aY_0Y_1 \ldots Y_{n-1}$ in ϵ-Greibach normal form, that is, $a \in \Sigma + \{\epsilon\}$, can be expressed by

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph $\boldsymbol{\Sigma_N}$ with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in \boldsymbol{N}$. ($\boldsymbol{\emptyset_N}$ is terminal.)

(1) A CFG à la Walters (CFW) γ over Σ is a faithful cm-graph morphism $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma_N}$ with $\boldsymbol{G}$ finite.

- Terminals (= elements of Σ) label the edges of $\boldsymbol{\Sigma_N}$, while the set $\mathcal{B}$ of variables is the set of $\boldsymbol{G}$-nodes.
- Classical CFG-productions $X \longrightarrow aY_0 Y_1 \ldots Y_{n-1}$ in ϵ-Greibach normal form, that is, $a \in \Sigma + \{\epsilon\}$, can be expressed by

$$(X \xrightarrow{\varphi} Y_0 \ldots Y_{n-1})\gamma = (H \xrightarrow{a} H^n)$$

Walters' approach slightly generalized

Definition

(0) Any set Σ induces a cm-graph $\boldsymbol{\Sigma_N}$ with a single node H and $\Sigma + \{\epsilon\}$ for all hom-sets $[H, H^n]$, $n \in \boldsymbol{N}$. ($\boldsymbol{\emptyset_N}$ is terminal.)

(1) A CFG à la Walters (CFW) γ over Σ is a faithful cm-graph morphism $\boldsymbol{G} \xrightarrow{\gamma} \boldsymbol{\Sigma_N}$ with $\boldsymbol{G}$ finite.

- Terminals (= elements of Σ) label the edges of $\boldsymbol{\Sigma_N}$, while the set $\mathcal{B}$ of variables is the set of $\boldsymbol{G}$ -nodes.
- Classical CFG-productions $X \longrightarrow aY_0Y_1 \dots Y_{n-1}$ in ϵ-Greibach normal form, that is, $a \in \Sigma + \{\epsilon\}$, can be expressed by

$$(X \xrightarrow{\varphi} Y_0 \dots Y_{n-1})\gamma = (H \xrightarrow{a} H^n)$$

or simply as $X \xrightarrow{a} Y_0 \dots Y_{n-1}$, since γ is faithful.

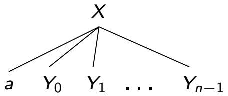
Trees and words

Trees and words

- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:

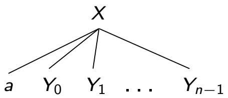
Trees and words

- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:

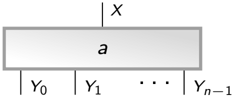


Trees and words

- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:

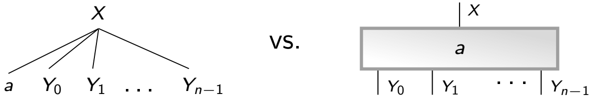


vs.



Trees and words

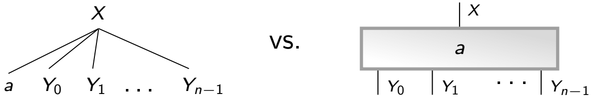
- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:



- for the language recognized by a G - node S , roughly speaking,

Trees and words

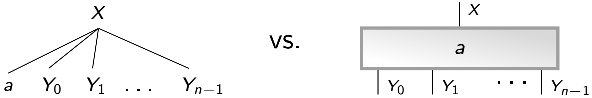
- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:



- for the language recognized by a G- node S, roughly speaking,
  - freely extend γ to a cm-functor $\gamma^{\star}$ between "free cm-categories over cm-graphs" (in analogy to forming free categories over a graphs);

Trees and words

▷ Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:

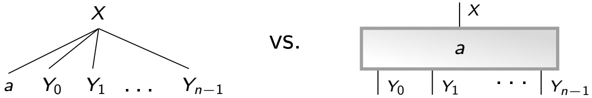


▷ for the language recognized by a $\boldsymbol{G}$ - node S , roughly speaking,

- freely extend γ to a cm-functor $\gamma^\star$ between "free cm-categories over cm-graphs" (in analogy to forming free categories over a graphs);
- consider the γ -image of the hom-set $\langle S, \epsilon \rangle G^\star$ in $\Sigma_N^\star$;

Trees and words

▷ Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:

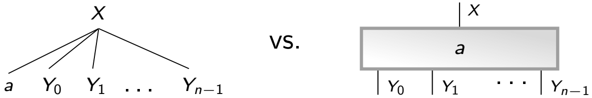


▷ for the language recognized by a G - node S , roughly speaking,

- freely extend γ to a cm-functor $\gamma^\star$ between "free cm-categories over cm-graphs" (in analogy to forming free categories over a graphs);
- consider the γ -image of the hom-set $\langle S, \epsilon \rangle G^\star$ in $\Sigma_N^\star$;
- extract words over Σ from the resulting diagrams in $\Sigma_N^\star$; these so-called yields constitute the string-language generated by γ and S .

