deterministic context free languages

how to generate parse trees (derivation trees)

- what cf languages are recognized by a *deterministic* pushdown automaton (dpda)?
- hopefully C0 is an example
- if true we can constrcut derivation tree for program of length n in time $O(n)$
- construction of such trees was left out in I2OS lectures
- following Michael Sipser: Introduction to the Theory of Computation, 3rd edition, CENGAGE learning 2013

MIT textbook

1 Derivation trees, derivations and reductions

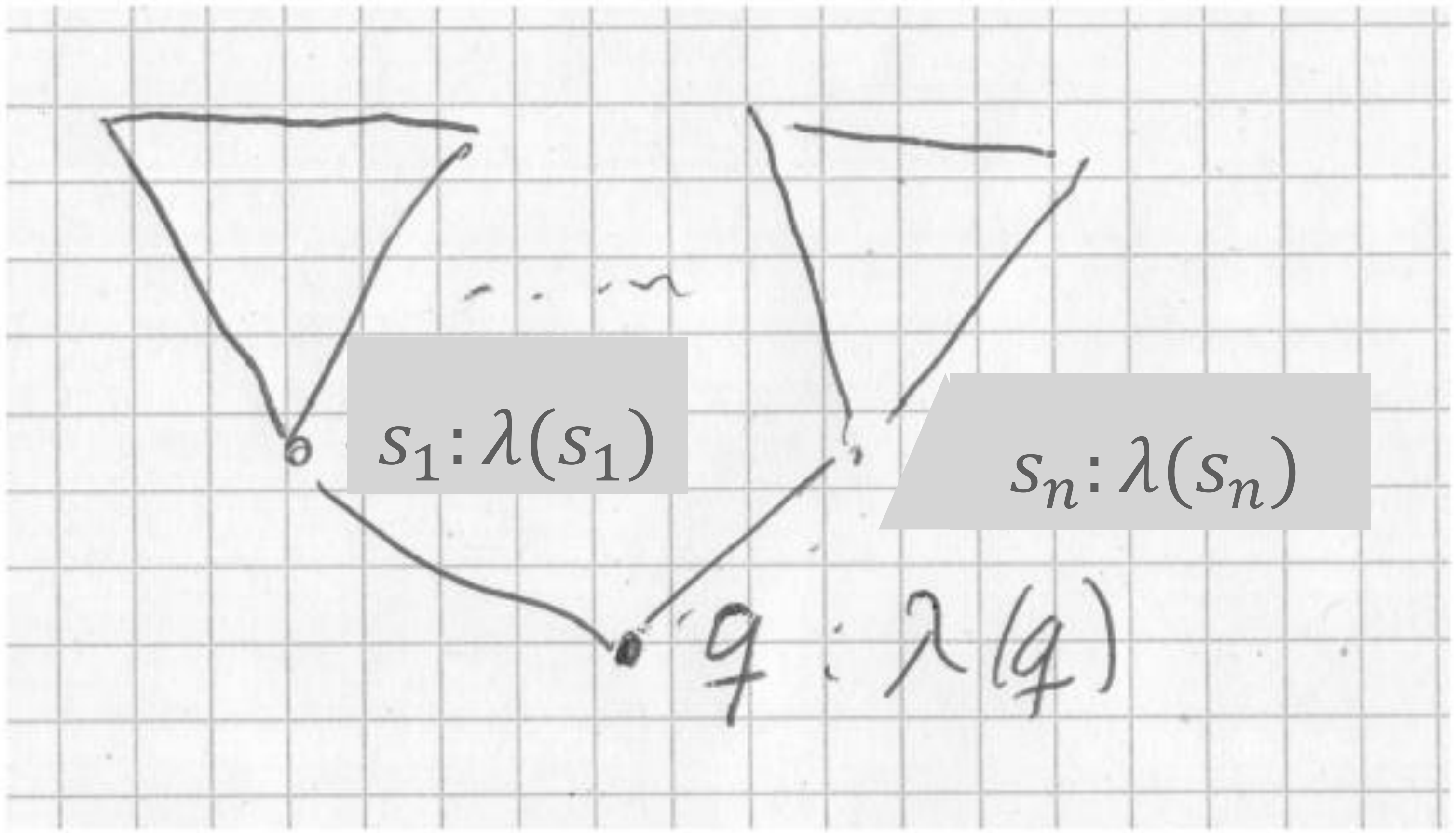


Figure 1: nodes x and their labels λ in a derivation tree.

reminder: derivation trees

- grammar

$$G = (N, T, P, S_1)$$

- tree (formally tree regions), nodes q with labels $\lambda(q) \in N \cup T$
- if node q has from left to right sons $s_1, \ldots, s_n$, then

$$\lambda(q) \rightarrow \lambda(s_1) \ldots \lambda(s_n) \quad \text{in } P$$

1 Derivation trees, derivations and reductions

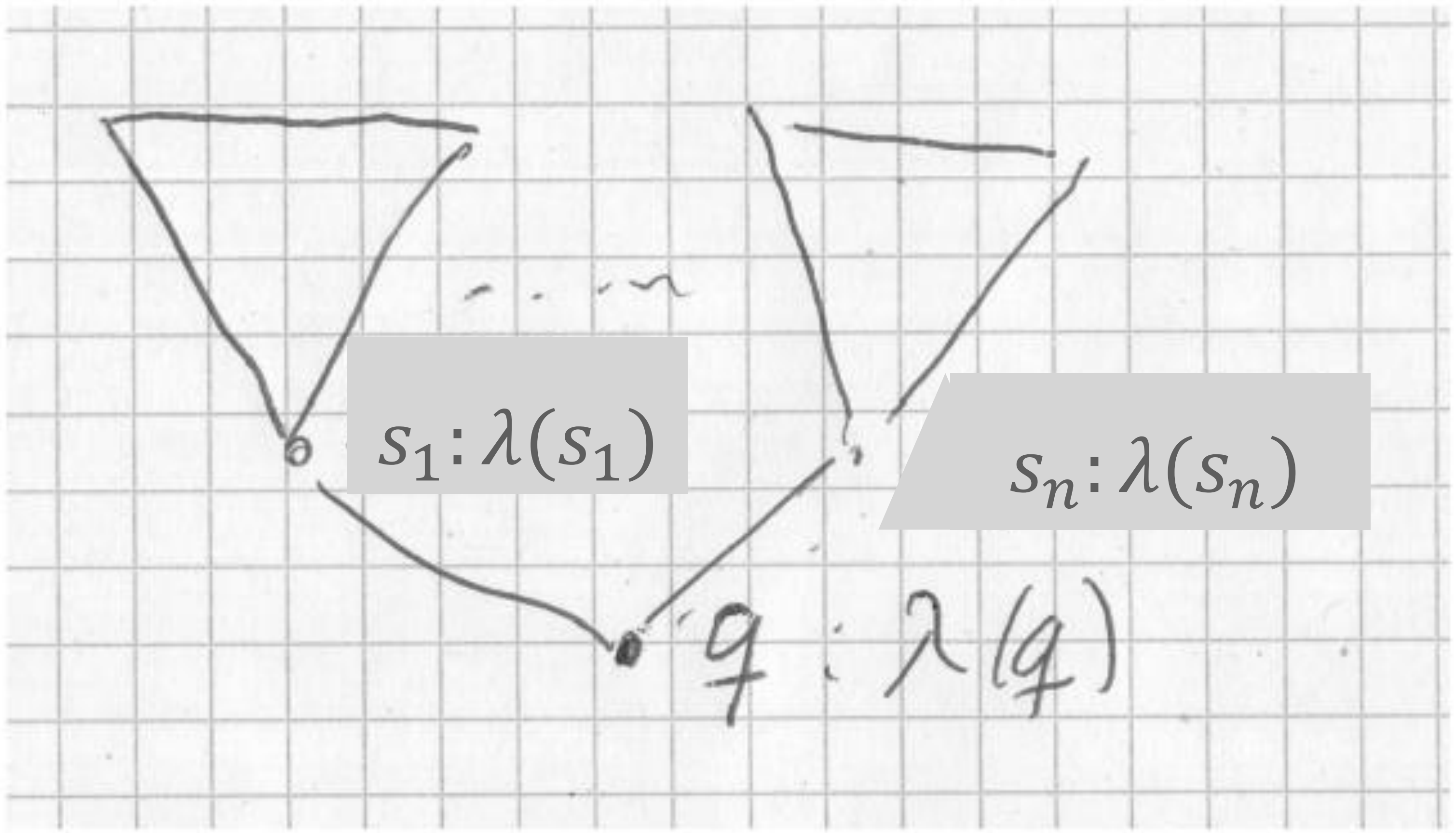


Figure 1: nodes x and their labels λ in a derivation tree.

reminder: derivation trees

- grammar

$$G = (N, T, P, S_1)$$

- tree (formally tree regions), nodes q with labels $\lambda(q) \in N \cup T$
- if node q has from left to right sons $s_1, \ldots, s_n$, then

$$\lambda(q) \to \lambda(s_1) \ldots \lambda(s_n) \quad \text{in } P$$

def: derivations

- For $w, w' \in (N \cup T)^*$ we define $w \to w'$ (w' is directly derived from w) if there are u, v, C, r with

$$w = uCv \ , \ w' = urv \ , \ C \to r \text{ in } P$$

- sequence of words

$$(w_1, \ldots, w_n)$$

is *derivation* if

$$w_i \to w_{i+1} \quad \text{for all } i < n$$

1 Derivation trees, derivations and reductions

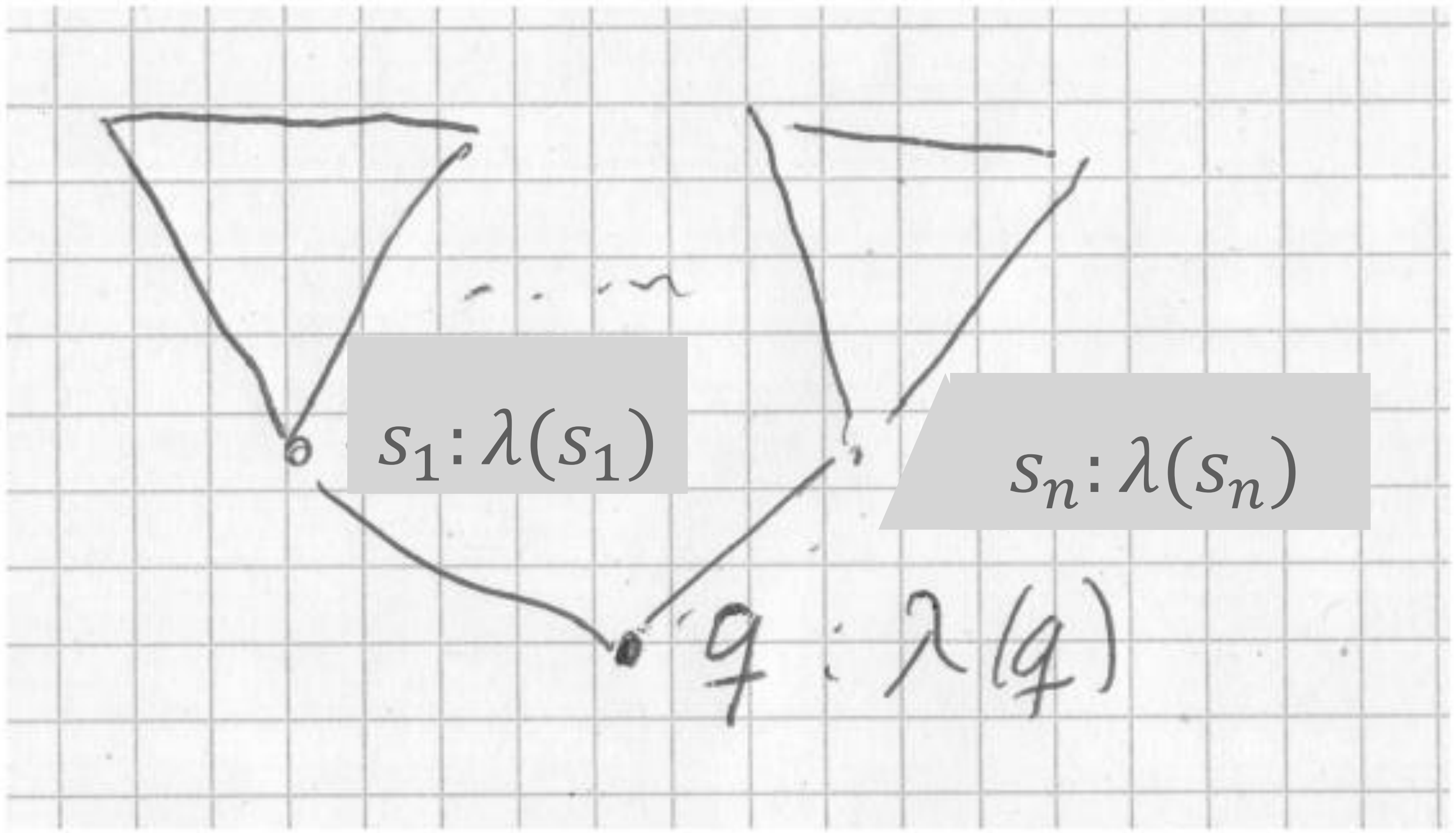


Figure 1: nodes x and their labels λ in a derivation tree.

reminder: derivation trees

- grammar
$$G = (N, T, P, S_1)$$
- tree (formally tree regions), nodes q with labels $\lambda(q) \in N \cup T$
- if node q has from left to right sons $s_1, \ldots, s_n$, then
$$\lambda(q) \to \lambda(s_1) \ldots \lambda(s_n) \quad \text{in } P$$

def: derivations

- For $w, w' \in (N \cup T)^*$ we define $w \to w'$ (w' is directly derived from w) if there are u, v, C, r with
$$w = uCv \ , \ w' = urv \ , \ C \to r \text{ in } P$$
- sequence of words
$$(w_1, \ldots, w_n)$$
is *derivation* if
$$w_i \to w_{i+1} \quad \text{for all } i < n$$
- it is a *rightmost derivation* if for all i in step $w_i \to w_{i+1}$ the rightmost non-terminal in w_i is replaced.

- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

preorder traversal: recursively treat

- sons of v, then v

postorder traversal: recursively treat

- v, then sons of v

- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

preorder traversal: recursively treat

- sons of v, then v

postorder traversal: recursively treat

- v, then sons of v

- example grammar for derivation tree in figure 2

$$\begin{aligned} F &\rightarrow V \mid -_1 F \mid (E) \\ T &\rightarrow F \mid T \cdot F \mid T/F \\ E &\rightarrow T \mid E+T \mid E-T \end{aligned}$$

- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

preorder traversal: recursively treat

- sons of v, then v

postorder traversal: recursively treat

- v, then sons of v

- example grammar for derivation tree in figure 2

$$
\begin{aligned}
F &\rightarrow V \mid -_1 F \mid (E) \\
T &\rightarrow F \mid T \cdot F \mid T/F \\
E &\rightarrow T \mid E+T \mid E-T
\end{aligned}
$$

have you seen this grammar?

where?

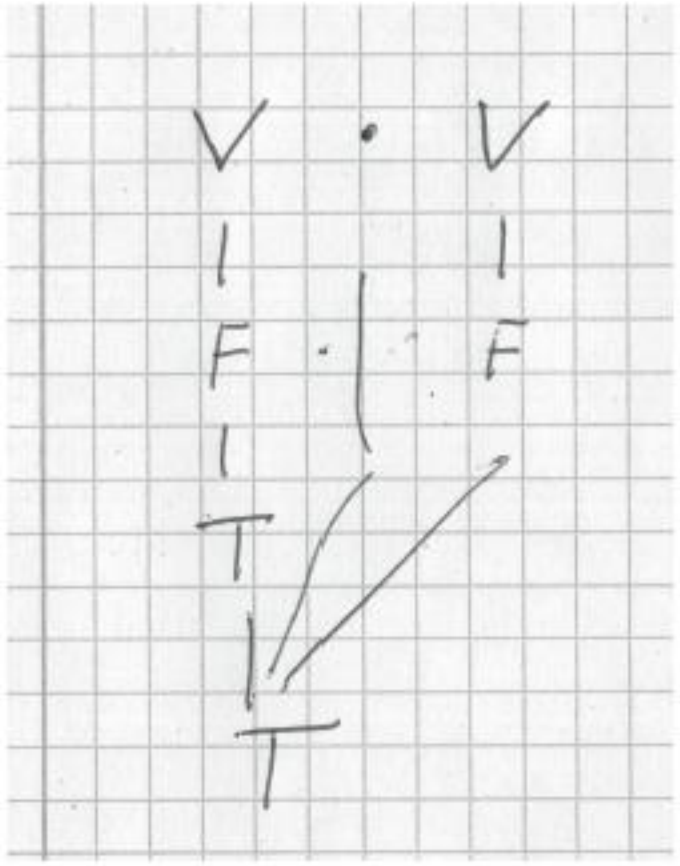
- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

- there is 1-1 relationship between rightmost derivations and derivation trees

exercise: make this statement precise and prove it

- example grammar for derivation tree in figure 2

$$
\begin{aligned}
F &\rightarrow V \mid -_1 F \mid (E) \\
T &\rightarrow F \mid T \cdot F \mid T/F \\
E &\rightarrow T \mid E+T \mid E-T
\end{aligned}
$$



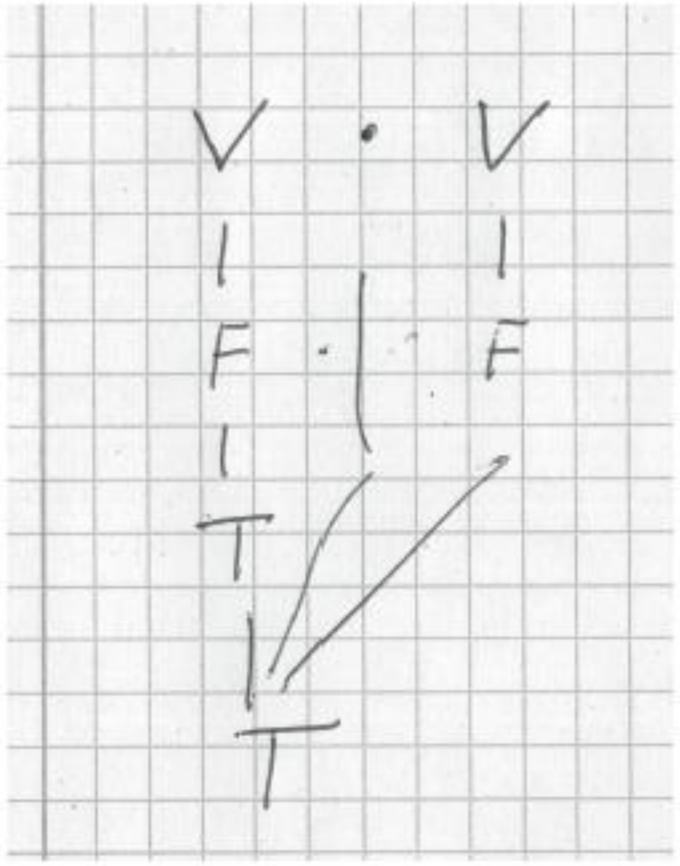
$$T \;,\; T \cdot F \;,\; T \cdot V \;,\; F \cdot V \;,\; V \cdot V$$

always replacing rightmost non terminal

- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

- example grammar for derivation tree in figure 2

$$
\begin{array}{rcl}
F & \to & V \mid -_1 F \mid (E) \\
T & \to & F \mid T \cdot F \mid T/F \\
E & \to & T \mid E+T \mid E-T
\end{array}
$$



$$T\ ,\ T\cdot F\ ,\ T\cdot V\ ,\ F\cdot V\ ,\ V\cdot V$$

always replacing rightmost non terminal

- there is 1-1 relationship between rightmost derivations and derivation trees

exercise: make this statement precise and prove it

def: reductions and valid strings

- For $w, w' \in (N \cup T)^*$ we define $w' \leftarrow w$ (w' is directly reducible to w) if $w \to w'$.

