Recognizability in functional programs

Sylvain Salvati
University of Lille

Journées annuelles GT Vérif et GT SCALP du GDR IFM 2024

λ-calculus

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)
- Logic, proof assistants (CoQ, Agda,...)

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)
- Logic, proof assistants (CoQ, Agda,...)
- Computability theory

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)
- Logic, proof assistants (CoQ, Agda,...)
- Computability theory
- ...

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)
- Logic, proof assistants (CoQ, Agda,...)
- Computability theory
- ...

λ-calculus can be use to study formal language theory.

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)
- Logic, proof assistants (CoQ, Agda,...)
- Computability theory
- ...

λ-calculus can be use to study formal language theory.
I focus here on two main fields of application:

- mathematics of language,

Applications of λ-calculus

- Programming languages (Lisp, Ocaml, SML,Haskell, ...)
- Logic, proof assistants (CoQ, Agda,...)
- Computability theory
- ...

λ-calculus can be use to study formal language theory.
I focus here on two main fields of application:

- mathematics of language,
- program verification.

Simply typed λ-calculus

Types: $\mathcal{A}$ is a finite set of atomic types and $(A \to B)$ is a type when A and B are types.

$$\mathrm{order}(A) = 1,\ \mathrm{order}(A \to B) = \max(\mathrm{order}(A) + 1, \mathrm{order}(B))$$

Higher-order signature $\Sigma = \{a^A, b^B, \dots\}$ is a set of typed constant.

λ-calculus

$$\Lambda : \qquad M^A, N^B \ ::= \ x^A \mid c^A \mid (\lambda x^A . M^B)^{A \to B} \mid (M^{A \to B} N^A)^B$$

$$(\beta) \qquad (\lambda x.M)N = M[N/x]$$

$$(\eta) \qquad \lambda x.Mx = M \text{ when } x \notin fv(M)$$

Simply typed λ-calculus

Types: $\mathcal{A}$ is a finite set of atomic types and $(A \to B)$ is a type when A and B are types.

$\text{order}(A) = 1$, $\text{order}(A \to B) = \max(\text{order}(A) + 1, \text{order}(B))$

Higher-order signature $\Sigma = \{a^A, b^B, \dots\}$ is a set of typed constant.

λ-calculus

$$\Lambda : \qquad M^A, N^B ::= x^A \mid c^A \mid (\lambda x^A.M^B)^{A \to B} \mid (M^{A \to B} N^A)^B$$

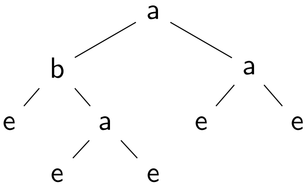
$$(\beta) \qquad (\lambda x.M)N = M[N/x]$$

$$(\eta) \qquad \lambda x.Mx = M \text{ when } x \notin fv(M)$$

$$(\delta) \qquad YM = M(YM)$$

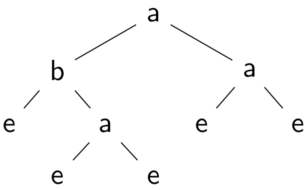
Syntax basics

Trees



Syntax basics

Trees

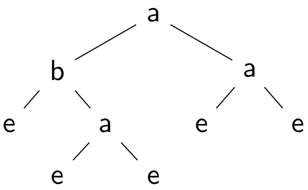


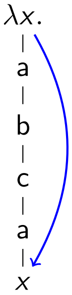
Strings

Syntax basics

Strings

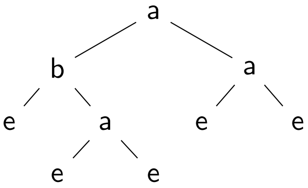
Trees



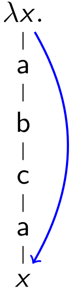


Syntax basics

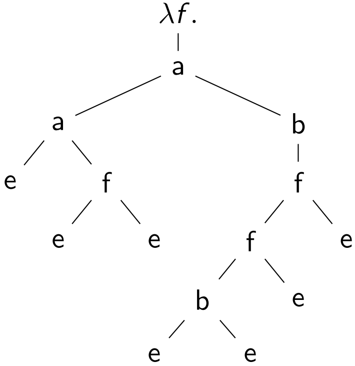
Trees



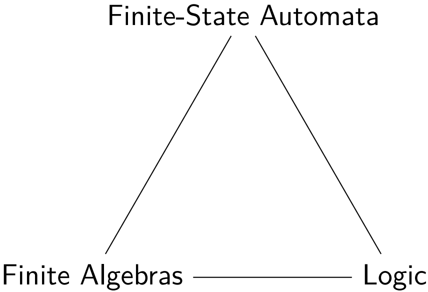
Strings



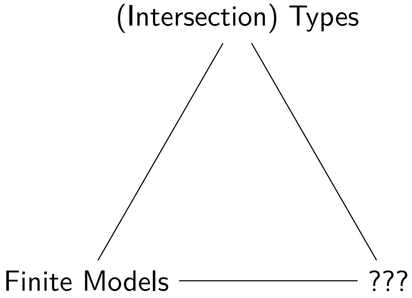
Complex operations



Recognizability

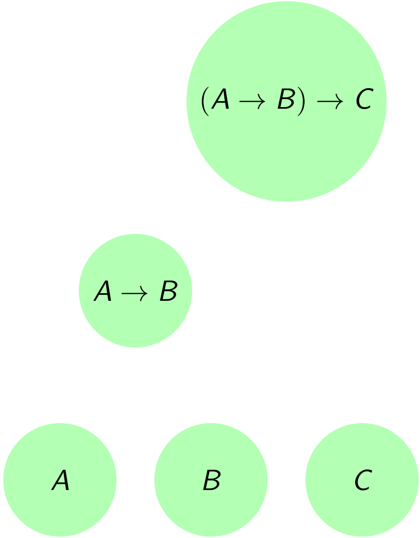


Recognizability in the simply typed λ-calculus [S.09]