Trees and words

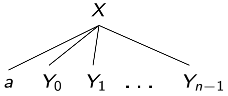
- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:



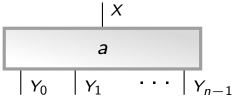
- for the language recognized by a G - node S , roughly speaking,
  - freely extend γ to a cm-functor $\gamma^\star$ between "free cm-categories over cm-graphs" (in analogy to forming free categories over a graphs);
  - consider the γ -image of the hom-set $\langle S, \epsilon\rangle G^\star$ in $\Sigma_N^\star$;
  - extract words over Σ from the resulting diagrams in $\Sigma_N^\star$; these so-called yields constitute the string-language generated by γ and S .
  - Optionally, one can view Σ_N as a reflexive cm-graph, which results in a somewhat simpler free cm-category $\Sigma_N^\star$.

Trees and words

- Instead of traditional node-labeled derivation trees, Poincaré duality now yields trees with somewhat different building blocks:



vs.



- for the language recognized by a $\boldsymbol{G}$ - node S, roughly speaking,
  - freely extend γ to a cm-functor $\gamma^\star$ between "free cm-categories over cm-graphs" (in analogy to forming free categories over a graphs);
  - consider the γ-image of the hom-set $\langle S, \epsilon \rangle G^\star$ in $\boldsymbol{\Sigma}_N^\star$;
  - extract words over Σ from the resulting diagrams in $\boldsymbol{\Sigma}_N^\star$; these so-called yields constitute the string-language generated by γ and S.
  - Optionally, one can view $\boldsymbol{\Sigma}_N$ as a reflexive cm-graph, which results in a somewhat simpler free cm-category $\boldsymbol{\Sigma}_N^\star$.
- As terminals are not limited to leaves, we need to switch from positional ordering of trees to temporal ordering (rotation by $\pi/2$ indicates this), which requires some notion of current position.

Strategy: towards 2PDAs

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.

Strategy: towards 2PDAs

- ▷ CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- ▷ Juxtaposing a second stack to the first one, the interface between them determines the current position:

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.
- We will employ two stack alphabets (= sets of variables) $\mathcal{B}$ and $\mathcal{C}$, which à priori need not be disjoint.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.
- We will employ two stack alphabets (= sets of variables) $\mathcal{B}$ and $\mathcal{C}$, which à priori need not be disjoint. But to indicate the current position, we use color-coded disjoint copies $\mathcal{B}$ and $\mathcal{C}$ for the lower/upper stack.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.
- We will employ two stack alphabets ($=$ sets of variables) $\mathcal{B}$ and $\mathcal{C}$, which à priori need not be disjoint. But to indicate the current position, we use color-coded disjoint copies $\mathcal{B}$ and $\mathcal{C}$ for the lower/upper stack. Their union makes up the set of $\boldsymbol{G}$- nodes.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.
- We will employ two stack alphabets (= sets of variables) $\mathcal{B}$ and $\mathcal{C}$, which à priori need not be disjoint. But to indicate the current position, we use color-coded disjoint copies $\mathcal{B}$ and $\mathcal{C}$ for the lower/upper stack. Their union makes up the set of $\boldsymbol{G}$- nodes. Moreover, we require the outputs of cm-edges to inherit the input's color.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.
- We will employ two stack alphabets (= sets of variables) $\mathcal{B}$ and $\mathcal{C}$, which à priori need not be disjoint. But to indicate the current position, we use color-coded disjoint copies $\mathcal{B}$ and $\mathcal{C}$ for the lower/upper stack. Their union makes up the set of $\boldsymbol{G}$- nodes. Moreover, we require the outputs of cm-edges to inherit the input's color.
- Transitions take the form $AB \xrightarrow{a} \Gamma\Delta$ with A, B not both empty (acceptance by empty stack), $a \in \Sigma + \{\epsilon\}$, and $\langle \Gamma, \Delta \rangle \in \mathcal{B}^\star \times \mathcal{C}^\star$.

Strategy: towards 2PDAs

- CFGs with the constraint of only left-derivations being allowed essentially are single state PDAs that accept by empty stack. Hence such pure PDAs may have been disregarded as uninteresting.
- Juxtaposing a second stack to the first one, the interface between them determines the current position: from here the first element on each side is visible, resp., the information that some stack is empty.
- We will employ two stack alphabets (= sets of variables) $\mathcal{B}$ and $\mathcal{C}$, which à priori need not be disjoint. But to indicate the current position, we use color-coded disjoint copies $\mathcal{B}$ and $\mathcal{C}$ for the lower/upper stack. Their union makes up the set of $\boldsymbol{G}$- nodes. Moreover, we require the outputs of cm-edges to inherit the input's color.
- Transitions take the form $AB \xrightarrow{a} \Gamma\Delta$ with A, B not both empty (acceptance by empty stack), $a \in \Sigma + \{\epsilon\}$, and $\langle \Gamma, \Delta \rangle \in \mathcal{B}^\star \times \mathcal{C}^\star$.
- Left and right moves $AB \xrightarrow{\epsilon} \epsilon AB$ and $AB \xrightarrow{\epsilon} AB\epsilon$ just change the current position.

