Syntax-Directed Translations and Quasi-alphabetic Tree Bimorphisms — Revisited

Andreas Maletti and **Cătălin Ionuţ Tîrnăucă**

Research Group on Mathematical Linguistics
Rovira i Virgili University
Tarragona, Spain
andreas.maletti@urv.cat
catalinionut.tirnauca@estudiants.urv.cat

20th of May, 2009

1. Introduction
2. Preliminaries
   - Tree Homomorphisms
   - Tree Bimorphisms
3. Syntax-Directed Translation Schemata
4. New Connection between SDTSs and Tree Bimorphisms
5. Closure under Composition

Syntax-Based Machine Translation

- **Syntax-based machine translation** was established by the demanding need of systems used in practical translations between natural languages [Knight 2007]

- An ideal such system should [Knight 2007]

  1. perform difficult rotations (reorder parts of sentences)
  2. model syntax-sensitive transformations (i.e., tree transformations)
  3. have composability (smaller parts easier to test, train, etc.)
  4. .....

Syntax-Based Machine Translation

- **Syntax-based machine translation** was established by the demanding need of systems used in practical translations between natural languages [Knight 2007]
- An ideal such system should [Knight 2007]
  1. perform difficult rotations (reorder parts of sentences)
  2. model syntax-sensitive transformations (i.e., tree transformations)
  3. have composability (smaller parts easier to test, train, etc.)
  4. .....

Syntax-Based Machine Translation

- **Syntax-based machine translation** was established by the demanding need of systems used in practical translations between natural languages [Knight 2007]

- An ideal such system should [Knight 2007]

  1. perform difficult rotations (reorder parts of sentences)
  2. model syntax-sensitive transformations (i.e., tree transformations)
  3. have composability (smaller parts easier to test, train, etc.)
  4. .....

Syntax-Based Machine Translation

- **Syntax-based machine translation** was established by the demanding need of systems used in practical translations between natural languages [Knight 2007]

- An ideal such system should [Knight 2007]

  1. perform difficult rotations (reorder parts of sentences)
  2. model syntax-sensitive transformations (i.e., tree transformations)
  3. have composability (smaller parts easier to test, train, etc.)
  4. .....

Syntax-Based Machine Translation

- **Syntax-based machine translation** was established by the demanding need of systems used in practical translations between natural languages [Knight 2007]
- An ideal such system should [Knight 2007]
  1. perform difficult rotations (reorder parts of sentences)
  2. model syntax-sensitive transformations (i.e., tree transformations)
  3. have composability (smaller parts easier to test, train, etc.)
  4. .....

How to Model Tree Transformations?

- **Tree transducers**
  - easy to implement: many available tools, e.g. TIBURON/ISI
  - closure under composition does not hold for the main types [Engelfriet 1975, Gécseg & Steinby 1984, Knight 2007]

- **Tree bimorphisms**
  - algebraic mechanisms, harder to implement (no available tools)
  - composition easier to establish by imposing suitable restrictions on their constituents [Arnold & Dauchet 1982, Bozapalidis 1992, Steinby 1986, Takahashi 1972]

- **Synchronous grammars**
  - naturally define difficult rotations: e.g. Arabic-English
  - quite easy to implement
  - very few composition results are known [Shieber 2004]

How to Model Tree Transformations?

- **Tree transducers**
  - easy to implement: many available tools, e.g. TIBURON/ISI
  - closure under composition does not hold for the main types [Engelfriet 1975, Gécseg & Steinby 1984, Knight 2007]
- **Tree bimorphisms**
  - algebraic mechanisms, harder to implement (no available tools)
  - composition easier to establish by imposing suitable restrictions on their constituents [Arnold & Dauchet 1982, Bozapalidis 1992, Steinby 1986, Takahashi 1972]
- **Synchronous grammars**
  - naturally define difficult rotations: e.g. Arabic-English
  - quite easy to implement
  - very few composition results are known [Shieber 2004]

How to Model Tree Transformations?

- **Tree transducers**
  - easy to implement: many available tools, e.g. TIBURON/ISI
  - closure under composition does not hold for the main types [Engelfriet 1975, Gécseg & Steinby 1984, Knight 2007]
- **Tree bimorphisms**
  - algebraic mechanisms, harder to implement (no available tools)
  - composition easier to establish by imposing suitable restrictions on their constituents [Arnold & Dauchet 1982, Bozapalidis 1992, Steinby 1986, Takahashi 1972]
- **Synchronous grammars**
  - naturally define difficult rotations: e.g. Arabic-English
  - quite easy to implement
  - very few composition results are known [Shieber 2004]

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  - is effectively equal to syntax-directed translation schemata of [Aho & Ullman 1972] (in terms of translations)
  - is closed under composition (restricted) and preserves recognizability
  - naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  - a smaller class of tree bimorphisms defines the same translations as SDTSs
  - a more general closure under composition (no restriction)
  - the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to syntax-directed translation schemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Synchronous Grammars as Tree Bimorphisms