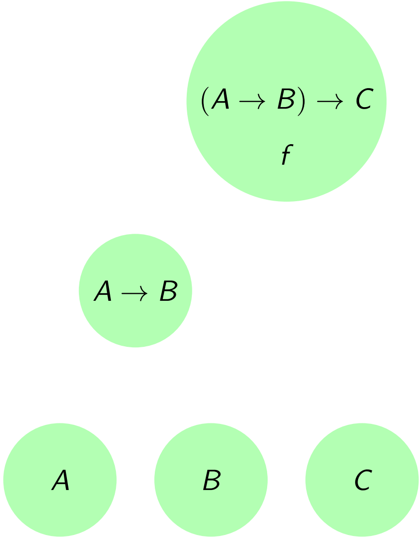
Finite models

$[\![M, \nu]\!]$



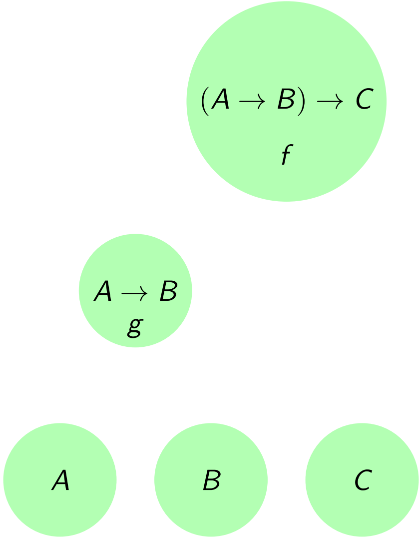
Finite models

$[\![M, \nu]\!]$



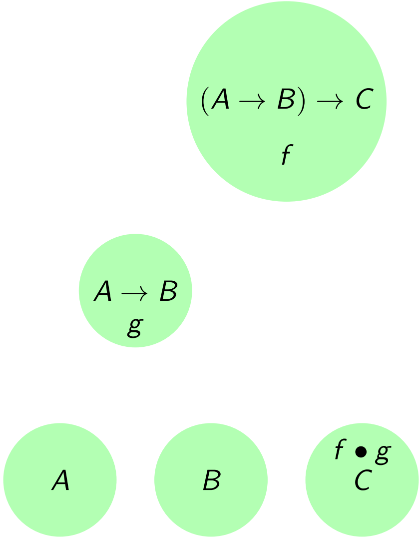
Finite models

$[\![M, \nu]\!]$



Finite models

$[\![M, \nu]\!]$

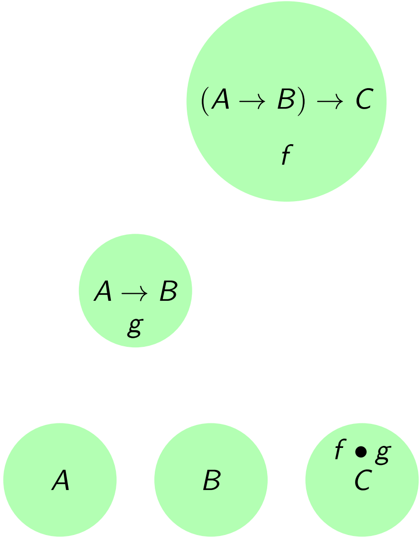


Finite models

$[\![M, \nu]\!]$

Axioms

$[\![MN, \nu]\!] = [\![M, \nu]\!] \bullet [\![N, \nu]\!]$

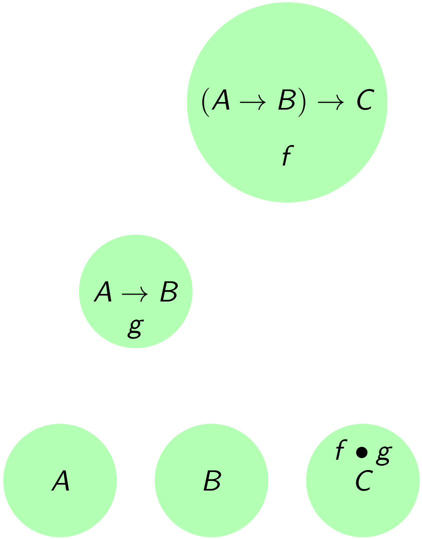


Finite models

$[\![M, \nu]\!]$

Axioms

$[\![MN, \nu]\!] = [\![M, \nu]\!] \bullet [\![N, \nu]\!]$
$[\![\lambda x.M, \nu]\!] \bullet f = [\![M, \nu[f/x]]\!]$



Finite models

$[\![M, \nu]\!]$

Axioms

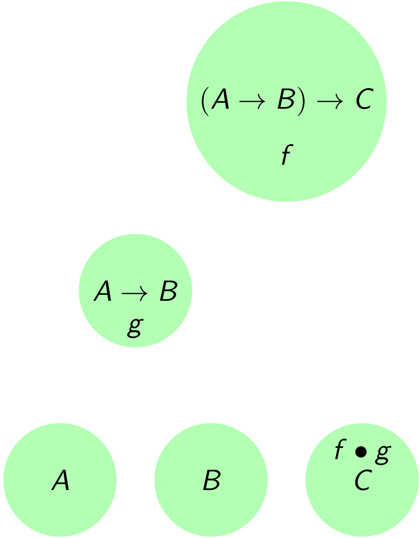
$[\![MN, \nu]\!] = [\![M, \nu]\!] \bullet [\![N, \nu]\!]$
$[\![\lambda x.M, \nu]\!] \bullet f = [\![M, \nu[f/x]]\!]$

Lemma (Correctness)

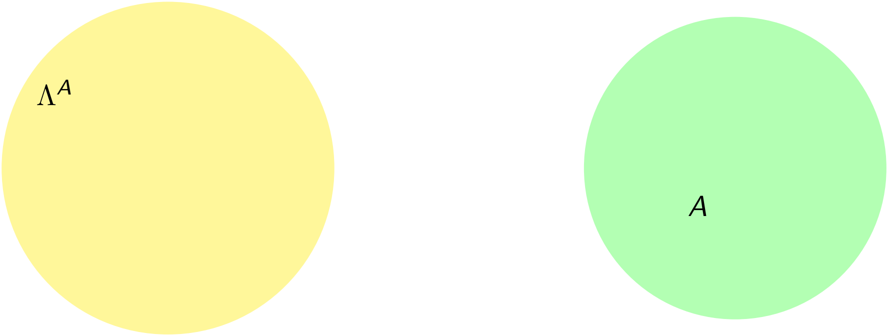
If $M =_\beta N$, then for every ν, $[\![M, \nu]\!] = [\![N, \nu]\!]$.

A wealth of possibilities

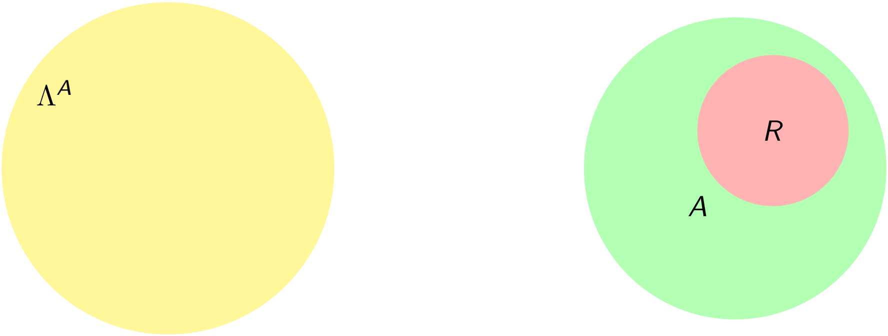
standard, monotonous, stable, strongly stable ...models, bi-domains etc.



Recognizability in the simply typed λ-calculus



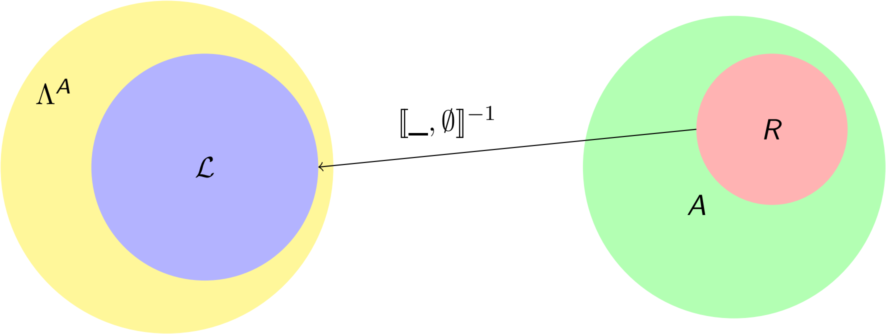
Recognizability in the simply typed λ-calculus



Recognizability in the simply typed λ-calculus

$\mathcal{L}$ is recognizable iff:

$$\mathcal{L} = \{M \mid [\![M, \emptyset]\!] \in R\}$$



Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,
- closed under inverse higher-order homomorphism,

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,
- closed under inverse higher-order homomorphism,
- not closed under relabeling.

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,
- closed under inverse higher-order homomorphism,
- not closed under relabeling.
- Singleton languages are recognizable [Statman 82]

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,
- closed under inverse higher-order homomorphism,
- not closed under relabeling.
- Singleton languages are recognizable [Statman 82]
- Emptiness is undecidable [Loader 01]

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,
- closed under inverse higher-order homomorphism,
- not closed under relabeling.