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC$$
$$\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA$$
$$\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB$$

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC \qquad @A \xrightarrow{a} @\epsilon$$
$$\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA \qquad @B \xrightarrow{b} @\epsilon$$
$$\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB \qquad @C \xrightarrow{c} @\epsilon$$

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{lll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @
\end{array}$$

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$
\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$
\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | *b* | *a* | *b* | *c* | *c* | *a* | *b* | *c* | *a* | can take the form:

Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$
\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:

$$\frac{S}{\star}$$ current position

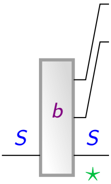
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X @ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



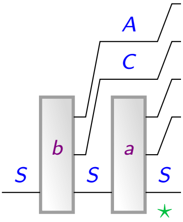
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



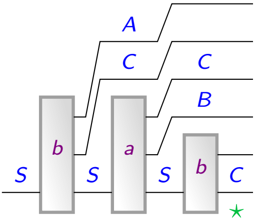
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$
\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



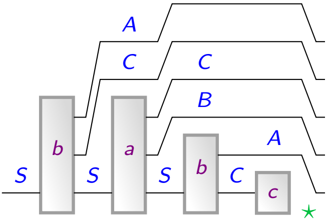
Example: $MIX = \{\, w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \,\}$

When using the initial stack $\epsilon\, S$ and the following transitions

$$
\begin{array}{llll}
\epsilon\, S \xrightarrow{a} \epsilon\, SBC \mid \epsilon\, BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon\, S \xrightarrow{b} \epsilon\, SCA \mid \epsilon\, CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X @ \\
\epsilon\, S \xrightarrow{c} \epsilon\, SAB \mid \epsilon\, AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

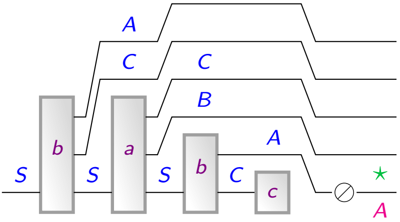
$$\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC \qquad @A \xrightarrow{a} @\epsilon \qquad A@ \xrightarrow{a} \epsilon @ \qquad @X \xrightarrow{\epsilon} @X\epsilon$$

$$\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA \qquad @B \xrightarrow{b} @\epsilon \qquad B@ \xrightarrow{b} \epsilon @ \qquad X@ \xrightarrow{\epsilon} \epsilon X@$$

$$\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB \qquad @C \xrightarrow{c} @\epsilon \qquad C@ \xrightarrow{c} \epsilon @ \qquad \text{moves!}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of b a b c c a b c a can take the form:



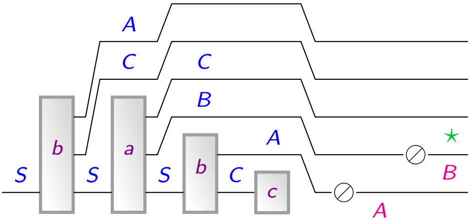
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC$	$@A \xrightarrow{a} @\epsilon$	$A@ \xrightarrow{a} \epsilon @$	$@X \xrightarrow{\epsilon} @X\epsilon$
$\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA$	$@B \xrightarrow{b} @\epsilon$	$B@ \xrightarrow{b} \epsilon @$	$X@ \xrightarrow{\epsilon} \epsilon X@$
$\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB$	$@C \xrightarrow{c} @\epsilon$	$C@ \xrightarrow{c} \epsilon @$	moves!

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of b a b c c a b c a can take the form:



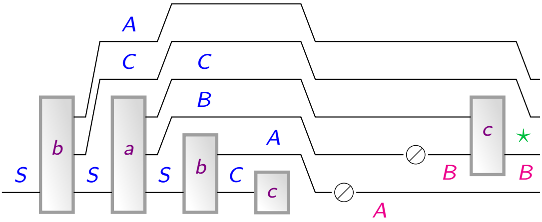
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



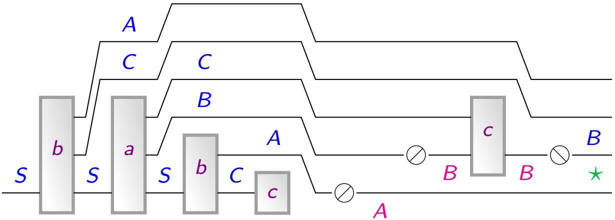
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X @ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



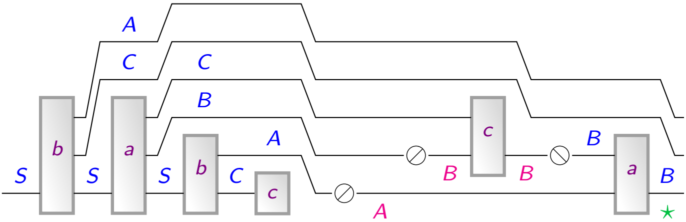
Example: $MIX = \{\, w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \,\}$

When using the initial stack $\epsilon\, S$ and the following transitions

$$
\begin{array}{llll}
\epsilon\, S \xrightarrow{a} \epsilon\, SBC \mid \epsilon\, BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon\, S \xrightarrow{b} \epsilon\, SCA \mid \epsilon\, CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon\, S \xrightarrow{c} \epsilon\, SAB \mid \epsilon\, AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



