Backtracking Machine (for Tree Computations)

- **If mode = ramify then**

  Let k = |alternatives (Params)|

  Let o_1 ,..., o_k =new (NODE)

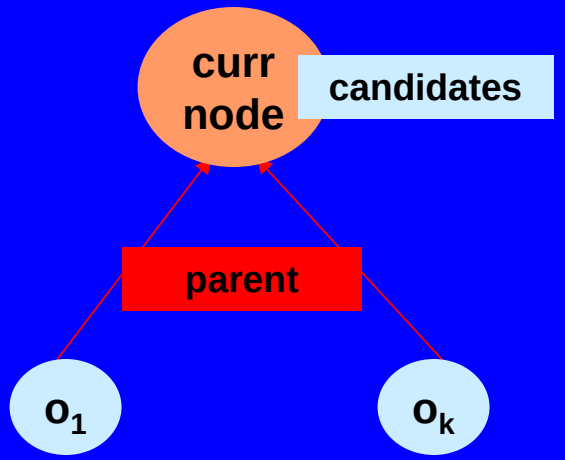
  candidates (currnode) := { o_1 ,..., o_k }

  forall $1 \leq i \leq k$ do

  parent (o_i) := currnode

  env (o_i) := i-th (alternatives (Params))

  mode := select



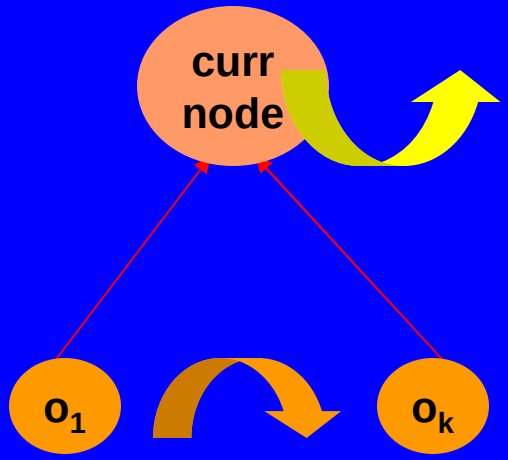
- **If mode = select then**

  If candidates (currnode) = ∅

  then backtrack

  else try-next-candidate

  mode := execute



Backtracking Machine

- backtrack ° if parent (currnode) = root
  then mode := Stop
  else currnode := parent (currnode)
- **try-next-candidate** ° depth-first tree traversal

  currnode:= next (candidates(currnode))

  delete next (candidates(currnode)) from candidates (currnode)
- The fctn next is a choice fct, possibly dynamic, which determines the order for trying out the alternatives.
- The fct alternatives, possibly dynamic and coming with parameters, determines the solution space.
- The execution machine may update mode again to ramify (in case of successful exec) or to select (for failed exec)

Backtracking Machine: logic progg instantiation

- Prolog Börger/Rosenzweig Science of Computer Programming 24 (1995)
  - alternatives = procdef (act,pgm), yielding a sequence of clauses in pgm, to be tried out in this order to execute the current statement ("goal") act
    - procdef (act,constr,pgm) in CLAM with constraints for indexing mechanism Börger/Salamone OUP 1995
  - next = first-of-sequence (depth-first left-to-right tree traversal)
  - execute mode resolves act against the head of the next candidate, if possible, replacing act by that clauses' body & proceeding in mode ramify, otherwise it deletes that candidate & switches to mode select

Backtracking Machine: functional progg instantiation

- Babel Börger et al. IFIP 13 World Computer Congress 1994, Vol.I
  - alternatives = fundef (currexp,pgm), yielding the list of defining rules provided in pgm for the outer fct of currexp
  - next = first-of-sequence
  - execute applies the defining rules in the given order to reduce currexp to normal form (using narrowing, a combination of unification and reduction)

Backtracking Machine: context free grammar instantiation

- Generating leftmost derivations of cf grammars G
  - alternatives (currnode,G), yields sequence of symbols Y1...Yk of the conclusion of a G-rule with premisse X labeling currnode. Includes a choice bw different rules X→w
  - env yields the label of a node: variable X or terminal letter a
  - next = first-of-sequence (depth-first left-to-right tree traversal)
  - execute mode
    - for nodes labeled by a variable triggers tree expansion
    - for terminal nodes extracts the yield, concatenating terminal word to output, continues derivation at parent node in mode select

If mode = execute then
  If env (currnode)∈ VAR
    then mode:=ramify
    else output:=output * env(currnode)
         currnode:= parent(currnode)
         mode := select

alternatives can be a dynamic fct (possibly monitored by the user) or static (with first argument in VAR)

Initially NODE = {root}
root=currnode
env(root)=G-axiom
mode=ramify

Backtracking Machine: instantiation for attribute grammars

- Synthesis of node attribute from children's attributes via backtrack ° if parent (currnode) = root then mode := Stop
  else currnode := parent (currnode)
  $X.a := f(Y_1.a_1, ..., Y_k.a_k)$
  - where X = env(parent(currnode)), Y_i =env(o_i) for children nodes
- Inheriting attribute from parent and siblings
  - included in update of env (e.g. upon node creation) generalized to update also node attributes
- Attribute conditions for grammar rules
  - included in execute-rules as additional guard to yielding output

