Introduction to Abstract Categorial Grammars

First Course

Motivations

The Categorial Grammar tradition

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition
- (syntactic) categories as types

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition
- (syntactic) categories as types
  - a (small) finite set of atomic types

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition
- (syntactic) categories as types
  - a (small) finite set of atomic types
  - together with type formation rules

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition
- (syntactic) categories as types
  - a (small) finite set of atomic types
  - together with type formation rules
- grammatical composition as a typing discipline

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition
- (syntactic) categories as types
  - a (small) finite set of atomic types
  - together with type formation rules
- grammatical composition as a typing discipline
  - grammatical composition obey the rules of a given typing system

The Categorial Grammar tradition

- a type-theoretic view of grammars and grammatical composition
- (syntactic) categories as types
  - a (small) finite set of atomic types
  - together with type formation rules
- grammatical composition as a typing discipline
  - grammatical composition obey the rules of a given typing system
  - the rules of grammatical composition are completely specified by the underlying type calculus

example: AB-grammars

Example: AB-grammars

Atomic types: N, NP, S

Example: AB-grammars

Atomic types: N, NP, S

Type formation rules: $\tau ::= a \mid (\tau \setminus \tau) \mid (\tau / \tau)$

Example: AB-grammars

Atomic types: N, NP, S

Type formation rules: $\tau ::= a \mid (\tau \setminus \tau) \mid (\tau / \tau)$

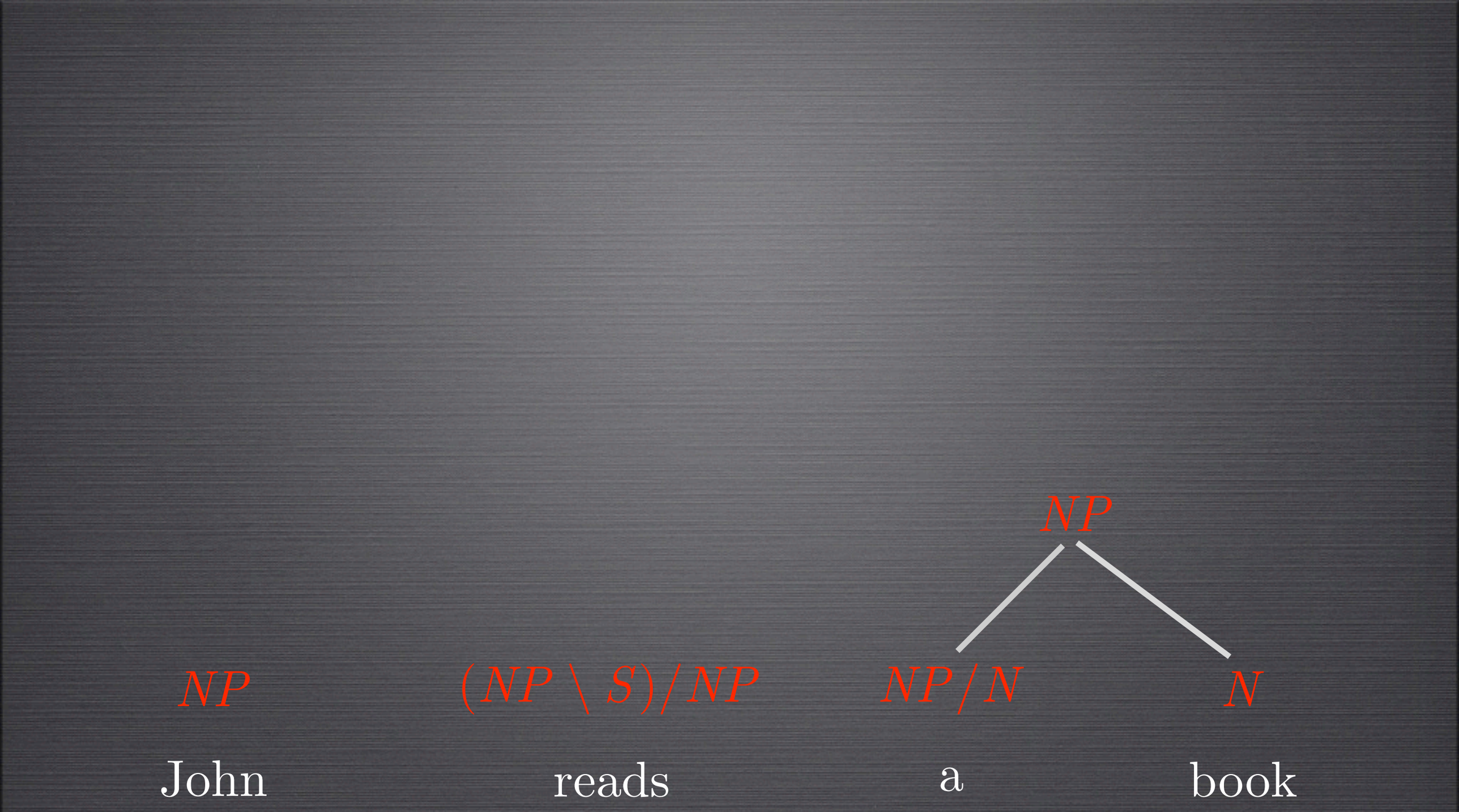
John	:	NP
reads	:	$(NP \setminus S)/NP$
a	:	NP/N
book	:	N