- How about describing synchronous grammars with the help of tree bimorphisms? [Shieber 2004]
- [Steinby & Tîrnăucă 2007] introduced the class of **quasi-alphabetic tree bimorphisms** which:
  1. is effectively equal to **s**yntax-**d**irected **t**ranslation **s**chemata of [Aho & Ullman 1972] (in terms of translations)
  2. is closed under composition (restricted) and preserves recognizability
  3. naturally describes the tree transformations defined by SDTSs
- Overall, we strengthen these results:
  1. a smaller class of tree bimorphisms defines the same translations as SDTSs
  2. a more general closure under composition (no restriction)
  3. the smaller class of tree bimorphisms describes tree transformations defined only by SDTSs in a normal form

Tree Homomorphisms - Basic Facts

Notations

- Σ ranked alphabet, V leaf alphabet (variables), X formal variables
- $X_k = \{x_1, x_2, \ldots, x_k\}$
- $\Sigma(V) = \{f(v_1, \ldots, v_k) \mid f \in \Sigma_k, v_1, \ldots, v_k \in V\}$
- $T_\Sigma(V)$ = set of all Σ-**trees** indexed by variables V
- **tree languages** = subsets of $T_\Sigma(V)$

Definition (Tree Homomorphism) [Gécseg & Steinby 1984]

A **tree homomorphism** $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$ is determined by a mapping $\varphi_V\colon V \to T_\Delta(Y)$ and mappings $\varphi_k\colon \Sigma_k \to T_\Delta(Y \cup X_k)$ for every $k \geqslant 0$ as follows:

1. $v\varphi = \varphi_V(v)$ for every $v \in V$
2. $f(t_1, \ldots, t_k)\varphi = \varphi_k(f)(x_1 \leftarrow t_1\varphi, \ldots, x_k \leftarrow t_k\varphi)$ for every $t_1, \ldots, t_k \in T_\Sigma(V)$ and $f \in \Sigma_k$.

Tree Homomorphisms - Basic Facts

Notations

- Σ ranked alphabet, V leaf alphabet (variables), X formal variables
- $X_k = \{x_1, x_2, \ldots, x_k\}$
- $\Sigma(V) = \{f(v_1, \ldots, v_k) \mid f \in \Sigma_k, v_1, \ldots, v_k \in V\}$
- $T_\Sigma(V)$ = set of all Σ-**trees** indexed by variables V
- **tree languages** = subsets of $T_\Sigma(V)$

Definition (Tree Homomorphism) [Gécseg & Steinby 1984]

A **tree homomorphism** $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$ is determined by a mapping $\varphi_V\colon V \to T_\Delta(Y)$ and mappings $\varphi_k\colon \Sigma_k \to T_\Delta(Y \cup X_k)$ for every $k \geqslant 0$ as follows:

1. $v\varphi = \varphi_V(v)$ for every $v \in V$
2. $f(t_1, \ldots, t_k)\varphi = \varphi_k(f)(x_1 \leftarrow t_1\varphi, \ldots, x_k \leftarrow t_k\varphi)$ for every $t_1, \ldots, t_k \in T_\Sigma(V)$ and $f \in \Sigma_k$.

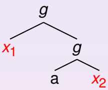
Types of a Tree Homomorphism $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$

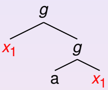
- **linear**: no copying (each x_i appears at most once)
- **complete**: no deletion (each x_i appears at least once)
- **normalized**: $\varphi_3(f(x_1, x_2, x_3)) = g(x_1, g(a, x_2))$ $\quad \varphi_3(f(x_1, x_2, x_3)) = g(x_1, g(a, x_1))$
- **quasi-alphabetic** (qaH): linear + complete + $\varphi_V(v) \in Y$ + $\varphi_k(f) \in \Delta(Y \cup X_k)$
- **symbol-to-symbol** (ssH): quasi-alphabetic + $\varphi_k(f) \in \Delta(X_k)$
- **alphabetic** (aH): symbol-to-symbol + normalized (relabelings)

Types of a Tree Homomorphism $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$

- **linear**: no copying (each x_i appears at most once)
- **complete**: no deletion (each x_i appears at least once)
- **normalized**: $\varphi_3(f(x_1, x_2, x_3)) =$ $\varphi_3(f(x_1, x_2, x_3)) =$
- **quasi-alphabetic** (qaH): linear + complete + $\varphi_V(v) \in Y$ + $\varphi_k(f) \in \Delta(Y \cup X_k)$
- **symbol-to-symbol** (ssH): quasi-alphabetic + $\varphi_k(f) \in \Delta(X_k)$
- **alphabetic** (aH): symbol-to-symbol + normalized (relabelings)

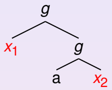
Types of a Tree Homomorphism $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$

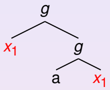
- **linear**: no copying (each x_i appears at most once)
- **complete**: no deletion (each x_i appears at least once)
- **normalized**: $\varphi_3(f(x_1, x_2, x_3)) =$ 

$\varphi_3(f(x_1, x_2, x_3)) =$ 

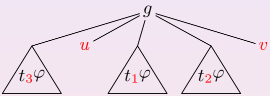
- **quasi-alphabetic** (qaH): linear + complete + $\varphi_V(v) \in Y$ + $\varphi_k(f) \in \Delta(Y \cup X_k)$
- **symbol-to-symbol** (ssH): quasi-alphabetic + $\varphi_k(f) \in \Delta(X_k)$
- **alphabetic** (aH): symbol-to-symbol + normalized (relabelings)

