Level Two Recursion Schemes and Finite Automata

Klaus Aehlig

(joint work with Jolie de Miranda and Luke Ong)

Supported by the German Academic Exchange Service (DAAD).

$\boxed{\textbf{The Problem}}$

- infinite tree t, given by a level 2 recursion scheme
  (idealised functional program)
- Monadic Second Order formula φ

Question: $$t \models \varphi \quad ?$$

...and find an algorithm deciding this.

Justification

- Automatic verification is important...
- Functional programming is natural, expressive, ...
- ...and it's an interesting piece of mathematics after all!

(Level 2) Recursion Schemes

Short answer: simply typed lambda terms with fixed points

Long answer: A level 2 recursion scheme is given by

- non-terminals $N\colon (\iota \to \ldots \to \iota) \to \ldots \to \iota \to \ldots \to \iota$
  one of them: the start symbol S

- terminals $\mathtt{f}\colon \iota \to \ldots \to \iota$

- Equations for all non-terminals N

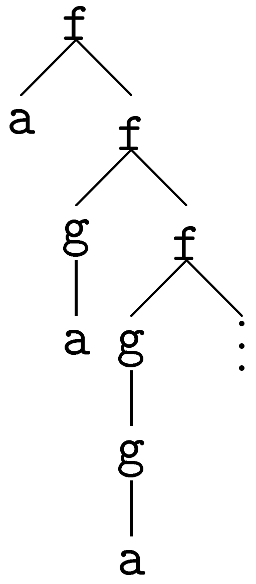
$$N\overrightarrow{\varphi}\overrightarrow{x} = e$$

  e built of terminals $\mathtt{f}, \mathtt{g}, \mathtt{a}, \ldots$, variables $\overrightarrow{\varphi}$, $\overrightarrow{x}$, and arbitrary non-terminals $N, M, \ldots$

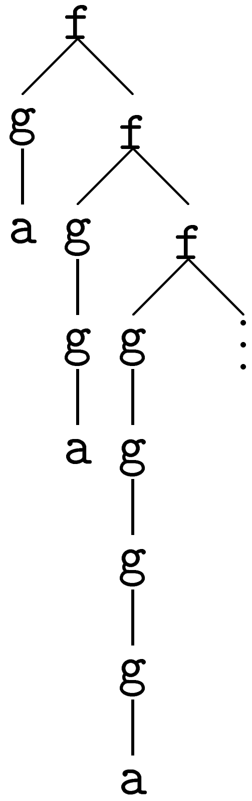
  Note: <u>NO</u> restrictions on the buildup of e.

Examples

$$S = F\mathtt{a}$$
$$Fx = \mathtt{f}x(F(\mathtt{g}x))$$



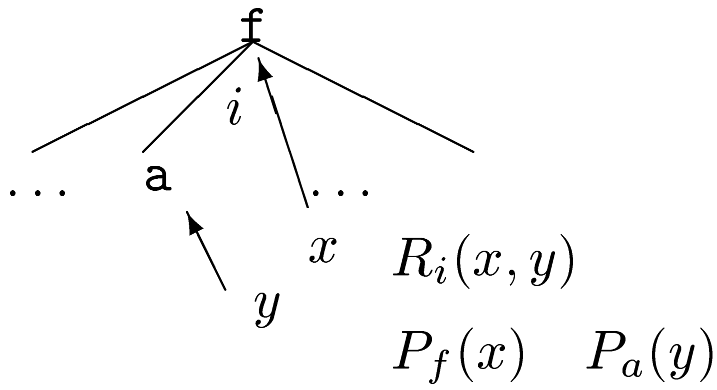
$$S = F\mathtt{g}$$
$$B\varphi\psi x = \varphi(\psi x)$$
$$F\varphi = \mathtt{f}(\varphi\mathtt{a})(F(B\varphi\varphi))$$



MSO Formulae...

- Child relation between nodes
- Labelling of a node
- propositional logic: $\wedge$, $\vee$, $\neg$, ...
- Quantification: $\forall x$, $\exists x$
- Sets of nodes X with new atomic formula $x \in X$ and quantification $\forall X$, $\exists X$

  like: "there is a path..."

