Program Schemes, Trees and Classes of Interpretations

Irène Guessarian

Workshop for Bruno Courcelle

OUTLINE

Outline

Ianov Flow Diagrams

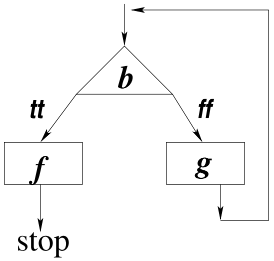


Figure: A Flow Diagram

Decisive step: Scott (1971) The Lattice of Flow Diagrams

Algebraic meaning for iterative program schemes à la Ianov

Semantics

Infinite Tree

Semantics

- complete lattice of (finite and infinite) flow diagrams
- meaning of a program (with iterations): a **least fixed point**

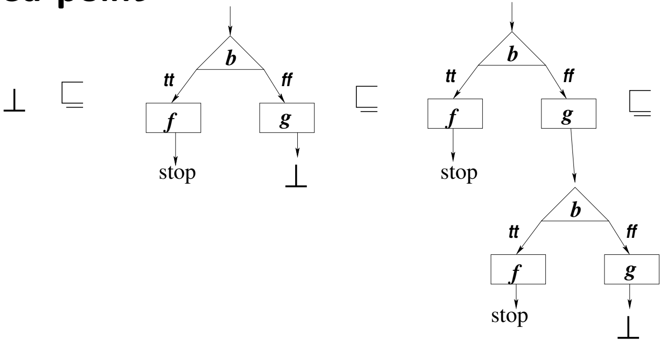


Figure: Trees unfolding a flow diagram

Fixed point theorems

Theorem (Knaster, Tarski, Kantorovich, . . . ?)

A monotone function $F\colon D \longrightarrow D$ on a complete lattice D has a least fixed point $\mu x.f(x) = \inf\{x | f(x) \leq x\}$.

If f preserves sups, i.e. $f(\sup\{x_n\}) = \sup\{f(x_n)\}$, then:

$$\mu x.f(x) = \sup\{f^n(\perp) | n \in \mathbb{N}\} = f^\omega(\perp)$$

Initial algebra (free algebra, Herbrand model)

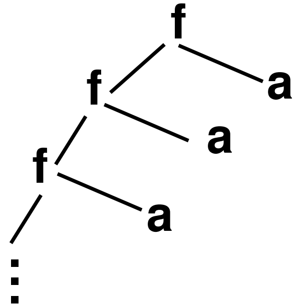
$$\begin{tikzcd}
\textbf{PROGRAM} \arrow[r, "abstract"] & \textbf{PROGRAM SCHEME} \arrow[d, "\text{solve in free algebra}"] \\
\textbf{function computed by PROGRAM} & \textbf{Infinite tree} \arrow[l, "morphism"']
\end{tikzcd}$$

Semantics

The initial algebra semantics of a deterministic program scheme is an infinite tree which is the least fixed point of the system of equations associated with the program scheme. It can be considered as the value of the program in the “Herbrand” domain.
Herbrand domain: trees, ordering $\sqsubseteq$ inductively generated by: $\forall t \quad \perp \sqsubseteq t$

Iterative program scheme

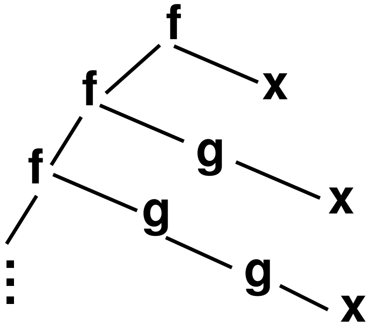
X=f(X,a)



Regular infinite tree

Recursive program scheme

$$F(x)=f(F(g(x)),x)$$



Algebraic infinite tree

Infinite Trees

Infinite trees were studied in the seminal paper
Fundamental Properties of Infinite Trees
Courcelle – TCS 1983
and have been extensively used in many areas of computer science - both theoretical and more applied (over 500 citations).

Outline

Regular trees

An infinite tree T is regular iff it is the solution of a regular system of equations of the form: $\varphi_i = t_i$ where function symbols in F are of type $D^n \longrightarrow D$, $n \in \mathbb{N}$ and function symbols in Φ are of type D

Theorem (Courcelle – TCS 1983)

T is regular iff the set of its branches is a regular language.
Equality of regular trees is decidable.

Regular Trees and ...

