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).

Recursion Schemes – Motivation

Idea

- Want to study functional programming

! But from a verification point of view

$\rightsquigarrow$ abstract away from data! *(in particular no case distinctions)*

... leaves the calling structure and the way the output is composed

- terminals to eventually obtain a tress *(results, observable events)*

Propaganda

- Automatic verification is important...
- Functional programming is natural, expressive, ...

Recursion Schemes

Here we only consider level 2 recursion schemes.

- non-terminals $N\colon (\iota \to \ldots \to \iota) \to \ldots \to \iota \to \ldots \to \iota$
- terminals $f\colon \iota \to \ldots \to \iota$
- variables $(x\colon \iota,\ \varphi\colon \iota \to \ldots \to \iota \to \iota)$
- Equations for all non-terminals N

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

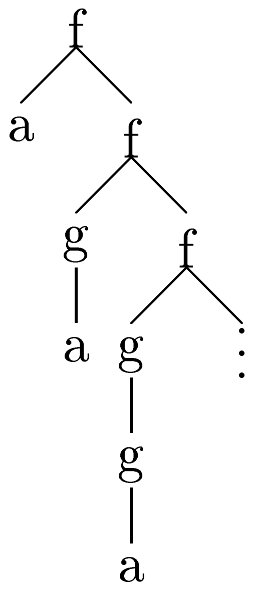
$\uparrow$

built of

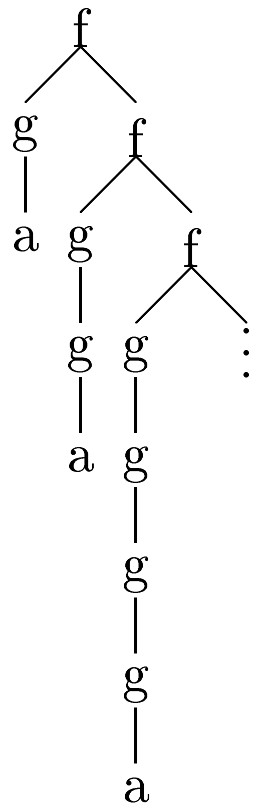
- terminals $f, g, a, \ldots$
- variables $\overrightarrow{\varphi}$, $\overrightarrow{x}$
- arbitrary non-terminals $N, M, \ldots$

Examples

$$
\begin{aligned}
S &= Fa \\
Fx &= fx(F(gx))
\end{aligned}
$$



$$
\begin{aligned}
S &= Fg \\
B\varphi\psi x &= \varphi(\psi x) \\
F\varphi &= f(\varphi a)(F(B\varphi\varphi))
\end{aligned}
$$



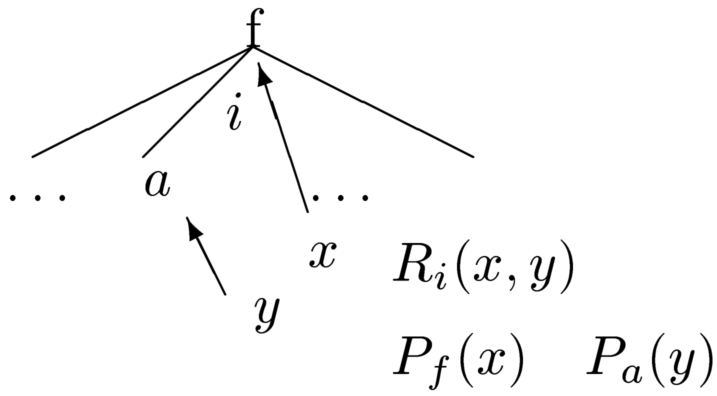
Properties of Trees

- 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 $x \in X$

  like: "there is a path...", "there is a descendent..."

! But only unary relations

  "There is a labelling $X(n, t)$ of the n'th square at time t of a Turing tape..."