- Singleton languages are recognizable [Statman 82]
- Emptiness is undecidable [Loader 01]
- Membership is non-elementary (with natural representations of the recognizing set).

Basic properties

Recognizable languages of λ-terms are:

- conservative extensions of recognizable languages of strings and trees,
- closed under boolean operations,
- closed under inverse higher-order homomorphism,
- not closed under relabeling.

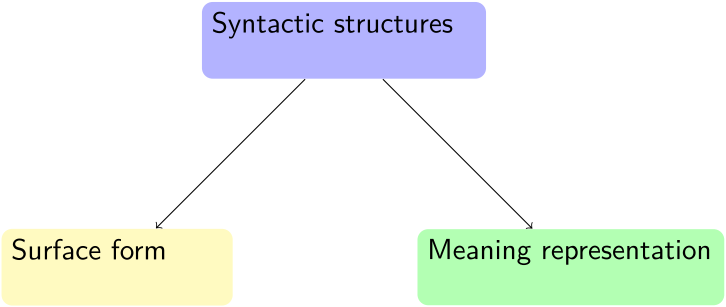
- Singleton languages are recognizable [Statman 82]
- Emptiness is undecidable [Loader 01]
- Membership is non-elementary (with natural representations of the recognizing set).

First application

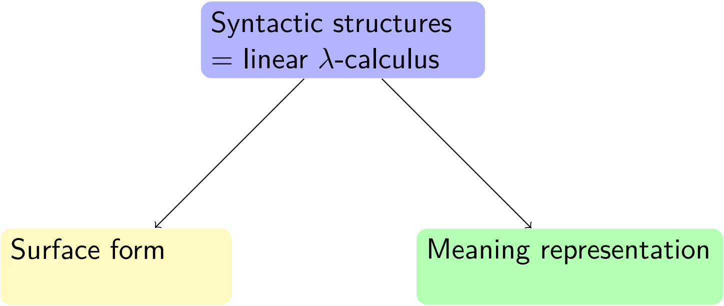
Simple proof of decidability of 4^{th} order matching.

Finiteness: parsing algorithms

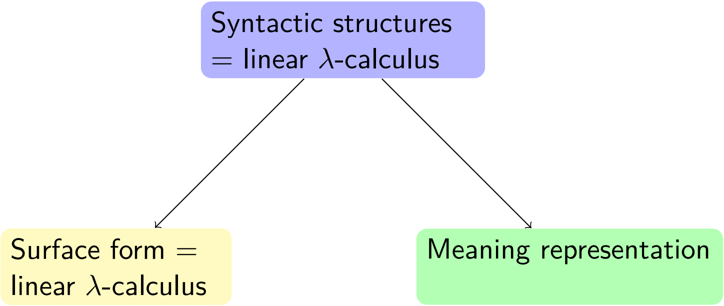
Abstract Categorial Grammars [de Groote 01, Muskens 01]



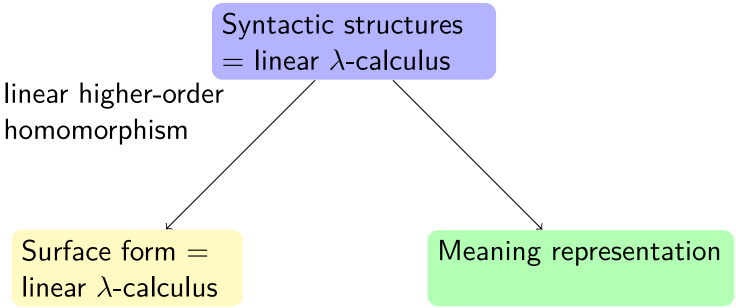
Abstract Categorial Grammars [de Groote 01, Muskens 01]



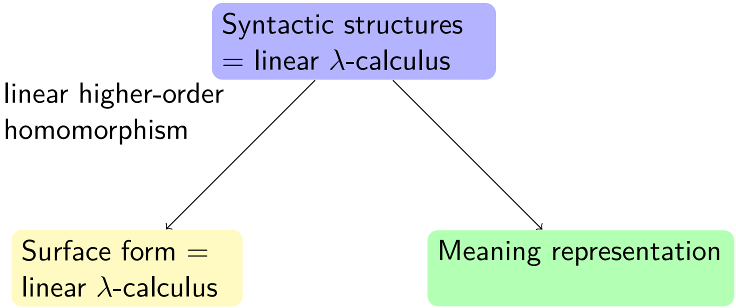
Abstract Categorial Grammars [de Groote 01, Muskens 01]



Abstract Categorial Grammars [de Groote 01, Muskens 01]

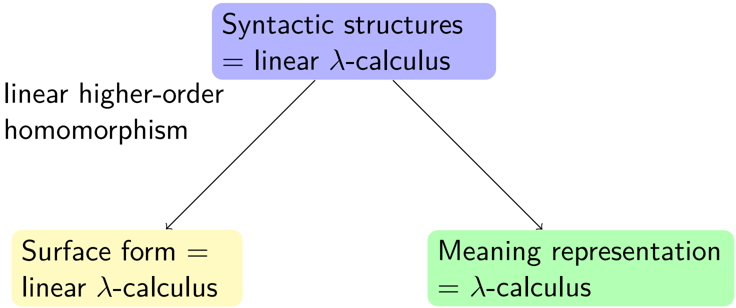


Abstract Categorial Grammars [de Groote 01, Muskens 01]



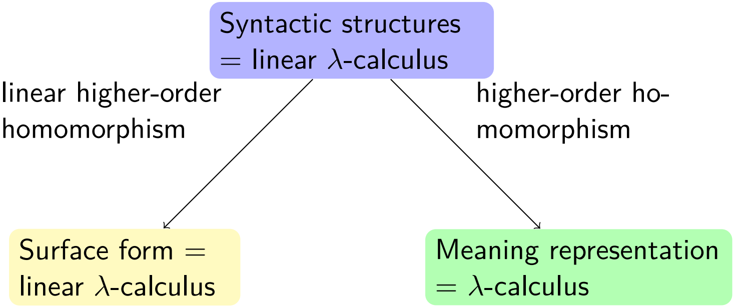
Generalizes many notions of grammars.

Abstract Categorial Grammars [de Groote 01, Muskens 01]



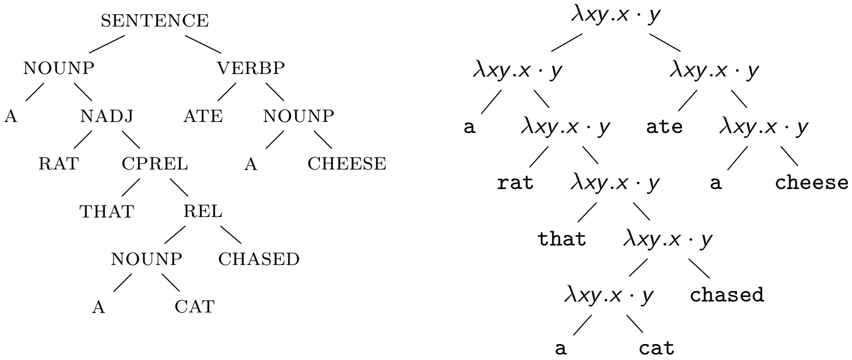
Generalizes many notions of grammars.