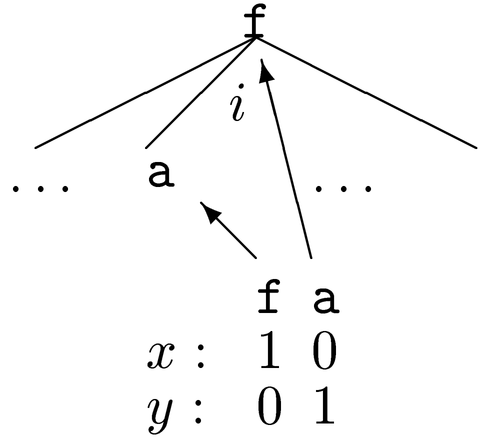


. . . can be understood as tree automata

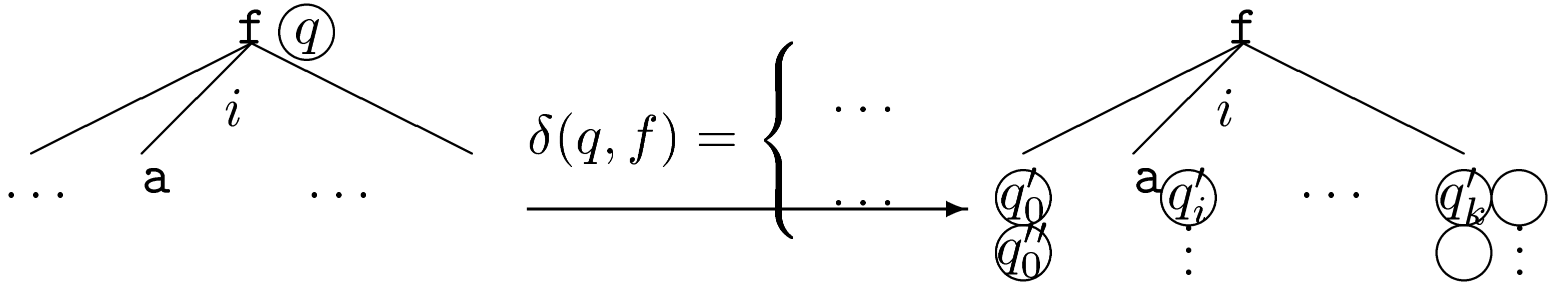
- first order variables $\longrightarrow$ singleton sets *(and $x \in X$ iff $\{x\} \subset X$)*

$\rightsquigarrow$ can be treated as additional labels

- $\exists X$ becomes projection of the alphabet *(non-deterministic choices)*



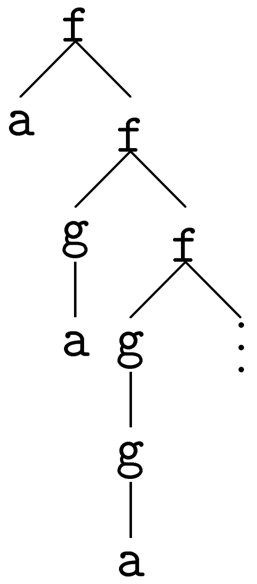
$\therefore$ validity can be understood as the successful run of an appropriate tree automaton (from a class that is closed under complement)



Essence: $\boxed{\text{MSO properties} \equiv \text{languages of finite tree automata}}$

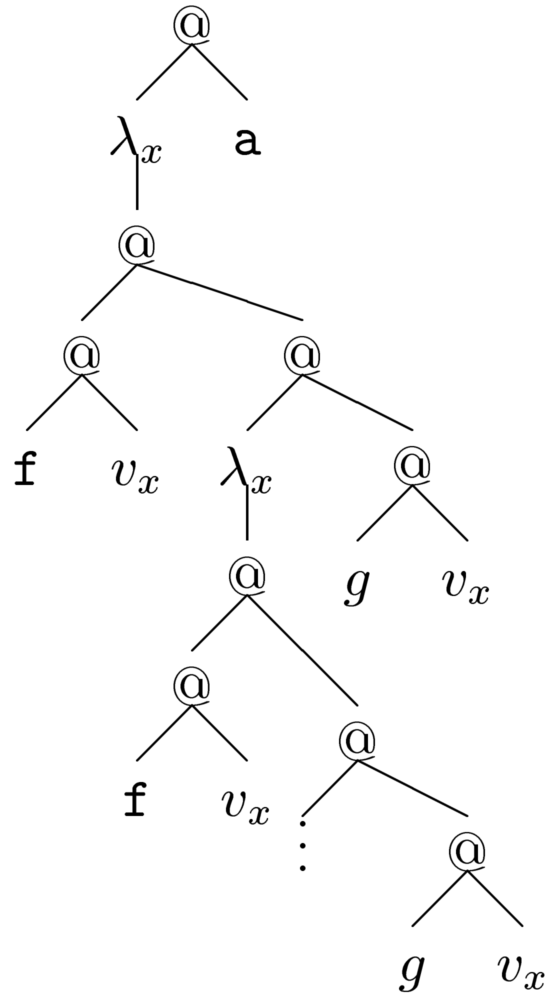
Making the trees regular

$$
\begin{aligned}
S &= F\mathtt{a} \\
Fx &= \mathtt{f}x(F(\mathtt{g}x))
\end{aligned}
$$