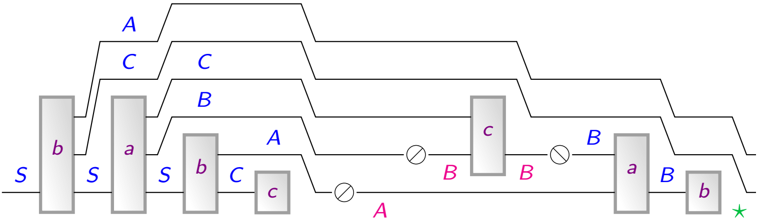
Example: $MIX = \{ w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \}$

When using the initial stack ϵS and the following transitions

$$\begin{array}{llll}
\epsilon S \xrightarrow{a} \epsilon SBC \mid \epsilon BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon S \xrightarrow{b} \epsilon SCA \mid \epsilon CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X@ \\
\epsilon S \xrightarrow{c} \epsilon SAB \mid \epsilon AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



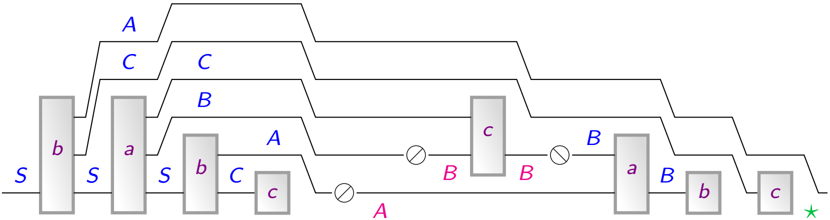
Example: $MIX = \{\, w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \,\}$

When using the initial stack $\epsilon\, S$ and the following transitions

$$
\begin{array}{llll}
\epsilon\, S \xrightarrow{a} \epsilon\, SBC \mid \epsilon\, BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon\, S \xrightarrow{b} \epsilon\, SCA \mid \epsilon\, CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X @ \\
\epsilon\, S \xrightarrow{c} \epsilon\, SAB \mid \epsilon\, AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}
$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of | b | a | b | c | c | a | b | c | a | can take the form:



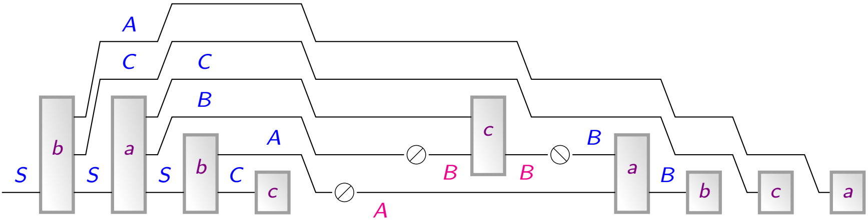
Example: $MIX = \{\, w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c \,\}$

When using the initial stack $\epsilon\, S$ and the following transitions

$$\begin{array}{llll}
\epsilon\, S \xrightarrow{a} \epsilon\, SBC \mid \epsilon\, BC & @A \xrightarrow{a} @\epsilon & A@ \xrightarrow{a} \epsilon @ & @X \xrightarrow{\epsilon} @X\epsilon \\
\epsilon\, S \xrightarrow{b} \epsilon\, SCA \mid \epsilon\, CA & @B \xrightarrow{b} @\epsilon & B@ \xrightarrow{b} \epsilon @ & X@ \xrightarrow{\epsilon} \epsilon X @ \\
\epsilon\, S \xrightarrow{c} \epsilon\, SAB \mid \epsilon\, AB & @C \xrightarrow{c} @\epsilon & C@ \xrightarrow{c} \epsilon @ & \text{moves!}
\end{array}$$

with $@ \in \{A, B, C, \epsilon\}$ and $X \in \{A, B, C\}$.

The derivation of b a b c c a b c a can take the form:

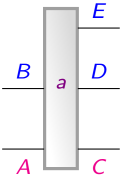


What's wrong with this picture?

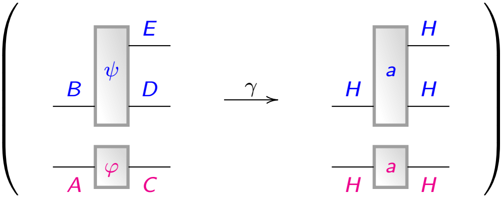
As the diagram above is not built from cm-edges of the proposed cm-graph G , we need to re-interpret its components, *e.g.*, by splitting them up. *E.g.*,

What's wrong with this picture?

As the diagram above is not built from cm-edges of the proposed cm-graph $\boldsymbol{G}$, we need to re-interpret its components, *e.g.*, by splitting them up. *E.g.*,

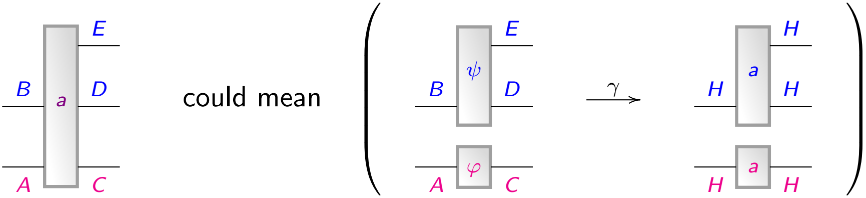


could mean



What's wrong with this picture?