- sequence of words

$$(w_1, \ldots, w_n)$$

is *reduction* if

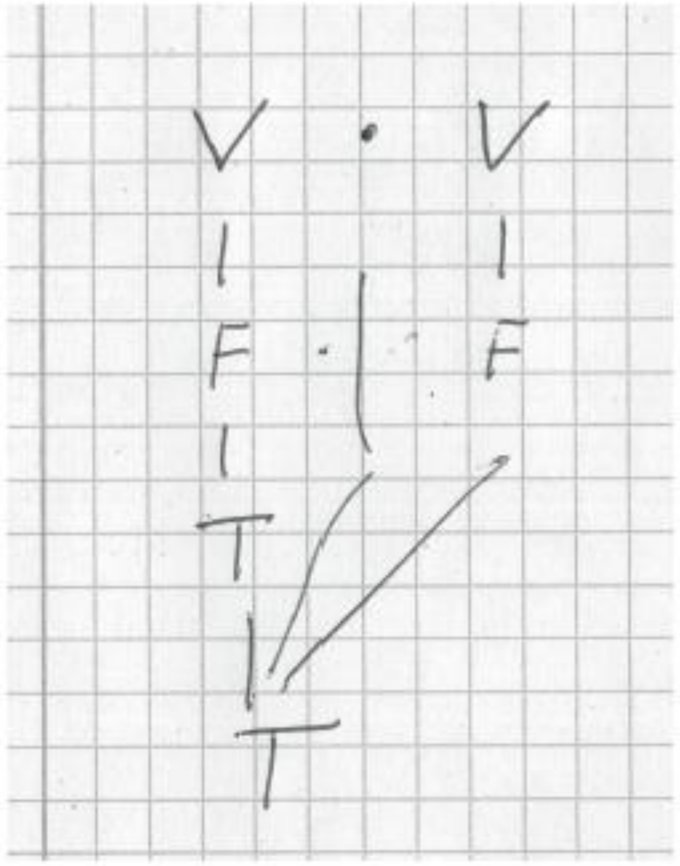
$$w_i \leftarrow w_{i+1} \quad \text{for all } i < n$$

it's just a derivation run backwards

- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

- example grammar for derivation tree in figure 2

$$\begin{array}{lcl} F & \to & V \mid -_1 F \mid (E) \\ T & \to & F \mid T \cdot F \mid T/F \\ E & \to & T \mid E+T \mid E-T \end{array}$$



$$T \;,\; T\cdot F \;,\; T\cdot V \;,\; F \cdot V \;,\; V \cdot V$$

always replacing rightmost non terminal

- there is 1-1 relationship between rightmost derivations and derivation trees

exercise: make this statement precise and prove it

def: reductions and valid strings

- For $w, w' \in (N \cup T)^*$ we define $w' \leftarrow w$ (w' is directly reducible to w) if $w \to w'$.
- sequence of words

$$(w_1, \dots, w_n)$$

is *reduction* if

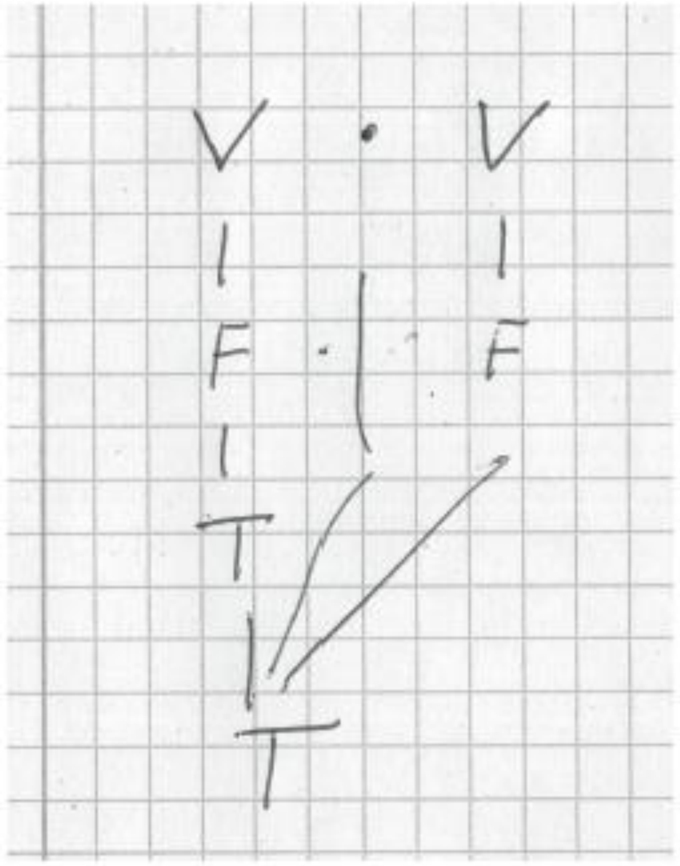
$$w_i \leftarrow w_{i+1} \quad \text{for all } i < n$$

- it is a *leftmost reduction* if for all i in step $w_i \leftarrow w_{i+1}$ the leftmost possible string r in w_{i+1} is replaced by a nonterminal C.
- leftmost reductions are rightmost derivations run backwards
- generating a leftmost reduction from a derivation tree by postorder traversal of trees, subtrees from left to right.

- generating a rightmost derivation from a derivation tree by preorder traversal of trees, subtrees from right to left.

- example grammar for derivation tree in figure 2

$$\begin{array}{rcl} F & \to & V \mid -_1 F \mid (E) \\ T & \to & F \mid T \cdot F \mid T/F \\ E & \to & T \mid E+T \mid E-T \end{array}$$



$$T \;,\; T\cdot F \;,\; T\cdot V \;,\; F\cdot V \;,\; V\cdot V$$

always replacing rightmost non terminal

- there is 1-1 relationship between rightmost derivations and derivation trees

exercise: make this statement precise and prove it

def: reductions and valid strings

- For $w, w' \in (N \cup T)^*$ we define $w' \leftarrow w$ (w' is directly reducible to w) if $w \to w'$.
- sequence of words
$$(w_1, \ldots, w_n)$$
is *reduction* if
$$w_i \leftarrow w_{i+1} \quad \text{for all } i < n$$

- it is a *leftmost reduction* if for all i in step $w_i \leftarrow w_{i+1}$ the leftmost possible string r in w_{i+1} is replaced by a nonterminal C.
- leftmost reductions are rightmost derivations run backwards
- generating a leftmost reduction from a derivation tree by postorder traversal of trees, subtrees from left to right.
- a string $v \in (N \cup T)^*$ is *valid* if it can occur in a leftmost reduction of a terminal string.

2 Handles and deterministic cfg's

def: handles A *handle* of a valid string v is a pair $(n, B \to h)$ specifies the first step in a leftmost reduction of v

- h occurs in v behind position n, i.e. there are x, y such that

$$v = xhy \quad \text{and} \quad n = |x|$$

- the production

$$B \to h \quad \text{in} \quad P$$

  is used

definitions in literature amazingly imprecise

2 Handles and deterministic cfg's

def: handles A *handle* of a valid string v is a pair $(n, B \rightarrow h)$ specifies the first step in a leftmost reduction of v

- h occurs in v behind position n, i.e. there are x, y such that

$$v = xhy \quad \text{and} \quad n = |x|$$

- the production

$$B \rightarrow h \quad \text{in} \quad P$$

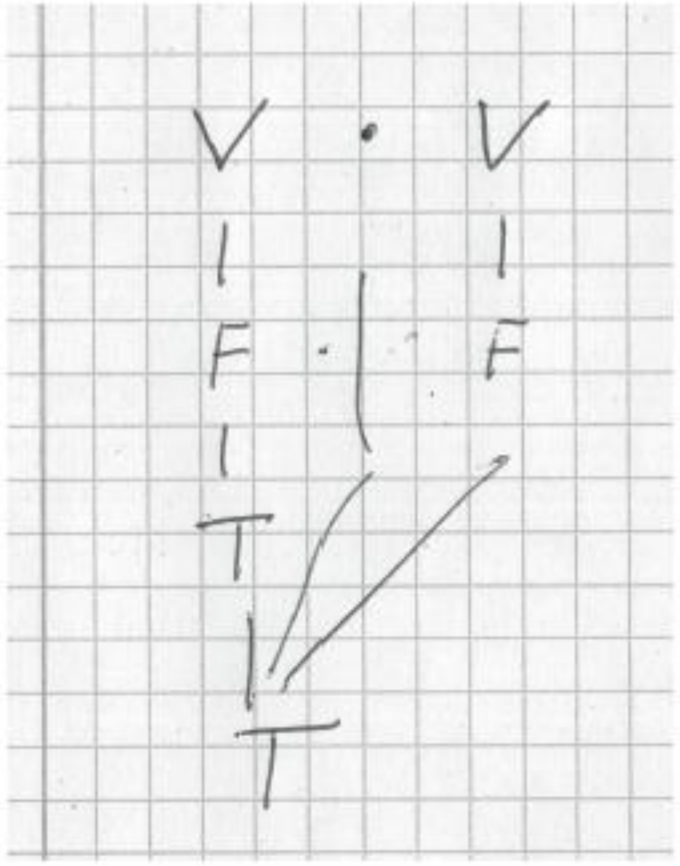
  is used

- note (proof exercise)

$$y \in T^*$$

- if the production is clear we only specify h
- in examples we specify n by underlining h

$$\underline{V} \cdot V \ , \ \underline{F} \cdot V \ , \ T \cdot \underline{V} \ , \ \underline{T \cdot F} \ , \ T$$



2 Handles and deterministic cfg's

def: handles A *handle* of a valid string v is a pair $(n, B \to h)$ specifies the first step in a leftmost reduction of v

- h occurs in v behind position n, i.e. there are x, y such that

$$v = xhy \quad \text{and} \quad n = |x|$$

- the production

$$B \to h \quad \text{in} \quad P$$

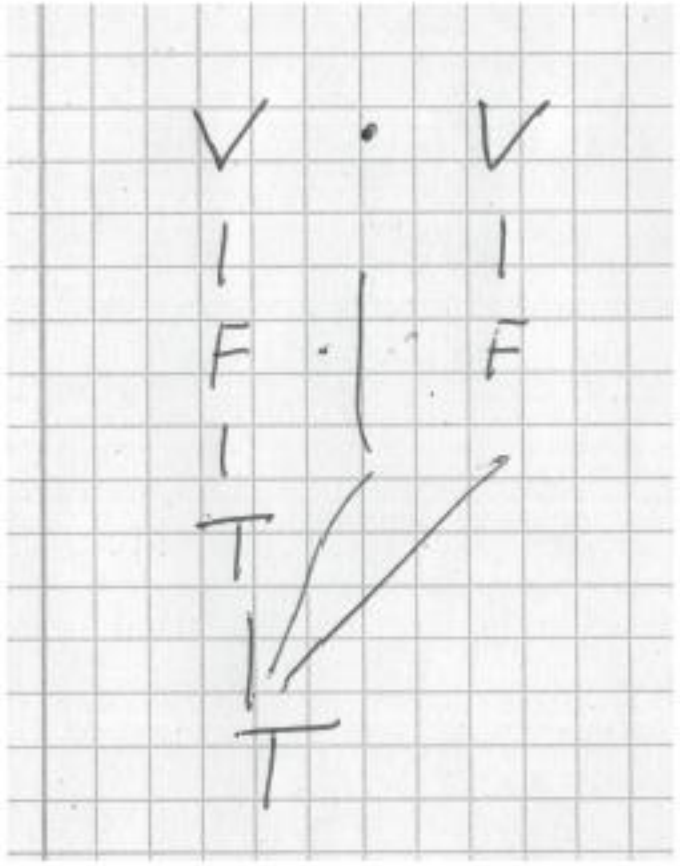
  is used

- note (proof exercise)

$$y \in T^*$$

- if the production is clear we only specify h
- in examples we specify n by underlining h

$$\underline{V} \cdot V \ , \ \underline{F} \cdot V \ , \ T \cdot \underline{V} \ , \ \underline{T \cdot F} \ , \ T$$



if we can identify handles, we can construct reductions and hence derivation trees

Lemma 1. *In unambiguous grammars handles for valid strings v are unique*

Proof. derivation trees are unique □

2 Handles and deterministic cfg's

def: handles A *handle* of a valid string v is a pair $(n, B \rightarrow h)$ specifies the first step in a leftmost reduction of v

- h occurs in v behind position n, i.e. there are x, y such that

$$v = xhy \quad \text{and} \quad n = |x|$$

- the production

$$B \rightarrow h \quad \text{in} \quad P$$

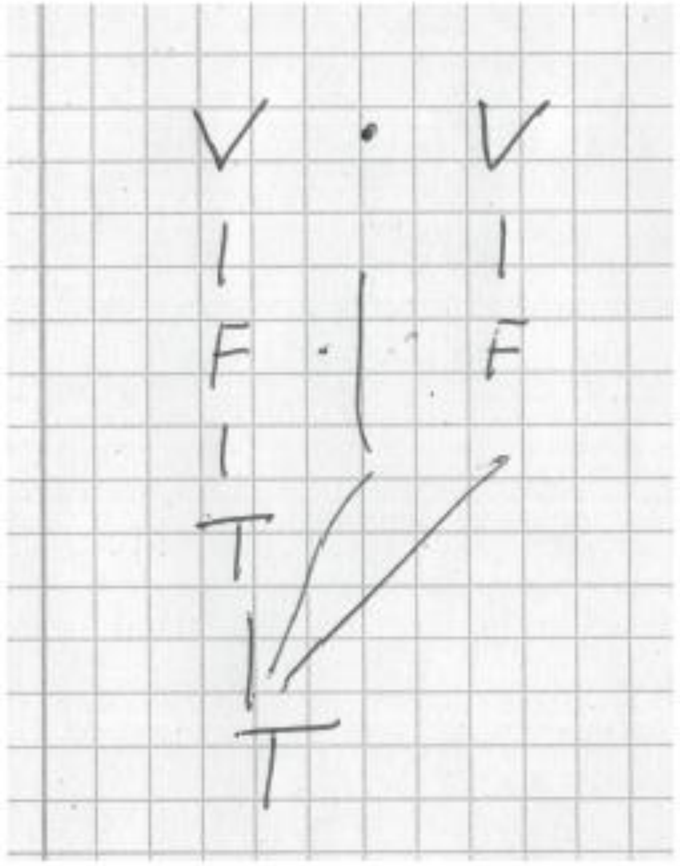
  is used

- note (proof exercise)

$$y \in T^*$$

- if the production is clear we only specify h
- in examples we specify n by underlining h

$$\underline{V} \cdot V \;,\; \underline{F} \cdot V \;,\; T \cdot \underline{V} \;,\; \underline{T \cdot F} \;,\; T$$



if we can identify handles, we can construct reductions and hence derivation trees

Lemma 1. *In unambiguous grammars handles for valid strings v are unique*

Proof. derivation trees are unique $\square$

is this grammar unambiguous

why or why not?

$$\begin{aligned} F &\rightarrow V \mid -_1 F \mid (E) \\ T &\rightarrow F \mid T \cdot F \mid T / F \\ E &\rightarrow T \mid E + T \mid E - T \end{aligned}$$

2 Handles and deterministic cfg's

recognizing unique handles might require large lookahead

$$\begin{array}{rcl} R & \rightarrow & S \mid T \\ S & \rightarrow & aSb \mid ab \\ T & \rightarrow & aTbb \mid abb \end{array}$$

$$aa\underline{ab}bb \rightarrow a\underline{aSb}b \rightarrow \underline{aSb} \rightarrow \underline{S} \rightarrow R$$

$$aa\underline{abb}bbbb \rightarrow a\underline{aTbb}bb \rightarrow \underline{aTbb} \rightarrow \underline{T} \rightarrow R$$

2 Handles and deterministic cfg's

recognizing unique handles might require large lookahead

$$
\begin{aligned}
R &\rightarrow S \mid T \\
S &\rightarrow aSb \mid ab \\
T &\rightarrow aTbb \mid abb
\end{aligned}
$$

$$aa\underline{ab}bb \rightarrow a\underline{aSb}b \rightarrow \underline{aSb} \rightarrow \underline{S} \rightarrow R$$

$$aa\underline{abb}bbbb \rightarrow a\underline{aTbb}bb \rightarrow \underline{aTbb} \rightarrow \underline{T} \rightarrow R$$

recognising a handle once one sees it (dpda-like) A handle $(|x|, h)$ of a valid string $v = xhy$ is called a *forced handle* if is the unique handle of each string

$$x h \hat{y} \quad \text{with} \quad \hat{y} \in T^*$$

def: deterministic cfg A cfg is *deterministic* if every valid string has a forced handle.

3 Testing if a cfg is deterministic

The (amazing) DK-test: for every cfg there is a fa DK which identifies handles. In particular DK accepts input z if

- z is prefix of a valid string $v = zy$
- $z = xh$ ends with a handle h of v

3 Testing if a cfg is deterministic

The (amazing) DK-test: for every cfg there is a fa DK which identifies handles.
In particular DK accepts input z if

- z is prefix of a valid string $v = zy$
- $z = xh$ ends with a handle h of v

3.1 The nondeterministic K-automaton

Construction of dfa DK from an nfa K by applying the power set construction.

running example

$$
\begin{array}{rcl}
S & \to & E \dashv \\
E & \to & E + T \mid T \\
T & \to & T \times a \mid a
\end{array}
$$

3 Testing if a cfg is deterministic

The (amazing) DK-test: for every cfg there is a fa DK which identifies handles. In particular DK accepts input z if

- z is prefix of a valid string $v = zy$
- $z = xh$ ends with a handle h of v

3.1 The nondeterministic K-automaton

Construction of dfa DK from an nfa K by applying the power set construction.

running example

$$\begin{array}{rcl} S & \rightarrow & E \dashv \\ E & \rightarrow & E + T \mid T \\ T & \rightarrow & T \times a \mid a \end{array}$$

states: using items of a cfg :

- constructed from productions in P. For each

$$B \rightarrow u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lcll} \langle B & \rightarrow & .u_1 \dots u_k \rangle & \text{(before processing)} \\ & \dots & & \\ \langle B & \rightarrow & u_1 \dots u_i.u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i) \\ & \dots & & \\ \langle B & \rightarrow & u_1 \dots u_k. \rangle & \text{(after processing)} \end{array}$$

3 Testing if a cfg is deterministic

The (amazing) DK-test: for every cfg there is a fa DK which identifies handles. In particular DK accepts input z if

- z is prefix of a valid string $v = zy$
- $z = xh$ ends with a handle h of v

3.1 The nondeterministic K-automaton

Construction of dfa DK from an nfa K by applying the power set construction.

running example

$$\begin{aligned} S &\to E \dashv \\ E &\to E + T \mid T \\ T &\to T \times a \mid a \end{aligned}$$

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{llll} \langle B & \to & .u_1 \dots u_k \rangle & \text{(before processing)} \\ & \dots & & \\ \langle B & \to & u_1 \dots u_i . u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i) \\ & \dots & & \\ \langle B & \to & u_1 \dots u_k . \rangle & \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

3 Testing if a cfg is deterministic

The (amazing) DK-test: for every cfg there is a fa DK which identifies handles. In particular DK accepts input z if

- z is prefix of a valid string $v = zy$
- $z = xh$ ends with a handle h of v

3.1 The nondeterministic K-automaton

Construction of dfa DK from an nfa K by applying the power set construction.

running example

$$\begin{array}{lcl} S & \to & E \dashv \\ E & \to & E+T \mid T \\ T & \to & T \times a \mid a \end{array}$$

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lcll} \langle B & \to & .u_1 \dots u_k \rangle & \text{(before processing)} \\ & \dots & & \\ \langle B & \to & u_1 \dots u_i.u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i) \\ & \dots & & \\ \langle B & \to & u_1 \dots u_k. \rangle & \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_1 \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \rangle$$

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{aligned}\langle B &\to .u_1 \dots u_k\rangle && \text{(before processing)}\\ &\dots\\ \langle B &\to u_1 \dots u_i.u_{i+1} \dots u_k\rangle && \text{(after processig until } u_i)\\ &\dots\\ \langle B &\to u_1 \dots u_k.\rangle && \text{(after processing)}\end{aligned}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av\rangle \xrightarrow{a} \langle B \to ua.v\rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv\rangle \xrightarrow{\varepsilon} \langle C \to .r\rangle$$

starting From start state z_0 and all production $S_\perp \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_\perp \to .u\rangle$$

$$\begin{aligned}S &\to E \dashv\\ E &\to E + T \mid T\\ T &\to T \times a \mid a\end{aligned}$$

$$\langle S \to .E \dashv\rangle \xrightarrow{E} \langle S \to E.\dashv\rangle \xrightarrow{\dashv} \langle S \to E \dashv.\rangle$$

$$\langle E \to .E + T\rangle \xrightarrow{E} \langle E \to E.+T\rangle \xrightarrow{+} \langle E \to E+.T\rangle \xrightarrow{T} \langle E \to E + T.\rangle$$

$$\langle E \to .T\rangle \xrightarrow{T} \langle E \to T.\rangle$$

$$\langle T \to .T \times a\rangle \xrightarrow{T} \langle T \to T.\times a\rangle \xrightarrow{\times} \langle T \to T \times.a\rangle \xrightarrow{a} \langle T \to T \times a.\rangle$$

$$\langle T \to .a\rangle \xrightarrow{a} \langle T \to a.\rangle$$

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lcll}
\langle B & \to & .u_1 \dots u_k \rangle & \text{(before processing)} \\
 & \dots & & \\
\langle B & \to & u_1 \dots u_i . u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i\text{)} \\
 & \dots & & \\
\langle B & \to & u_1 \dots u_k . \rangle & \text{(after processing)}
\end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_1 \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \rangle$$

$$\begin{array}{lcl}
S & \to & E \dashv \\
E & \to & E + T \mid T \\
T & \to & T \times a \mid a
\end{array}$$

$$\langle S \to .E \dashv \rangle \xrightarrow{E} \langle S \to E. \dashv \rangle \xrightarrow{\dashv} \langle S \to E \dashv . \rangle$$

$$\langle E \to .E + T \rangle \xrightarrow{E} \langle E \to E. + T \rangle \xrightarrow{+} \langle E \to E + .T \rangle \xrightarrow{T} \langle E \to E + T. \rangle$$

$$\langle E \to .T \rangle \xrightarrow{T} \langle E \to T. \rangle$$

$$\langle T \to .T \times a \rangle \xrightarrow{T} \langle T \to T. \times a \rangle \xrightarrow{\times} \langle T \to T \times .a \rangle \xrightarrow{a} \langle T \to T \times a. \rangle$$

$$\langle T \to .a \rangle \xrightarrow{a} \langle T \to a. \rangle$$

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lll} \langle B & \to & .u_1 \dots u_k \rangle \quad \text{(before processing)} \\ & \dots & \\ \langle B & \to & u_1 \dots u_i . u_{i+1} \dots u_k \rangle \quad \text{(after processig until } u_i) \\ & \dots & \\ \langle B & \to & u_1 \dots u_k . \rangle \quad \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