Types of a Tree Homomorphism $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$

- **linear**: no copying (each x_i appears at most once)
- **complete**: no deletion (each x_i appears at least once)
- **normalized**: $\varphi_3(f(x_1, x_2, x_3)) =$ 

$\varphi_3(f(x_1, x_2, x_3)) =$ 

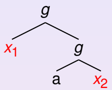
- **quasi-alphabetic** (qaH): linear + complete + $\varphi_V(v) \in Y$ + $\varphi_k(f) \in \Delta(Y \cup X_k)$

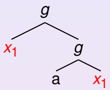
$\varphi(f(t_1, t_2, t_3)) =$ 

with $u, v \in Y$

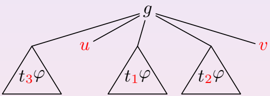
- **symbol-to-symbol** (ssH): quasi-alphabetic + $\varphi_k(f) \in \Delta(X_k)$
- **alphabetic** (aH): symbol-to-symbol + normalized (relabelings)

Types of a Tree Homomorphism $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$

- **linear**: no copying (each x_i appears at most once)
- **complete**: no deletion (each x_i appears at least once)
- **normalized**: $\varphi_3(f(x_1, x_2, x_3)) =$ 

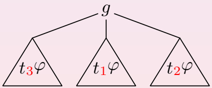
$\varphi_3(f(x_1, x_2, x_3)) =$ 

- **quasi-alphabetic** (qaH): linear + complete + $\varphi_V(v) \in Y$ + $\varphi_k(f) \in \Delta(Y \cup X_k)$

$\varphi(f(t_1, t_2, t_3)) =$ 

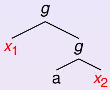
with $u, v \in Y$

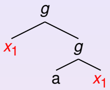
- **symbol-to-symbol** (ssH): quasi-alphabetic + $\varphi_k(f) \in \Delta(X_k)$

$\varphi(f(t_1, t_2, t_3)) =$ 

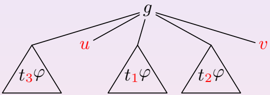
- **alphabetic** (aH): symbol-to-symbol + normalized (relabelings)

Types of a Tree Homomorphism $\varphi \colon T_\Sigma(V) \to T_\Delta(Y)$

- **linear**: no copying (each x_i appears at most once)
- **complete**: no deletion (each x_i appears at least once)
- **normalized**: $\varphi_3(f(x_1, x_2, x_3)) =$ 

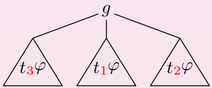
$\varphi_3(f(x_1, x_2, x_3)) =$ 

- **quasi-alphabetic** (qaH): linear + complete + $\varphi_V(v) \in Y$ + $\varphi_k(f) \in \Delta(Y \cup X_k)$

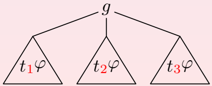
$\varphi(f(t_1, t_2, t_3)) =$ 

with $u, v \in Y$

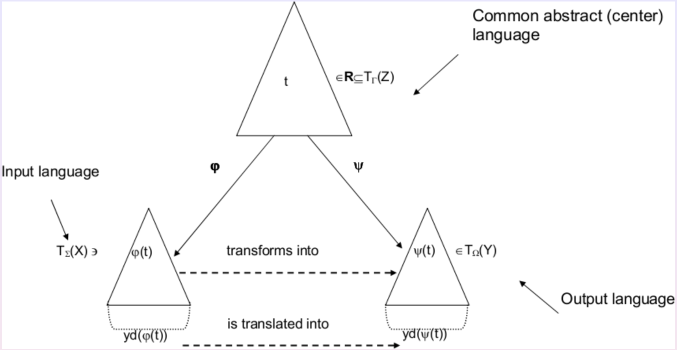
- **symbol-to-symbol** (ssH): quasi-alphabetic + $\varphi_k(f) \in \Delta(X_k)$

$\varphi(f(t_1, t_2, t_3)) =$ 

- **alphabetic** (aH): symbol-to-symbol + normalized (relabelings)