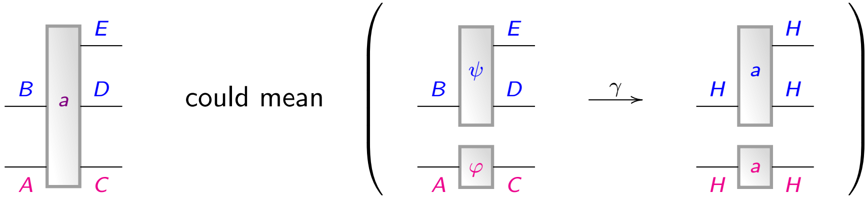
As the diagram above is not built from cm-edges of the proposed cm-graph G, we need to re-interpret its components, *e.g.*, by splitting them up. *E.g.*,



- With G a disjoint union of a red and a blue component, any pairing $\langle \varphi, \psi \rangle$ with the same Σ- labels under γ would produce a valid transition for the ss2PDA.

What's wrong with this picture?

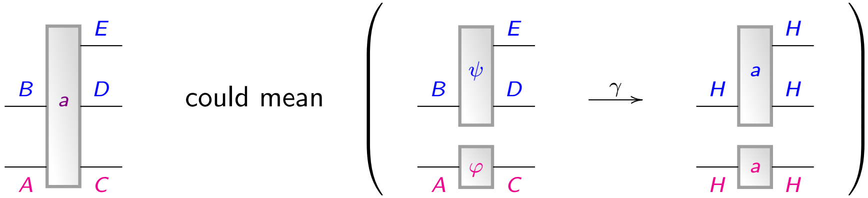
As the diagram above is not built from cm-edges of the proposed cm-graph $\boldsymbol{G}$, we need to re-interpret its components, *e.g.*, by splitting them up. *E.g.*,



- With $\boldsymbol{G}$ a disjoint union of a red and a blue component, any pairing $\langle \varphi, \psi \rangle$ with the same Σ- labels under γ would produce a valid transition for the ss2PDA. Is this really what we want?

What's wrong with this picture?

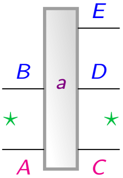
As the diagram above is not built from cm-edges of the proposed cm-graph G , we need to re-interpret its components, *e.g.*, by splitting them up. *E.g.*,



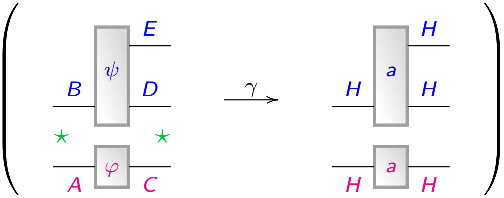
- With G a disjoint union of a red and a blue component, any pairing $\langle \varphi, \psi \rangle$ with the same Σ- labels under γ would produce a valid transition for the ss2PDA. Is this really what we want?
- Another problem is that at least here the current position does not really move to a different region of the diagram.

What’s wrong with this picture?

As the diagram above is not built from cm-edges of the proposed cm-graph $\boldsymbol{G}$, we need to re-interpret its components, *e.g.*, by splitting them up. *E.g.*,



could mean

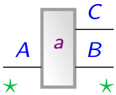


- ▷ With $\boldsymbol{G}$ a disjoint union of a red and a blue component, any pairing $\langle\varphi,\psi\rangle$ with the same Σ- labels under γ would produce a valid transition for the ss2PDA. Is this really what we want?
- ▷ Another problem is that at least here the current position does not really move to a different region of the diagram.

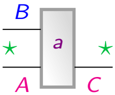
The missing ingredient

The missing ingredient

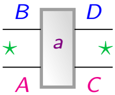
Instead of drawing, *e.g.*,



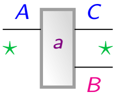
or



or



(but not

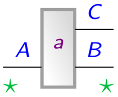


!)

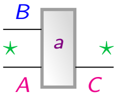
where the region of the current position does not really change,

The missing ingredient

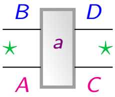
Instead of drawing, *e.g.*,



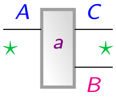
or



or



(but not

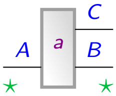


!)

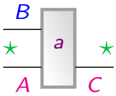
where the region of the current position does not really change, let us introduce explicit vertical separations,

The missing ingredient

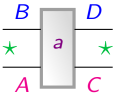
Instead of drawing, *e.g.*,



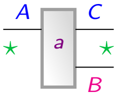
or



or

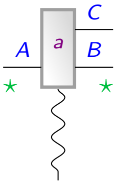


(but not

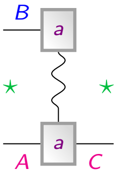


!)

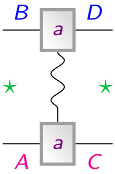
where the region of the current position does not really change, let us introduce explicit vertical separations,



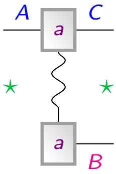
or



or



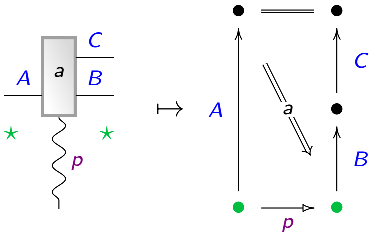
(but not



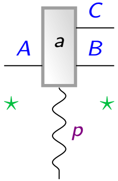
!)