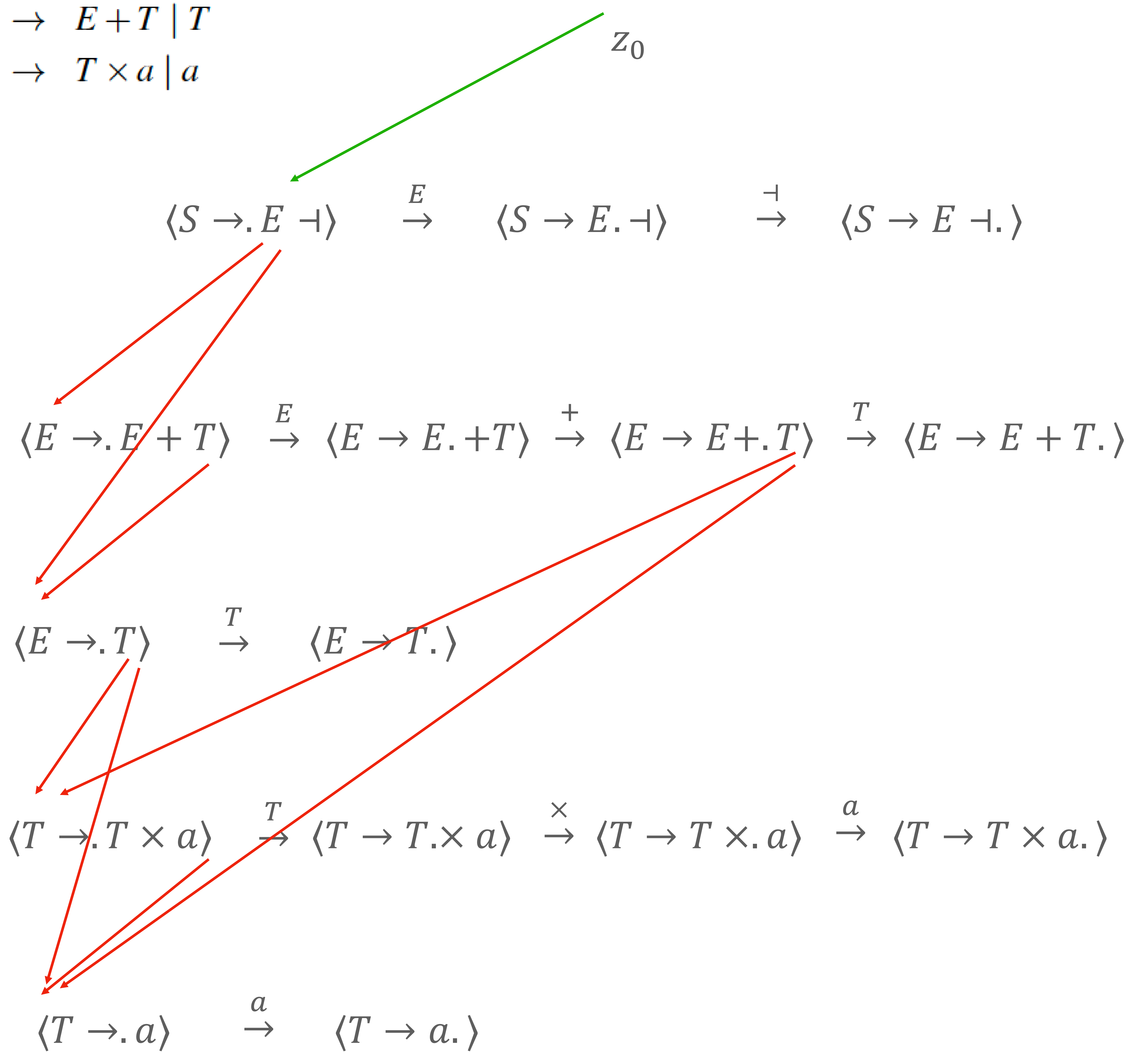
ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_\perp \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_\perp \to .u \rangle$$

$$\begin{array}{lll} S & \to & E \dashv \\ E & \to & E + T \mid T \\ T & \to & T \times a \mid a \end{array}$$



states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \ldots u_k$$

include *dotted rules*

$$\begin{array}{llll} \langle B & \to & .u_1 \ldots u_k \rangle & \text{(before processing)} \\ & \ldots & & \\ \langle B & \to & u_1 \ldots u_i . u_{i+1} \ldots u_k \rangle & \text{(after processig until } u_i\text{)} \\ & \ldots & & \\ \langle B & \to & u_1 \ldots u_k . \rangle & \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_\perp \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_\perp \to .u \rangle$$

$$\begin{array}{lll} S & \to & E \dashv \\ E & \to & E + T \mid T \\ T & \to & T \times a \mid a \end{array}$$

z_0

$$\langle S \to .E \dashv \rangle \xrightarrow{E} \langle S \to E. \dashv \rangle \xrightarrow{\dashv} \langle S \to E \dashv . \rangle$$

$$\langle E \to .E + T \rangle \xrightarrow{E} \langle E \to E.+T \rangle \xrightarrow{+} \langle E \to E+.T \rangle \xrightarrow{T} \langle E \to E + T. \rangle$$

$$\langle E \to .T \rangle \xrightarrow{T} \langle E \to T. \rangle$$

$$\langle T \to .T \times a \rangle \xrightarrow{T} \langle T \to T.\times a \rangle \xrightarrow{\times} \langle T \to T \times .a \rangle \xrightarrow{a} \langle T \to T \times a. \rangle$$

$$\langle T \to .a \rangle \xrightarrow{a} \langle T \to a. \rangle$$

accepting end states: states

$$\langle B \to u. \rangle$$

corresponding to a completed production

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lcll} \langle B & \to & .u_1 \dots u_k \rangle & \text{(before processing)} \\ & \dots & & \\ \langle B & \to & u_1 \dots u_i . u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i) \\ & \dots & & \\ \langle B & \to & u_1 \dots u_k . \rangle & \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_1 \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \rangle$$

$$\begin{array}{lcl} S & \to & E \dashv \\ E & \to & E + T \mid T \\ T & \to & T \times a \mid a \end{array}$$

z_0

$\langle S \to .E \dashv \rangle \xrightarrow{E} \langle S \to E.\dashv \rangle \xrightarrow{\dashv} \underline{\langle S \to E \dashv . \rangle}$

$\langle E \to .E + T \rangle \xrightarrow{E} \langle E \to E.+T \rangle \xrightarrow{+} \langle E \to E+.T \rangle \xrightarrow{T} \underline{\langle E \to E + T. \rangle}$

$\langle E \to .T \rangle \xrightarrow{T} \langle E \to T. \rangle$

$\langle T \to .T \times a \rangle \xrightarrow{T} \langle T \to T.\times a \rangle \xrightarrow{\times} \langle T \to T \times .a \rangle \xrightarrow{a} \underline{\langle T \to T \times a. \rangle}$

$\langle T \to .a \rangle \xrightarrow{a} \underline{\langle T \to a. \rangle}$

accepting end states: states

$$\langle B \to u. \rangle$$

corresponding to a completed production

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lll} \langle B & \to & .u_1 \dots u_k\rangle \quad \text{(before processing)} \\ & \dots & \\ \langle B & \to & u_1 \dots u_i.u_{i+1} \dots u_k\rangle \quad \text{(after processig until } u_i) \\ & \dots & \\ \langle B & \to & u_1 \dots u_k.\rangle \quad \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av\rangle \xrightarrow{a} \langle B \to ua.v\rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv\rangle \xrightarrow{\varepsilon} \langle C \to .r\rangle$$

starting From start state z_0 and all production $S_{\perp} \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_{\perp} \to .u\rangle$$

accepting end states: states

$$\langle B \to u.\rangle$$

corresponding to a completed production

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{lll} \langle B & \to & .u_1 \dots u_k\rangle \quad \text{(before processing)} \\ & \dots & \\ \langle B & \to & u_1 \dots u_i.u_{i+1} \dots u_k\rangle \quad \text{(after processig until } u_i) \\ & \dots & \\ \langle B & \to & u_1 \dots u_k.\rangle \quad \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av\rangle \xrightarrow{a} \langle B \to ua.v\rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv\rangle \xrightarrow{\varepsilon} \langle C \to .r\rangle$$

starting From start state z_0 and all production $S_\perp \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_\perp \to .u\rangle$$

accepting end states: states

$$\langle B \to u.\rangle$$

corresponding to a completed production

K recognizes handles:

Lemma 2. *K started with input z can reach state $\langle T \to u.v\rangle$ iff*

- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{llll} \langle B & \to & .u_1 \dots u_k \rangle & \text{(before processing)} \\ & \dots & & \\ \langle B & \to & u_1 \dots u_i.u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i) \\ & \dots & & \\ \langle B & \to & u_1 \dots u_k. \rangle & \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_1 \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \rangle$$

accepting end states: states

$$\langle B \to u. \rangle$$

corresponding to a completed production

K recognizes handles:

Lemma 2. *K started with input z can reach state $\langle T \to u.v \rangle$ iff*

- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

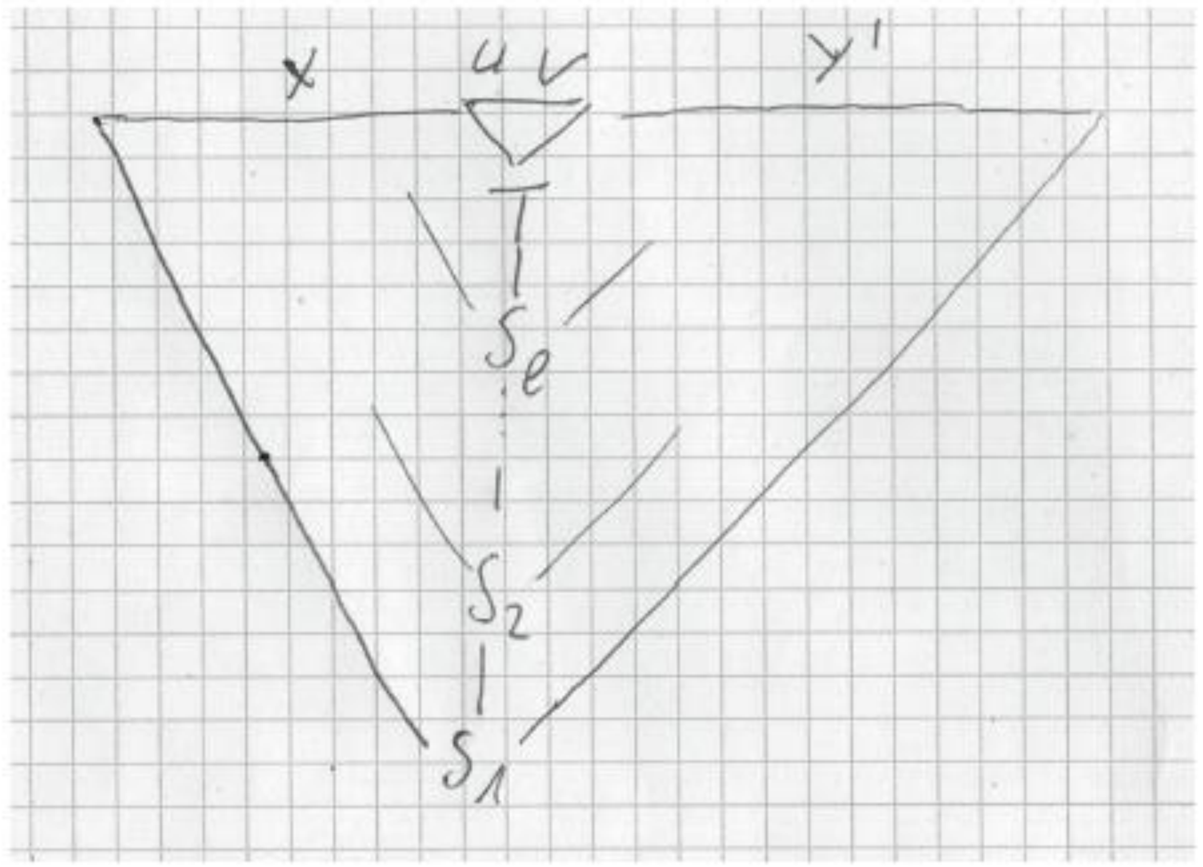
$\to$**:**

- Partition path from z_0 to $\langle T \to u.v \rangle$ into *runs* of shift moves separated by ε-moves.
- run i belongs to a production $S_i \to u_i S_{i+1} v_i$
- last production $T \to uv$
- production before last: $S_\ell \to u_\ell S_\ell v_\ell$
- input z processed so far:

$$z = u_1 \dots u_\ell u$$

set

$$y' = v_\ell \dots v_1$$



states: using items of a cfg :

- constructed from productions in P. For each

$$B \to u_1 \dots u_k$$

include *dotted rules*

$$\begin{array}{llll} \langle B & \to & .u_1 \dots u_k \rangle & \text{(before processing)} \\ & \dots & & \\ \langle B & \to & u_1 \dots u_i.u_{i+1} \dots u_k \rangle & \text{(after processig until } u_i) \\ & \dots & & \\ \langle B & \to & u_1 \dots u_k. \rangle & \text{(after processing)} \end{array}$$

shift moves for $a \in N \cup T$ and every production $B \to uav$ transition

$$\langle B \to u.av \rangle \xrightarrow{a} \langle B \to ua.v \rangle$$

ε moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \rangle \xrightarrow{\varepsilon} \langle C \to .r \rangle$$

starting From start state z_0 and all production $S_1 \to u$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \rangle$$

accepting end states: states

$$\langle B \to u. \rangle$$

corresponding to a completed production

K recognizes handles:

Lemma 2. *K started with input z can reach state $\langle T \to u.v \rangle$ iff*

- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

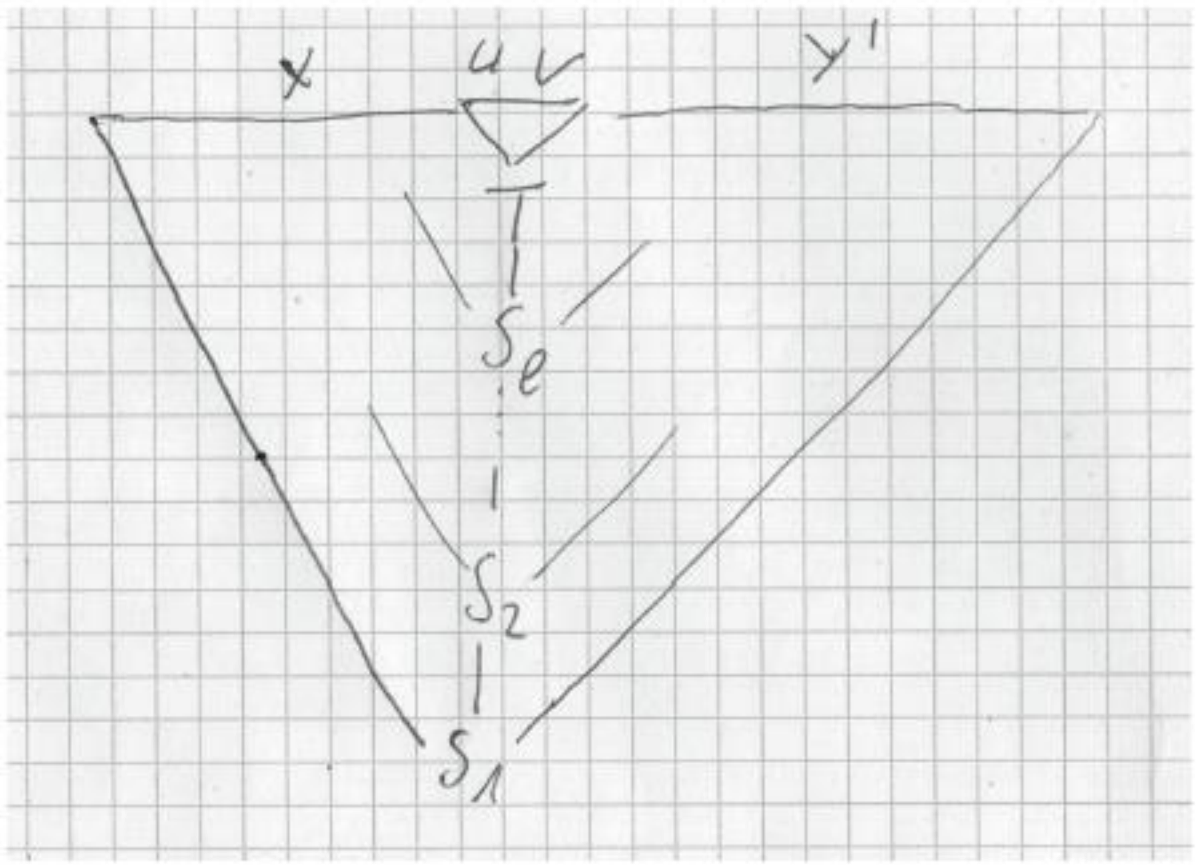
$\to$:

- Partition path from z_0 to $\langle T \to u.v \rangle$ into *runs* of shift moves separated by ε-moves.
- run i belongs to a production $S_i \to u_i S_{i+1} v_i$
- last production $T \to uv$
- production before last: $S_\ell \to u_\ell S_\ell v_\ell$
- input z processed so far:

$$z = u_1 \dots u_\ell u$$

set

$$y' = v_\ell \dots v_1$$



- then $S_1 \to^* xuvy'$, i.e. $xuvy'$ derivable in G by derivation tree of figure 3 and construction of automaton.

Lemma 2. *K started with input z can reach state $\langle T \to u.v \rangle$ iff*

- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

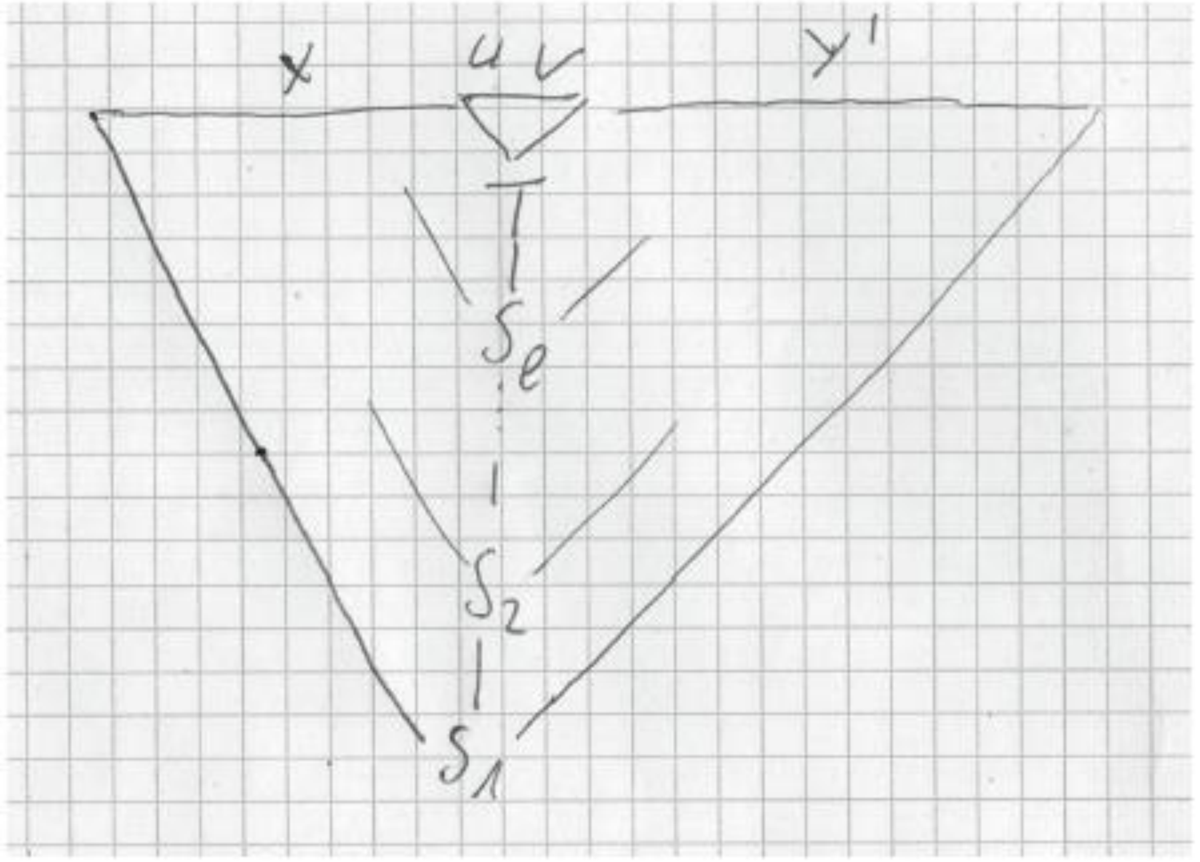
$\to$:

- Partition path from z_0 to $\langle T \to u.v \rangle$ into *runs* of shift moves separated by ε-moves.
- run i belongs to a production $S_i \to u_i S_{i+1} v_i$
- last production $T \to uv$
- production before last: $S_\ell \to u_\ell \overset{T}{\cancel{S}}_\ell v_\ell$
- input z processed so far:

$$z = u_1 \dots u_\ell u$$

set

$$y' = v_\ell \dots v_1$$



- then $S_1 \to^* xuvy'$, i.e. $xuvy'$ derivable in G by derivation tree of figure 3 and construction of automaton.