- word languages, orderings (Thomas, ..)
- logics (Caucal, Colcombet, ...)
- abstract data types (Goguen, ADJ, ...)
- functional programming ML, Haskell,... (Holderman, Rodriguez, Morris, Jeuring,..)
- XML(Castagna, David, Segoufin, ..)
- program transformations (Gallagher, Visser,...)
- Model checking (Bouajjani, ..)
- Theorem provers (Casteran, Klarlund, Møller,...)
- *Recursive types* (Amadio, Cardelli, Dyjber, ...)
- etc.

Recursive types

Recursive types intensively used in

- functional languages (ML, Haskell)
- Object oriented languages (JAVA)
- XML
- Theorem provers (Coq, Isabelle,MONA)

Some recursive types

Examples ($+$ denotes "or" union)

- Binary trees :
  $T = int + T \times T$ or $T = \mu\alpha.\,(int + \alpha \times \alpha)$
- Lists :
  $L = NIL + int \times L$ or $L = \mu\alpha.\,(NIL + int \times \alpha)$
- Streams : $S = int \times S$ or $S = \mu\alpha.\,(int \times \alpha)$
- Integers :
  $int = unit + int$ or $int = \mu\alpha.\,(unit + \alpha)$

Types and Trees

Non-recursive types $\iff$ Finite trees
Subtyping $\iff$ Ordering on trees

$$\bot \prec \alpha \prec \top$$

$\bot$ empty type (divergence)
$\top$ universal type (all values)

$$\alpha \prec \alpha' \ , \ \beta \prec \beta' \Longrightarrow \alpha \times \beta \prec \alpha' \times \beta'$$

$$\alpha' \prec \alpha \ , \ \beta \prec \beta' \Longrightarrow \alpha \longrightarrow \beta \prec \alpha' \longrightarrow \beta'$$

Example:

$$\begin{tikzcd}
 & \longrightarrow \arrow[dl, dash] \arrow[dr, dash] & & & & \longrightarrow \arrow[dl, dash] \arrow[dr, dash] & \\
\top & & \bot & \prec & \bot & & \bot
\end{tikzcd}$$

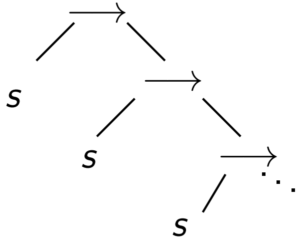
Recursive types and regular trees

- Order and metric on trees [Arnold –Nivat]: contracting maps have unique fixed points
- Recursive type $\Longleftrightarrow$ a regular infinite tree (unique solution of a contracting system of equations)
- Examples : $\mu t.\ t = \perp$

$$s \longrightarrow \mu t.\ t \quad = \quad \begin{tikzcd} & \longrightarrow \arrow[dl, no head] \arrow[dr, no head] & \\ s & & \perp \end{tikzcd}$$

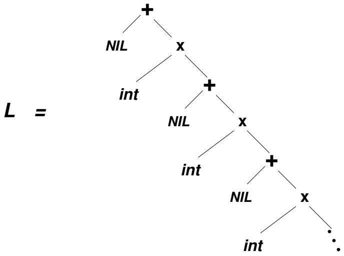
Examples (continued)

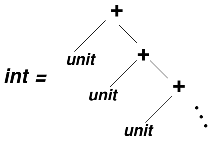
$\mu t.\ (s \longrightarrow t) \quad =$

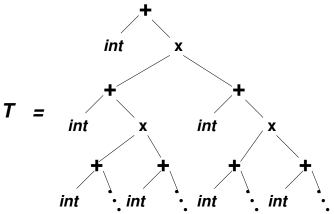


$$= \ \mu t.\ (s \longrightarrow (s \longrightarrow t))$$

Examples (continued)







Equivalence of recursive types

- Equivalence of recursive types reduces to equality of regular trees
- Equivalence is decidable in time $O(n \log n)$ and nicely axiomatizable (sound, complete and simple axiom system)
- Subtyping of recursive types (constructors: $\longrightarrow$ and least fixed point) is also decidable in time $O(n^2)$

References: Abadi, Amadio, Castagna, Cardelli, Bertot, Brandt-Henglein, Cardone, Casteran, Compagnoni, Coppo, DiCosmo, Fiore, Jha, Zhao, Kozen, Palsberg, Pottier-Gauthier, Swartzbach, Tyurin, Wadler,etc.