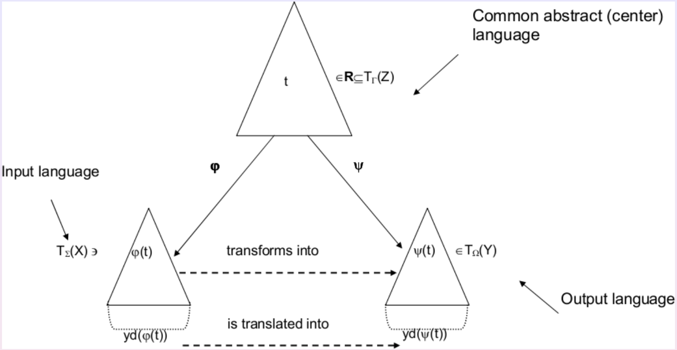
$\varphi(f(t_1, t_2, t_3)) =$ 

Graphical Representation and Notations



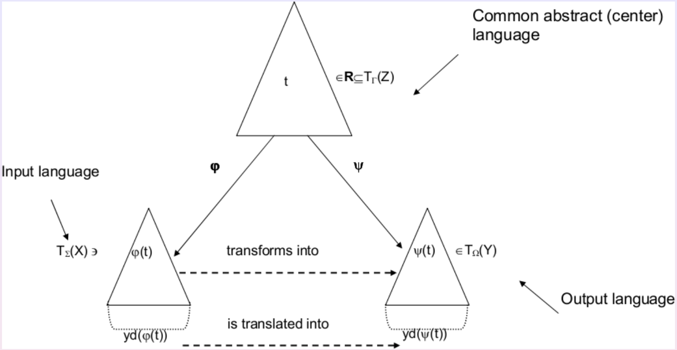
- $B = (\varphi, L, \psi)$ a tree bimorphism
- **tree transformation** defined by B: $\tau_B = \{(\varphi(t), \psi(t)) \mid t \in L\}$
- **translation** defined by B: $\mathrm{yd}(\tau_B) = \{(\mathrm{yd}_{V\setminus\{e\}}(s), \mathrm{yd}_{Y\setminus\{e\}}(t)) \mid (s, t) \in \tau_B\}$
- e special variable, never output, acts as the empty string
- (φ, L, ψ) is **quasi-alphabetic** (**symbol-to-symbol**, **alphabetic**) if both φ and ψ have this property and L is a recognizable tree language

Graphical Representation and Notations



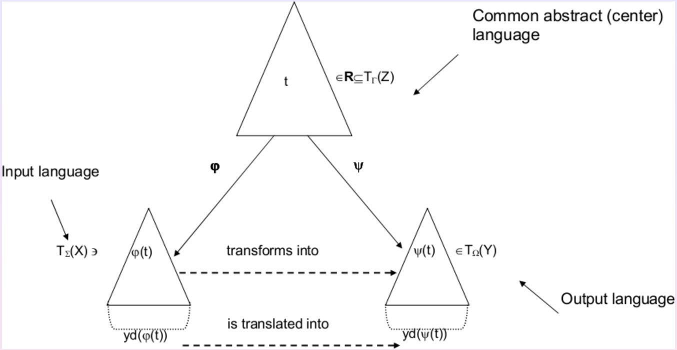
- $B = (\varphi, L, \psi)$ a tree bimorphism
- **tree transformation** defined by B: $\tau_B = \{(\varphi(t), \psi(t)) \mid t \in L\}$
- **translation** defined by B: $\text{yd}(\tau_B) = \{(\text{yd}_{V \setminus \{e\}}(s), \text{yd}_{Y \setminus \{e\}}(t)) \mid (s, t) \in \tau_B\}$
- e special variable, never output, acts as the empty string
- (φ, L, ψ) is **quasi-alphabetic** (**symbol-to-symbol**, **alphabetic**) if both φ and ψ have this property and L is a recognizable tree language

Graphical Representation and Notations



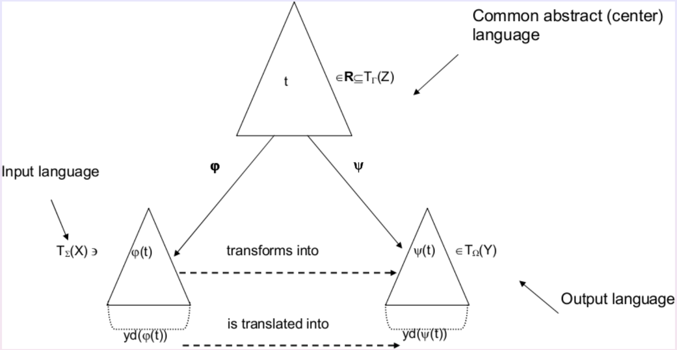
- $B = (\varphi, L, \psi)$ a tree bimorphism
- **tree transformation** defined by B: $\tau_B = \{(\varphi(t), \psi(t)) \mid t \in L\}$
- **translation** defined by B: $\text{yd}(\tau_B) = \{(\text{yd}_{V\setminus\{e\}}(s), \text{yd}_{Y\setminus\{e\}}(t)) \mid (s, t) \in \tau_B\}$
- e special variable, never output, acts as the empty string
- (φ, L, ψ) is **quasi-alphabetic** (**symbol-to-symbol**, **alphabetic**) if both φ and ψ have this property and L is a recognizable tree language

Graphical Representation and Notations



- $B = (\varphi, L, \psi)$ a tree bimorphism
- **tree transformation** defined by B: $\tau_B = \{(\varphi(t), \psi(t)) \mid t \in L\}$
- **translation** defined by B: $\mathrm{yd}(\tau_B) = \{(\mathrm{yd}_{V \setminus \{e\}}(s), \mathrm{yd}_{Y \setminus \{e\}}(t)) \mid (s, t) \in \tau_B\}$
- e special variable, never output, acts as the empty string
- (φ, L, ψ) is **quasi-alphabetic** (**symbol-to-symbol**, **alphabetic**) if both φ and ψ have this property and L is a recognizable tree language

Graphical Representation and Notations



- $B = (\varphi, L, \psi)$ a tree bimorphism
- **tree transformation** defined by B: $\tau_B = \{(\varphi(t), \psi(t)) \mid t \in L\}$
- **translation** defined by B: $\mathrm{yd}(\tau_B) = \{(\mathrm{yd}_{V\setminus\{e\}}(s), \mathrm{yd}_{Y\setminus\{e\}}(t)) \mid (s, t) \in \tau_B\}$
- e special variable, never output, acts as the empty string
- (φ, L, ψ) is **quasi-alphabetic** (**symbol-to-symbol**, **alphabetic**) if both φ and ψ have this property and L is a recognizable tree language

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T** $= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \rightarrow u, w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is simple if in each production the nonterminals occur in same order in u and w

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T**$= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \to u; w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is **simple** if in each production the nonterminals occur in same order in u and w.