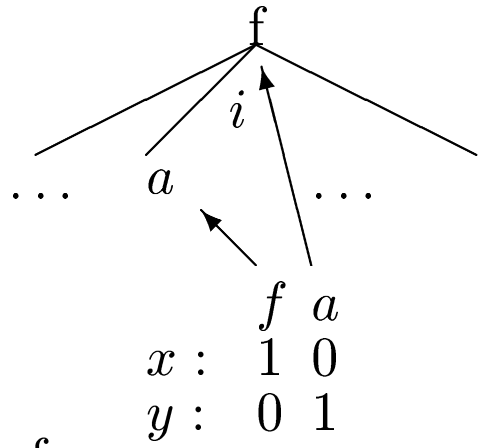
Tree Automata

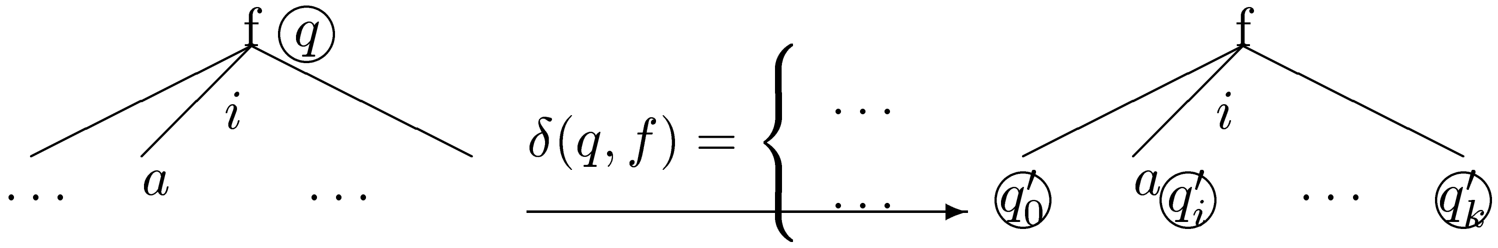
- first order variables $\longrightarrow$ singleton sets
  (and $x \in X$ iff $\{x\} \subset X$)

$\leadsto$ can be treated as additional labels

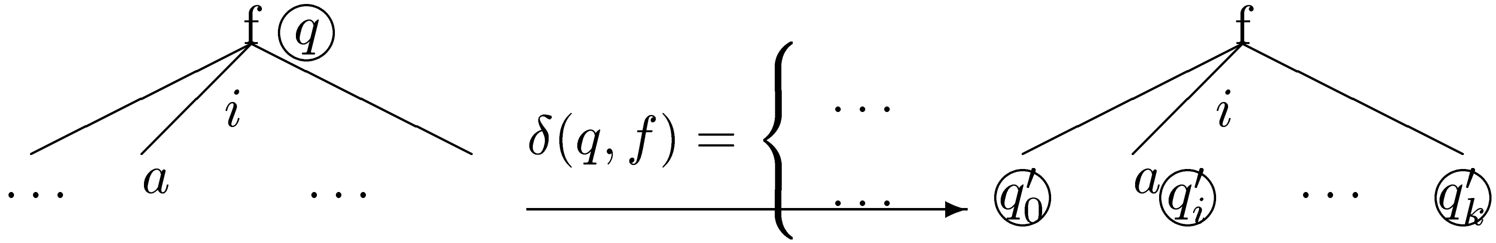
- $\exists X$ becomes projection of the alphabet

$\therefore$ validity can be understood as the successful run of an appropriate non-deterministic tree automaton





Tree Automata (continued)



- choices at every step (∴ projection, i.e., $\exists X$)

$\rightsquigarrow$ runs of the automaton

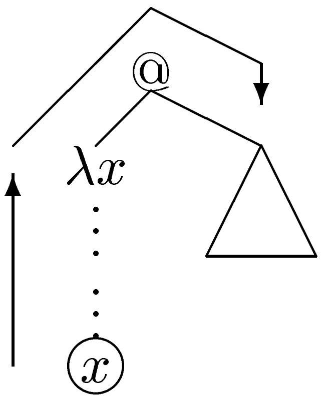
- conjunction by an appropriate product construction
- acceptance condition by the set of states visited infinitely often on each path (requiring $\mathrm{Inf}(\pi) \in \Omega \subset \mathfrak{P}(Q)$)

$\rightsquigarrow$ closure under complement (negation!)

actually the hard part of the (known) proof

Essence: MSO properties $\equiv$ languages of tree automata

Introducing β-Redexes



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

  (Naturally given by λ-terms;
  unfolding of β-redexes can be hard)

$\leadsto$ 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...