Equational Recursive types

Example: associativity, commutativity, distributivity equations should hold, even though the types are not subtypes of each other

$$\begin{aligned}\alpha \times (\beta_1 \times \beta_2) &\equiv (\alpha \times \beta_1) \times \beta_2 \\ \alpha \times (\beta_1 + \beta_2) &\equiv \alpha \times \beta_1 + \alpha \times \beta_2\end{aligned}$$

- deciding equality and subtyping in classes of models (Palsberg-Zhao, Fiore,...)
- reducing subtyping with intersection types to equations (Castagna,...)

Outline

Algebraic trees

An infinite tree T is algebraic iff it is the solution of a deterministic system of equations of the form: $\varphi_i(x_1, \dots, x_i) = t_i$ where all function symbols in $F \cup \Phi$ are of type $D^n \longrightarrow D$, $n \in \mathbb{N}$

Theorem (Courcelle – TCS 1983)

T is algebraic iff the set of its branches is a deterministic context-free language.

Theorem (Sénizergues – 1997)

The equivalence problem for DPDA is solvable.
The equality problem for algebraic trees is solvable.

Generalizations

- Non Determinism (Arnold, Nivat,...)
- Higher Order : fixed point semantics can be adapted by using more complex signatures: each level n tree is the image by a canonical morphism of a regular level (n+1) tree (Engelfriet-Schmidt, Damm, ...)
- Higher Order Schemes and Model Checking: Caucal, Knapik-Niwinski-Urzyczyn, Kobayashi, Ong, Walukiewicz)

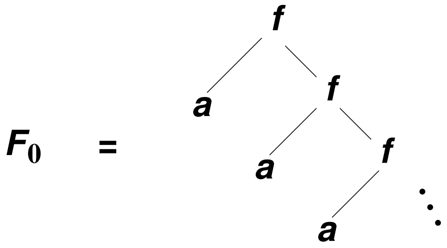
Algebraic Trees - applications

- functional programming
- Object Oriented languages (OBJ : Goguen, Meseguer, Jouannaud,..)
- order sorted algebras (Meseguer, Erwig, ...)
- Higher order extensions (Damm, Gallier, Haxthausen, Meinke, Serre,...)
- Category theory (Winskell,...)
- *Algebraic Orderings* (Bloom, Esik, Braud, Carayol,...)
- etc.

Algebraic Orderings

$$F_0 = f(a, F_0)$$

F_0 is the regular tree



The Frontier of F_0 is isomorphic to $\mathcal{F}_0 = 1^*0$ (ordinal ω).

$$0 < 10 < 110 < \cdots < 1^n0 < 1^{n+1}0 < \cdots$$

Algebraic Orderings

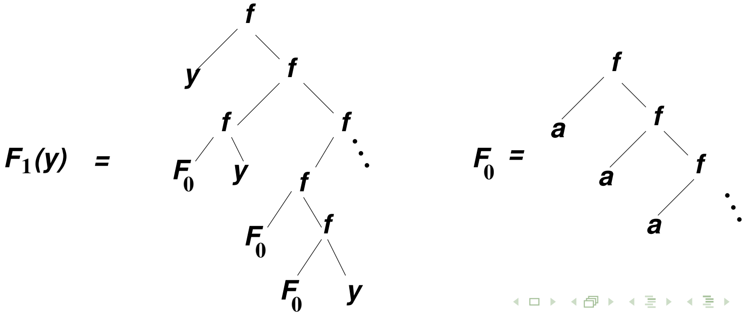
Theorem (Bloom-Choffrut)

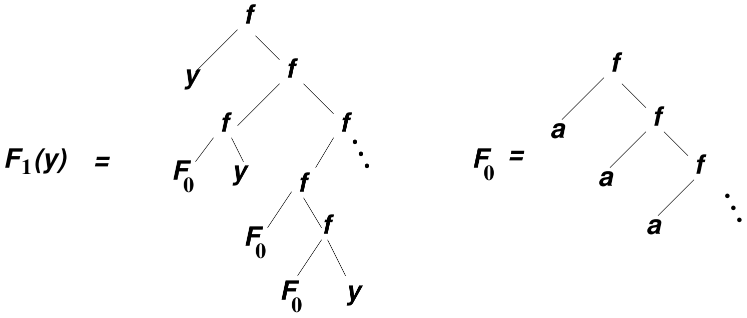
An ordinal α is regular iff it satisfies one of the equivalent conditions

- α is isomorphic to the frontier of a regular tree
- $\alpha < \omega^\omega$.