Lemma 2. *K started with input z can reach state $\langle T \to u.v \rangle$ iff*

- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

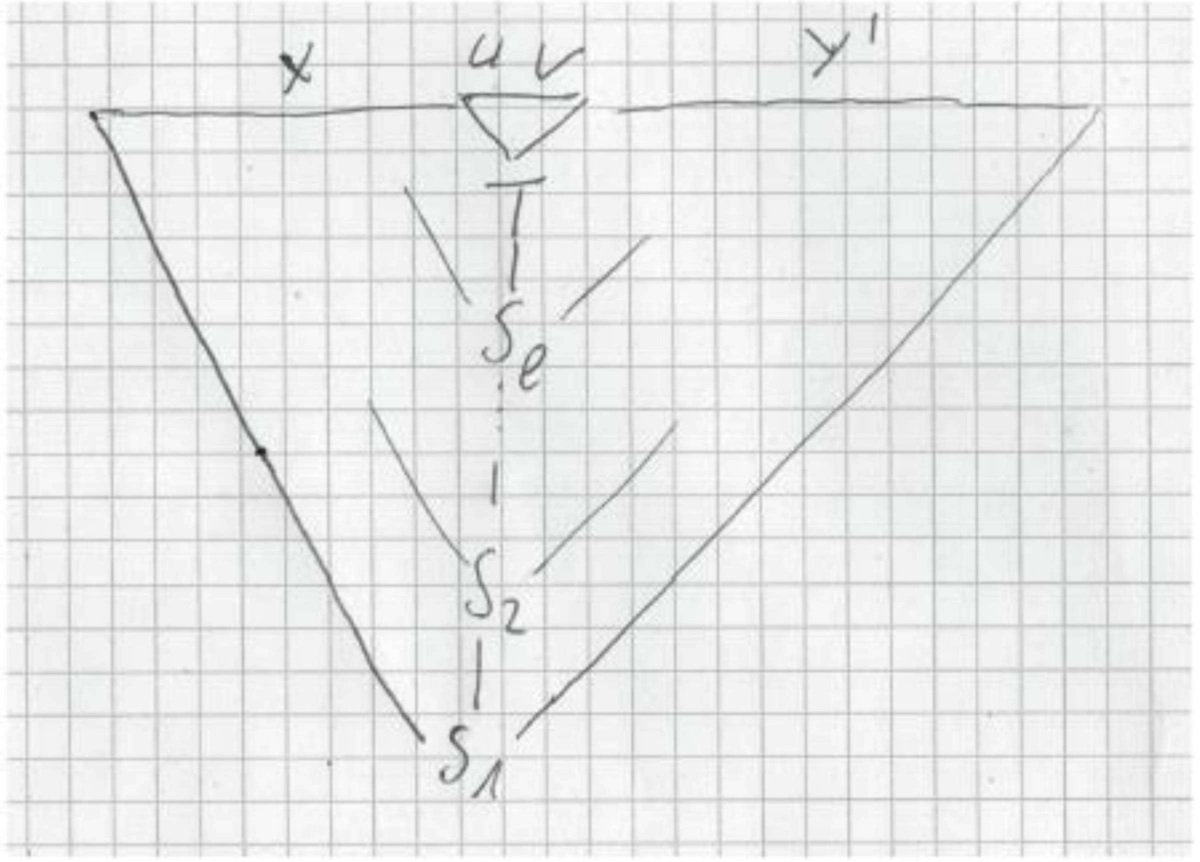
$\to$:

- Partition path from z_0 to $\langle T \to u.v \rangle$ into *runs* of shift moves separated by ε-moves.
- run i belongs to a production $S_i \to u_i S_{i+1} v_i$
- last production $T \to uv$
- production before last: $S_\ell \to u_\ell S_\ell v_\ell$
- input z processed so far:

$$z = u_1 \dots u_\ell u$$

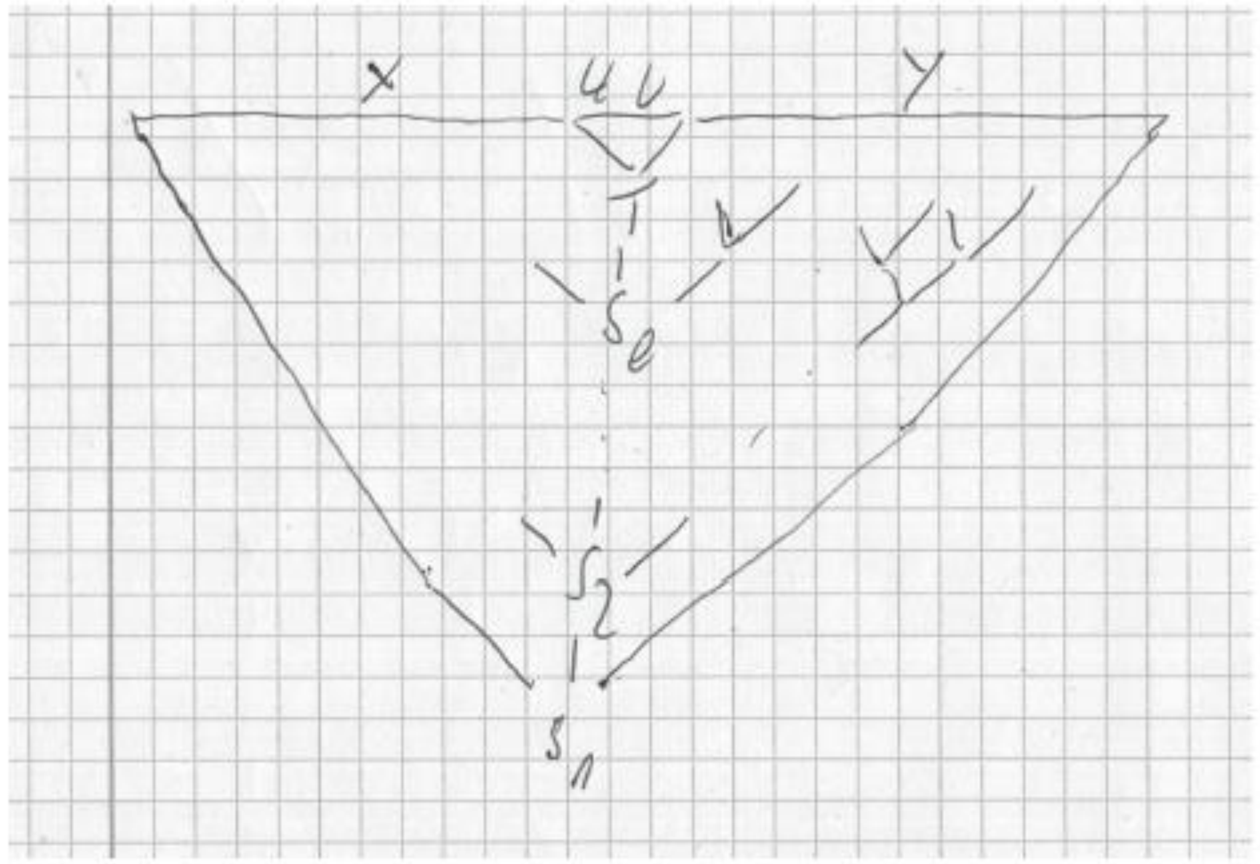
set

$$y' = v_\ell \dots v_1$$



- then $S_1 \to^* xuvy'$, i.e. $xuvy'$ derivable in G by derivation tree of figure 3 and construction of automaton.

- obtain y by expanding all nonterminals in y' to terminal strings with derivation tree of figure 4 (w.l.o.g there are no redundant nonterminals, which cannot be expanded to terminal string).



Lemma 2. *K started with input z can reach state $\langle T \to u.v\rangle$ iff*

- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

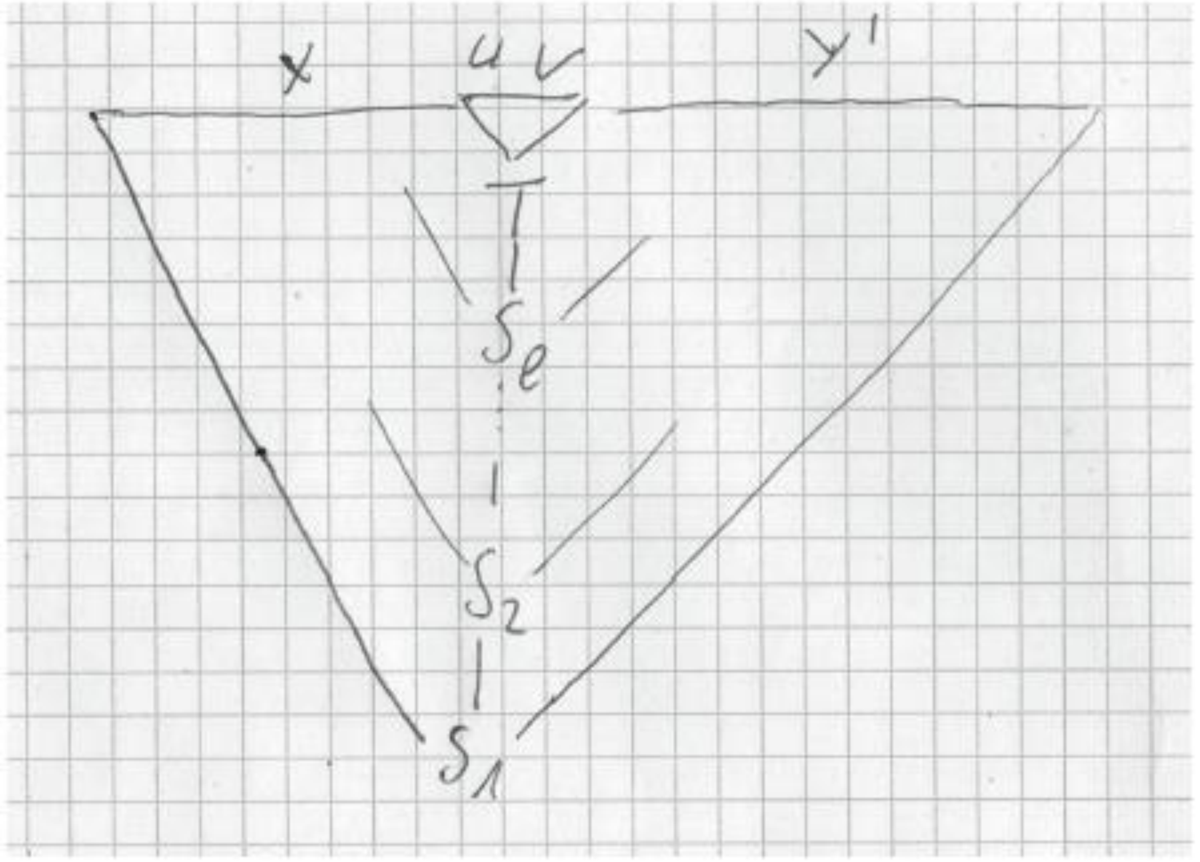
$\to$:

- Partition path from z_0 to $\langle T \to u.v\rangle$ into *runs* of shift moves separated by ε-moves.
- run i belongs to a production $S_i \to u_i S_{i+1} v_i$
- last production $T \to uv$
- production before last: $S_\ell \to u_\ell \overset{T}{S}_\ell v_\ell$
- input z processed so far:

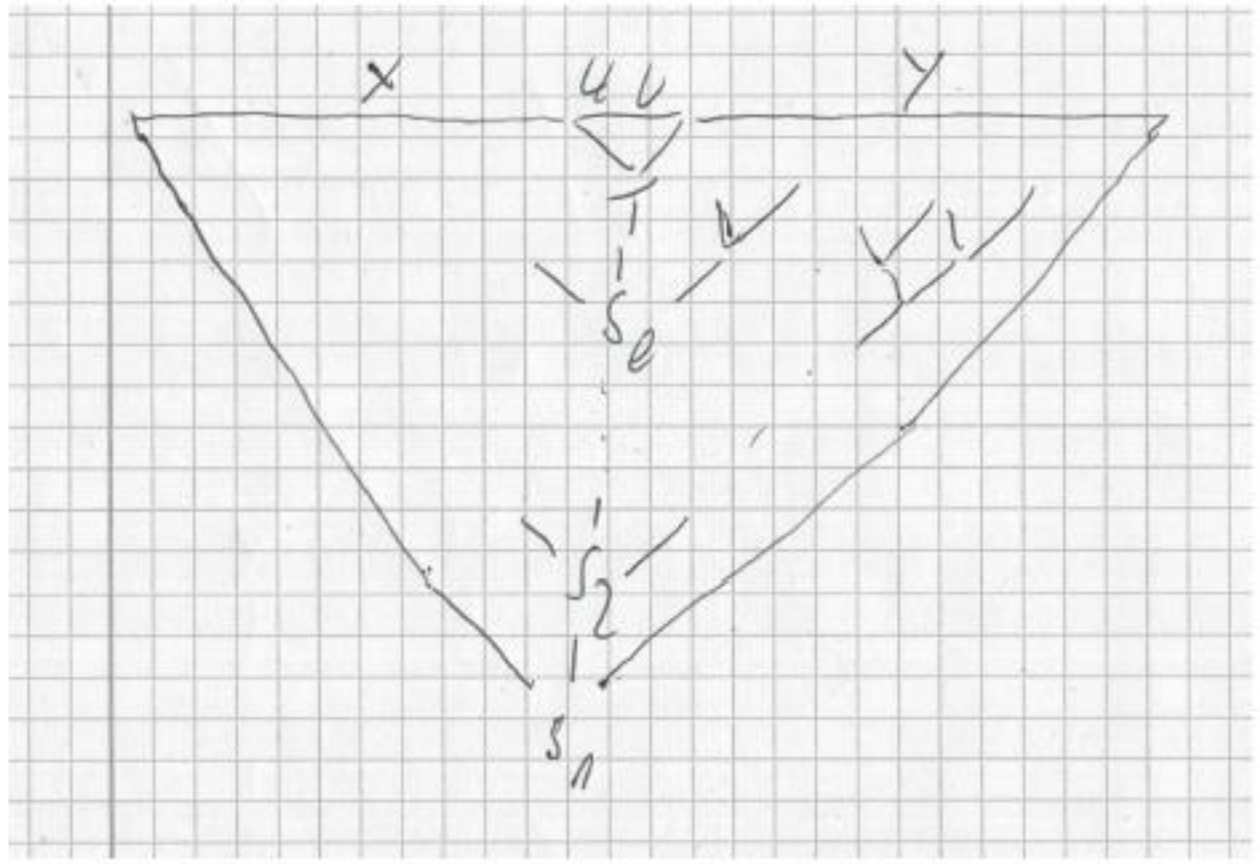
$$z = u_1 \dots u_\ell u$$

set

$$y' = v_\ell \dots v_1$$



- then $S_1 \to^* xuvy'$, i.e. $xuvy'$ derivable in G by derivation tree of figure 3 and construction of automaton.
- obtain y by expanding all nonterminals in y' to terminal strings with derivation tree of figure 4 (w.l.o.g there are no redundant nonterminals, which cannot be expanded to terminal string).



- turn tree into leftmost reduction.
- $xuvy$ is valid, $(|x|, T \to uv)$ is handle

Lemma 2. *K started with input z can reach state $\langle T \rightarrow u.v \rangle$ iff*

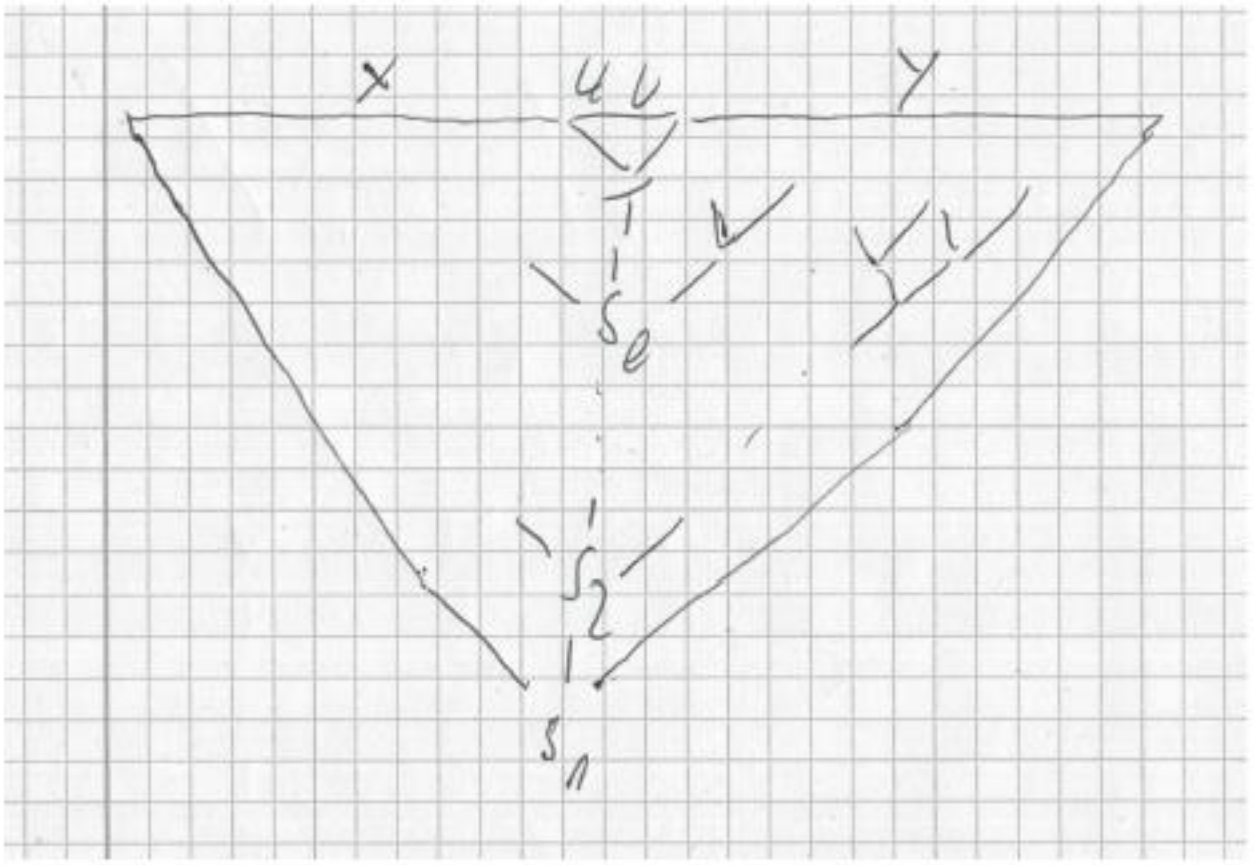
- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \rightarrow uv)$*

$\leftarrow$:

- derivation tree of $xuvy$ with handle $(|x|, T \rightarrow uv)$ has path

$$(T, S_\ell, \ldots, S_1)$$

as shown in figure 4.