Abstract Categorial Grammars [de Groote 01, Muskens 01]



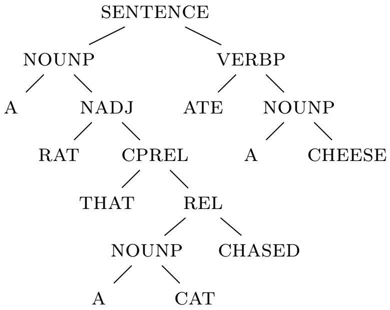
Generalizes many notions of grammars.

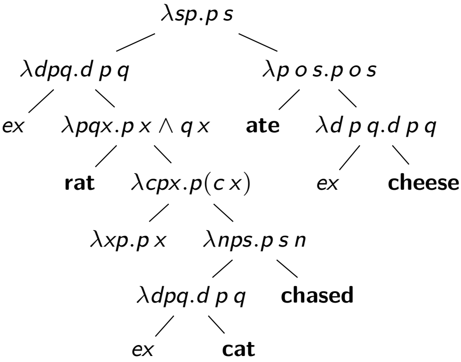
Example: Surface Realization



`a rat that a cat saw ate a cheese`

Example: Montague Semantics

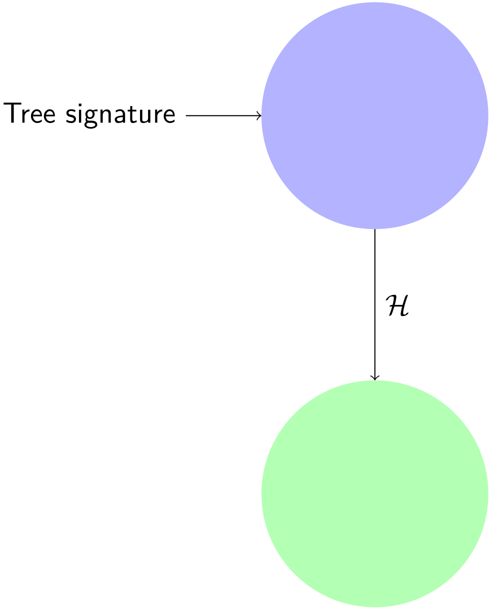




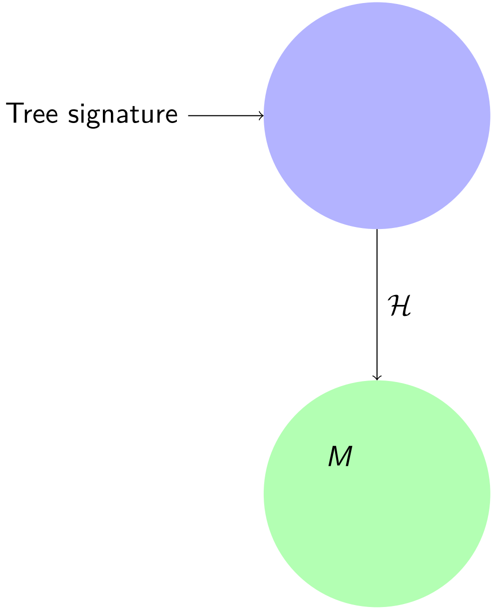
where $ex = \lambda pq.\exists x.p\,x \wedge q\,x$.

$$\exists x.\mathbf{rat}\,x \wedge (\exists y.\mathbf{cat}\,y \wedge \mathbf{chased}\,y\,x \wedge (\exists u.\mathbf{cheese}\,u \wedge \mathbf{ate}\,x\,u))$$

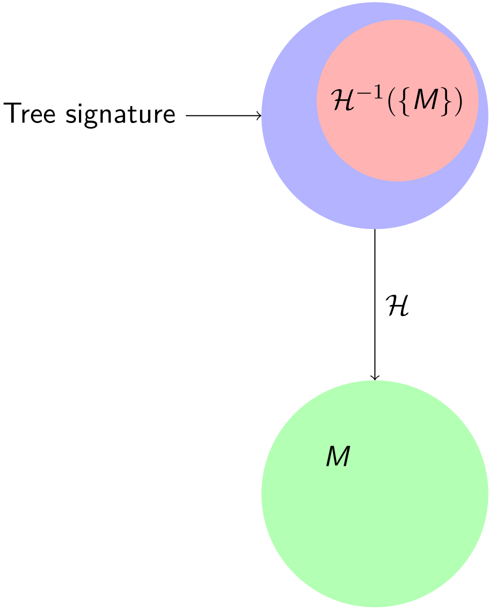
Decidability of parsing and text generation



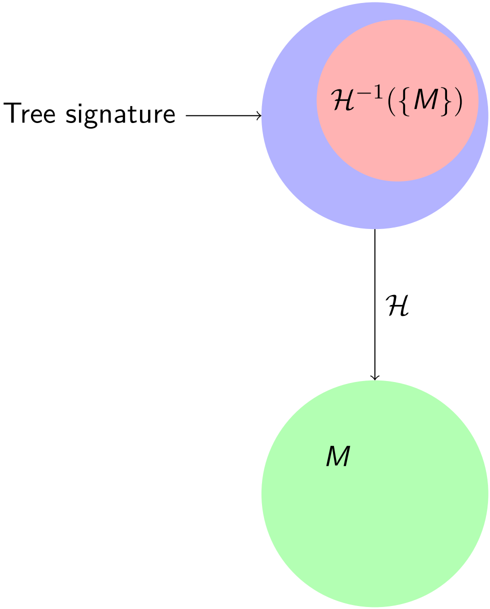
Decidability of parsing and text generation



Decidability of parsing and text generation

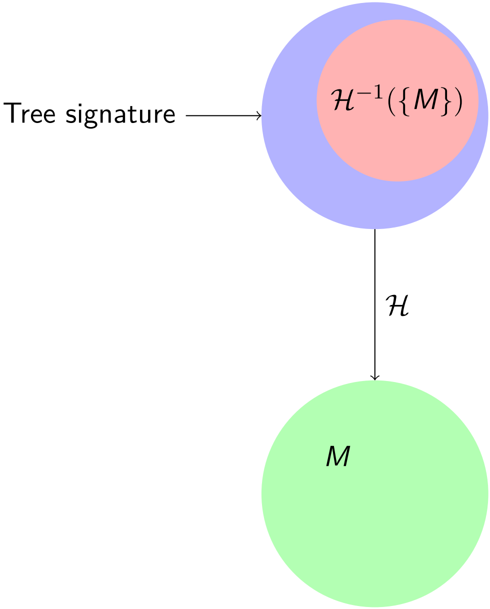


Decidability of parsing and text generation



- The complexity is non-elementary (but polynomial in the linear case)

Decidability of parsing and text generation



- The complexity is non-elementary (but polynomial in the linear case)
- A semantic argument extends the decidability result to higher-order OI grammars *[Kobele, S. 15]*

Efficient algorithms

- The construction yields efficient algorithms for particular cases:

Efficient algorithms

- The construction yields efficient algorithms for particular cases:
  - linear case [S. 05],

Efficient algorithms

- The construction yields efficient algorithms for particular cases:
  - linear case [S. 05],
  - almost linear case [Kanazawa 07],

Efficient algorithms

- The construction yields efficient algorithms for particular cases:
  - linear case [S. 05],
  - almost linear case [Kanazawa 07],
  - almost affine case [Bourreau, S. 11]

Efficient algorithms

- The construction yields efficient algorithms for particular cases:
  - linear case [S. 05],
  - almost linear case [Kanazawa 07],
  - almost affine case [Bourreau, S. 11]
- Datalog [Kanazawa 07] [Bourreau, S. 11][Ball *et al.* 14] and program transformation give efficient parsers.