Algebraic Orderings

$$\begin{aligned} F_0 &= f(a, F_0) \\ F_1(y) &= f(y, F_1(f(F_0, y))) \end{aligned}$$





Frontier of F_0 is $\mathcal{F}_0 = 1^*0$
Frontier of $F_1(y)$ is $0 < 100\mathcal{F}_0 \cdots < 101 < 1100\mathcal{F}_0 \cdots < 11010\mathcal{F}_0 \cdots < 11011 < \cdots$

The Frontier of $F_1(y)$ is isomorphic to

$$\mathbf{y_1}\ \mathcal{F}_\mathbf{0}\ \mathbf{y_2}\ \mathbf{2}\mathcal{F}_\mathbf{0}\ \mathbf{y_3}\ \mathbf{3}\mathcal{F}_\mathbf{0}\ \cdots$$

(ordinal ω^ω).

Theorem (Bloom et al.)

An ordinal α is algebraic iff it satisfies one of the equivalent conditions

- α is isomorphic to the frontier of an algebraic tree
- $\alpha < \omega^{\omega^\omega}$.

Generalization to Higher Order : Braud-Carayol

Outline

Mathematical semantics

Algebraic, Denotational, Fixpoint,. . .
Goals:

- **give precise meaning to programs**
- **prove equivalences and properties of programs**

Decisive step: Scott (1971) The Lattice of Flow Diagrams

Algebraic meaning for iterative program schemes à la Ianov

Scott domain

$(D, \leq)$ Scott domain

- every directed set has a least upper bound
- the set of finite elements is countable (x finite iff $x \sqsubseteq \sup\{y | y \in Y\} \Longrightarrow \exists y \in Y \ , \ x \sqsubseteq y$)
- every element in D is the least upper bound of the directed set of its finite approximations (algebraic domain)

$f : D^n \longrightarrow D$ continuous $\iff f$ monotone and f preserves sup.

Interpretation

Assume a set F of function symbols, an interpretation is a Scott domain D where all function symbols in F are interpreted as continuous functions $f_i \colon D^n \longrightarrow D$ (n the arity of f).

The notion of interpretation is expressed by a second order formula: D ordered, every directed set has a sup, set of finite elements countable, every x is the sup of its finite approximants, every f is sup-preserving.

Scott ... or Park... or...?

Park 1970

Fixpoint Induction and Proofs of program Properties

- meaning of **recursive schemes** as least fixpoints
- proofs of properties of schemes by fixpoint induction
- considers also greatest fixpoints

Nivat 1975

On the Interpretation of Recursive Polyadic Program Schemes

Example: $f(\perp) \leq a$ and $\forall n \left[f^n(\perp) \leq a \Rightarrow f^{n+1}(\perp) \leq a \right]$
$\Longrightarrow \mu x \, . \, f(x) = f^\omega(\perp) = \sup\{f^n(\perp)\} \leq a$

Semantics in the 70s

Many researchers worked on the algebraic, logic and automata theory approach to semantics:
Nivat, Park, Scott, Strachey, Luckham, Paterson, Plaisted, Milner, Ashcroft, Hennessy, Winskell, Milne, Morris, Strong, de Bakker, de Roever, Courcelle, Berry, Kotts, Downey, Engelfriet, Indermark, Damm, Fehr, Arnold, Dauchet, Guessarian, Gallier, Manna, Chandra, Vuillemin, Cadiou, Raoult, Burstall, Darlington, Andreka, Nemeti, Tiuryn, ... and even Knuth.

Two program schemes are equivalent iff they compute the same function in every interpretation: this implies that they compute the same function in the Herbrand interpretation, *i.e.* that the associated algebraic trees are equal.

consequence

The equivalence problem for program schemes and the equality of infinite trees are interreducible problems.

Other more interesting equivalences (unfortunately more complex) by restricting the class of interpretations with respect to which the programs are equivalent.

Classes of interpretations

Equivalence: P, P' program schemes , T, T' the associated trees,

$$P \approx P' \quad \Longleftrightarrow T = T'$$

Too restrictive: relax the demand that P and P' should compute the same function for every interpretation.

$\mathcal{C}$ **a class of interpretations:** $P \approx_{\mathcal{C}} P'$ iff P and P' compute the same function for every interpretation in $\mathcal{C}$.