...but we're interested in level 2

- Level 1 recursion schemes are well understood.

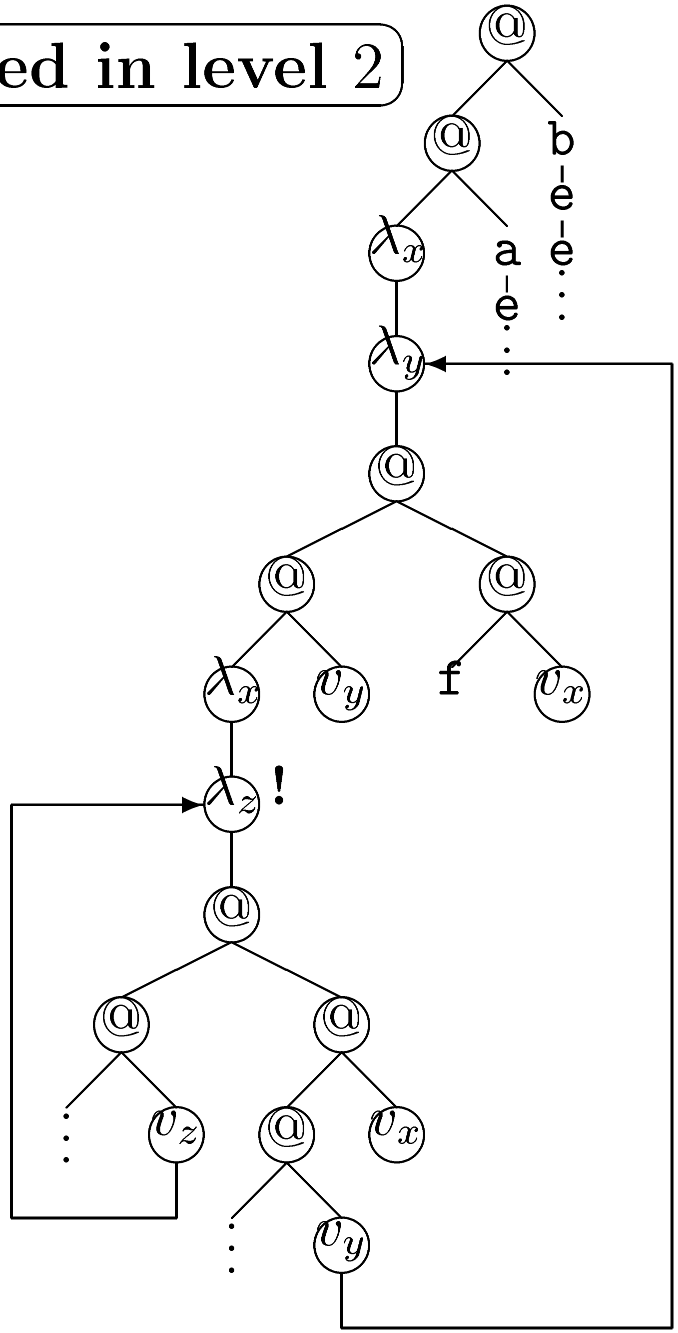
? Why not just reduce the level by one, by generating the corresponding lambda tree

! Name capture! Consider the following grammar.