Efficient algorithms

- The construction yields efficient algorithms for particular cases:
  - linear case [S. 05],
  - almost linear case [Kanazawa 07],
  - almost affine case [Bourreau, S. 11]
- Datalog [Kanazawa 07] [Bourreau, S. 11][Ball *et al.* 14] and program transformation give efficient parsers.

Parsing is mostly about:

- Evaluating fixpoint with non-determinism,
- Memoizing computation.

Efficient algorithms

- The construction yields efficient algorithms for particular cases:
  - linear case [S. 05],
  - almost linear case [Kanazawa 07],
  - almost affine case [Bourreau, S. 11]
- Datalog [Kanazawa 07] [Bourreau, S. 11][Ball *et al.* 14] and program transformation give efficient parsers.

Parsing is mostly about:

- Evaluating fixpoint with non-determinism,
- Memoizing computation.

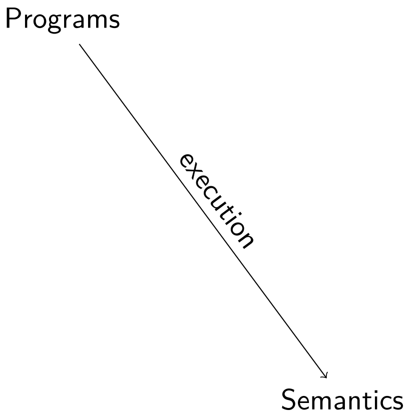
With intentional models of λY-calculus, part of the properties can be used in parsing can be internalized in the semantics.