$\approx_{\mathcal{C}}$ equivalence modulo $\mathcal{C}$

- Captures more interesting equivalences.
- Even harder to study.

wishes

- $\approx_{\mathcal{C}}$ decidable
- $\approx_{\mathcal{C}}$ characterized by its restriction to finite trees: *i.e.* $\mathcal{C}$ is algebraic
- $\approx_{\mathcal{C}}$ axiomatizable

Algebraic classes

To prove $T \approx_{\mathcal{C}} T'$ it suffices to prove $T \leq_{\mathcal{C}} T'$ and $T' \leq_{\mathcal{C}} T$

Definition: $T \leq_{\mathcal{C}} T'$ iff for every interpretation I in $\mathcal{C}$, $T_I \leq T'_I$ (T_I function computed by T in I).

Intuition: $\mathcal{C}$ is algebraic iff $\leq_{\mathcal{C}}$ is always "provable by induction". Formally, t finite tree:

$$t \leq_{\mathcal{C}} \sup_{n \in \mathbb{N}} t'_n \quad \text{iff} \quad \exists n \; t \leq_{\mathcal{C}} t'_n$$

Algebraicity vs induction

Fixed point induction: IF $f : D \longrightarrow D$ continuous , D Scott domain , $a \in D$, **THEN:**
$f(\perp) \leq a$ and $\forall n \left[f^n(\perp) \leq a \Rightarrow f^{n+1}(\perp) \leq a\right]$
$\Longrightarrow f^\omega(\perp) = \sup\{f^n(\perp)\} \leq a$

Algebraicity: IF $\mathcal{C}$ algebraic , and t finite tree, **THEN:**

$$\left[t \leq_{\mathcal{C}} \sup\{f^n(\perp)\}\right] \Longrightarrow \exists\, n \;\; t \leq_{\mathcal{C}} f^n(\perp)$$

Some results

Courcelle-Guessarian : On some classes of interpretations (1978)

- $\mathcal{D}$ (the class of discrete interpretations) is algebraic and $\leq_{\mathcal{D}}$ is decidable
- any first-order definable class is algebraic.
- equational classes are algebraic.
  $\mathcal{C}_R$ is a equational class *iff* $\exists R \subset FTrees \times FTrees$ such that
  $\mathcal{C}_R = \{I \mid \forall\, t, t' \in R \quad t_I = t'_I\}$
- Relational classes are algebraic.
  $\mathcal{C}_R$ is a relational class *iff* $\exists R \subset FTrees \times FTrees$ such that
  $\mathcal{C}_R = \{I \mid \forall\, t, t' \in R \quad t_I \leq t'_I\}$

Developments

- "IF...THEN...ELSE..." classes of interpretations (Guessarian-Meseguer 1987), applications to rewriting and proof systems
- "IF...THEN...ELSE..." classes of interpretations in category-theory framework (Adamek-Nelson-Reiterman 1987)
- Conditional term rewriting systems (Kaplan-Jouannaud 1988)

Related Work – Extensions

- Equational classes (R set of equalities): rewriting (Baader et al. 1996, 2002), regular equational tree languages (Hendrix-Ohsaki 2008)
- R is defined by infinite trees : equational constraints on Recursive Data Types, functions on the quotient (initial) algebra; automated reasoning, verification, theorem provers (Paulson 2004)
- Object oriented languages, order - sorted algebras, many-sorted equational logic (Goguen-Meseguer 1982)
- Category-theory framework (Bloom-Tindell 1983, Adamek-Nelson-Reiterman 1987)
- Order-sorted + Higher-order specifications (Haxthausen 1997)

Outline

Summary

Some applications of infinite trees and program schemes

- Applications: object oriented languages (Java) and types
- languages and algebra
- semantic equivalences
- regular model checking: analyse/verify program using tree automata
- logics
- ...

Extensions - Open problems

- deciding equality and subtyping in classes of models
- higher-order schemes, and model checking
- higher-order recursive types (object encodings) but associated trees non regular... ;
- recursive polymorphic types

Programming = μ study. (TREES + study)

THANKS FOR YOUR ATTENTION

Extensions - Open problems

- deciding equality and subtyping in classes of models
- higher-order schemes, and model checking
- higher-order recursive types (object encodings) but associated trees non regular... ;
- recursive polymorphic types

Programming = μ study. (TREES + study)

THANKS FOR YOUR ATTENTION