$$F\varphi xy = F(F\varphi y)y(\varphi x) \qquad A = aE$$

$$F\varphi xy = y \qquad B = bE$$

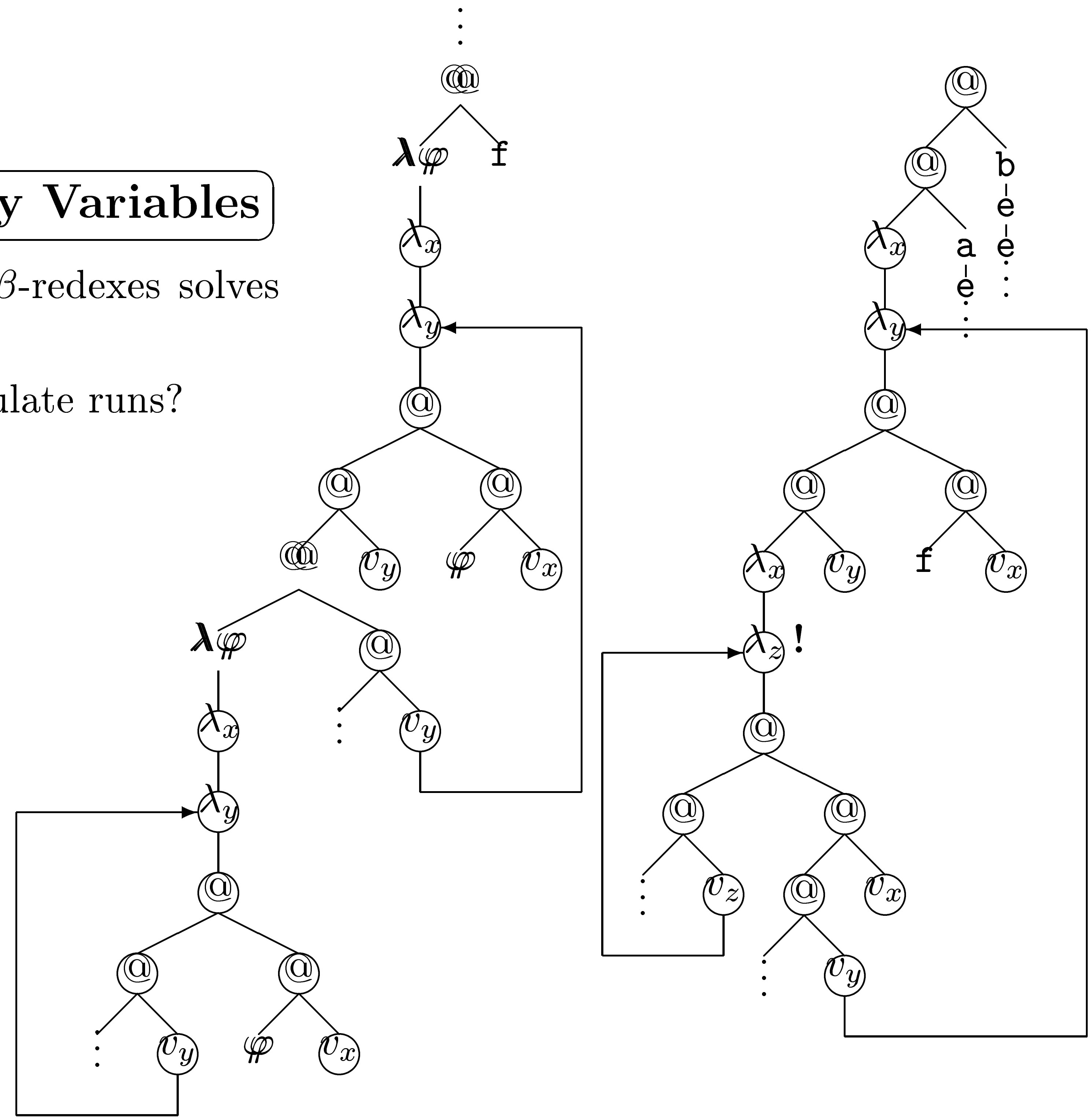
$$S = FfAB \qquad E = eE$$



Finitely Many Variables

Introducing more β-redexes solves the problem…

…but who to simulate runs?