Poincaré duality now yields, *e.g.*,

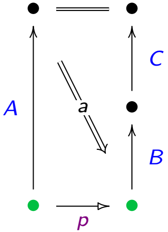
Poincaré duality now yields, *e.g.*,



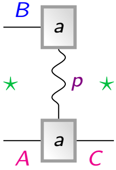
Poincaré duality now yields, *e.g.*,



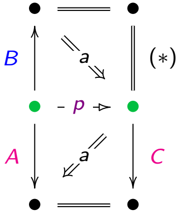
$\mapsto$



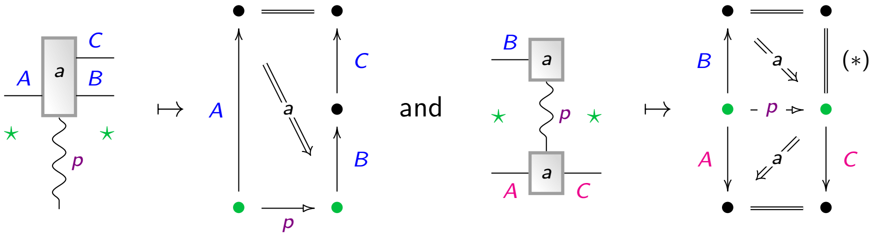
and



$\mapsto$

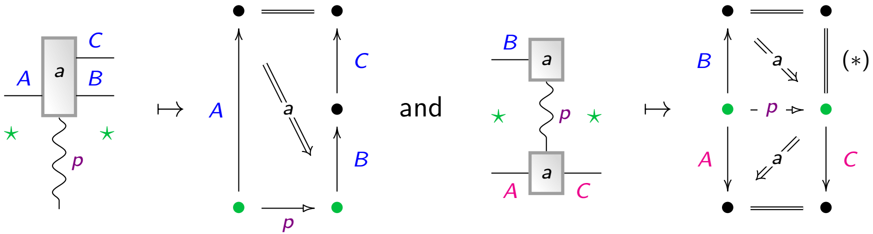


Poincaré duality now yields, *e.g.*,



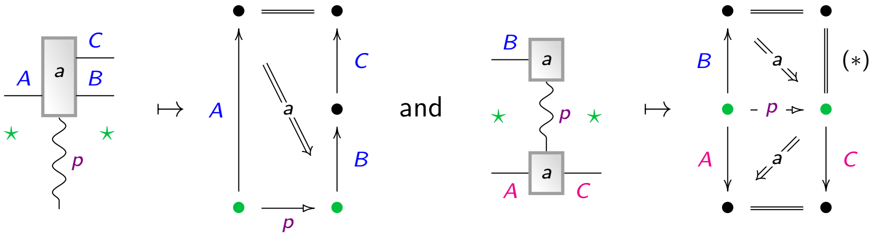
The regions (=positions) have not yet been named.

Poincaré duality now yields, *e.g.*,



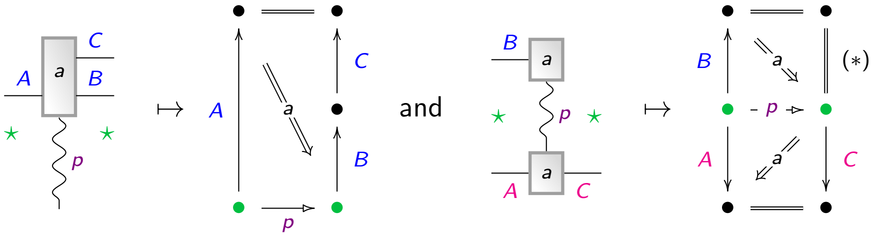
The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells.

Poincaré duality now yields, *e.g.*,

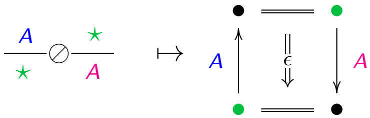


The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests

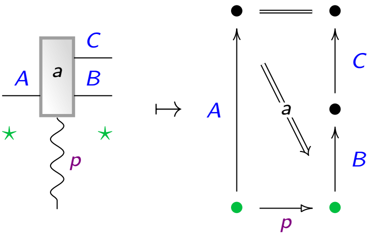
Poincaré duality now yields, *e.g.*,



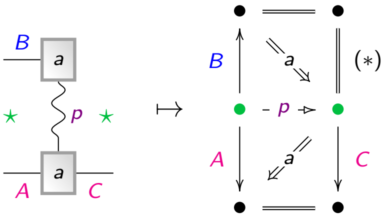
The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests



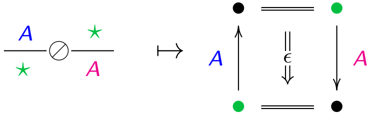
Poincaré duality now yields, *e.g.*,



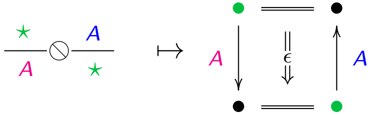
and



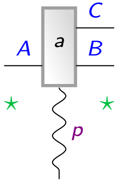
The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests