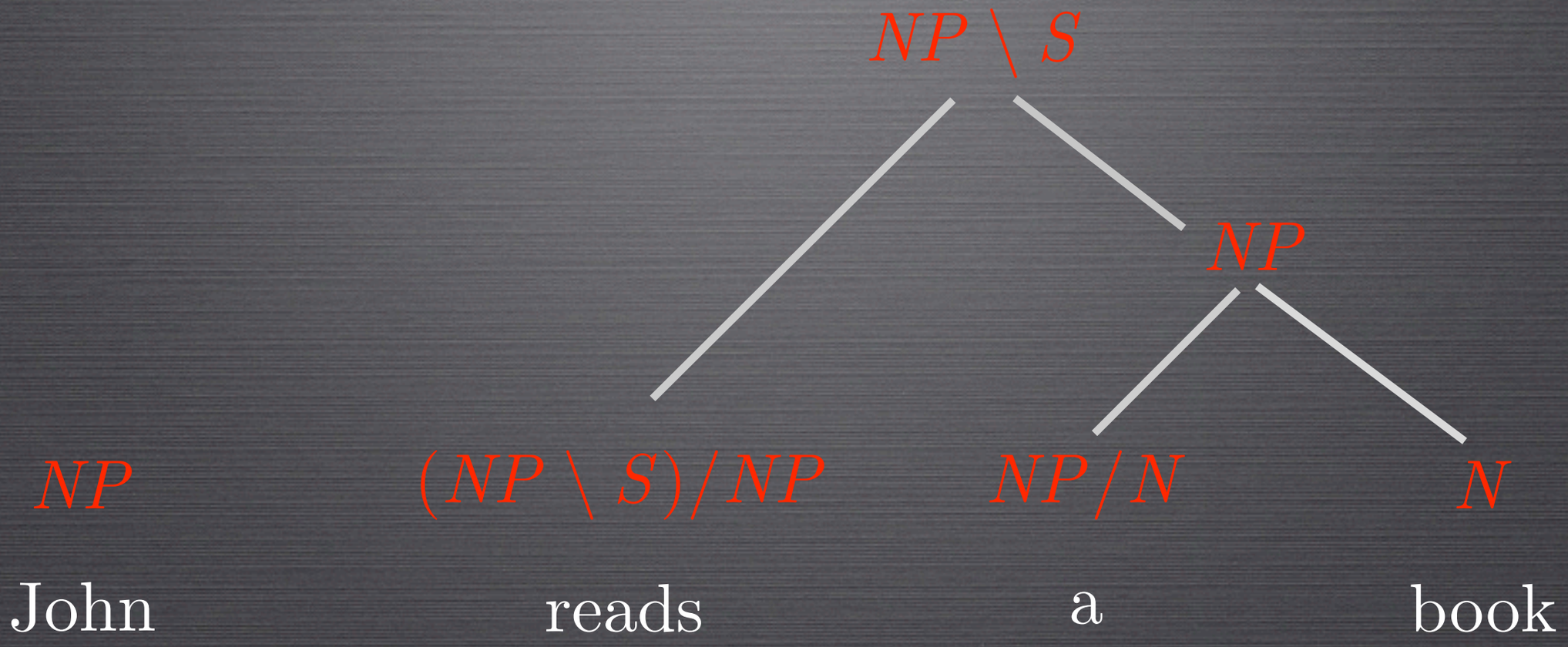
Type calculus: $\alpha,\ \alpha\backslash\beta \vdash \beta$