$\rightsquigarrow$

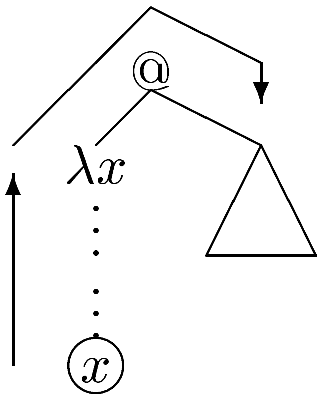
$$
\begin{aligned}
S &= F@\mathtt{a} \\
F &= \lambda_x(\mathtt{f}@v_x@(F@(\mathtt{g}@v_x)))
\end{aligned}
$$



The trees we consider are <u>not</u> regular

– but their definition is!

$\boxed{\textbf{Introducing } \beta\textbf{-Redexes}}$



- We are interested in automata runs over trees
- But our grammar is a definition of functions

  (Naturally given by λ-terms.)

$\rightsquigarrow$ when hitting a variable go and look for its argument!

$\therefore$ can work with a lambda tree – and don't have to unfold the grammar

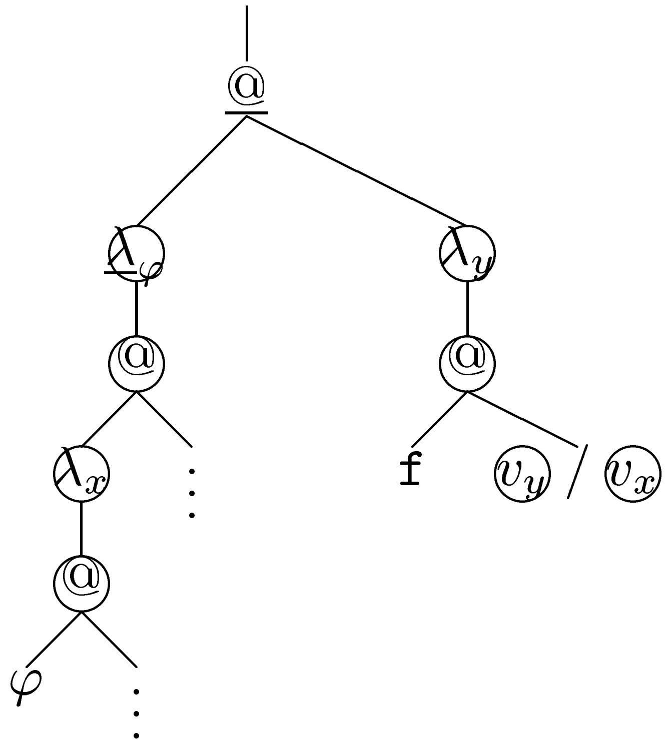
At least for redexes of ground types...

. . . and more β-redexes

- MSO-define the relation "variable – denoted tree"
- But we have function abstraction as well!

$$N\overrightarrow{\varphi}\,\overrightarrow{x} = e \qquad \rightsquigarrow \qquad N = \lambda\overrightarrow{\varphi}\,\lambda\overrightarrow{x}\,e$$

- Moreover: name capture. . .

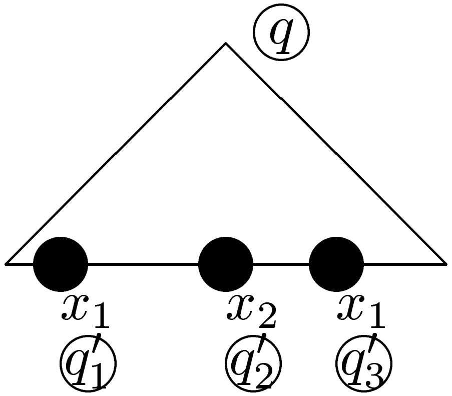


How to describe a function variable φ?