Infiniteness: Program Verification

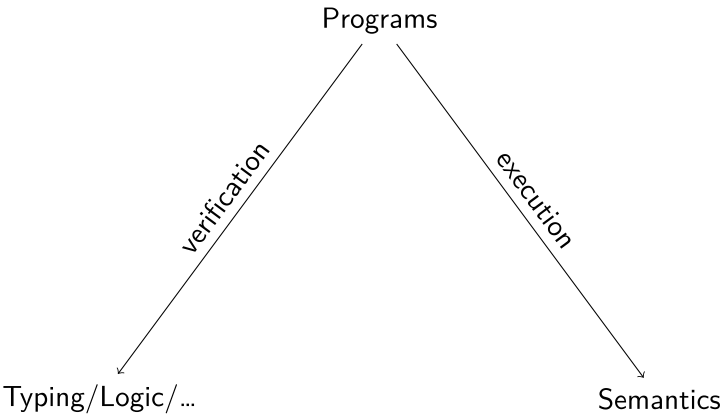
Schematology

Programs

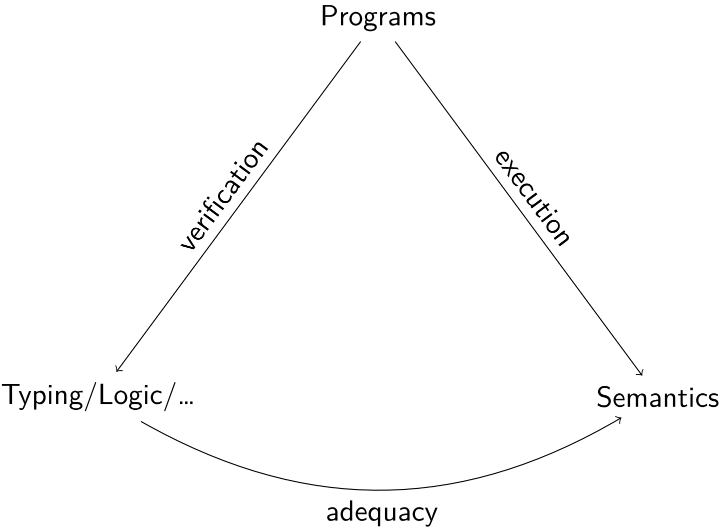
Schematology



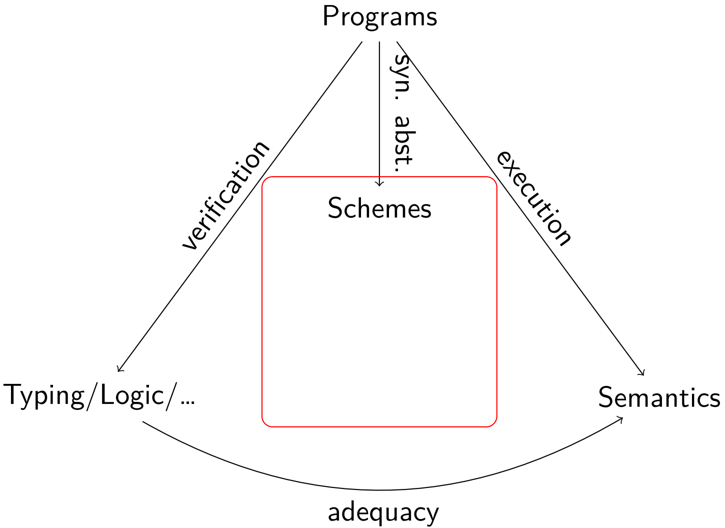
Schematology



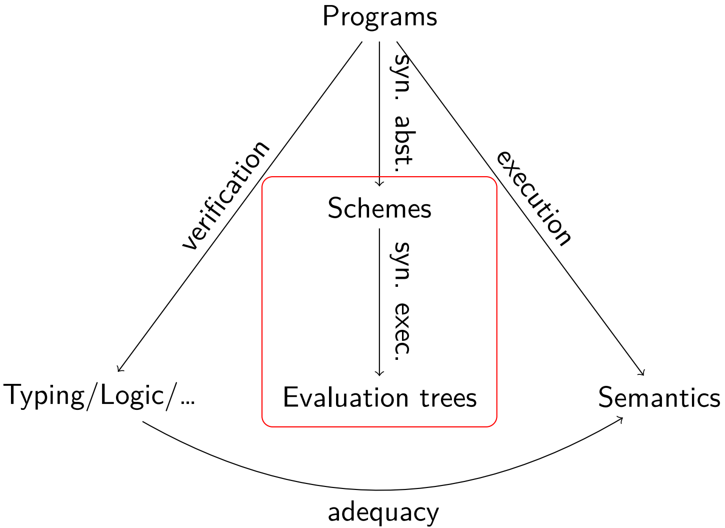
Schematology



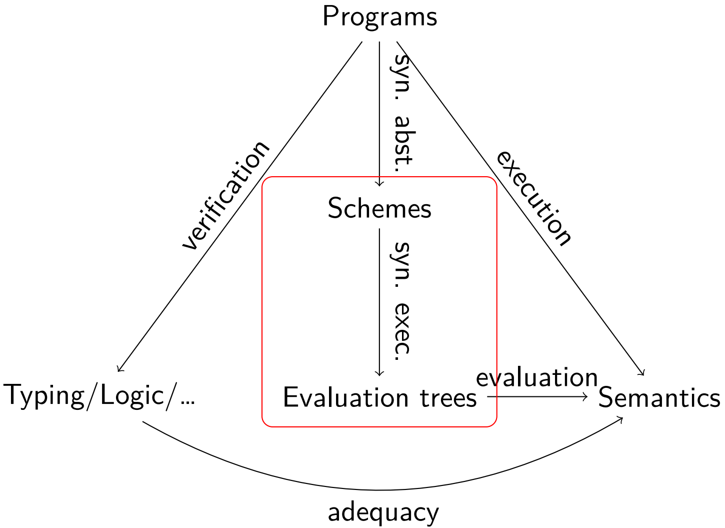
Schematology



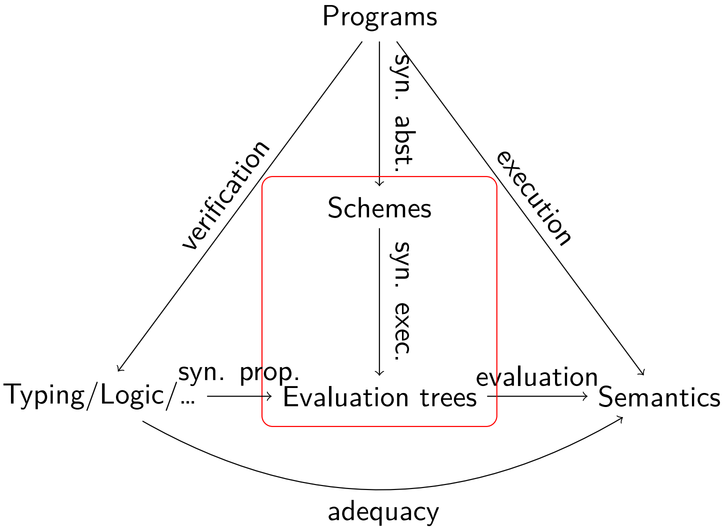
Schematology



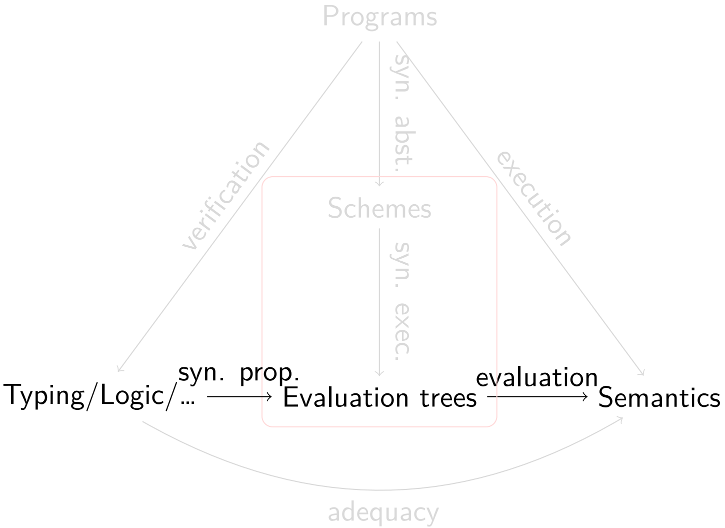
Schematology



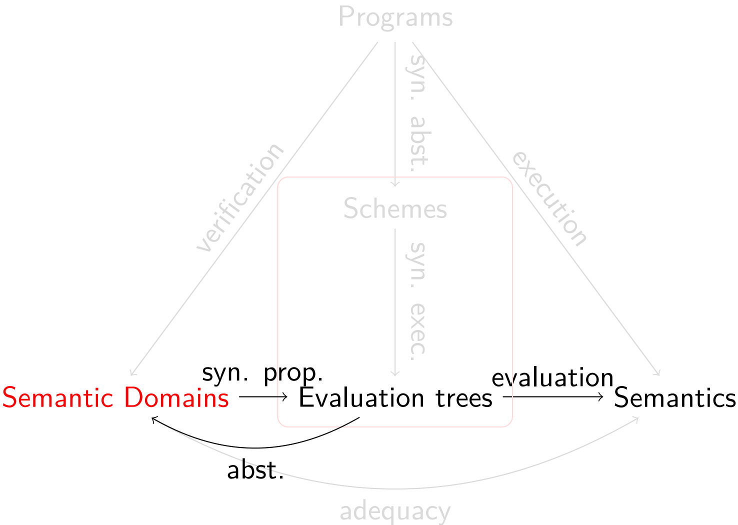
Schematology



Schematology



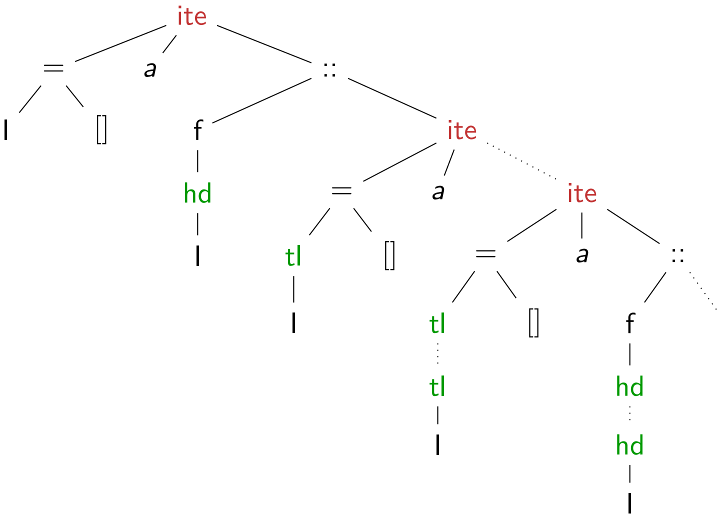
Schematology



Higher-order control flow

$$\text{fold f a l} = \text{if l=[] then a else f (hd l) (fold f a (tl l))}$$

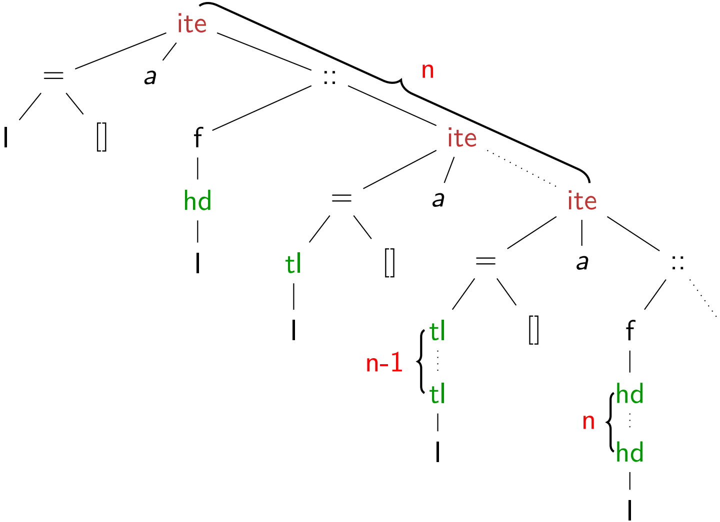
$$M = Y\lambda\text{fold f a l.ite } (=\text{l } []) \text{ a } (\text{f } (\text{hd l}) (\text{fold f a } (\text{tl l})))$$



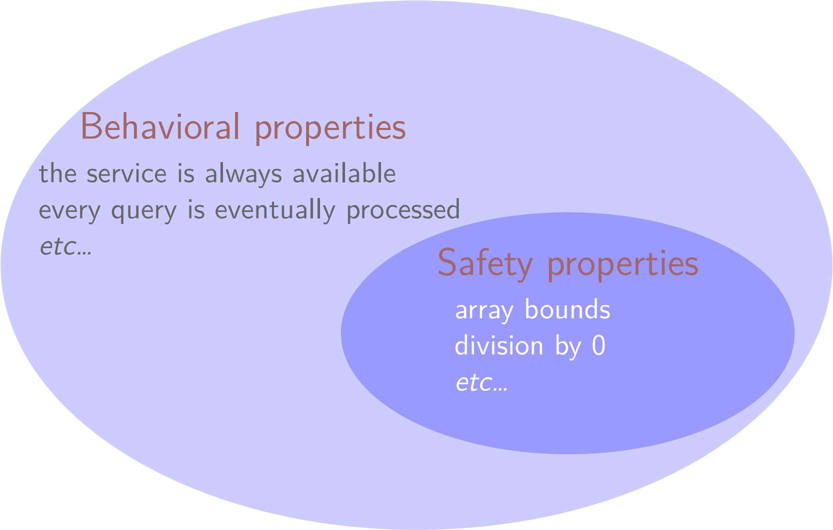
Higher-order control flow

fold f a l = if l=[] then a else f (hd l) (fold f a (tl l))