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T**$= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \to u; w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is **simple** if in each production the nonterminals occur in same order in u and w.

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T**$= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \to u; w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is **simple** if in each production the nonterminals occur in same order in u and w.

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T**$= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \rightarrow u; w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is **simple** if in each production the nonterminals occur in same order in u and w.

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T**$= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \rightarrow u; w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is **simple** if in each production the nonterminals occur in same order in u and w.

Definitions

What Is an SDTS?

Two CFGs over a common set of nonterminals (productions have **associated nonterminals**). Derivations are obtained by applying 2 suitable rules to associated nonterminals [Aho & Ullman 1972].

Definition (Syntax-Directed Translation Schema)[Aho & Ullman 1972]

An **SDTS** is a device **T**$= (N, V, Y, P, S)$, where:

- N is a finite set of **nonterminal symbols**,
- V is a finite **input alphabet**,
- Y is a finite **output alphabet**,
- $S \in N$ is the **start symbol**, and
- P is a finite set of **productions** of the form:

$$p = A \rightarrow u; w$$

where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$ and the nonterminals in w are a permutation of those in u.

T is **simple** if in each production the nonterminals occur in same order in u and w.

An Example

Let $T = (\{S, A, B\}, \{0, 1\}, \{a, b, c\}, P, S)$, where P has the rules:

$$p_1 = \mathbf{S} \to 0\,\mathbf{A}\,11\,\mathbf{B}\,0\,\mathbf{B}\;;\;\mathbf{B}\,\mathbf{A}\,\mathrm{cba}\,\mathbf{B}\,\mathrm{aa}$$
$$p_2 = \mathbf{A} \to \mathbf{A}\,\mathbf{A}\;;\;\mathbf{A}\,\mathrm{a}\,\mathbf{A}$$
$$p_3 = \mathbf{A} \to \epsilon\;;\;\epsilon$$
$$p_4 = \mathbf{B} \to 01\;;\;\epsilon$$

A **derivation** in T is:

$$\begin{aligned}(S, S) &\overset{p_1}{\Longrightarrow}_T (0A11B0B, BAcbaBaa)\\ &\overset{p_2}{\Longrightarrow}_T (0AA11B0B, BAaAcbaBaa)\\ &\overset{p_3}{\Longrightarrow}{}^{*}_T (011B0B, BacbaBaa)\\ &\overset{p_4}{\Longrightarrow}{}^{*}_T (01101001, acbaaa)\ .\end{aligned}$$

Definition (Syntax-Directed Translation)

The **translation** defined by a (simple) SDTS T is the relation

$$\tau_T = \{(u, w) \in V^* \times Y^* \mid (S, S) \Rightarrow^*_T (u, w)\},$$

and it will be called **(simple) syntax-directed translation**.

An Example

Let $T = (\{S, A, B\}, \{0, 1\}, \{a, b, c\}, P, S)$, where P has the rules:

$$p_1 = \mathbf{S} \rightarrow 0\,\mathbf{A}\,11\,\mathbf{B}\,0\,\mathbf{B}\,;\ \mathbf{B}\,\mathbf{A}\,\mathrm{cba}\,\mathbf{B}\,\mathrm{aa}$$
$$p_2 = \mathbf{A} \rightarrow \mathbf{A}\,\mathbf{A}\,;\ \mathbf{A}\,\mathrm{a}\,\mathbf{A}$$
$$p_3 = \mathbf{A} \rightarrow \epsilon\,;\ \epsilon$$
$$p_4 = \mathbf{B} \rightarrow 01\,;\ \epsilon$$

A **derivation** in T is:

$$\begin{aligned}(S, S) &\xRightarrow{p_1}_T (0A11B0B, BAcbaBaa)\\ &\xRightarrow{p_2}_T (0AA11B0B, BAaAcbaBaa)\\ &\xRightarrow{p_3}{}^*_T (011B0B, BacbaBaa)\\ &\xRightarrow{p_4}{}^*_T (01101001, acbaaa)\ .\end{aligned}$$

Definition (Syntax-Directed Translation)

The **translation** defined by a (simple) SDTS T is the relation

$$\tau_T = \{(u, w) \in V^* \times Y^* \mid (S, S) \Rightarrow^*_T (u, w)\},$$

and it will be called **(simple) syntax-directed translation**.

Normal Form

Normal Form [Aho & Ullman 1969b]

A (simple) SDTS (N, V, Y, P, S) is in **normal form** if for every production $A \to u \, ; w$ in P

- $u, w \in N^*$ or
- $u \in V \cup \{\varepsilon\}$ and $w \in Y \cup \{\varepsilon\}$.

