Briefly on Bottom-up [Algoritmi da]

Paola Quaglia
University of Trento

October 8, 2017

Definition 0.1. *Let $\mathcal{Q}$, V, and τ be, respectively, the set of states, the vocabulary, and the transition function of a characteristic automaton. Also, let $\mathcal{LA}_i$ be an actual instance of the lookahead function. Then, the* parsing table for *the pair constisting of the given characteristic automaton and the given lookahead function is the matrix $\mathcal{Q} \times (V \cup \{\$\})$ obtained by filling in each entry (P, Y) after the following rules.*

- *Insert* "Shift Q" *if Y is a terminal and $\tau(P, Y) = Q$.*
- *Insert* "Reduce $A \to \beta$" *if P contains a reducing item for $A \to \beta$ and $Y \in \mathcal{LA}_i(P, A \to \beta)$.*
- *Set to* "Accept" *if P contains the accepting item and $Y = \$$.*
- *Set to* "Error" *if Y is a terminal or \$, and none of the above applies.*
- *Set to* "Goto Q" *if Y is a nonterminal and $\tau(P, Y) = Q$.*

Algorithm 1: Construction of either LR(0)-automaton or LR(1)-automaton (P_0 and closure(_) to be instantiated accordingly)

```
initialize 𝒬 to contain P_0;
tag P_0 as unmarked;
while there is an unmarked state P in 𝒬 do
    mark P ;
    foreach grammar symbol Y do
        Tmp ⟵ ∅;
                              /* Compute the kernel-set of the Y-target of P */
        foreach A → α · Yβ ∈ P do
            add A → αY · β to Tmp;
        if Tmp ≠ ∅ then
                              /* Check whether τ(P, Y) has already been collected */
            if Tmp = kernel(Q) for some Q in 𝒬 then
                τ(P, Y) ⟵ Q;
            else
                New_state ⟵ closure(Tmp);
                τ(P, Y) ⟵ New_state;
                add New_state as an unmarked state to 𝒬 ;
```

Algorithm 2: Computation of $\text{closure}_0(Q)$

```
function closure₀(P)
    tag every item in P as unmarked ;
    while there is an unmarked item I in P do
        mark I ;
        if I has the form A → α · Bβ then
            foreach B → γ ∈ 𝒫′ do
                if B → ·γ ∉ P then
                    add B → ·γ as an unmarked item to P ;
    return P ;
```

Algorithm 3: Computation of $\text{closure}_1(P)$

```
function closure₁(P)
    tag every item in P as unmarked ;
    while there is an unmarked item I in P do
        mark I ;
        if I has the form [A → α · Bβ, Δ] then
            Δ₁ ⟵ ⋃_{d∈Δ} first(βd) ;
            foreach B → γ ∈ 𝒫′ do
                if B → ·γ ∉ prj(P) then
                    add [B → ·γ, Δ₁] as an unmarked item to P ;
                else
                    if ([B → ·γ, Γ] ∈ P and Δ₁ ⊈ Γ) then
                        update [B → ·γ, Γ] to [B → ·γ, Γ ∪ Δ₁] in P ;
                        set [B → ·γ, Γ ∪ Δ₁] as unmarked ;
    return P ;
```

Algorithm 4: Construction of the symbolic automaton

```
x_0 ⟵ newVar();
Vars ⟵ {x_0};
P_0 ⟵ closure_1({[S' → ·S, {x_0}]});
initialize Eqs to contain the equation x_0 ≐ {$};
initialize 𝒬 to contain P_0;
tag P_0 as unmarked;
while there is an unmarked state P in 𝒬 do
    mark P ;
    foreach grammar symbol Y do
                                /* Compute the kernel-set of the Y-target of P */
        Tmp ⟵ ∅;
        foreach [A → α · Yβ, Δ] in P do
            add [A → αY · β, Δ] to Tmp;
        if Tmp ≠ ∅ then
            if prj(Tmp) = prj(kernel(Q)) for some Q in 𝒬 then
                                /* Refine Eqs to let Q be the Y-target of P */
                foreach ([A → αY · β, Δ] ∈ Tmp , [A → αY · β, {x}] ∈ kernel(Q)) do
                    update (x ≐ Γ) to (x ≐ Γ ∪ Δ) in Eqs;
                τ(P, Y) ⟵ Q;
            else
                                /* Generate the Y-target of P */
                foreach [A → αY · β, Δ] ∈ Tmp do
                    x ⟵ newVar();
                    Vars ⟵ Vars ∪ {x};
                    enqueue (x ≐ Δ) into Eqs;
                    replace [A → αY · β, Δ] by [A → αY · β, {x}] in Tmp;
                τ(P, Y) ⟵ closure_1(Tmp);
                add τ(P, Y) as an unmarked state to 𝒬 ;
```

Algorithm 5: Reduced system of equations *REqs* for the variables in $RVars \subseteq Vars$

```
inizialize RVars and REqs to ∅ ;
while Eqs not empty do
    x ≐ Δ ⟵ dequeue(Eqs) ;
    if Δ \ {x} = {x'} then
        class(x) ⟵ class(x') ;
    else
        class(x) ⟵ x ;
        add x to RVars ;
foreach x ∈ RVars such that x ≐ Δ ∈ Eqs do
    update each x' in Δ to class(x') ;
    add x ≐ Δ \ {x} to REqs ;
```

Algorithm 6: Computation of the actual values of variables

```
foreach x do
    D(x) ⟵ 0 ;
foreach x in RVars do
    if D(x) = 0 then
        traverse(x) ;

where

function traverse(x)
    push x onto stack S ;
    depth ⟵ number of elements in S ;
    D(x) ⟵ depth ;
    val(x) ⟵ init(x) ;
    foreach x' such that there is an edge in G from x to x' do
        if D(x') = 0 then
            traverse(x')
        D(x) ⟵ min(D(x), D(x')) ;
        val(x) ⟵ val(x) ∪ val(x') ;
    if D(x) = depth then
        repeat
            D(top(S)) ⟵ ∞ ;
            val(top(S)) ⟵ val(x) ;
        until pop(S) = x;
```