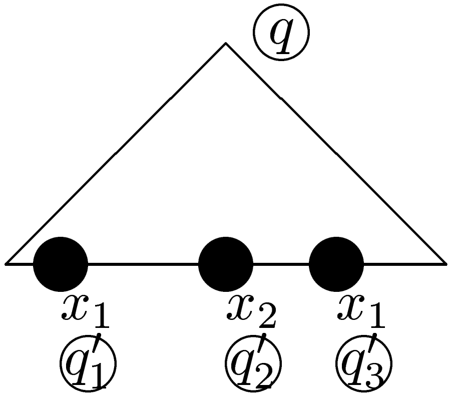


How to describe a function order variable φ?

- 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

$$\varphi x_1 x_2 \quad = \qquad q \mapsto q_1' q_2' q_3'$$



$\therefore$ finite description of φ.

Simulating a Run

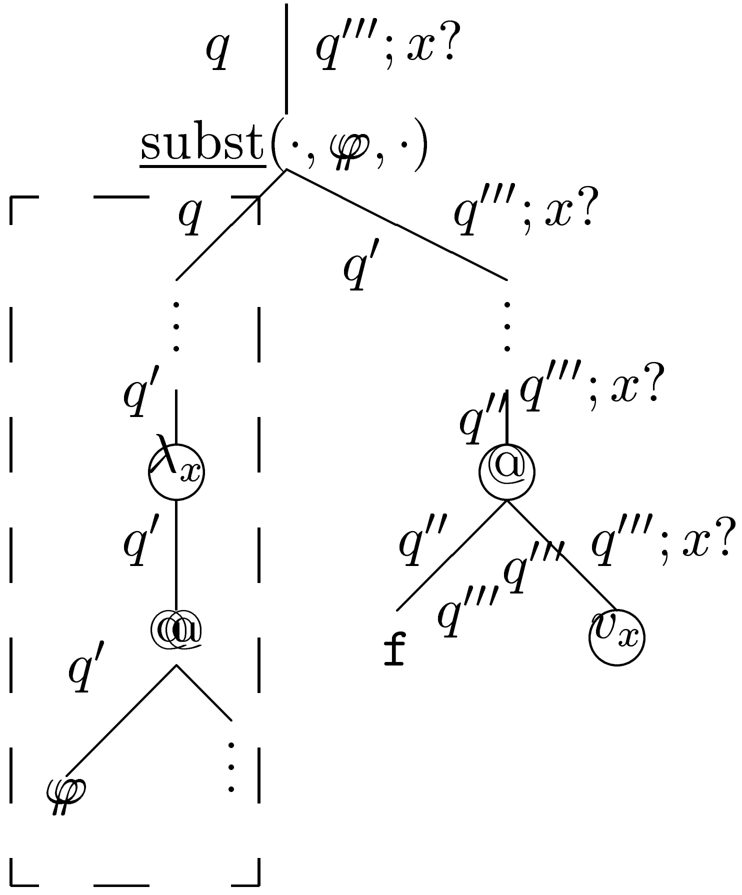
- All non-terminals take at most function arguments; right hand side of ground type

$\therefore$ all function arguments present

- write $\underline{\text{subst}}(t, \overrightarrow{\varphi}, \overrightarrow{t})$ for $(\boldsymbol{\lambda}\overrightarrow{\varphi}.t)\overrightarrow{t}$
- At each $\underline{\text{subst}}(\cdot, \overrightarrow{\varphi}, \cdot)$ guess the transition function for the φ_i

$\ldots$ In the main branch simulate the run with the "knowledge" of the transitions in the φ_i