- A function $\varphi\colon \iota \to \ldots \to \iota \to \iota$ is a tree with "holes" for the arguments
- For a run we only need to know with which states we continue at the various arguments.

! Note that we can identify several occurrences of the same argument

$$\begin{array}{ccc} & \varphi\, x_1 x_2 \\ q & \mapsto & q_1' \; q_2' \\ & & q_3' \end{array} \quad =$$



$\therefore$ finite description of φ.

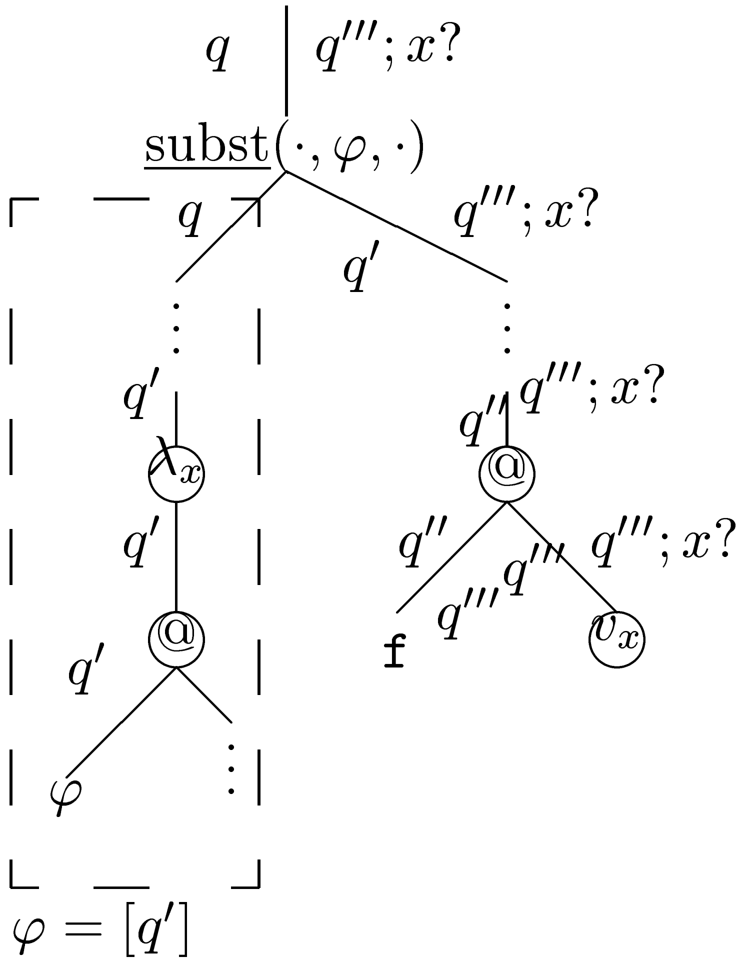
Proving the existence of a Run

- All non-terminals take at most function arguments and right hand side of ground type
  $\therefore$ all function arguments present

$\rightsquigarrow$ write $\underline{\text{subst}}(t, \overrightarrow{\varphi}, \overrightarrow{t})$ for $(\underline{\lambda}\overrightarrow{\varphi}.t)\overrightarrow{t}$

- At each $\underline{\text{subst}}(\cdot, \overrightarrow{\varphi}, \cdot)$ guess the transition function for the φ_i. *(cut formula)*
  fixed size $\rightsquigarrow$ store in state!

... In the main branch verify the run with the "knowledge" of the transitions in the φ_i
In the side branches verify the guess.



Summary of Argument

- Recursion scheme $\to$ lambda tree with $\underline{\lambda}$, $\underline{@}$

  This tree is regular.

  $\leadsto$ described by some automaton

- MSO formula $\to$ automaton accepting trees with this property

  Simulation...

  $\to$ another automaton accepting trees that denote (after β-reductions) trees with the given property

- Intersect these two automata

  $\leadsto$ automaton accepting trees that come from the grammar and have the given property

- Check emptiness of the last automaton *(decidable!)*

Questions and Further Work

- Used that for level-2 recursion schemes all arguments are present.
  Is this really necessary?

- Main idea was: get a finite semantics for functions
  Finite semantics for funcitonals, higher type objects?
  $\rightsquigarrow$ *does this idea generalise to all levels?*

- What is the optimal complexity of this decission procedure?
  Which hard problems can we encode?