Lemma 2. *K started with input z can reach state $\langle T \to u.v \rangle$ iff*

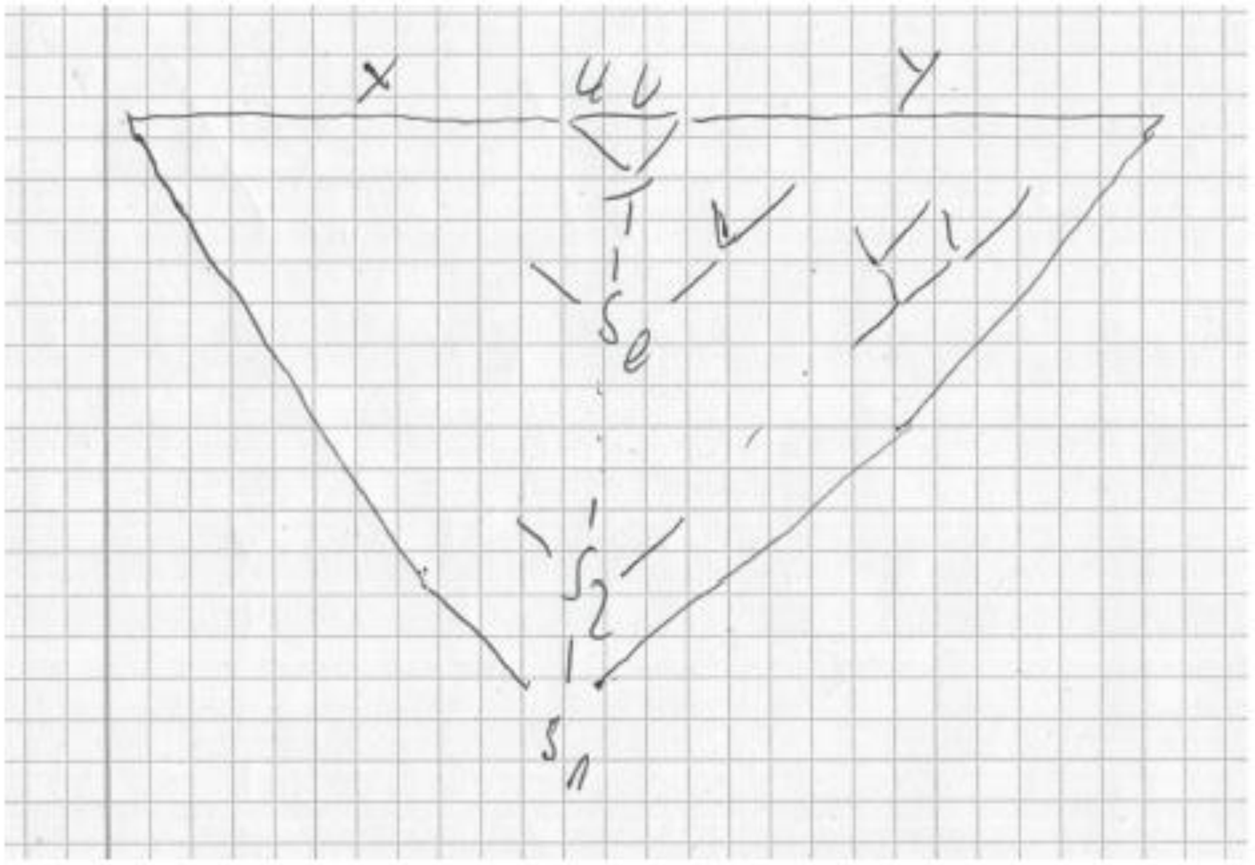
- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

$\leftarrow$:

- derivation tree of $xuvy$ with handle $(|x|, T \to uv)$ has path

$$(T, S_\ell, \ldots, S_1)$$

as shown in figure 4.



- portion x left of this path is directly derived from the S_i; otherwise $(|x|, T \to uv)$ is not handle.

$$\begin{aligned} S_i &\to u_i S_{i+1} v_i \quad \text{for } i < \ell \\ S_\ell &\to u_\ell T v_\ell \\ T &\to uv \end{aligned}$$

Lemma 2. *K started with input z can reach state $\langle T \to u.v\rangle$ iff*

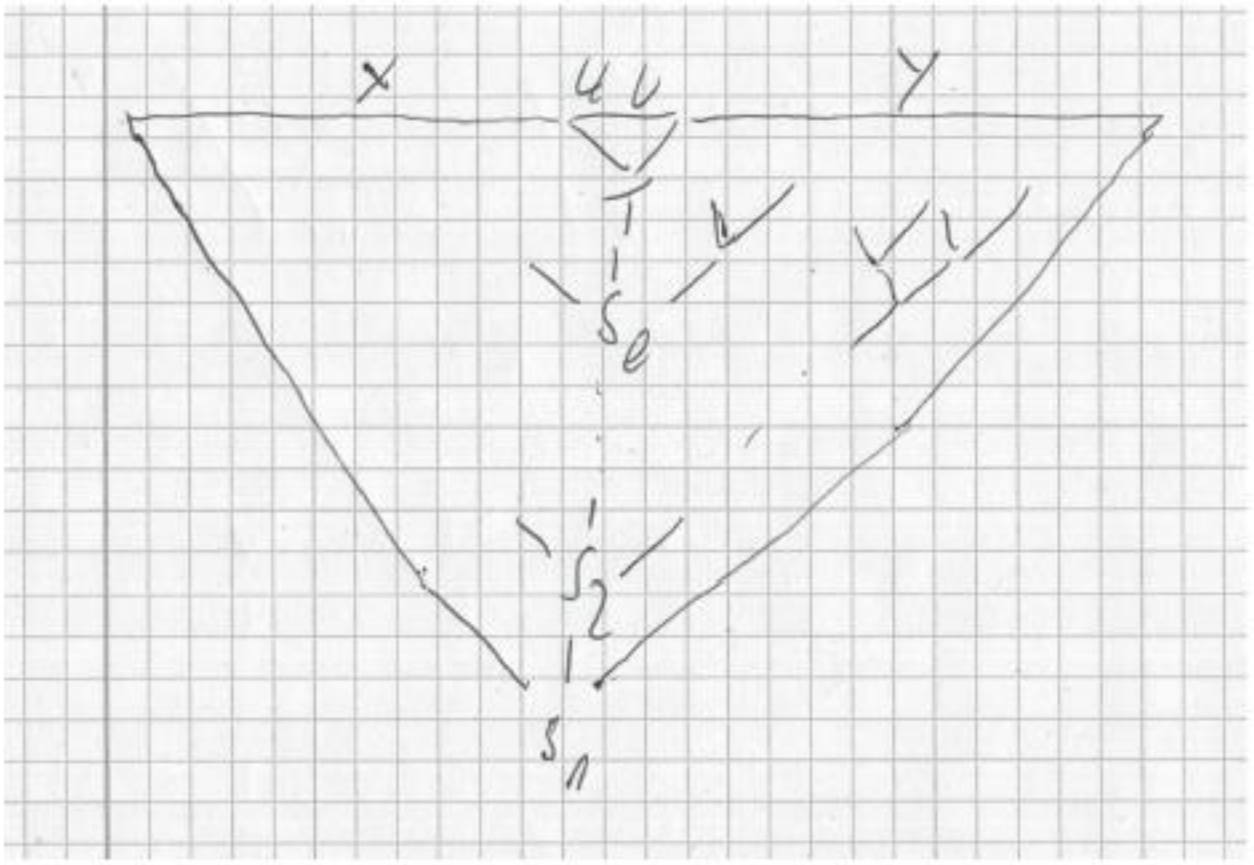
- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

$\leftarrow$:

- derivation tree of $xuvy$ with handle $(|x|, T \to uv)$ has path

$$(T, S_\ell, \ldots, S_1)$$

as shown in figure 4.



- portion x left of this path is directly derived from the S_i; otherwise $(|x|, T \to uv)$ is not handle.

$$\begin{aligned} S_i &\to u_i S_{i+1} v_i \quad \text{for } i < \ell \\ S_\ell &\to u_\ell T v_\ell \\ T &\to uv \end{aligned}$$

- path of K with input $z = xu$

$$\begin{aligned} z_0 &\xrightarrow{\varepsilon}{}^* \langle S_1 \to .u_1 S_2 v_1\rangle \\ &\xrightarrow{u_1}{}^* \langle S_1 \to u_1.S_2 v_1\rangle \\ &\xrightarrow{\varepsilon}{}^* \langle S_2 \to .u_2 S_3 v_2\rangle \\ &\xrightarrow{u_2}{}^* \langle S_2 \to u_2.S_3 v_2\rangle \\ &\ldots \\ &\xrightarrow{u_\ell}{}^* \langle S_\ell \to u_\ell.T v_\ell \\ &\xrightarrow{\varepsilon}{}^* \langle T \to .uv\rangle \\ &\xrightarrow{u}{}^* \langle T \to u.v\rangle \end{aligned}$$

Lemma 2. *K started with input z can reach state $\langle T \to u.v\rangle$ iff*

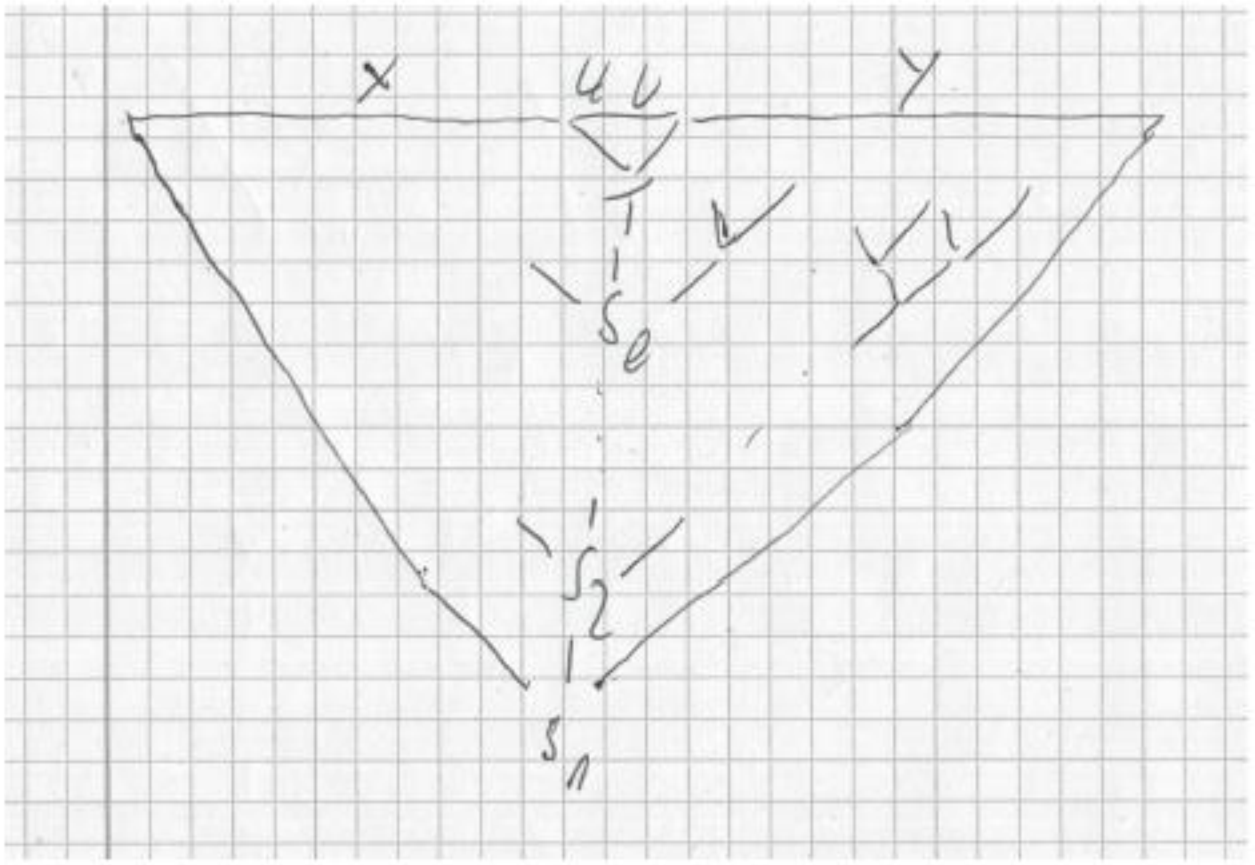
- *z ends with u, i.e. $z = xu$*
- *and there is $y \in T^*$ such that $xuvy$ is a valid string*
- *with handle $(|x|, T \to uv)$*

$\leftarrow$:

- derivation tree of $xuvy$ with handle $(|x|, T \to uv)$ has path

$$(T, S_\ell, \dots, S_1)$$

as shown in figure 4.



- portion x left of this path is directly derived from the S_i; otherwise $(|x|, T \to uv)$ is not handle.

$$\begin{aligned} S_i &\to u_i S_{i+1} v_i \quad \text{for } i < \ell \\ S_\ell &\to u_\ell T v_\ell \\ T &\to uv \end{aligned}$$

- path of K with input $z = xu$

$$\begin{aligned} z_0 &\xrightarrow{\varepsilon}{}^* \langle S_1 \to .u_1 S_2 v_1\rangle \\ &\xrightarrow{u_1}{}^* \langle S_1 \to u_1.S_2 v_1\rangle \\ &\xrightarrow{\varepsilon}{}^* \langle S_2 \to .u_2 S_3 v_2\rangle \\ &\xrightarrow{u_2}{}^* \langle S_2 \to u_2.S_3 v_2\rangle \\ &\dots \\ &\xrightarrow{u_\ell}{}^* \langle S_\ell \to u_\ell.T v_\ell \\ &\xrightarrow{\varepsilon}{}^* \langle T \to .uv\rangle \\ &\xrightarrow{u}{}^* \langle T \to u.v\rangle \end{aligned}$$

Lemma 3. *K started with z can reach state $\langle T \to h.\rangle$ iff $z = xh$ and $(|x|, T \to h)$ is handle off a valid string xhy.*

Proof. Lemma 2 with $u = h, v = \varepsilon$ and handle$(|x|, T \to h)$. $\square$

Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

$$\begin{aligned} S &\rightarrow E\dashv \\ E &\rightarrow E+T \mid T \\ T &\rightarrow T\times a \mid a \end{aligned}$$

z_0

$$\langle S \rightarrow .E\dashv\rangle \xrightarrow{E} \langle S \rightarrow E.\dashv\rangle \xrightarrow{\dashv} \underline{\langle S \rightarrow E\dashv.\rangle}$$

$$\langle E \rightarrow .E+T\rangle \xrightarrow{E} \langle E \rightarrow E.+T\rangle \xrightarrow{+} \langle E \rightarrow E+.T\rangle \xrightarrow{T} \underline{\langle E \rightarrow E+T.\rangle}$$

$$\langle E \rightarrow .T\rangle \xrightarrow{T} \langle E \rightarrow T.\rangle$$

$$\langle T \rightarrow .T\times a\rangle \xrightarrow{T} \langle T \rightarrow T.\times a\rangle \xrightarrow{\times} \langle T \rightarrow T\times .a\rangle \xrightarrow{a} \underline{\langle T \rightarrow T\times a.\rangle}$$

$$\langle T \rightarrow .a\rangle \xrightarrow{a} \underline{\langle T \rightarrow a.\rangle}$$

Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

z_0

$$\boxed{\begin{array}{l} \langle S \rightarrow . E \dashv \rangle \\ \langle E \rightarrow . E + T \rangle \\ \langle E \rightarrow . T \rangle \\ \langle T \rightarrow . T \times a \rangle \\ \langle T \rightarrow . a \rangle \end{array}}$$

you can read this off directly from the grammar

$$\begin{aligned} S &\rightarrow E \dashv \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T \times a \mid a \end{aligned}$$

z_0

$$\langle S \rightarrow . E \dashv \rangle \xrightarrow{E} \langle S \rightarrow E . \dashv \rangle \xrightarrow{\dashv} \underline{\langle S \rightarrow E \dashv . \rangle}$$

$$\langle E \rightarrow . E + T \rangle \xrightarrow{E} \langle E \rightarrow E . + T \rangle \xrightarrow{+} \langle E \rightarrow E + . T \rangle \xrightarrow{T} \underline{\langle E \rightarrow E + T . \rangle}$$

$$\langle E \rightarrow . T \rangle \xrightarrow{T} \langle E \rightarrow T . \rangle$$

$$\langle T \rightarrow . T \times a \rangle \xrightarrow{T} \langle T \rightarrow T . \times a \rangle \xrightarrow{\times} \langle T \rightarrow T \times . a \rangle \xrightarrow{a} \underline{\langle T \rightarrow T \times a . \rangle}$$

$$\langle T \rightarrow . a \rangle \xrightarrow{a} \underline{\langle T \rightarrow a . \rangle}$$

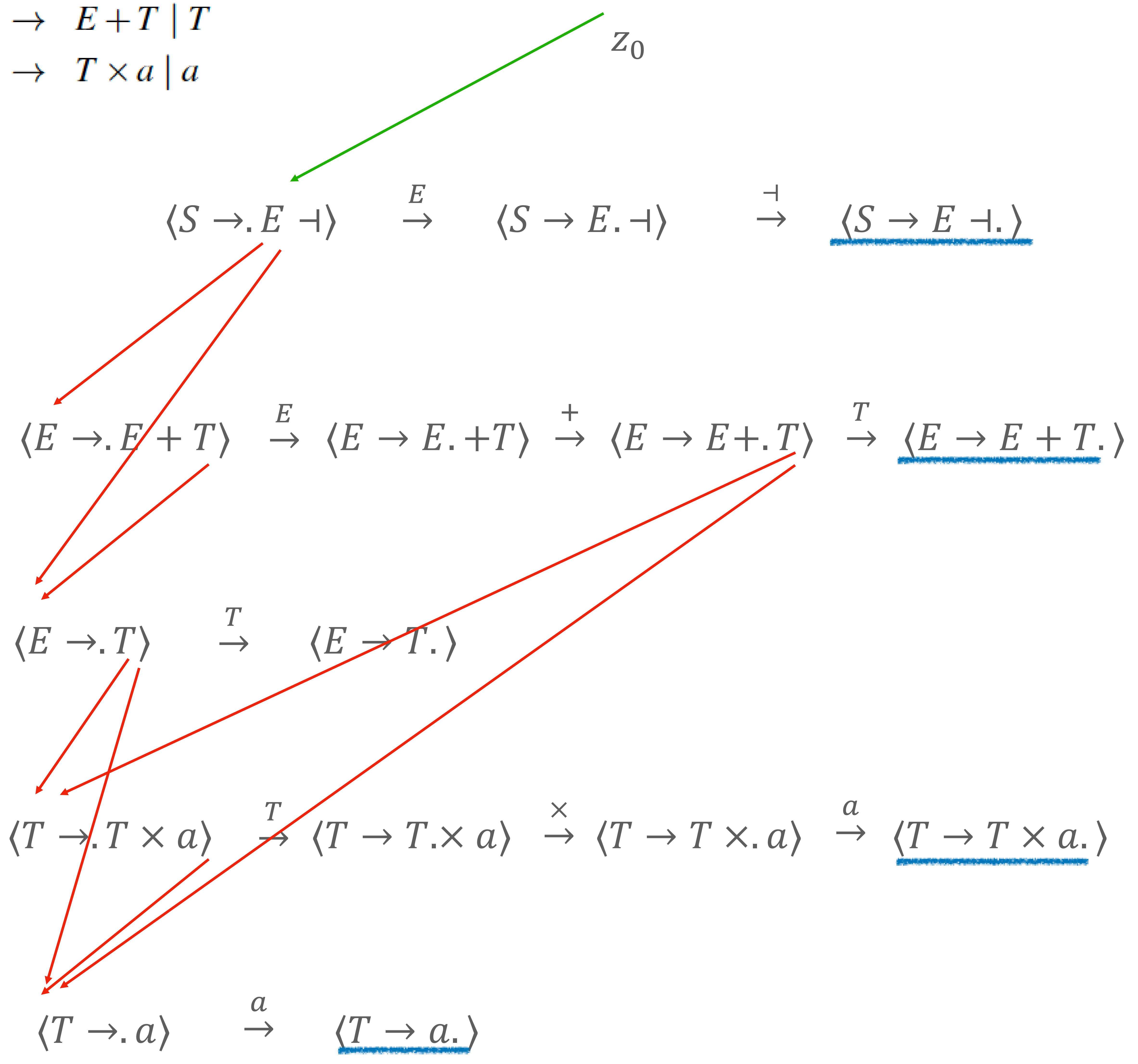
Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \;:\; K \text{ can be in state } z \text{ after it has read } z\}$$

$$
\begin{array}{c} z_0 \\ \downarrow \\ \boxed{\begin{array}{l} \langle S \to . E \dashv \rangle \\ \langle E \to . E + T \rangle \\ \langle E \to . T \rangle \\ \langle T \to . T \times a \rangle \\ \langle T \to . a \rangle \end{array}} \end{array}
\xrightarrow{E}
\boxed{\begin{array}{c} \langle S \to E . \dashv \rangle \\ \langle E \to E . + T \rangle \end{array}}
$$

$$
\begin{aligned}
S &\to E \dashv \\
E &\to E + T \mid T \\
T &\to T \times a \mid a
\end{aligned}
$$



Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

z_0

$$\boxed{\begin{array}{c}\langle S \to . E \dashv\rangle \\ \langle E \to . E + T\rangle \\ \langle E \to . T\rangle \\ \langle T \to . T \times a\rangle \\ \langle T \to . a\rangle\end{array}} \xrightarrow{E} \boxed{\begin{array}{c}\langle S \to E . \dashv\rangle \\ \langle E \to E . + T\rangle\end{array}} \xrightarrow{\dashv} \boxed{\langle S \to E \dashv .\rangle}$$

$$\begin{array}{lcl} S & \to & E \dashv \\ E & \to & E + T \mid T \\ T & \to & T \times a \mid a \end{array}$$

z_0

$$\langle S \to . E \dashv\rangle \xrightarrow{E} \langle S \to E . \dashv\rangle \xrightarrow{\dashv} \underline{\langle S \to E \dashv .\rangle}$$

$$\langle E \to . E + T\rangle \xrightarrow{E} \langle E \to E . + T\rangle \xrightarrow{+} \langle E \to E + . T\rangle \xrightarrow{T} \underline{\langle E \to E + T .\rangle}$$

$$\langle E \to . T\rangle \xrightarrow{T} \langle E \to T .\rangle$$

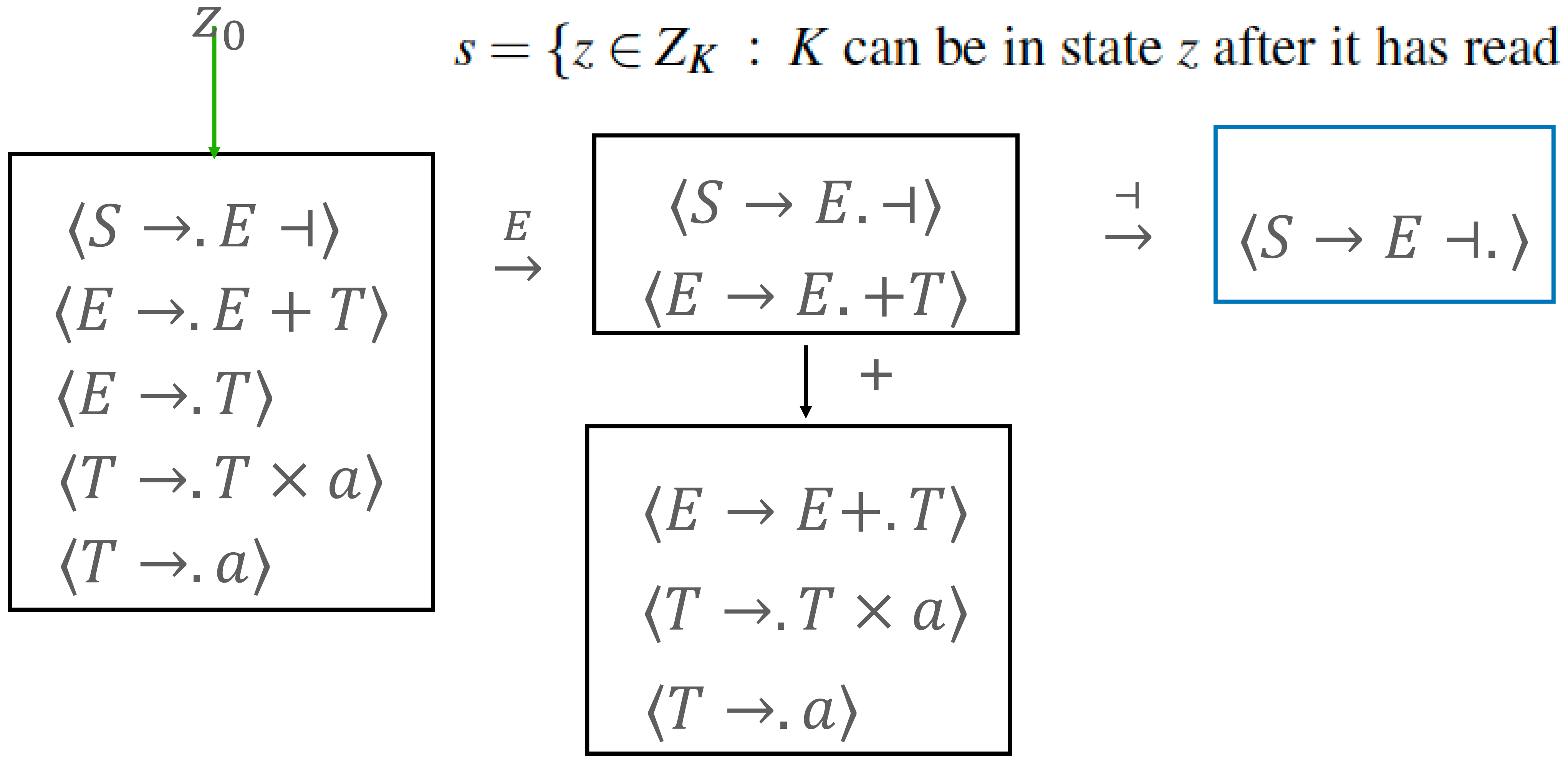
$$\langle T \to . T \times a\rangle \xrightarrow{T} \langle T \to T . \times a\rangle \xrightarrow{\times} \langle T \to T \times . a\rangle \xrightarrow{a} \underline{\langle T \to T \times a .\rangle}$$

$$\langle T \to . a\rangle \xrightarrow{a} \underline{\langle T \to a .\rangle}$$

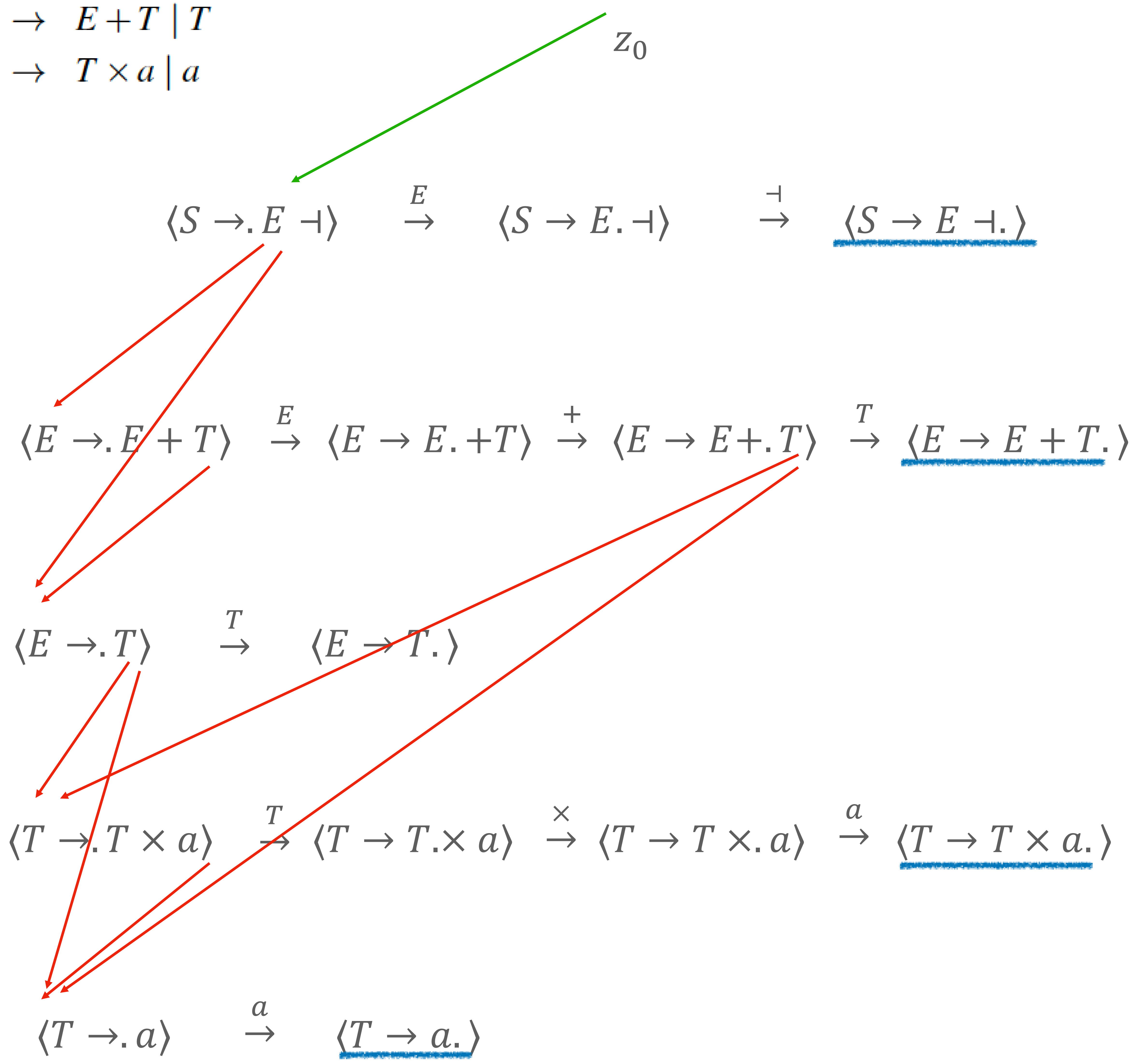
Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$



$$
\begin{aligned}
S &\to E \dashv \\
E &\to E + T \mid T \\
T &\to T \times a \mid a
\end{aligned}
$$

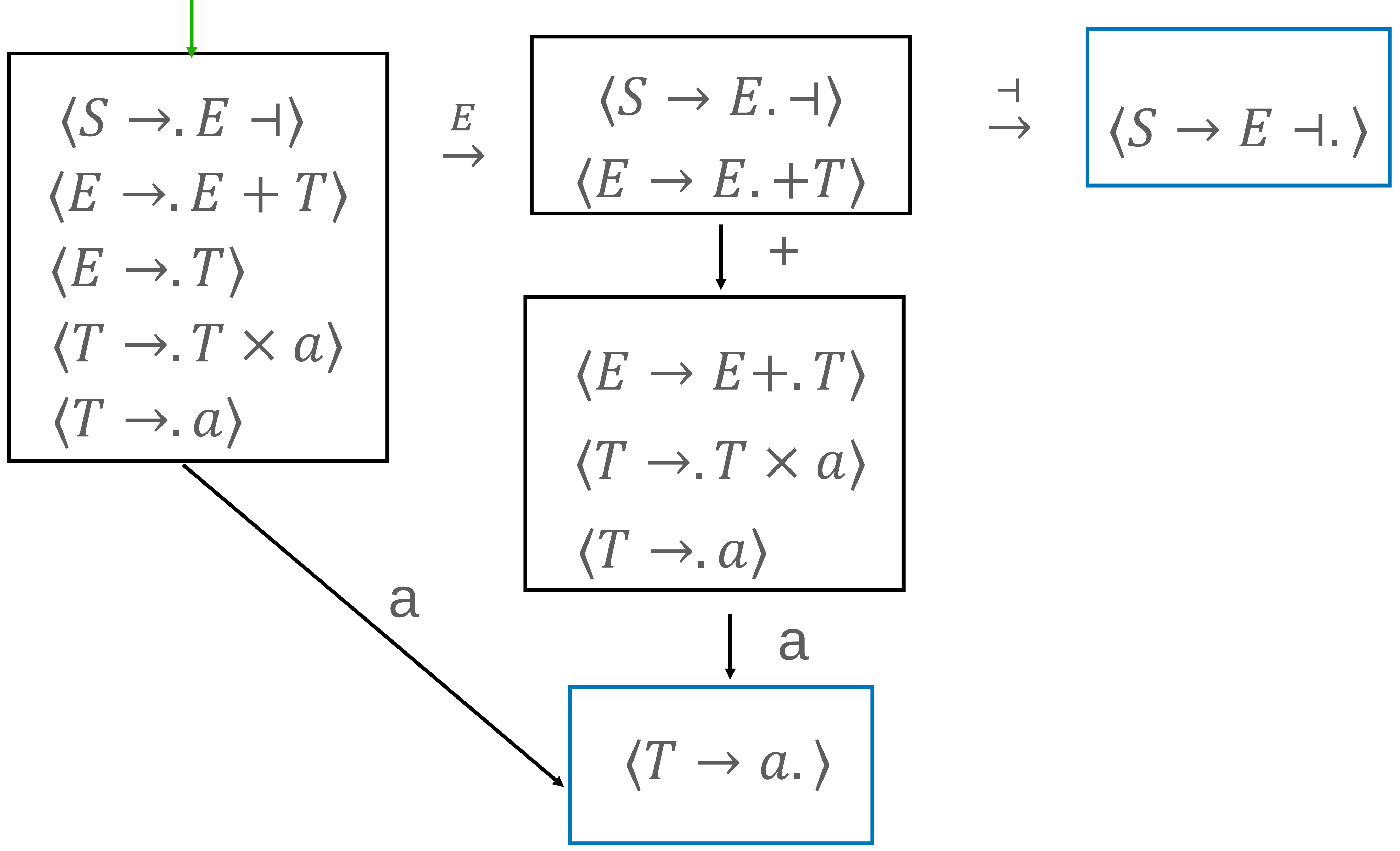


Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

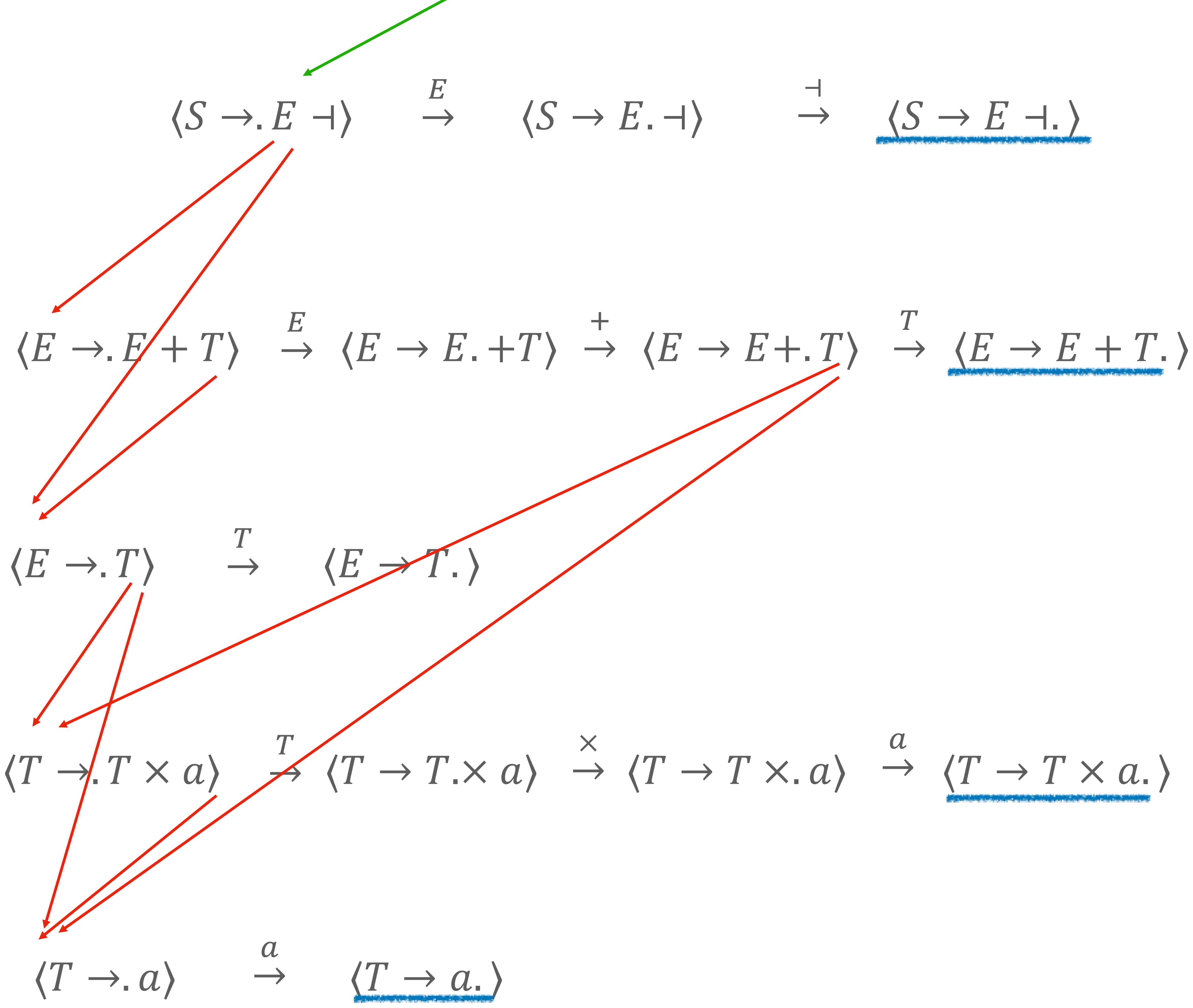
$$s = \{z \in Z_K \;:\; K \text{ can be in state } z \text{ after it has read } z\}$$

z_0



$$
\begin{aligned}
S &\to E \dashv \\
E &\to E + T \mid T \\
T &\to T \times a \mid a
\end{aligned}
$$