Proposition [Aho & Ullman 1969b, Lemma 3.1]

For every SDTS T there exists an SDTS T' in normal form such that $\tau_T = \tau_{T'}$. If T is simple, then T' can be chosen to be simple as well.

Theorem [Aho & Ullman 1969a, Theorem 2]

The class of all simple syntax-directed translations is properly contained in the class of all syntax-directed translations.

Normal Form

Normal Form [Aho & Ullman 1969b]

A (simple) SDTS (N, V, Y, P, S) is in **normal form** if for every production $A \to u\,;\,w$ in P

- $u, w \in N^*$ or
- $u \in V \cup \{\varepsilon\}$ and $w \in Y \cup \{\varepsilon\}$.

Proposition [Aho & Ullman 1969b, Lemma 3.1]

For every SDTS T there exists an SDTS T' in normal form such that $\tau_T = \tau_{T'}$. If T is simple, then T' can be chosen to be simple as well.

Theorem [Aho & Ullman 1969a, Theorem 2]

The class of all simple syntax-directed translations is properly contained in the class of all syntax-directed translations.

Normal Form

Normal Form [Aho & Ullman 1969b]

A (simple) SDTS (N, V, Y, P, S) is in **normal form** if for every production $A \to u\,;\, w$ in P

- $u, w \in N^*$ or
- $u \in V \cup \{\varepsilon\}$ and $w \in Y \cup \{\varepsilon\}$.

Proposition [Aho & Ullman 1969b, Lemma 3.1]

For every SDTS T there exists an SDTS T' in normal form such that $\tau_T = \tau_{T'}$. If T is simple, then T' can be chosen to be simple as well.

Theorem [Aho & Ullman 1969a, Theorem 2]

The class of all simple syntax-directed translations is properly contained in the class of all syntax-directed translations.

Syntax-Directed Translations and Tree Bimorphisms

Previous Result [Steinby & Tîrnăucă 2007, Theorem 5.7]

The class of syntax-directed translations is effectively equal to the class of translations defined by quasi-alphabetic tree bimorphisms.

New Result

The class of syntax-directed translations (respectively simple) coincides with the class of translations defined by symbol-to-symbol (respectively, alphabetic) tree bimorphisms.

Immediate Consequence

The class of all translations defined by alphabetic tree bimorphisms is properly contained in the class of all translations defined by symbol-to-symbol tree bimorphisms.

Syntax-Directed Translations and Tree Bimorphisms

Previous Result [Steinby & Tîrnăucă 2007, Theorem 5.7]

The class of syntax-directed translations is effectively equal to the class of translations defined by quasi-alphabetic tree bimorphisms.

New Result

The class of syntax-directed translations (respectively simple) coincides with the class of translations defined by symbol-to-symbol (respectively, alphabetic) tree bimorphisms.

Immediate Consequence

The class of all translations defined by alphabetic tree bimorphisms is properly contained in the class of all translations defined by symbol-to-symbol tree bimorphisms.

Syntax-Directed Translations and Tree Bimorphisms

Previous Result [Steinby & Tîrnăucă 2007, Theorem 5.7]

The class of syntax-directed translations is effectively equal to the class of translations defined by quasi-alphabetic tree bimorphisms.

New Result

The class of syntax-directed translations (respectively simple) coincides with the class of translations defined by symbol-to-symbol (respectively, alphabetic) tree bimorphisms.

Immediate Consequence

The class of all translations defined by alphabetic tree bimorphisms is properly contained in the class of all translations defined by symbol-to-symbol tree bimorphisms.

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! ▸ Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

1. every symbol-to-symbol tree bimorphism is quasi-alphabetic
2. the definition of translation of a bimorphism is slightly modified: special symbol
3. the SDTS of [Steinby & Tîrnăucă 2007] is simple if the bimorphism is alphabetic

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

- every symbol-to-symbol tree bimorphism is quasi-alphabetic
- the definition of translation of a bimorphism is slightly modified: special symbol e
- the SDTS of [Steinby & Tîrnăucă 2007] is simple if the bimorphism is alphabetic

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! ▸ Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

- every symbol-to-symbol tree bimorphism is quasi-alphabetic

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! ▸ Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

- every symbol-to-symbol tree bimorphism is quasi-alphabetic
- the definition of translation of a bimorphism is slightly modified: special symbol e
- the SDTS of [Steinby & Tîrnăucă 2007] is simple if the bimorphism is alphabetic

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! ▸ Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

- every symbol-to-symbol tree bimorphism is quasi-alphabetic
- the definition of translation of a bimorphism is slightly modified: special symbol e
- the SDTS of [Steinby & Tîrnăucă 2007] is simple if the bimorphism is alphabetic

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! ▸ Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

- every symbol-to-symbol tree bimorphism is quasi-alphabetic
- the definition of translation of a bimorphism is slightly modified: special symbol e
- the SDTS of [Steinby & Tîrnăucă 2007] is simple if the bimorphism is alphabetic

The Construction

From SDTSs to Tree Bimorphisms

1. Given a (simple) SDTS, assume it is in normal form
2. The difference from [Steinby & Tîrnăucă 2007] is in the homomorphisms: change the behaviour of the productions that only have terminals on the right-hand sides! ▸ Example
3. With previous constructions, do a similar proof as the one in [Steinby & Tîrnăucă 2007] (it works)