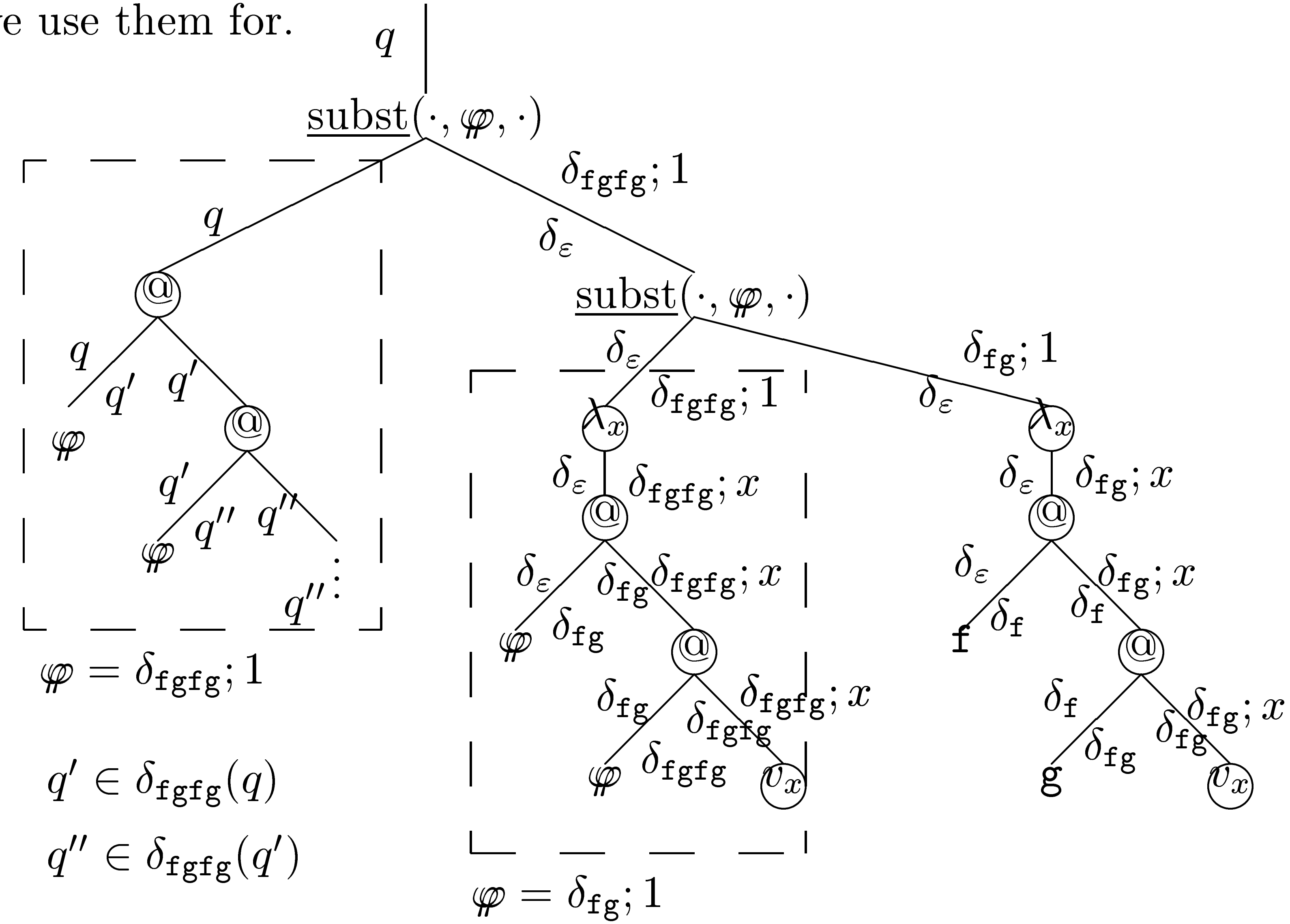
In the side branches verify the guess.



$\varphi = [q']$

Guesses Within Guesses

No problem! The guesses are absolute ones. We don't have to care what we use them for.



Summary of Argument

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

  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-reduction) 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!)*