z_0

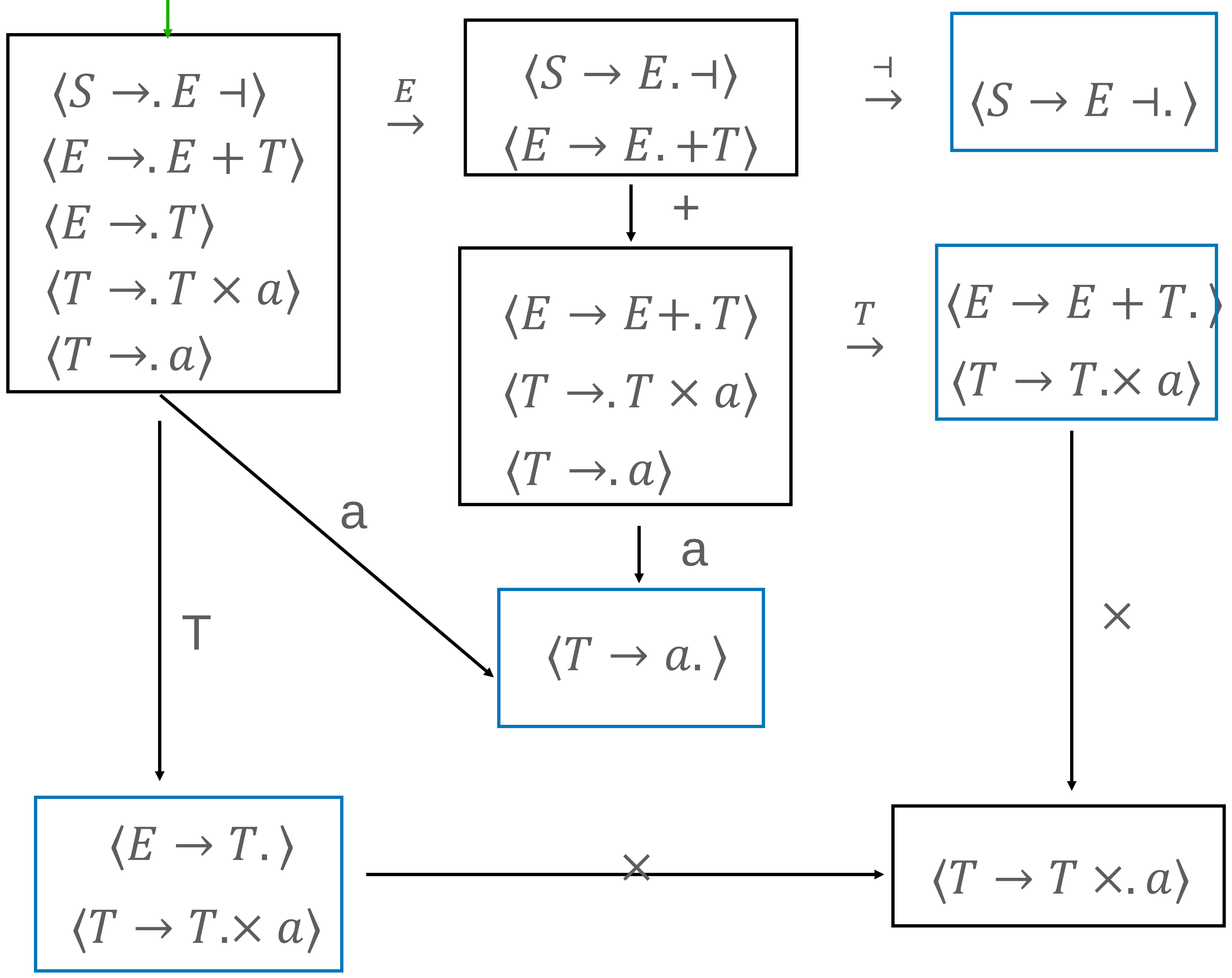


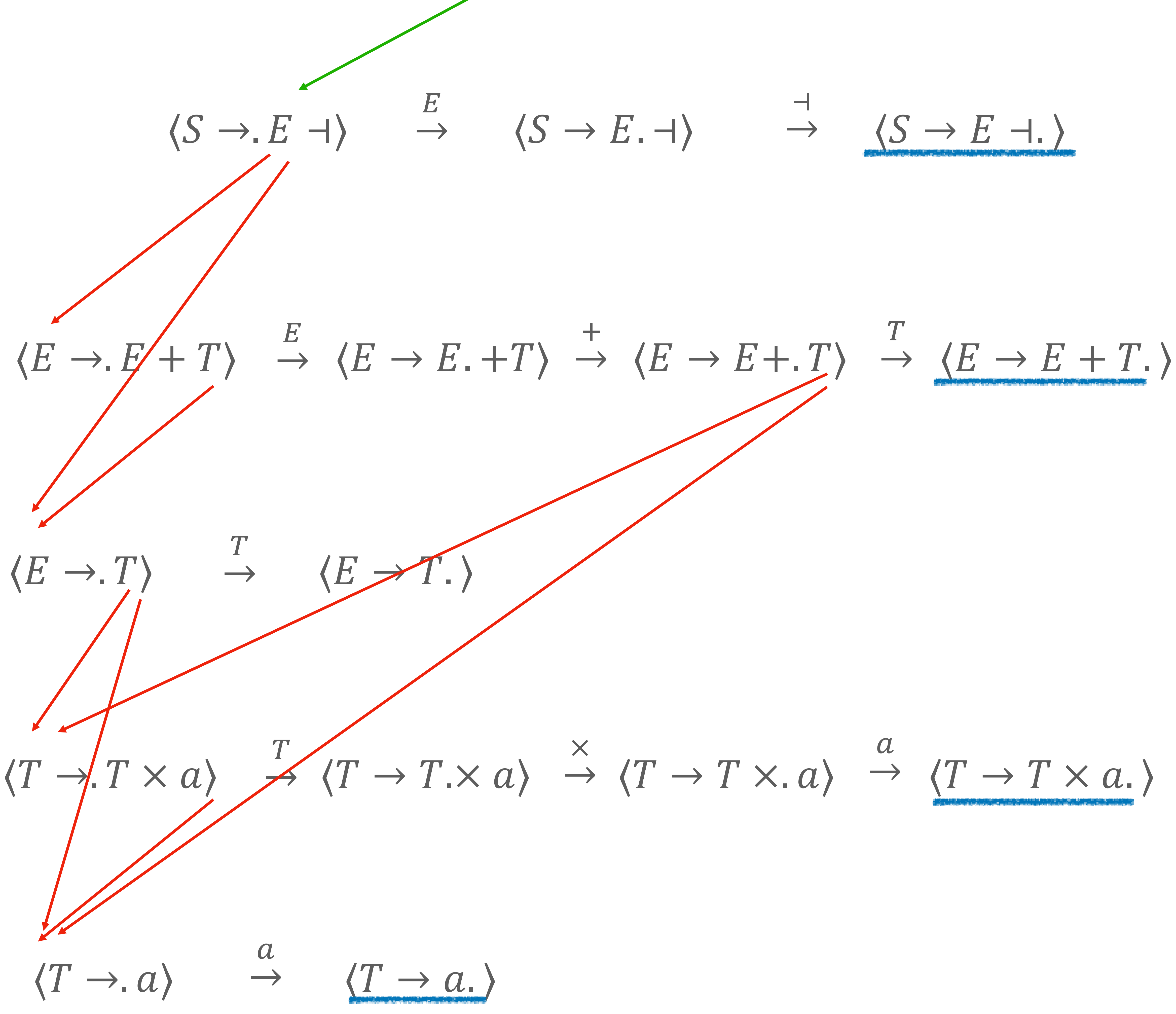
Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

$$\begin{aligned} S &\rightarrow E \dashv \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T \times a \mid a \end{aligned}$$

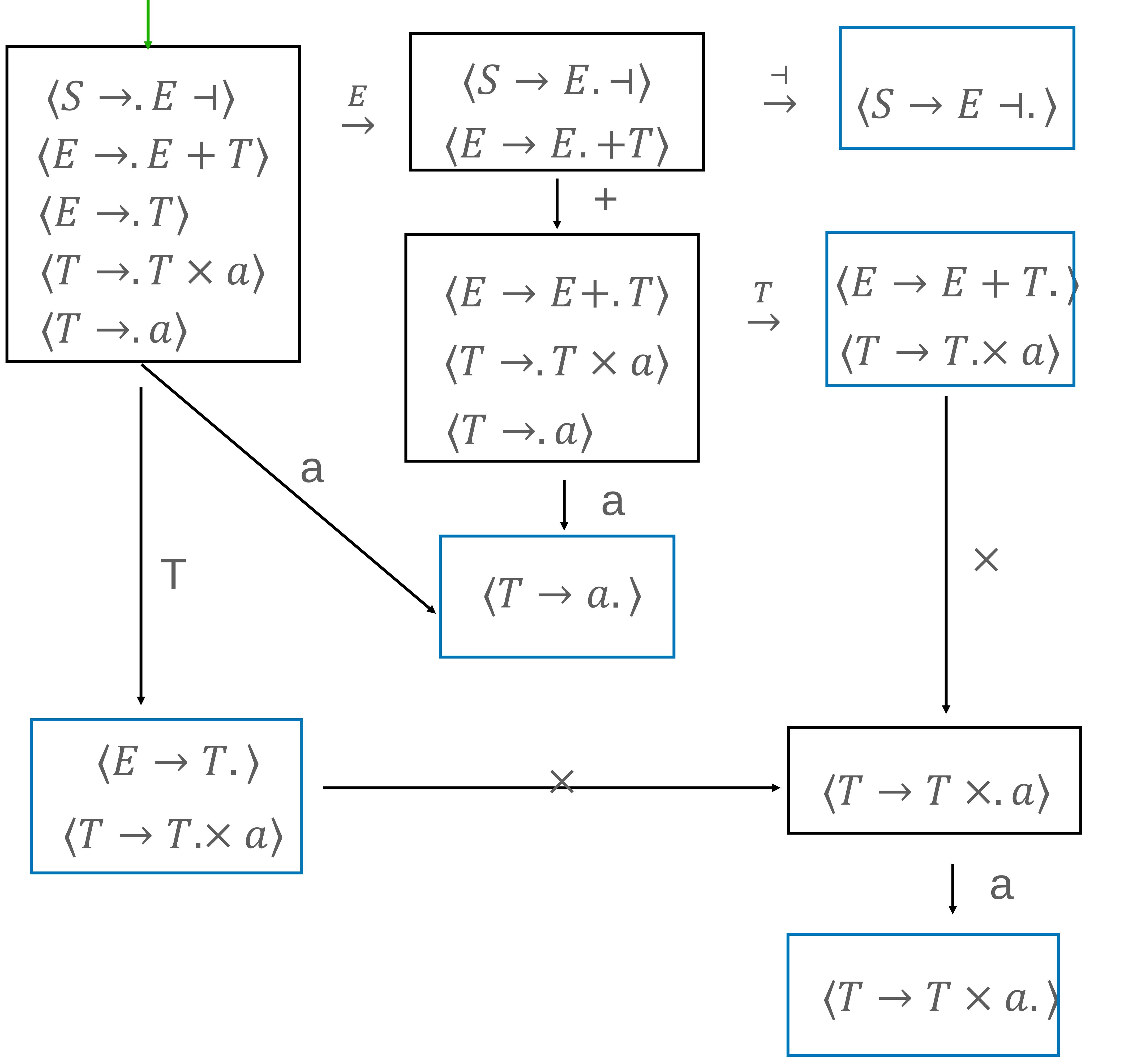




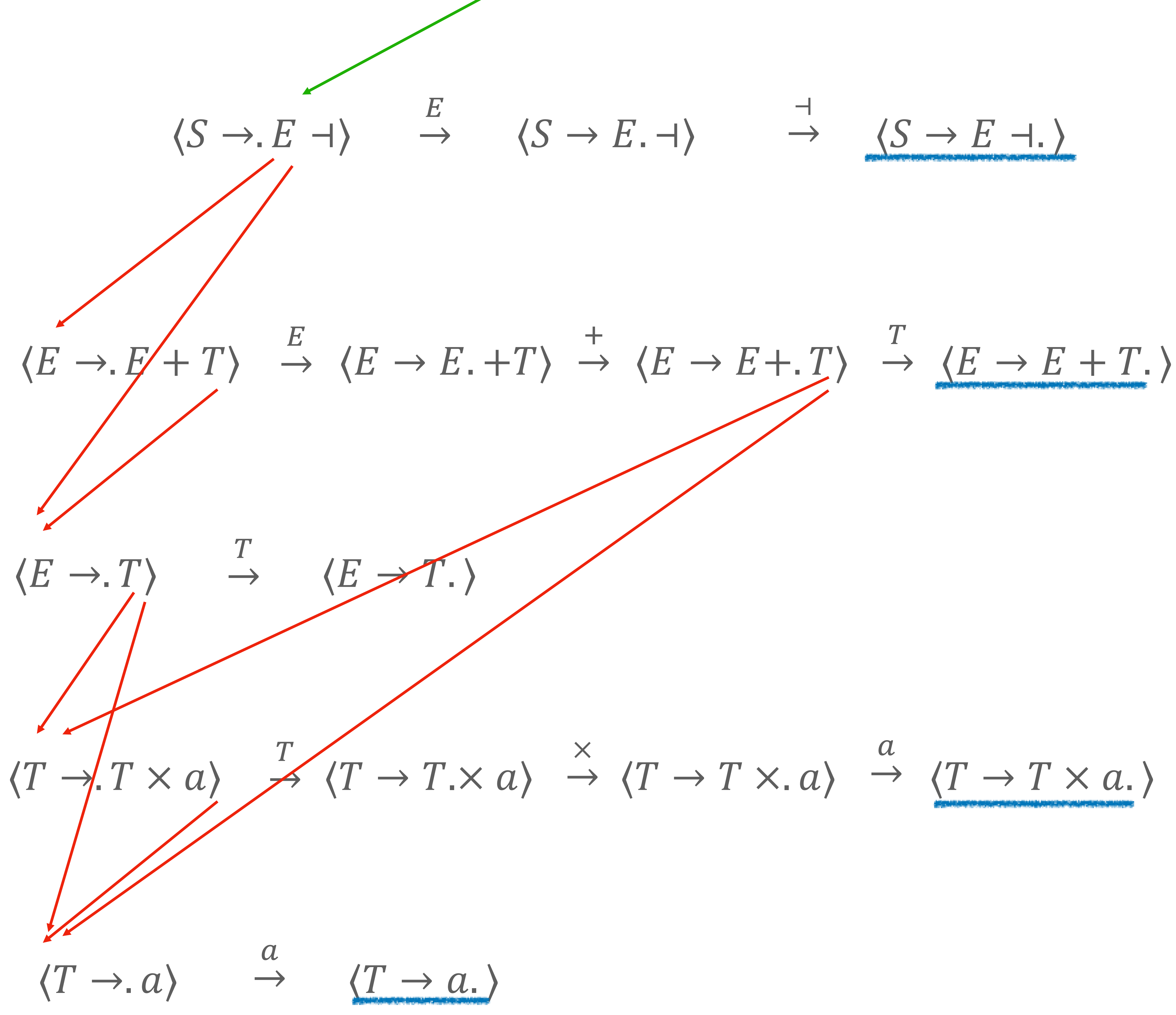
Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \;:\; K \text{ can be in state } z \text{ after it has read } z\}$$



$$\begin{aligned}
S &\to E\dashv\\
E &\to E+T \mid T\\
T &\to T\times a \mid a
\end{aligned}$$



3.2 DK-test

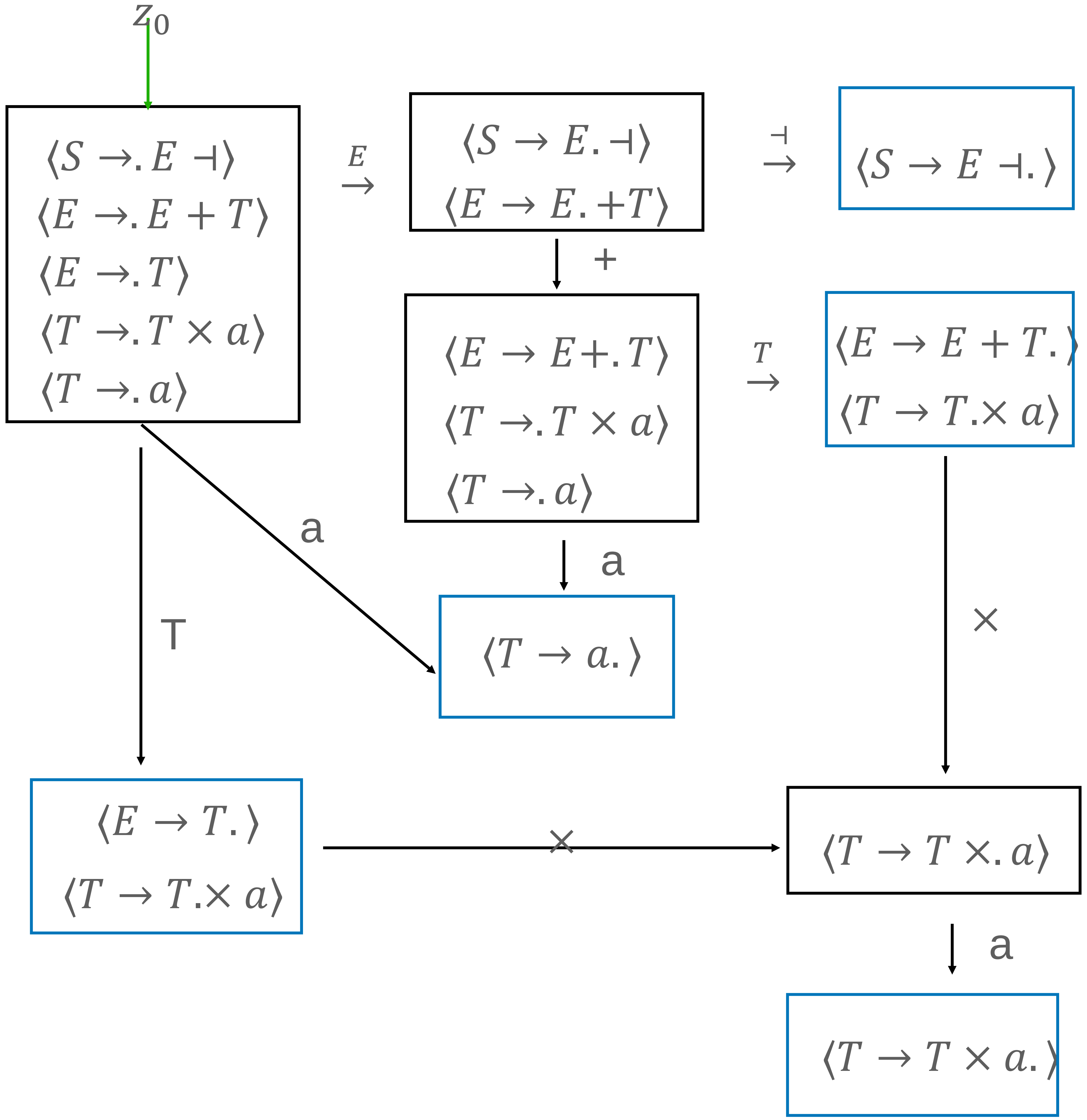
Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

DK-test A cfg G passes the DK-test if every accepting state of DK contains

- exactly one completed rule $B \to u$ (unique production for handle at this place)
- no rule $B \to u.av$ with $a \in T$ (no later handle)



3.2 DK-test

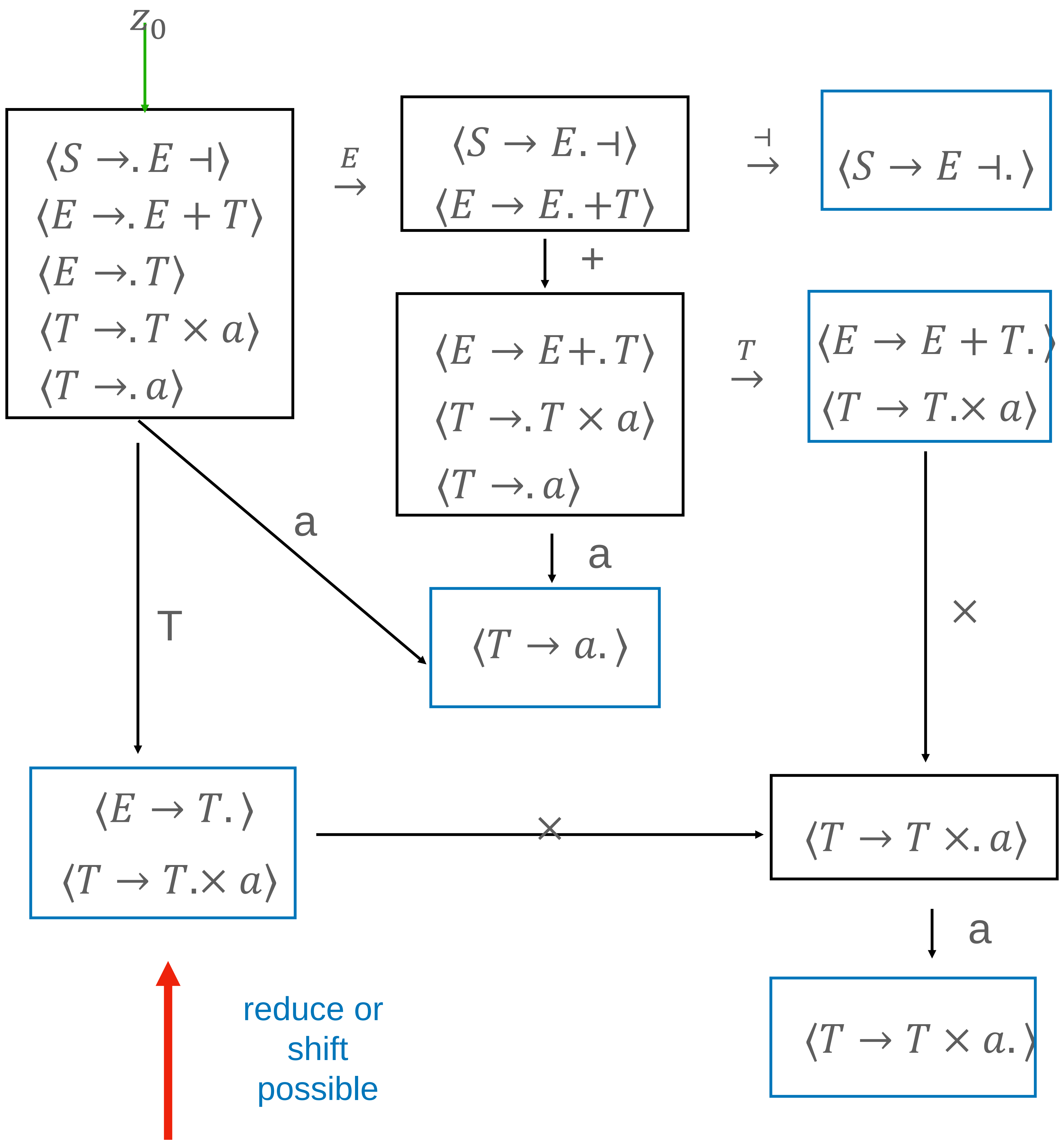
Get fa *DK* from nfa *K* by power set construction .