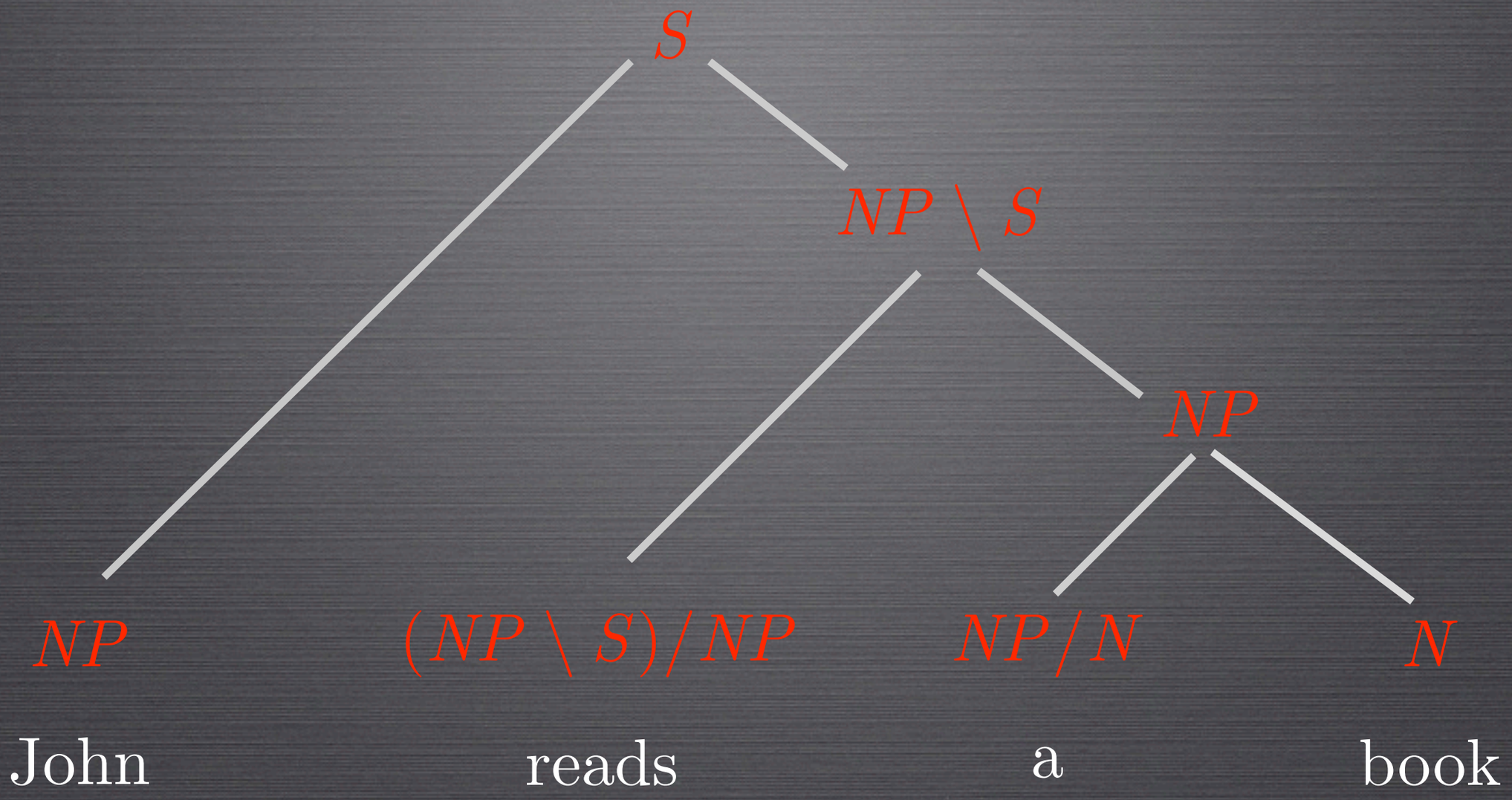
$\beta/\alpha,\ \alpha \vdash \beta$

John reads a book

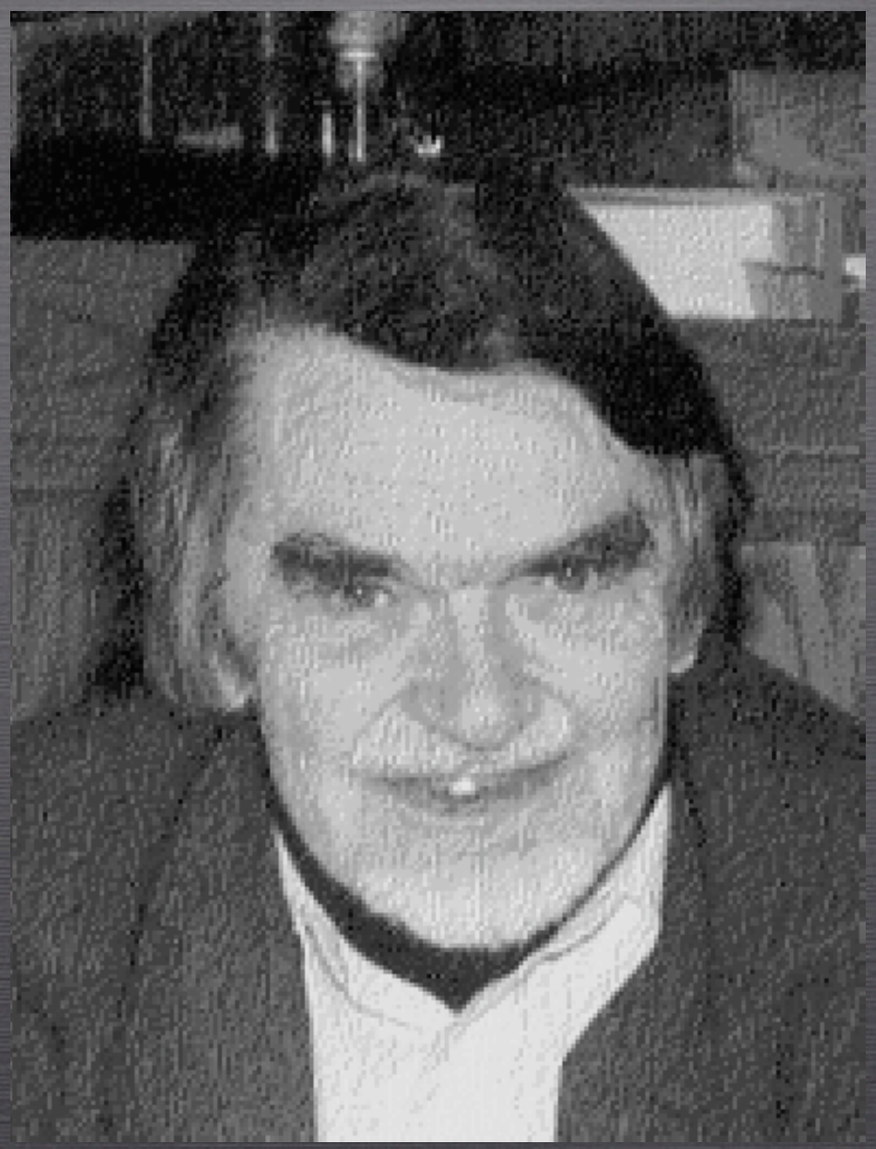
NP	$(NP \setminus S)/NP$	NP/N	N
John	reads	a	book