From Tree Bimorphisms to SDTSs

A minor change of the proof of [Steinby & Tîrnăucă 2007] yields the result since

- every symbol-to-symbol tree bimorphism is quasi-alphabetic
- the definition of translation of a bimorphism is slightly modified: special symbol e
- the SDTS of [Steinby & Tîrnăucă 2007] is simple if the bimorphism is alphabetic

Closure under Composition

Two More Notations

Let Σ be a ranked alphabet, V a set of variables

- **variable-free** tree languages = subsets of T_Σ
- **almost variable-free** tree languages = subsets of $T_\Sigma \cup V$

Previous Result [Steinby & Tîrnăucă 2007]

The class of tree transformations defined by quasi-alphabetic tree bimorphisms with a variable-free center tree language is closed under composition.

New Result

The class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition.

Closure under Composition

Two More Notations

Let Σ be a ranked alphabet, V a set of variables

- **variable-free** tree languages = subsets of T_Σ
- **almost variable-free** tree languages = subsets of $T_\Sigma \cup V$

Previous Result [Steinby & Tîrnăucă 2007]

The class of tree transformations defined by quasi-alphabetic tree bimorphisms with a variable-free center tree language is closed under composition.

New Result

The class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition.

Closure under Composition

Two More Notations

Let Σ be a ranked alphabet, V a set of variables

- **variable-free** tree languages = subsets of T_Σ
- **almost variable-free** tree languages = subsets of $T_\Sigma \cup V$

Previous Result [Steinby & Tîrnăucă 2007]

The class of tree transformations defined by quasi-alphabetic tree bimorphisms with a variable-free center tree language is closed under composition.

New Result

The class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition.

Closure under Composition

Two More Notations

Let Σ be a ranked alphabet, V a set of variables

- **variable-free** tree languages = subsets of T_Σ
- **almost variable-free** tree languages = subsets of $T_\Sigma \cup V$

Previous Result [Steinby & Tîrnăucă 2007]

The class of tree transformations defined by quasi-alphabetic tree bimorphisms with a variable-free center tree language is closed under composition.

New Result

The class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition.

Closure under Composition: Sketch of the Proof

1. One homomorphism of a quasi-alphabetic tree bimorphism can always be normalized (for details, see Proposition 8 of the paper)

2. Get rid of variables as much as possible: the class of tree transformations defined by quasi-alphabetic tree bimorphisms is equal with the class of tree transformations defined by quasi-alphabetic tree bimorphisms with an almost variable-free center tree language (see Lemma 9)

3. All almost variable-free trees with the same image under two normalized quasi-alphabetic tree homomorphisms can be paired up in a product data structure $T_{\Sigma \times \Delta}(V \times Y)$ (see Lemma 10)

4. Condition for closure under composition (see Lemmata 11 and 12): the class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition if

$$\{t \in T_\Omega \cup V \mid \varphi(t) = \psi(t)\}$$

is a recognizable tree language for every ranked alphabet Ω, set V of variables, and pair (φ, ψ) of normalized quasi-alphabetic tree homomorphisms

Closure under Composition: Sketch of the Proof

1. One homomorphism of a quasi-alphabetic tree bimorphism can always be normalized (for details, see Proposition 8 of the paper)

2. Get rid of variables as much as possible: the class of tree transformations defined by quasi-alphabetic tree bimorphisms is equal with the class of tree transformations defined by quasi-alphabetic tree bimorphisms with an almost variable-free center tree language (see Lemma 9)

3. All almost variable-free trees with the same image under two normalized quasi-alphabetic tree homomorphisms can be paired up in a product data structure $T_{\Sigma \times \Delta}(V \times Y)$ (see Lemma 10)

4. Condition for closure under composition (see Lemmata 11 and 12): the class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition if

$$\{t \in T_\Omega \cup V \mid \varphi(t) = \psi(t)\}$$

is a recognizable tree language for every ranked alphabet Ω, set V of variables, and pair (φ, ψ) of normalized quasi-alphabetic tree homomorphisms

Closure under Composition: Sketch of the Proof

1. One homomorphism of a quasi-alphabetic tree bimorphism can always be normalized (for details, see Proposition 8 of the paper)

2. Get rid of variables as much as possible: the class of tree transformations defined by quasi-alphabetic tree bimorphisms is equal with the class of tree transformations defined by quasi-alphabetic tree bimorphisms with an almost variable-free center tree language (see Lemma 9)

3. All almost variable-free trees with the same image under two normalized quasi-alphabetic tree homomorphisms can be paired up in a product data structure $T_{\Sigma \times \Delta}(V \times Y)$ (see Lemma 10)

4. Condition for closure under composition (see Lemmata 11 and 12): the class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition if

$$\{t \in T_\Omega \cup V \mid \varphi(t) = \psi(t)\}$$

is a recognizable tree language for every ranked alphabet Ω, set V of variables, and pair (φ, ψ) of normalized quasi-alphabetic tree homomorphisms

Closure under Composition: Sketch of the Proof

1. One homomorphism of a quasi-alphabetic tree bimorphism can always be normalized (for details, see Proposition 8 of the paper)