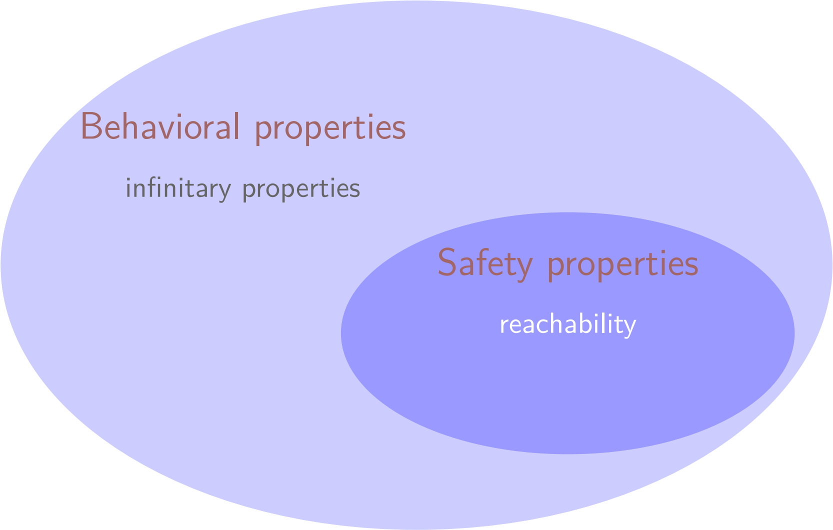
$M = Y\lambda$fold f a l.ite (=l []) a (f (hd l) (fold f a (tl l)))



Two kinds of properties



Two kinds of properties

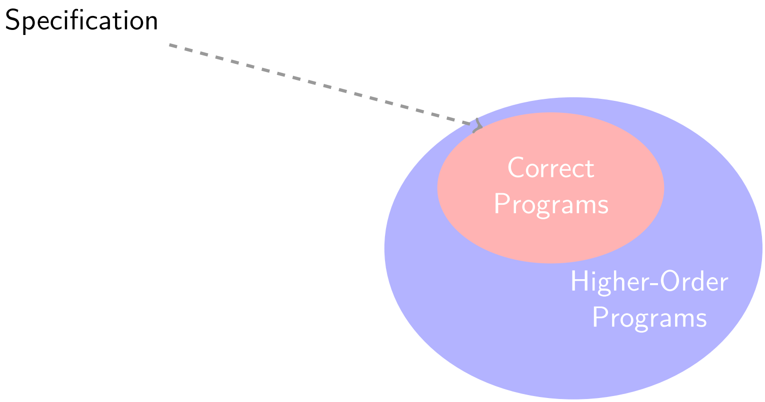


Finite abstractions

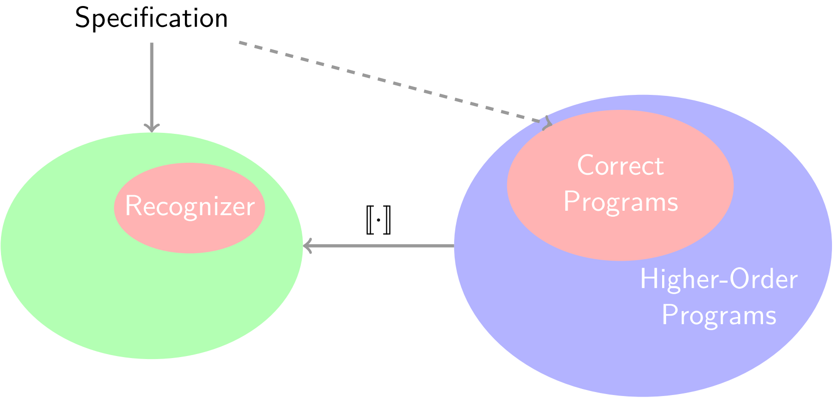
Reachability ⟶ Finite state automata

Infinitary properties ⟶ Parity automata
Monadic Second
Order Logic

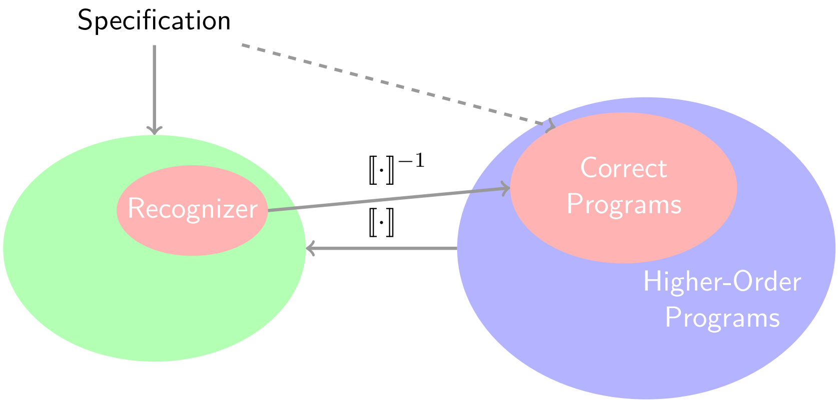
Programs and recognizability



Programs and recognizability



Programs and recognizability



Motivations and results

- Relating finite state methods with denotational methods
- Reveal the invariants behind behavioral properties
- Obtain decidability results by finiteness properties

Some results:

- New proof of Ong's Theorem with Krivine Machine and semantics [Walukiewicz S. 11]
- Limitation of Scott models [Walukiewicz S. 13]
- Transfer Theorem for term evaluation [Walukiewicz S. 13]
- Finite models for weak MSOL based on wreath products of models [Walukiewicz S. 15]
- Finite models for MSOL [Walukiewicz S. 15]

Example: unfolding

$$\text{Graph} \xrightarrow{\textit{unfold}} \text{Tree}$$

MSOL-compatibility of unfolding

For all Σ.
For all φ there is $\hat{\varphi}$ s.t. for all $G \in Graph(\Sigma)$:

$$G \models \hat{\varphi} \text{ iff } \mathit{Unf}(G) \models \varphi$$

Remark: this theorem implies Rabin's Theorem.

Example: unfolding

$$\text{Graph} \xrightarrow{\textit{unfold}} \text{Tree}$$

MSOL-compatibility of unfolding

For all Σ.
For all φ there is $\hat{\varphi}$ s.t. for all $G \in Graph(\Sigma)$:

$$G \models \hat{\varphi} \text{ iff } Unf(G) \models \varphi$$

Remark: this theorem implies Rabin's Theorem.
Other example: Muchnik iteration.

$$M \xrightarrow{eval} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.
For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

$$M \xrightarrow{eval} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.
For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature

$$M \xrightarrow{eval} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.

For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature
- $\mathcal{T}$ is a finite set of types

$$M \xrightarrow{eval} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.
For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature
- $\mathcal{T}$ is a finite set of types
- $\mathcal{X}$ is a finite set of λ-variables

$$M \xrightarrow{eval} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.

For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature
- $\mathcal{T}$ is a finite set of types
- $\mathcal{X}$ is a finite set of λ-variables
- $Terms(\Sigma, \mathcal{T}, \mathcal{X})$: terms of type 0 over Σ with

$$M \xrightarrow{\textit{eval}} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.
For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature
- $\mathcal{T}$ is a finite set of types
- $\mathcal{X}$ is a finite set of λ-variables
- $Terms(\Sigma, \mathcal{T}, \mathcal{X})$: terms of type 0 over Σ with
    - all subterms having type in $\mathcal{T}$

$$M \xrightarrow{\textit{eval}} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.