NP
NP
(NP \ S)/NP
NP/N
N
John
reads
a
book


NP \ S
NP
NP
NP
(NP \ S)/NP
NP/N
N
John
reads
a
book


S
NP \ S
NP
NP
(NP \ S)/NP
NP/N
N
John
reads
a
book

Lambek's legacy



Lambek's legacy

Functorial types, $A \setminus B$ and B / A, as implications:

Lambek's legacy

Functorial types, $A \setminus B$ and B / A, as implications:

- hypothetical reasoning

Lambek's legacy

Functorial types, $A \setminus B$ and B / A, as implications:

- hypothetical reasoning
- type calculus as a substructural logic
  (no weakening and no contraction)

Lambek's legacy

Functorial types, $A \setminus B$ and B / A, as implications:

- hypothetical reasoning
- type calculus as a substructural logic (no weakening and no contraction)
- Geach rules, type raising, ..., for free

Lambek's legacy

Functorial types, $A \setminus B$ and B / A, as implications:

- hypothetical reasoning
- type calculus as a substructural logic (no weakening and no contraction)
- Geach rules, type raising, ..., for free
- Syntax/semantics interface based on the Curry-Howard isomorphism

John	:	NP
loves	:	$(NP \setminus S)/NP$
Mary	:	NP
passionately	:	$S \setminus S$

John	:	NP
loves	:	$(NP \setminus S)/NP$
Mary	:	NP
passionately	:	$S \setminus S$

John loves : S/NP ???

John	:	NP
loves	:	$(NP \setminus S)/NP$
Mary	:	NP
passionately	:	$S \setminus S$

John loves : S/NP ???

We need: $(\mathrm{NP} \setminus \mathrm{S})/\mathrm{NP} \vdash \mathrm{NP} \setminus (\mathrm{S}/\mathrm{NP})$

John	:	NP
loves	:	$(NP \setminus S)/NP$
Mary	:	NP
passionately	:	$S \setminus S$

loves Mary passionately : $NP \setminus S$???

John	:	NP
loves	:	$(NP \setminus S)/NP$
Mary	:	NP
passionately	:	$S \setminus S$

loves Mary passionately : $NP \setminus S$???

We need: $\mathrm{NP} \setminus \mathrm{S}, \mathrm{S} \setminus \mathrm{S} \vdash \mathrm{NP} \setminus \mathrm{S}$

Add the rules:

$$\textsc{if}\quad \alpha, \Gamma \vdash \beta \quad \textsc{then}\quad \Gamma \vdash \alpha \backslash \beta$$

$$\textsc{if}\quad \Gamma, \alpha \vdash \beta \quad \textsc{then}\quad \Gamma \vdash \alpha / \beta$$

Curry's legacy

Curry's legacy

Distinction between two levels:

Curry's legacy

Distinction between two levels:

- **Tectogrammatics:** *"the study of* [abstract] *grammatical structure in itself"*

Curry's legacy

Distinction between two levels:

- **Tectogrammatics:** *"the study of* [abstract] *grammatical structure in itself"*
- **Phenogrammatics:** *"the study of the way* [abstract] *phrases are represented by expressions"*

Curry's legacy

Distinction between two levels:

- **Tectogrammatics:** *"the study of* [abstract] *grammatical structure in itself"*
- **Phenogrammatics:** *"the study of the way* [abstract] *phrases are represented by expressions"*

This allows/constraints the functorial types to be seen as **undirected** implications.

The logical framework approach

The logical framework approach

A logical framework does not consist of a single specific logic, but provides a means of defining various logics. It amounts to a general theory of logical systems that isolates the uniformities of a wide class of logics.

The logical framework approach

A logical framework does not consist of a single specific logic, but provides a means of defining various logics. It amounts to a general theory of logical systems that isolates the uniformities of a wide class of logics.

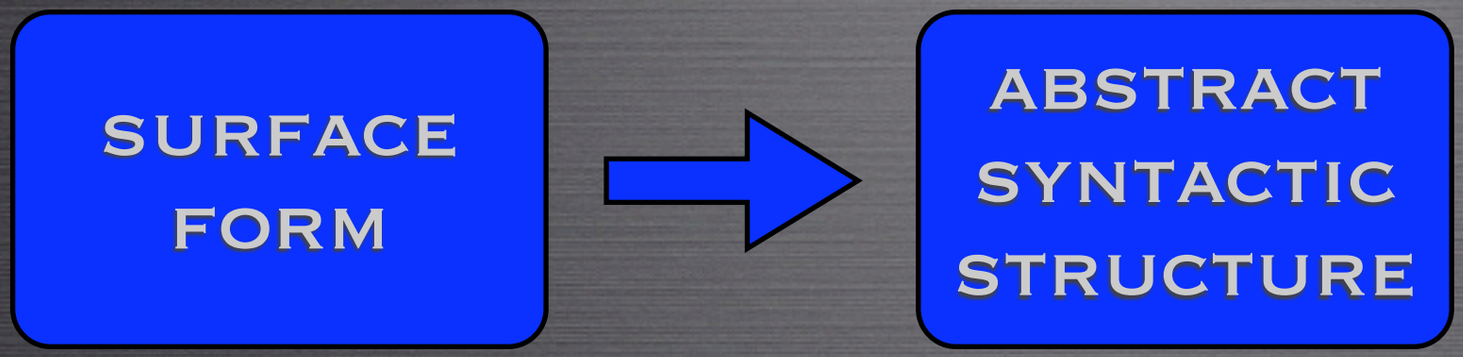
Similarly, a grammatical framework should provide a means of defining various grammatical formalisms.