2. Get rid of variables as much as possible: the class of tree transformations defined by quasi-alphabetic tree bimorphisms is equal with the class of tree transformations defined by quasi-alphabetic tree bimorphisms with an almost variable-free center tree language (see Lemma 9)

3. All almost variable-free trees with the same image under two normalized quasi-alphabetic tree homomorphisms can be paired up in a product data structure $T_{\Sigma \times \Delta}(V \times Y)$ (see Lemma 10)

4. Condition for closure under composition (see Lemmata 11 and 12): the class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition if

$$\{t \in T_\Omega \cup V \mid \varphi(t) = \psi(t)\}$$

is a recognizable tree language for every ranked alphabet Ω, set V of variables, and pair (φ, ψ) of normalized quasi-alphabetic tree homomorphisms

References I

- Alfred V. Aho and Jeffrey D. Ullman.
  Properties of syntax directed translations.
  J. Comput. Syst. Sci., 3(3):319–334, 1969.
- Alfred V. Aho and Jeffrey D. Ullman.
  Syntax directed translations and the pushdown assembler.
  J. Comput. Syst. Sci., 3(1):37–56, 1969.
- Alfred V. Aho and Jeffrey D. Ullman.
  Parsing, volume 1 of *The Theory of Parsing, Translation, and Compiling*.
  Prentice Hall, 1972.
- André Arnold and Max Dauchet.
  Morphismes et bimorphismes d'arbres.
  Theor. Comput. Sci., 20(1):33–93, 1982.
- Symeon Bozapalidis.
  Alphabetic tree relations.
  Theor. Comput. Sci., 99(2):177–211, 1992.
- Joost Engelfriet.
  Bottom-up and top-down tree transformations: A comparison.
  Math. Syst. Theory, 9(3):198–231, 1975.
- Ferenc Gécseg and Magnus Steinby.
  Tree Automata.
  Akadémiai Kiadó, Budapest, 1984.

References II

Kevin Knight.
Capturing practical natural language transformations.
Machine Translation, 21(2):121–133, 2007.

Kevin Knight and Jonathan Graehl.
An overview of probabilistic tree transducers for natural language processing.
In *Proc. CICLing*, volume 3406 of LNCS, pages 1–24. Springer, 2005.

Stuart M. Shieber.
Synchronous grammars as tree transducers.
In *Proc. TAG+7*, pages 88–95, 2004.

Magnus Steinby.
On certain algebraically defined tree transformations.
In *Proc. Algebra, Combinatorics and Logic in Computer Science*, volume 42 of *Colloquia Mathematica Societatis János Bolyai*, pages 745–764. North-Holland, 1986.

Magnus Steinby and Cătălin Ionuţ Tîrnăucă.
Defining syntax-directed translations by tree bimorphisms.
Theor. Comput. Sci., 2009.
to appear.
`http://dx.doi.org/10.1016/j.tcs.2009.03.009`.

Masako Takahashi.
Primitive transformations of regular sets and recognizable sets.
In *Proc. ICALP*, pages 475–480. North-Holland, 1972.

That's all folks!

Thank you!

From SDTS to Tree Bimorphism

Back

Let $T = (\{A, B\}, \{a, b\}, \{0, 1\}, P, A)$, where P has the rules:

$p_1 = \mathbf{A} \to \mathbf{A}\,\mathbf{B}\,\mathbf{A}\,;\ \mathbf{B}\,\mathbf{A}\,\mathbf{A}\,,\ \sigma = (2, 3, 1)$

$p_2 = \mathbf{B} \to \mathbf{B}\,\mathbf{B}\,;\ \mathbf{B}\,\mathbf{B},\ \sigma = (2, 1),$

$p_3 = \mathbf{A} \to \mathrm{aba}\,;\ \epsilon$, and

$p_4 = \mathbf{B} \to \mathrm{b}\,;\ 10.$

We turn P into a ranked alphabet: productions are symbols, number of its nonterminals gives the rank (e.g., $\mathrm{rk}(p_2) = 2$).
We construct two symbol-to-symbol tree homomorphisms

$$\varphi\colon T_{\{p_1, p_2\}}(\{p_3, p_4\}) \to T_{\{p_1, p_2\}}(\{a, b, e\}) \qquad \psi\colon T_{\{p_1, p_2\}}(\{p_3, p_4\}) \to T_{\{p_1, p_2\}}(\{0, 1, e\})$$

defined by

$$\begin{aligned} \varphi_{\{p_3, p_4\}}(p_3) &= aba \\ \varphi_{\{p_3, p_4\}}(p_4) &= b \\ \varphi_3(p_1) &= p_1(x_1, \ldots, x_3) \\ \varphi_2(p_2) &= p_2(x_1, \ldots, x_2) \end{aligned} \qquad \begin{aligned} \psi_{\{p_3, p_4\}}(p_3) &= e \\ \psi_{\{p_3, p_4\}}(p_4) &= 10 \\ \psi_3(p_1) &= p_1(x_2, x_3, x_1) \\ \psi_2(p_2) &= p_2(x_2, x_1) \end{aligned}$$

If T is simple (all permutations are identity), then clearly φ and ψ are alphabetic.