For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature
- $\mathcal{T}$ is a finite set of types
- $\mathcal{X}$ is a finite set of λ-variables
- $Terms(\Sigma, \mathcal{T}, \mathcal{X})$: terms of type 0 over Σ with
  - all subterms having type in $\mathcal{T}$
  - all λ-variables from $\mathcal{X}$

$$M \xrightarrow{eval} BT(M)$$

Transfer Theorem

For all Σ, $\mathcal{T}$, $\mathcal{X}$.
For all φ there is $\hat{\varphi}$ s.t. for all $M \in Terms(\Sigma, \mathcal{T}, \mathcal{X})$:

$$M \models \hat{\varphi} \text{ iff } BT(M) \models \varphi$$

- Σ is a tree signature
- $\mathcal{T}$ is a finite set of types
- $\mathcal{X}$ is a finite set of λ-variables
- $Terms(\Sigma, \mathcal{T}, \mathcal{X})$: terms of type 0 over Σ with
  - all subterms having type in $\mathcal{T}$
  - all λ-variables from $\mathcal{X}$

Note: no limitation on Y-variables.

Transducers

Deforestation

```
q = sum(filter p (map f l))
```

```
def query(l):
    res = 0
    for e in l:
        if p(f e):
            res += f e
    return res

q = query(l)
```

Higher-order transducers

```
sum :: [a] -> Int
sum [] = 0
sum (n:l) = n + sum l

filter :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (a:as) | p a       = a : filter p as
                | otherwise = filter p as

map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (a:as) = f a : map f as
```

The composed transducer

```
q = sum(filter p (map f l))
query [] = 0
query (a:as) | p(f a)     = f a + query as
             | otherwise = query as
```

Closure of higher-order transducers under composition

- Higher-order transducers are closed under composition, provided they can inspect the input with regular properties: this implements *deforestation* in a very general setting.
- Finite models and recognizability give direct constructions of the compositions of transducers (Gallot, Lemay, S.).
- Several properties need to be investigated:
  - How complex are the regular properties to be checked on input?
  - When is the size of the composition reasonable?

We have Encouraging results for the transducers equivalent to MSOT (Gallot, Lemay, S. 20)

Perspectives

Evaluation of terms in finite models

Parsing HO Grammars
and HO verification

evaluation of higher-order
programs in finite models

Evaluation of terms in finite models

Parsing HO Grammars and HO verification

evaluation of higher-order programs in finite models

Parsing in the almost affine case

fixpoint computation strategy via datalog program transformation

Evaluation of terms in finite models

Parsing HO Grammars and HO verification	Parsing in the almost affine case
evaluation of higher-order programs in finite models	fixpoint computation strategy via datalog program transformation
Abstract Interpretation - abstraction refinements - fixpoint acceleration techniques	

Evaluation of terms in finite models

Parsing HO Grammars and HO verification

evaluation of higher-order programs in finite models

Parsing in the almost affine case

fixpoint computation strategy via datalog program transformation

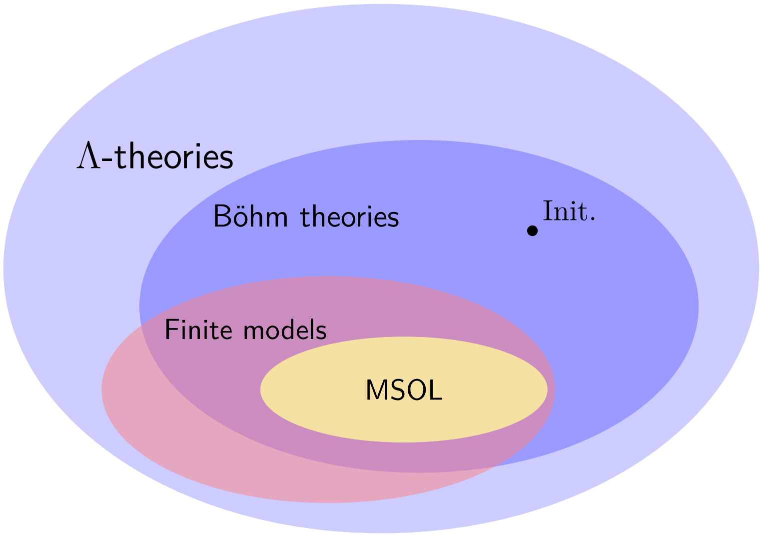
Abstract Interpretation

- abstraction refinements
- fixpoint acceleration techniques

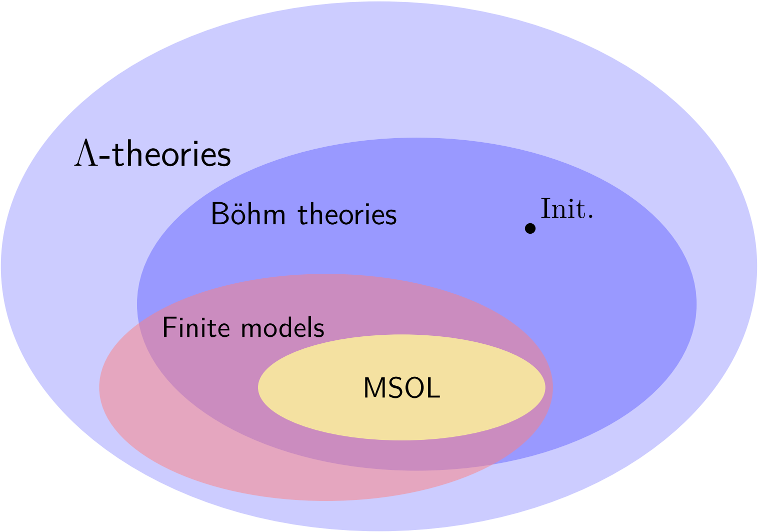
Structure of models

- sequential algorithms, etc...
- linear logic

Theory of Böhm trees

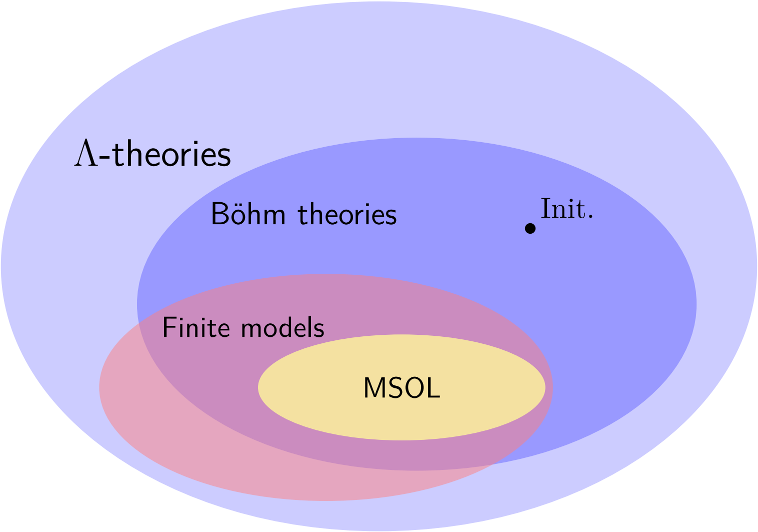


Theory of Böhm trees



- expressiveness of finite Böhm models?

Theory of Böhm trees



- expressiveness of finite Böhm models?
- axiomatization of finite Böhm models?