Architecture of grammatical systems

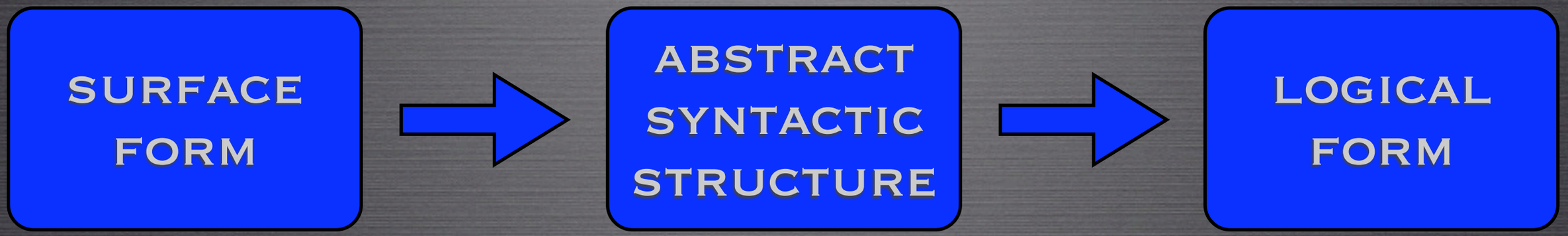
Architecture of grammatical systems

SURFACE FORM

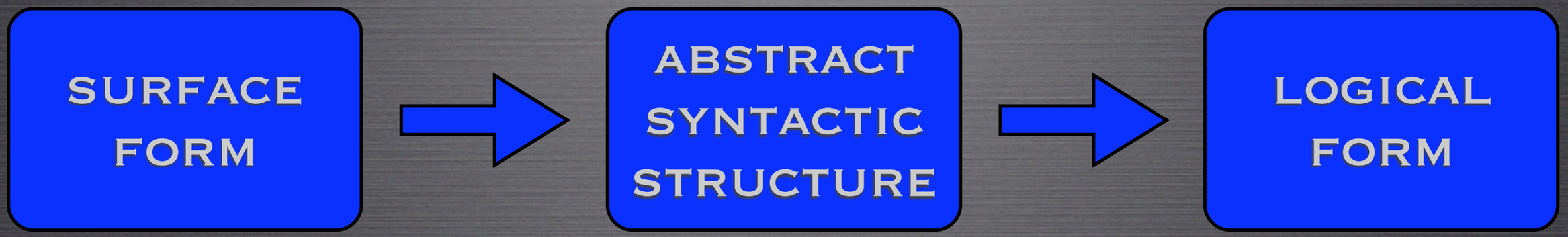
Architecture of grammatical systems



Architecture of grammatical systems

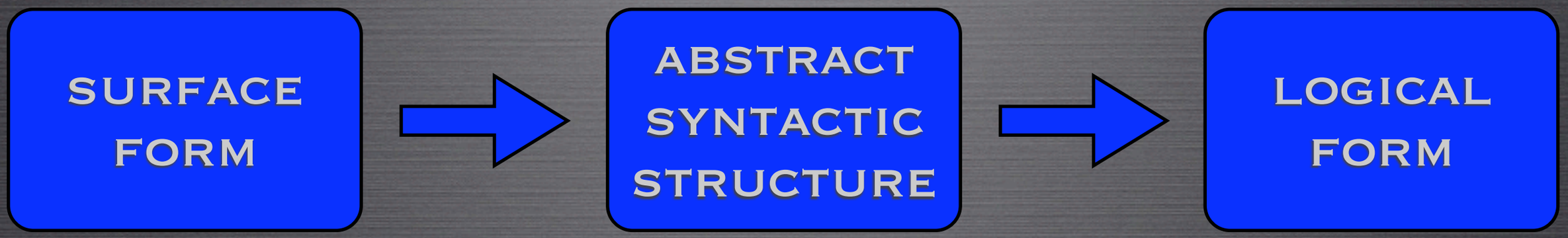


Architecture of grammatical systems



string

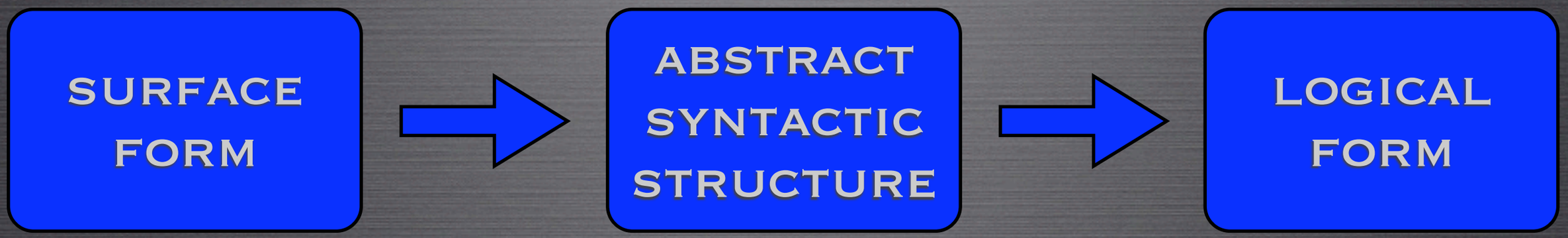
Architecture of grammatical systems



string

parse tree

Architecture of grammatical systems

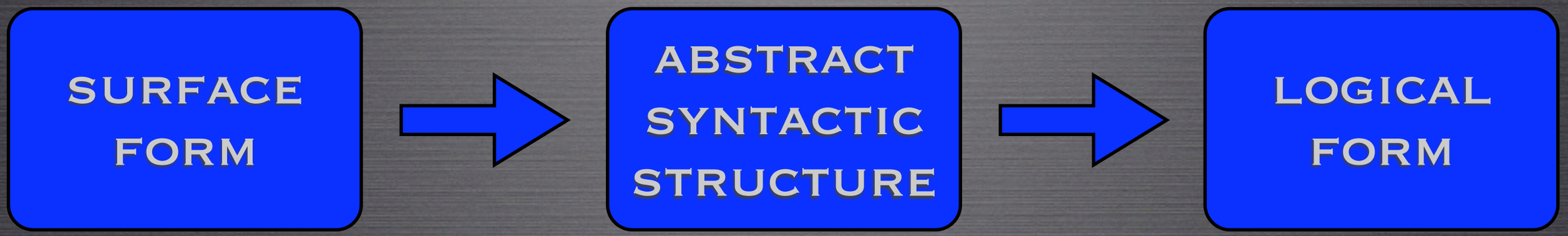


STRING

PARSE TREE

LOGICAL FORMULA

Architecture of grammatical systems



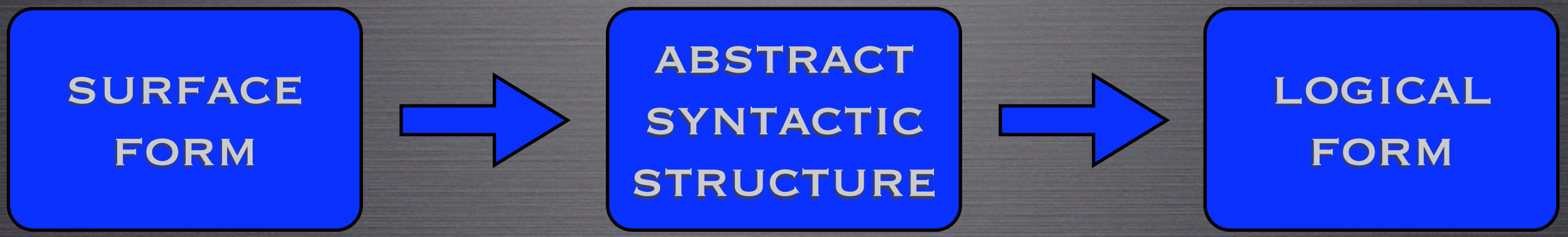
string

parse tree

logical formula

term from some prosodic algebra

Architecture of grammatical systems



string

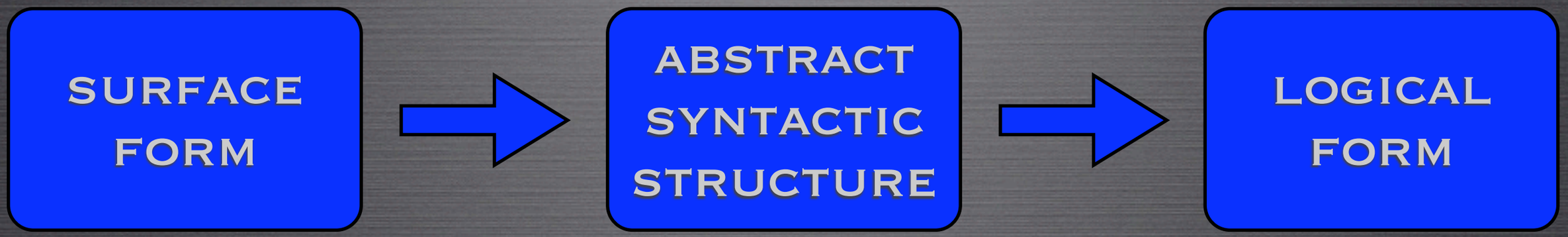
parse tree

logical formula

term from some prosodic algebra

formal proof (deduction tree)

Architecture of grammatical systems



string	parse tree	logical formula
term from some prosodic algebra	formal proof (deduction tree)	(first-order) (higher-order) (modal) (intensional)

Mathematical preliminaries

Simple types

Simple types

Let $\mathcal{A}$ be a (finite) set of atomic types

Simple types

Let $\mathcal{A}$ be a (finite) set of atomic types

The set $\mathcal{T}_{\mathcal{A}}$ of simple types built upon $\mathcal{A}$ is inductively defined as follows:

$$\mathcal{T}_{\mathcal{A}} ::= \mathcal{A} \mid (\mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{A}})$$

Higher-order signatures

Higher-order signatures

A higher-order signature is defined to be a triple:

$$\Sigma = \langle A, C, \tau \rangle$$

where:

Higher-order signatures

A higher-order signature is defined to be a triple:

$$\Sigma = \langle A, C, \tau \rangle$$

where:

- A is a finite set of atomic types

Higher-order signatures

A higher-order signature is defined to be a triple:

$$\Sigma = \langle A, C, \tau \rangle$$

where:

A is a finite set of atomic types

C is a finite set of constants

Higher-order signatures

A higher-order signature is defined to be a triple:

$$\Sigma = \langle A, C, \tau \rangle$$

where:

A is a finite set of atomic types

C is a finite set of constants

$\tau : C \to \mathscr{T}_A$ is a type assignment

Higher-order signatures

A higher-order signature is defined to be a triple:

$$\Sigma = \langle A, C, \tau \rangle$$

where:

A is a finite set of atomic types

C is a finite set of constants

$\tau : C \to \mathscr{T}_A$ is a type assignment

We write $\mathscr{T}_\Sigma$ for $\mathscr{T}_A$

Simply typed lambda-calculus

Simply typed lambda-calculus

Syntax:

Simply typed lambda-calculus

Syntax:

Let $\mathcal{X}$ be a infinite countable set of variables, and let $\Sigma = \langle A, C, \tau \rangle$ be a higher-order signature

Simply typed lambda-calculus

Syntax:

Let $\mathcal{X}$ be a infinite countable set of variables, and let $\Sigma = \langle A, C, \tau \rangle$ be a higher-order signature

The set of lambda-terms built upon Σ is inductively defined as follows:

$$T ::= x \mid c \mid (\lambda x.T) \mid (T\,T)$$

where $x \in \mathcal{X}$ and $c \in C$

Typing rules:

Typing rules:

$$\Gamma \vdash_\Sigma c : \tau_\Sigma(c) \quad \text{(cons)}$$

Typing rules:

$$\Gamma \vdash_\Sigma c : \tau_\Sigma(c) \quad \text{(cons)}$$

$$\Gamma, x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)}$$

Typing rules:

$$\Gamma \vdash_\Sigma c : \tau_\Sigma(c) \quad \text{(cons)}$$

$$\Gamma, x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.\, t) : (\alpha \to \beta)} \quad \text{(abs)}$$

Typing rules:

$$\Gamma \vdash_\Sigma c : \tau_\Sigma(c) \quad \text{(cons)}$$

$$\Gamma, x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.\, t) : (\alpha \to \beta)} \quad \text{(abs)}$$

$$\frac{\Gamma \vdash_\Sigma t : (\alpha \to \beta) \quad \Gamma \vdash_\Sigma u : \alpha}{\Gamma \vdash_\Sigma (t\, u) : \beta} \quad \text{(app)}$$

Properties:

Properties:

- Church-Rosser property

Properties:

- Church-Rosser property
- Subject reduction

Properties:

- Church-Rosser property
- Subject reduction
- Decidability of type inference

Properties:

- Church-Rosser property
- Subject reduction
- Decidability of type inference
- Principal type

Linear lambda-calculus

Linear lambda-calculus

A lambda-abstraction $\lambda x.t$ is linear if and only if there is exactly one free occurrence of x in t

Linear lambda-calculus

A lambda-abstraction $\lambda x.t$ is linear if and only if there is exactly one free occurrence of x in t

A (closed) lambda-term is linear if and only if all the lambda-abstractions it contains are linear

Typing rules:

Typing rules:

$$x : \alpha \vdash_{\Sigma} x : \alpha \quad (\text{var})$$

Typing rules:

$$x : \alpha \vdash_\Sigma x : \alpha \quad (\text{var})$$

$$\vdash_\Sigma c : \tau_\Sigma(c) \quad (\text{cons})$$

Typing rules:

$$x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)}$$

$$\vdash_\Sigma c : \tau_\Sigma(c) \quad \text{(cons)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.\, t) : (\alpha \to \beta)} \; x \notin \mathrm{dom}(\Gamma) \quad \text{(abs)}$$

Typing rules:

$$x : \alpha \vdash_\Sigma x : \alpha \quad (\text{var})$$

$$\vdash_\Sigma c : \tau_\Sigma(c) \quad (\text{cons})$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.\, t) : (\alpha \to \beta)} \; x \notin \mathrm{dom}(\Gamma) \quad (\text{abs})$$

$$\frac{\Gamma \vdash_\Sigma t : (\alpha \to \beta) \quad \Delta \vdash_\Sigma u : \alpha}{\Gamma, \Delta \vdash_\Sigma (t\, u) : \beta} \; \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \varnothing \quad (\text{app})$$

Properties:

Properties:

- Subject expansion

Properties:

- Subject expansion
- Every linear lambda-term is simply typable

Properties:

- Subject expansion
- Every linear lambda-term is simply typable
- The principal type of a linear lambda-term specifies uniquely this term (coherence)

Properties:

- Subject expansion
- Every linear lambda-term is simply typable
- The principal type of a linear lambda-term specifies uniquely this term (coherence)

We write Λ_Σ for the set of simply-typed linear lambda-terms built upon a signature Σ

Lexicons

Lexicons

Given two higher-order signatures:

$$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle \quad \text{and} \quad \Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$$

A lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a pair $\mathscr{L} = \langle \eta, \theta \rangle$ such that:

Lexicons

Given two higher-order signatures:

$$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle \quad \text{and} \quad \Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$$

A lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a pair $\mathscr{L} = \langle \eta, \theta \rangle$ such that:

$$\eta : A_1 \to \mathscr{T}_{A_2}$$

Lexicons

Given two higher-order signatures:

$$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle \quad \text{and} \quad \Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$$

A lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a pair $\mathscr{L} = \langle \eta, \theta \rangle$ such that:

$$\eta : A_1 \to \mathscr{T}_{A_2}$$

$$\theta : C_1 \to \Lambda_{\Sigma_2}$$

Lexicons

Given two higher-order signatures:

$$\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle \quad \text{and} \quad \Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$$

A lexicon $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a pair $\mathscr{L} = \langle \eta, \theta \rangle$ such that:

$$\eta : A_1 \to \mathscr{T}_{A_2}$$

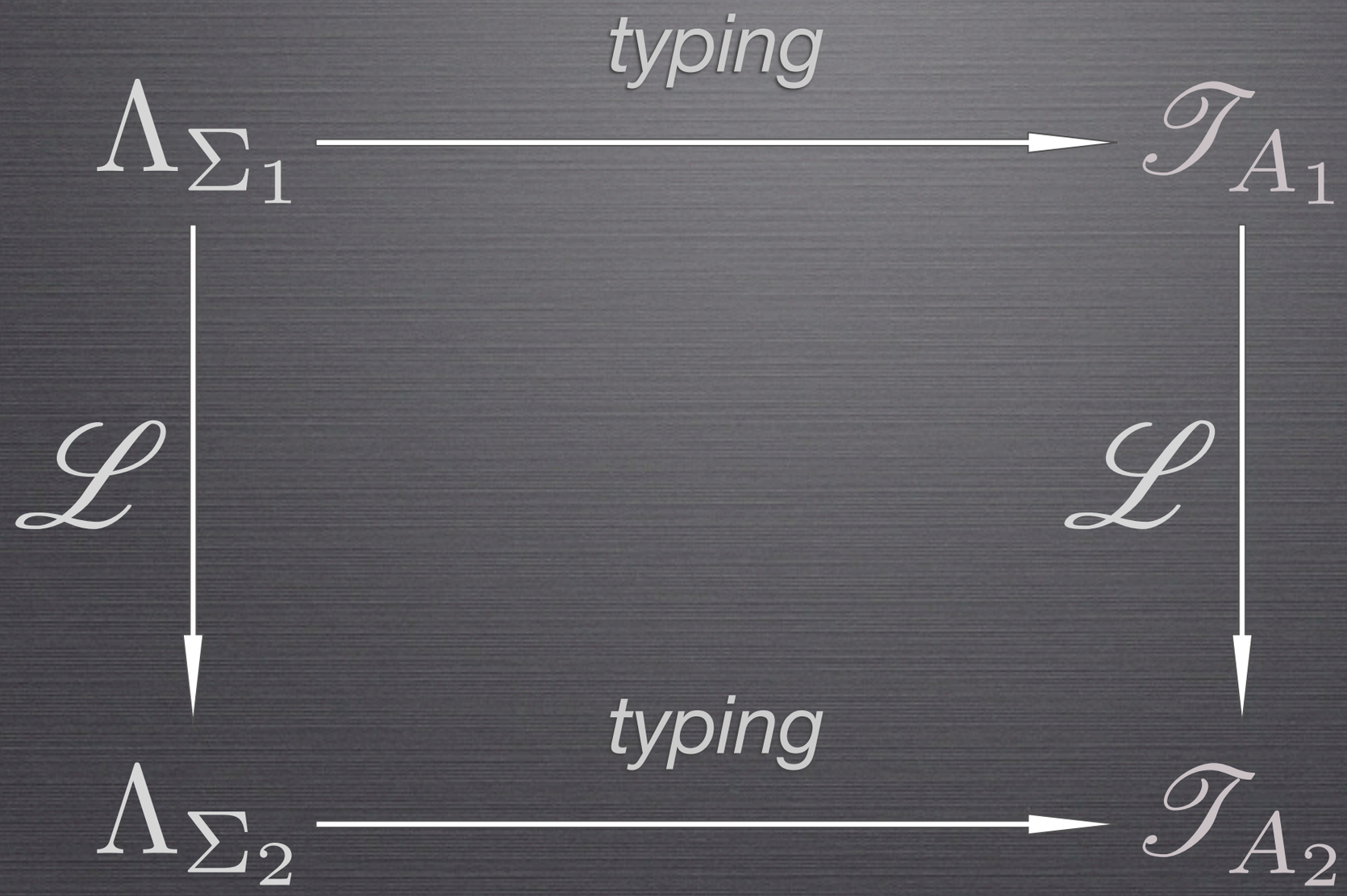
$$\theta : C_1 \to \Lambda_{\Sigma_2}$$

$$\vdash_{\Sigma_2} \theta(c) : \eta(\tau_1(c))$$


typing
Λ_{Σ_1}
$\mathscr{T}_{A_1}$


typing
Λ_{Σ_1}
$\mathscr{T}_{A_1}$


typing
Λ_{Σ_2}
$\mathscr{T}_{A_2}$


typing
Λ_{Σ_1}
$\mathscr{T}_{A_1}$
$\mathscr{L}$
$\mathscr{L}$
typing
Λ_{Σ_2}
$\mathscr{T}_{A_2}$