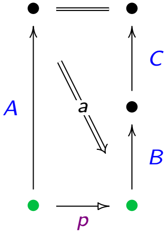
and



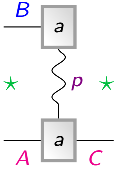
Poincaré duality now yields, *e.g.*,



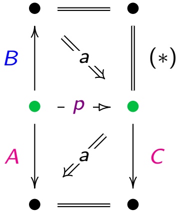
$\mapsto$



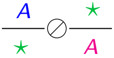
and



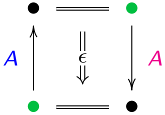
$\mapsto$



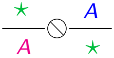
The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests



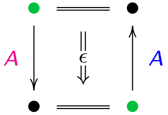
$\mapsto$



and

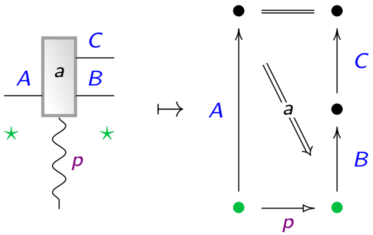


$\mapsto$

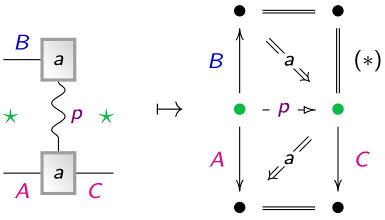


which resembles adjunctions.

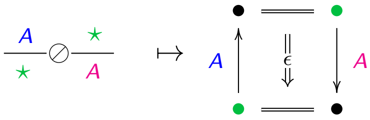
Poincaré duality now yields, *e.g.*,



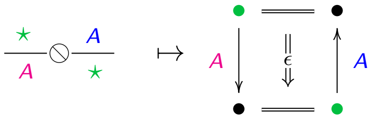
and



The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests

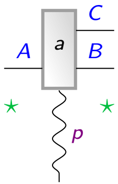


and

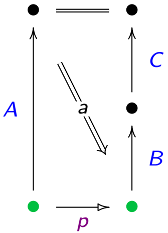


which resembles adjunctions. Vertical $=$-arrows are empty words $(*)$,

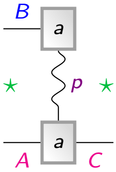
Poincaré duality now yields, *e.g.*,



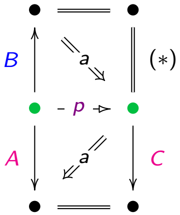
$\mapsto$



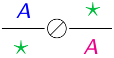
and



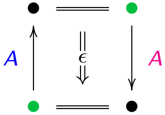
$\mapsto$



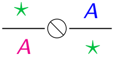
The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests



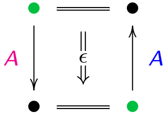
$\mapsto$



and

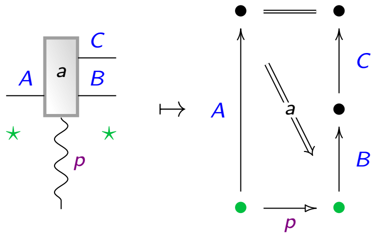


$\mapsto$

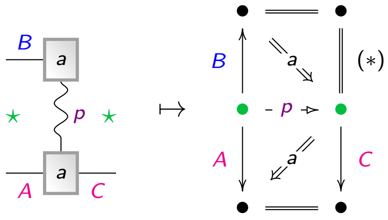


which resembles adjunctions. Vertical $=$-arrows are empty words $(*)$, what about the horizontal ones?

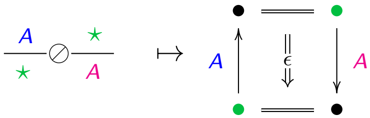
Poincaré duality now yields, *e.g.*,



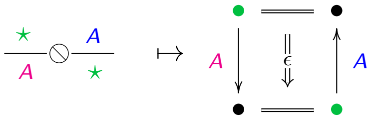
and



The regions (=positions) have not yet been named. Notice the opposite orientations of the red and blue vertical 1-cells. For moves this suggests



and



which resembles adjunctions. Vertical $=$-arrows are empty words $(*)$, what about the horizontal ones? The underlying graph has to be reflexiv!

Putting it all together (0)

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories.

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an “fc-cm-graph”, in analogy to Tom Leinster’s fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as “states”.
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.
- History need not be linear, but is distributed in as many threads as the current stack size.

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.
- History need not be linear, but is distributed in as many threads as the current stack size. The current state is visible from the current position,

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.
- History need not be linear, but is distributed in as many threads as the current stack size. The current state is visible from the current position, changing the latter jumps to another history thread.

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.
- History need not be linear, but is distributed in as many threads as the current stack size. The current state is visible from the current position, changing the latter jumps to another history thread.
- There is some resemblance with Gadducci and Montanari's tile model:

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.
- History need not be linear, but is distributed in as many threads as the current stack size. The current state is visible from the current position, changing the latter jumps to another history thread.
- There is some resemblance with Gadducci and Montanari's tile model:
  - for vertical arrows we have more freedom, as the horizontal codomain of a tile can be a word of stack symbols, or the direction can flip;

Putting it all together (0)

- Instead of of a cm-graph we seem to need a 2-dimensional structure, an "fc-cm-graph", in analogy to Tom Leinster's fc-multi-categories. Objects are the positions, vertical arrows are finite sequences of stack symbols, while horizontal arrows might be interpreted as "states".
- So the framework for handling more than one state is alread in place; states do not have to be grafted on artificially.
- While vertically the stack can grow and shrink and be traversed, horizontally, history can only grow, one state at each tick of the clock.
- History need not be linear, but is distributed in as many threads as the current stack size. The current state is visible from the current position, changing the latter jumps to another history thread.
- There is some resemblance with Gadducci and Montanari's tile model:
  - for vertical arrows we have more freedom, as the horizontal codomain of a tile can be a word of stack symbols, or the direction can flip;
  - non-trivial horizontal arrows are constrained to vertical domains of tiles.