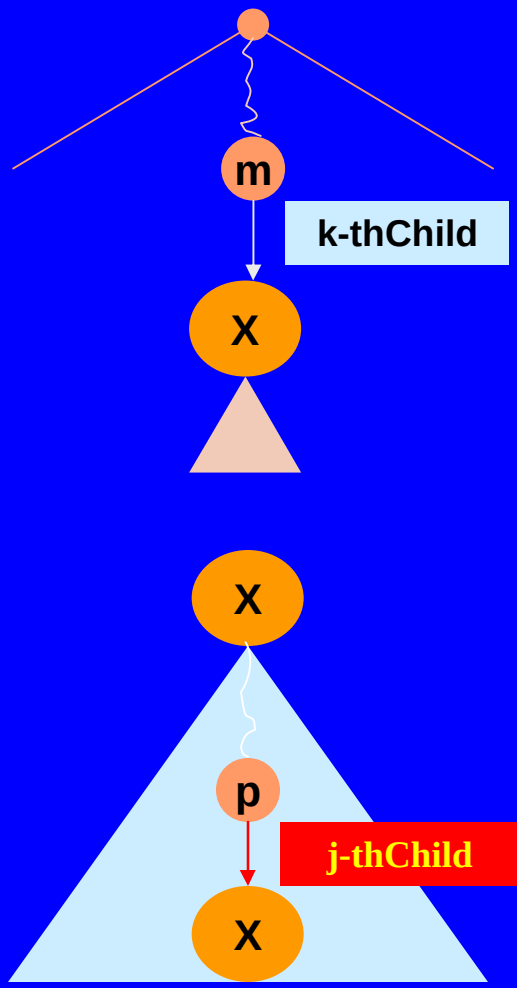
Johnson/ Moss Linguistics &Philosophy 17 (1994) 537-560

If mode = execute then …

else If Cond(currnode.a, parent(currnode).b, siblings(currnode).c)
then output:=output * env(currnode)
currnode:= parent(currnode) , mode := select

Tree Adjoining Grammars

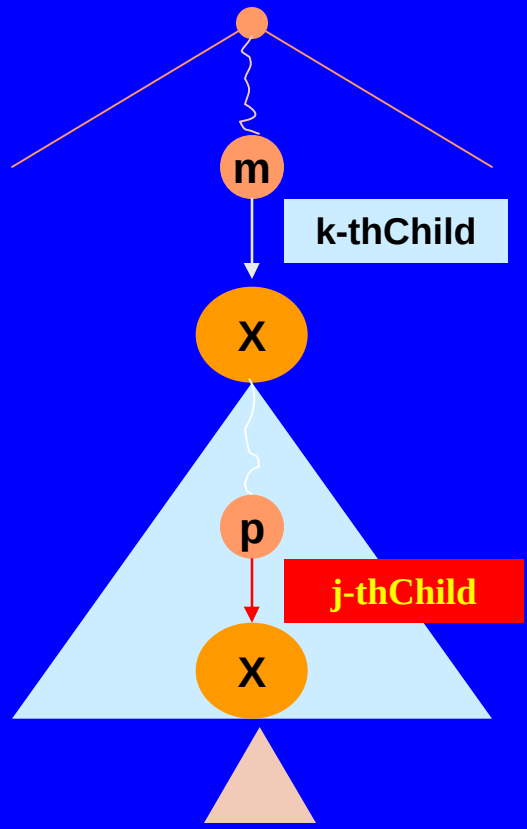
Generalizing Parikh's analysis of context free languages by pumping of cf trees from basis trees (with terminal yield) and recursion trees (with terminal yield except for the root variable)



If n=k-thChild(m) & symb(n)=symb(root(T)) & T Î RecTree & foot(T) = j-thChild(p)

Then

Let T'=new copy(T) in k-thChild(m):=root(T') j-thChild(p'):=n



References

- E.Börger and D. Rosenzweig: Mathematical Definition of Full Prolog
  - In: Science of Computer Programming 24 (1995) 249-286
- E.Börger and R.F.Salamone: CLAM Specification for Provably Correct Compilation of CLP (R) Programs
  - In: E.Börger (Ed.) Specification and Validation Methods. Oxford University Press, 1995, 97-130
- E.Börger, F.J.Lopez-Fraguas, M.Rodrigues-Artalejo: A Model for Mathematical Analysis of Functional Programs and their Implementations
  - In: B.Pehrson and I.Simon (Eds.): IFIP 13 World Computer Congress 1994, Vol.I: Technology/Foundations, 410-415
- D. Johnson and L. Moss: Grammar Formalisms Viewed als Evolving Algebras
  - Linguistics and Philosophy 17 (1994) 537-560