- If *DK* is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

DK-test A cfg G passes the DK-test if every accepting state of *DK* contains

- exactly one completed rule $B \to u$ (unique production for handle at this place)
- no rule $B \to u.av$ with $a \in T$ (no later handle)



3.2 DK-test

Get fa *DK* from nfa *K* by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

DK-test A cfg G passes the DK-test if every accepting state of DK contains

- exactly one completed rule $B \to u$ (unique production for handle at this place)
- no rule $B \to u.av$ with $a \in T$ (no later handle)

Lemma 4. *Grammar G passes the DK-test iff G is a dcfg.*

- so C0 is not deterministic
- this *alone* won't work

3.2 DK-test

Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \ : \ K \text{ can be in state } z \text{ after it has read } z\}$$

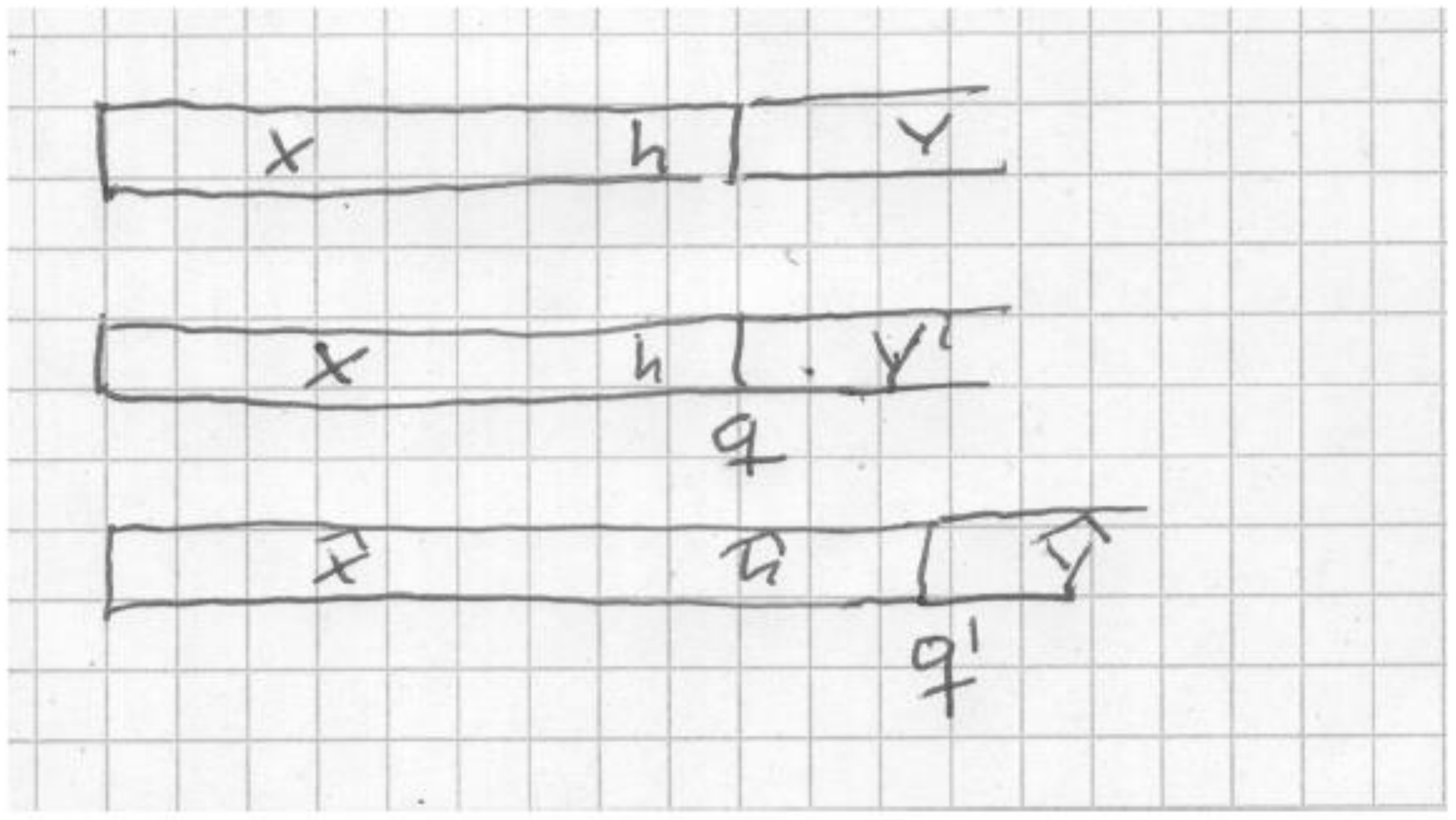
DK-test A cfg G passes the DK-test if every accepting state of DK contains

- exactly one completed rule $B \to u$ (unique production for handle at this place)
- no rule $B \to u.av$ with $a \in T$ (no later handle)

Lemma 4. *Grammar G passes the DK-test iff G is a dcfg.*

we show only $\to$ Assume valid string $v = xhy$ has unforced hanle $(|x|, T \to h)$. Hence valid string $v' = xhy'$ has handle $(|\hat{x}|, \hat{T} \to \hat{h})$

$$xhy' = \hat{x}\hat{h}\hat{y}$$



3.2 DK-test

Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \;:\; K \text{ can be in state } z \text{ after it has read } z\}$$

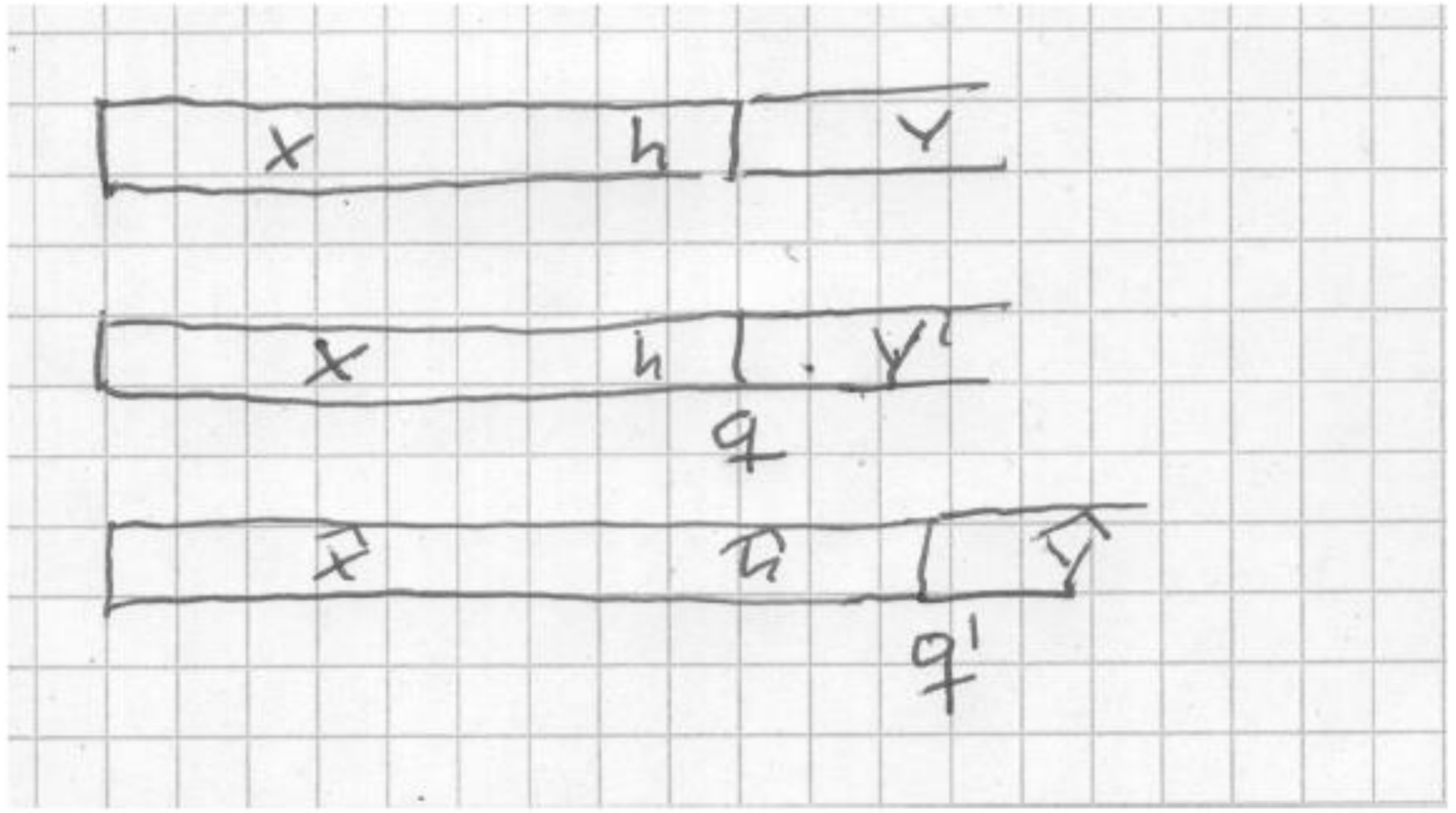
DK-test A cfg G passes the DK-test if every accepting state of DK contains

- exactly one completed rule $B \to u$ (unique production for handle at this place)
- no rule $B \to u.av$ with $a \in T$ (no later handle)

Lemma 4. *Grammar G passes the DK-test iff G is a dcfg.*

we show only $\to$ Assume valid string $v = xhy$ has unforced hanle $(|x|, T \to h)$. Hence valid string $v' = xhy'$ has handle $(|\hat{x}|, \hat{T} \to \hat{h})$

$$xhy' = \hat{x}\hat{h}\hat{y}$$



- $xh = \hat{x}\hat{h}$: Then $T \neq \hat{T}$. Test fails

3.2 DK-test

Get fa DK from nfa K by power set construction .

- If DK is in state $s \in Z_{DK}$ after it has read read z, then

$$s = \{z \in Z_K \;:\; K \text{ can be in state } z \text{ after it has read } z\}$$

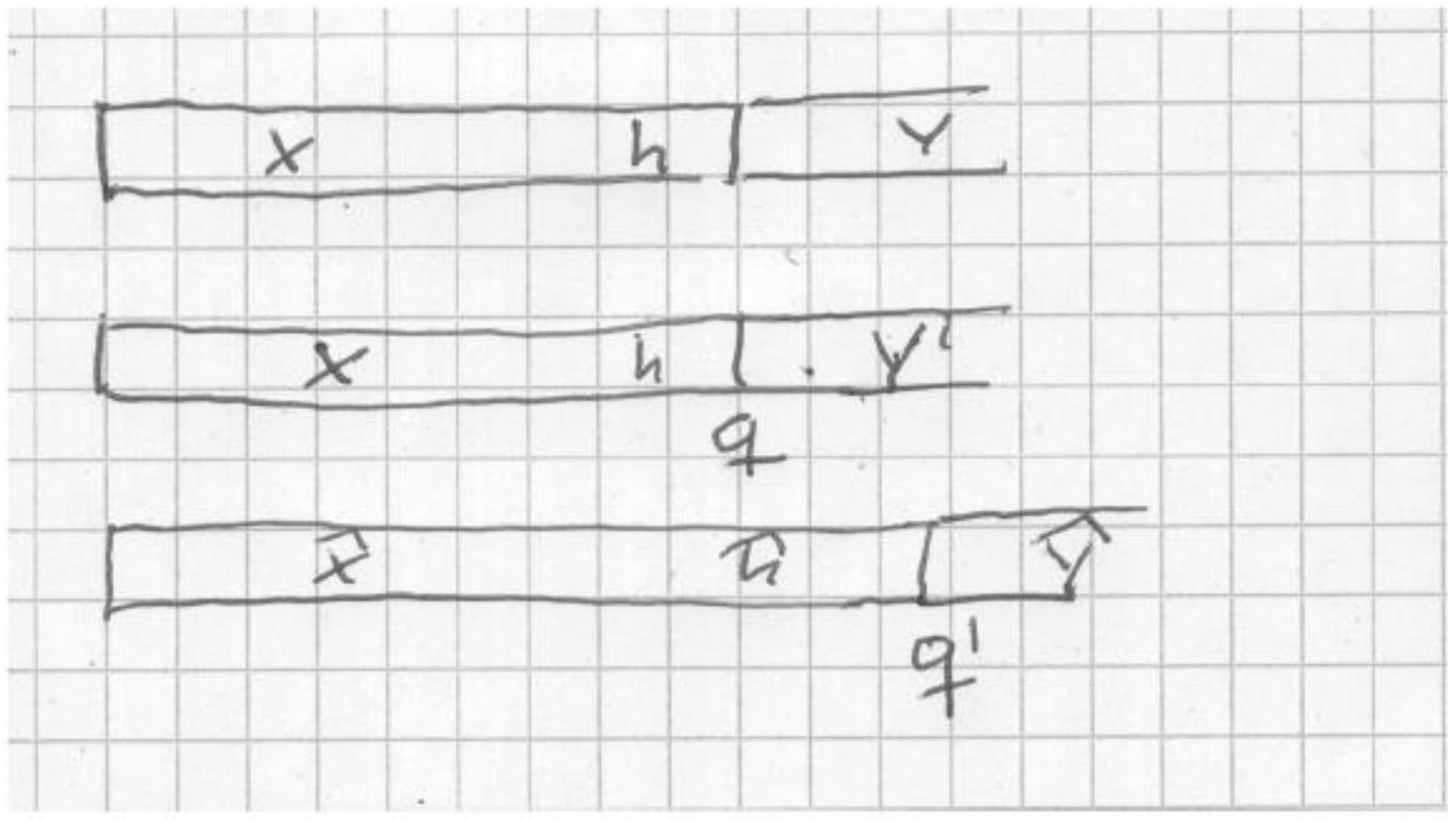
DK-test A cfg G passes the DK-test if every accepting state of DK contains

- exactly one completed rule $B \to u$ (unique production for handle at this place)
- no rule $B \to u.av$ with $a \in T$ (no later handle)

Lemma 4. *Grammar G passes the DK-test iff G is a dcfg.*

we show only $\to$ Assume valid string $v = xhy$ has unforced hanle $(|x|, T \to h)$. Hence valid string $v' = xhy'$ has handle $(|\hat{x}|, \hat{T} \to \hat{h})$

$$xhy' = \hat{x}\hat{h}\hat{y}$$



- $xh = \hat{x}\hat{h}$: Then $T \neq \hat{T}$. Test fails
- w.l.o.g xh prefix of $\hat{x}\hat{h}$: DK accepts xh in state q

There is path in DK from q to state q' leaving q with input symbol $y'_1 \in T$. Thus q contains a rule $B \to u.y'_1v$. Test fails.

4 Constructing leftmost derivations (and derivation trees) in linear time

Lemma 5. *If G passes the DK-test, then $L(G)$ is accepted by a dpda.*

naive solution in rounds i. Maintain valid strings v_i.
Let w be the input and $n = |w|$.

- round 0: v_0= input

- round $i > 0$: run DK on string v_i. If it finds handle $(n, T \to b)$ apply it to v_i to obtain v_{i+1}

run time may is $O(n^2)$; better than Younger...

4 Constructing leftmost derivations (and derivation trees) in linear time

Lemma 5. *If G passes the DK-test, then $L(G)$ is accepted by a dpda.*

naive solution in rounds i. Maintain valid strings v_i.
Let w be the input and $n = |w|$.

- round 0: v_0= input

- round $i > 0$: run DK on string v_i. If it finds handle $(n, T \to b)$ apply it to v_i to obtain v_{i+1}

run time may is $O(n^2)$; better than Younger...

using the stack

- run *DK* in finite control of dpda. Beferory every shift move of *DK* push state of *DK* on stack.

- if handle with production $T \to h$ is found: popping $|h|$ symbols from stack gives new top of stack q. Continue running *DK* with state q and next input symbol T.

run time $O(n)$.

4 Constructing leftmost derivations (and derivation trees) in linear time

Lemma 5. *If G passes the DK-test, then $L(G)$ is accepted by a dpda.*

naive solution in rounds i. Maintain valid strings v_i.
Let w be the input and $n = |w|$.

- round 0: v_0= input
- round $i > 0$: run DK on string v_i. If it finds handle $(n, T \to b)$ apply it to v_i to obtain v_{i+1}

run time may is $O(n^2)$; better than Younger...

using the stack

- run DK in finite control of dpda. Beferory every shift move of DK push state of DK on stack.
- if handle with production $T \to h$ is found: popping $|h|$ symbols from stack gives new top of stack q. Continue running DK with state q and next input symbol T.

run time $O(n)$.

examples get complicated very quickly. DK is generated by programs; parser generators like 'yacc'.

does C0 grammar pass DK test? no

5 $LR(k)$ grammars

here treated for $k = 1$.

def: handle forced by lookahead Let $H = (|x|, T \to h)$ be handle of a valid string xhy. We say H is *forced by lookahead* k if if it is the unique handle of every valid string $xh\hat{y}$, where y and $\hat{y}$ agree on the first k symbols (if either string is shorter than k they must agree on the symbold of the shorter string).

$LR(k)$**-grammar:** a cfg, where the handle of each valid string is forced by lookahead k.

5.1 DK_1-automaton

constructed from nfa K_1:

states have the form

$$\langle B \to u.v \quad a \rangle \quad \text{with lookahead symbol } a \in T$$

starting From start state z_0 and all production $S_1 \to u$ and all $a \in T$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \quad a \rangle$$

shift moves for $x \in N \cup T$ and every production $B \to uav$ and all $a \in T$

$$\langle B \to u.xv \quad a \rangle \xrightarrow{x} \langle B \to ux.v \quad a \rangle$$

ε-moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \quad a \rangle \xrightarrow{\varepsilon} \langle C \to .r \quad b \rangle$$

for all $b \in T$ which are first symbol of a string of terminals derived from v.
If v produces ε add

$$\langle B \to u.Cv \quad a \rangle \xrightarrow{\varepsilon} \langle C \to .r \quad a \rangle$$

accepting end states: states

$$\langle B \to u. \quad a \rangle$$

corresponding to a completed production for $a \in T$

5 $LR(k)$ grammars

here treated for $k = 1$.

def: handle forced by lookahead Let $H = (|x|, T \to h)$ be handle of a valid string xhy. We say H is *forced by lookahead* k if if it is the unique handle of every valid string $xh\hat{y}$, where y and $\hat{y}$ agree on the first k symbols (if either string is shorter than k they must agree on the symbold of the shorter string).

***LR*(k)-grammar:** a cfg, where the handle of each valid string is forced by lookahead k.

5.1 DK_1-automaton

- read u
- part of handle uv
- if v follows u
- and a follows v

constructed from nfa K_1:

states have the form

$$\langle B \to u.v \quad a\rangle \quad \text{with lookahead symbol } a \in T$$

starting From start state z_0 and all production $S_1 \to u$ and all $a \in T$

$$z_0 \xrightarrow{\varepsilon} \langle S_1 \to .u \quad a\rangle$$

shift moves for $x \in N \cup T$ and every production $B \to uav$ and all $a \in T$

$$\langle B \to u.xv \quad a\rangle \xrightarrow{x} \langle B \to ux.v \quad a\rangle$$

ε-moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \quad a\rangle \xrightarrow{\varepsilon} \langle C \to .r \quad b\rangle$$

for all $b \in T$ which are first symbol of a string of terminals derived from v.
If v produces ε add

$$\langle B \to u.Cv \quad a\rangle \xrightarrow{\varepsilon} \langle C \to .r \quad a\rangle$$

accepting end states: states

$$\langle B \to u. \quad a\rangle$$

corresponding to a completed production for $a \in T$

5 $LR(k)$ grammars

5.1 DK_1-automaton

constructed from nfa K_1:

states have the form

$$\langle B \to u.v \quad a\rangle \quad \text{with lookahead symbol } a \in T$$

starting From start state z_0 and all production $S_1 \to u$ and all $a \in T$

$$z_0 \xrightarrow{\varepsilon} \quad \langle S_1 \to .u \quad a\rangle$$

shift moves for $x \in N \cup T$ and every production $B \to uav$ and all $a \in T$

$$\langle B \to u.xv \quad a\rangle \xrightarrow{x} \langle B \to ux.v \quad a\rangle$$

ε-moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \quad a\rangle \xrightarrow{\varepsilon} \langle C \to .r \quad b\rangle$$

for all $b \in T$ which are first symbol of a string of terminals derived from v.
If v produces ε add

$$\langle B \to u.Cv \quad a\rangle \xrightarrow{\varepsilon} \langle C \to .r \quad a\rangle$$

accepting end states: states

$$\langle B \to u. \quad a\rangle$$

corresponding to a completed production for $a \in T$

def: consistent dotted rules Let

$$R = \langle B \to u. \quad a\rangle$$

be a completed dotted rule and

$$R' = \langle B' \to u'.cv' \quad a'\rangle$$

R and R' are *consistent* if

- R' is completed and $a = a'$ or
- R' is not completed and $v' = av''$

DK_1 test passed if no end state of DK_1 contains consistent dotted rules.

this works for example grammar and possibly C0 grammar

5 $LR(k)$ grammars

5.1 DK_1-automaton

constructed from nfa K_1:

states have the form

$$\langle B \to u.v \quad a\rangle \quad \text{with lookahead symbol } a \in T$$

starting From start state z_0 and all production $S_1 \to u$ and all $a \in T$

$$z_0 \xrightarrow{\varepsilon} \quad \langle S_1 \to .u \quad a\rangle$$

shift moves for $x \in N \cup T$ and every production $B \to uav$ and all $a \in T$

$$\langle B \to u.xv \quad a\rangle \xrightarrow{x} \langle B \to ux.v \quad a\rangle$$

ε-moves for all productions $B \to uCv$ and $C \to r$ transition

$$\langle B \to u.Cv \quad a\rangle \xrightarrow{\varepsilon} \langle C \to .r \quad b\rangle$$

for all $b \in T$ which are first symbol of a string of terminals derived from v.
If v produces ε add

$$\langle B \to u.Cv \quad a\rangle \xrightarrow{\varepsilon} \langle C \to .r \quad a\rangle$$

accepting end states: states

$$\langle B \to u. \quad a\rangle$$

corresponding to a completed production for $a \in T$

def: consistent dotted rules Let

$$R = \langle B \to u. \quad a\rangle$$

be a completed dotted rule and

$$R' = \langle B' \to u'.cv' \quad a'\rangle$$

R and R' are *consistent* if

- R' is completed and $a = a'$ or
- R' is not completed and $v' = av''$

DK_1 test passed if no end state of DK_1 contains consistent dotted rules.

this works for example grammar and possibly C0 grammar

Lemma 6. *If cfg G passes the DK_1-test, then*

- *G is an $LR(1)$ grammar*
- *$L(G)$ is recognized by a dpda*

- handles are identified as before independent of lookahead symbols
- pda reads 1 symbol ahead and stores it in its finite control, then disambiguates rules using this symbol.