Putting it all together (1)

Putting it all together (1)

- In principle, both states and stack symbols can be typed (by means of positions).

Putting it all together (1)

- In principle, both states and stack symbols can be typed (by means of positions). The usefulness of this flexibility still needs to be exploited.

Putting it all together (1)

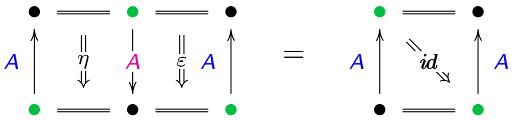
- In principle, both states and stack symbols can be typed (by means of positions). The usefulness of this flexibility still needs to be exploited.
- The term "adjoints" was used delibertately, even though we are just working with graphs.

Putting it all together (1)

- In principle, both states and stack symbols can be typed (by means of positions). The usefulness of this flexibility still needs to be exploited.
- The term "adjoints" was used delibertately, even though we are just working with graphs. Just like the distinguished loops of a reflexive graph are intended to become identities in the free category, the distinguished cells are intended to become units, resp., counits of adjuncions between vertical arrows in the free fc-cm-category:

Putting it all together (1)

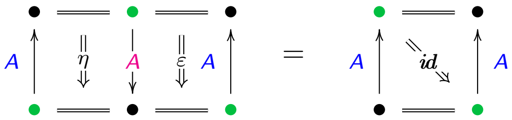
- In principle, both states and stack symbols can be typed (by means of positions). The usefulness of this flexibility still needs to be exploited.
- The term "adjoints" was used delibertately, even though we are just working with graphs. Just like the distinguished loops of a reflexive graph are intended to become identities in the free category, the distinguished cells are intended to become units, resp., counits of adjuncions between vertical arrows in the free fc-cm-category:



(and the dual)

Putting it all together (1)

- In principle, both states and stack symbols can be typed (by means of positions). The usefulness of this flexibility still needs to be exploited.
- The term "adjoints" was used delibertately, even though we are just working with graphs. Just like the distinguished loops of a reflexive graph are intended to become identities in the free category, the distinguished cells are intended to become units, resp., counits of adjuncions between vertical arrows in the free fc-cm-category:

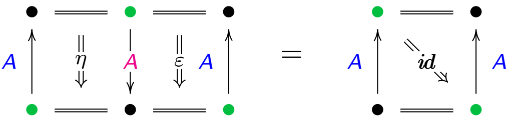


(and the dual)

- The corresponding fc-cm-graph generated by Σ has one position, vertical arrows H and H, and no non-trivial horizontal arrows.

Putting it all together (1)

- In principle, both states and stack symbols can be typed (by means of positions). The usefulness of this flexibility still needs to be exploited.
- The term "adjoints" was used delibertately, even though we are just working with graphs. Just like the distinguished loops of a reflexive graph are intended to become identities in the free category, the distinguished cells are intended to become units, resp., counits of adjuncions between vertical arrows in the free fc-cm-category:



(and the dual)

- The corresponding fc-cm-graph generated by Σ has one position, vertical arrows H and H, and no non-trivial horizontal arrows. The unit/counit cells are only labeled by ϵ.

To do

To do

- Does it make sense to have conditional moves?

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean?

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean? Perhaps transducers can be modeled?

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean? Perhaps transducers can be modeled?
- Surely, other targets than Σ-induced fc-cm-graphs must make sense.

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean? Perhaps transducers can be modeled?
- Surely, other targets than Σ-induced fc-cm-graphs must make sense.
- Formulate everything properly for fc-cm-categories.

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean? Perhaps transducers can be modeled?
- Surely, other targets than Σ-induced fc-cm-graphs must make sense.
- Formulate everything properly for fc-cm-categories.
- How does the other side of the Gothendieck construction look like?

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean? Perhaps transducers can be modeled?
- Surely, other targets than Σ-induced fc-cm-graphs must make sense.
- Formulate everything properly for fc-cm-categories.
- How does the other side of the Gothendieck construction look like?
- Dropping faithfulness requires the use of $\boldsymbol{spn}$ instead of $\boldsymbol{rel}$.

To do

- Does it make sense to have conditional moves?
- The mechanism allows for the processing of pairs of input symbols, what does that mean? Perhaps transducers can be modeled?
- Surely, other targets than Σ-induced fc-cm-graphs must make sense.
- Formulate everything properly for fc-cm-categories.
- How does the other side of the Gothendieck construction look like?
- Dropping faithfulness requires the use of $\boldsymbol{spn}$ instaed of $\boldsymbol{rel}$. Other bicategories may be usable as well.