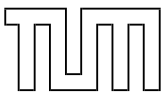
Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2009



Übersicht

Offene Referenzen

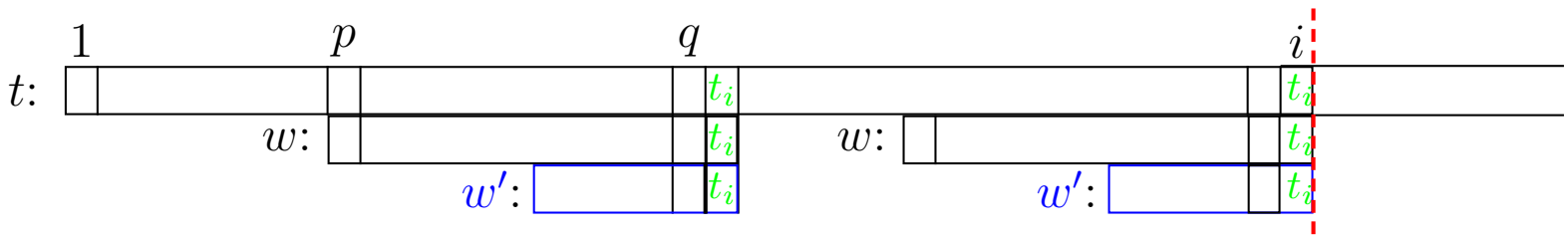
- Für Blätter des Suffix-Baumes $\hat{T}^i$ werden alle Referenzen offen angegeben, d.h. es wird $(s,(k,\infty))$ statt $(s,(k,i))$ in $\hat{T}^i$ verwendet.
- Dadurch spart man sich dann die Arbeit, die Kantenlabels der Kanten, die zu Blättern führen, während der Konstruktion des Suffixbaumes zu verlängern.

Definition

Sei $w = t_j \cdots t_n$ ein Suffix von $t_1 \cdots t_n$ mit $w = uv$, wobei $u = t_j \cdots t_{k-1}$ und $v = t_k \cdots t_n$, so dass $(s,(k,n))$ die kanonische Referenz von $\overline{w}$ ist und w als Teilwort nur einmal in $t_1 \cdots t_n$ auftritt. Dann heißt $(s,(k,\infty))$ die offene Referenz des Blattes $\overline{w}$.

Kindknotenerzeugung

- Wenn einmal ein innerer Knoten erreicht wurde, kommt man dann über die Suffix-Links wirklich nur noch zu *inneren* Knoten oder müsste vielleicht doch noch ein *Blatt* um ein Kind erweitert werden?
- Besitzt jeder weitere Knoten auch eine Kante mit dem Label t_i nachdem der Endknoten erreicht wurde?



Skizze: Endknoten beendet Erweiterung von T^{i-1}

Kindknotenerzeugung (Anfangsknoten und folgende)

- Bisher betrachteter Teil des Texts: $t' = t_1 \cdots t_{i-1}$
- Annahme: ein Suffix-Link zeigt auf einen inneren Knoten w.

$\Rightarrow$ Die Kantenlabels bis zu diesem Knoten ergeben ein Teilwort von t', das nicht nur Suffix von t' ist, sondern auch sonst irgendwo in t' auftritt (siehe w in der Skizze).

- Jeder weitere Suffix-Link verweist auf einen Knoten, der ein Suffix des zuvor gefundenen Teilwortes darstellt (blauer String w' in der Skizze).

$\Rightarrow$ Auch dieser Knoten muss ein innerer sein, da seine Kantenlabels ebenfalls nicht nur Suffix von t' sind, sondern auch noch an der anderen Stelle in t' auftauchen.

Kindknotenerzeugung (Endknoten und folgende)

- Hat man nun den Endknoten erreicht, dann besitzt der betreffende Knoten bereits eine Kante mit Kantenlabel t_i.
- Somit ergeben die Kantenlabels von der Wurzel bis zu diesem Knoten ein Teilwort von t', das nicht nur Suffix von t' ist, sondern auch an anderer Stelle in t' auftritt.
- Außerdem kann aufgrund der Kante mit Kantenlabel t_i an das Teilwort das Zeichen t_i angehängt werden.
- Da nun jeder weitere Suffix-Link auf einen Knoten zeigt, der ein Suffix des vom Endknoten repräsentierten Wortes darstellt, muss auch an das neue Teilwort das Zeichen t_i angehängt werden können.
- Also muss der betreffende Knoten eine Kante mit Kantenlabel t_i besitzen.

Referenz Endknoten / aktiver Knoten

Lemma

Sei

- $t = t_1 \cdots t_n \in \Sigma^*$
- $\hat{T}^{i-1}$ *bzw.* $\hat{T}^i$ *der Suffix-Baum für* $t_1 \cdots t_{i-1}$ *bzw.* $t_1 \cdots t_i$ *für ein* $i \in [1 : n]$.

Ist $(s, (k, i-1))$ *eine Referenz in* $\hat{T}^{i-1}$ *auf den Endknoten, dann ist* $(s, (k, i))$ *eine Referenz in* $\hat{T}^i$ *auf den aktiven Knoten.*

Referenz Endknoten / aktiver Knoten

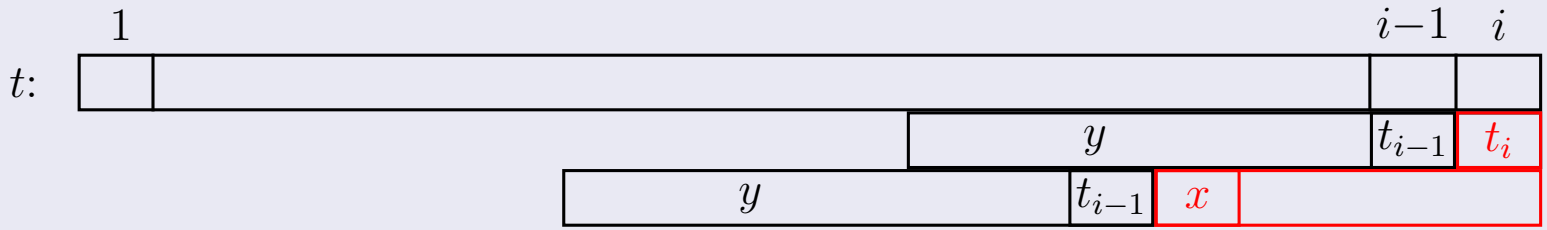
Beweis.

- Der aktive Knoten von T^i ist nach Definition der erste Knoten mit mindestens einem Kind auf dem Pfad bestehend aus Suffix-Links beginnend am Blatt $\overline{t_1 \cdots t_i}$.
- Wie man der vorletzten Abbildung ('aktiver Knoten', 'Endknoten') sofort entnimmt, sind alle neuen Knoten, die bis zum Endknoten (ausschließlich) in T^{i-1} angehängt wurden, Blätter.
- Damit ist das Kind über die Kante t_i vom Endknoten in T^{i-1} der erste Kandidat für den aktiven Knoten von T^i.

Referenz Endknoten / aktiver Knoten

Beweis.

- Wir müssen also nur noch nachweisen, dass dieser Knoten wirklich kein Blatt ist.
- Dazu betrachten wir die folgende Skizze:



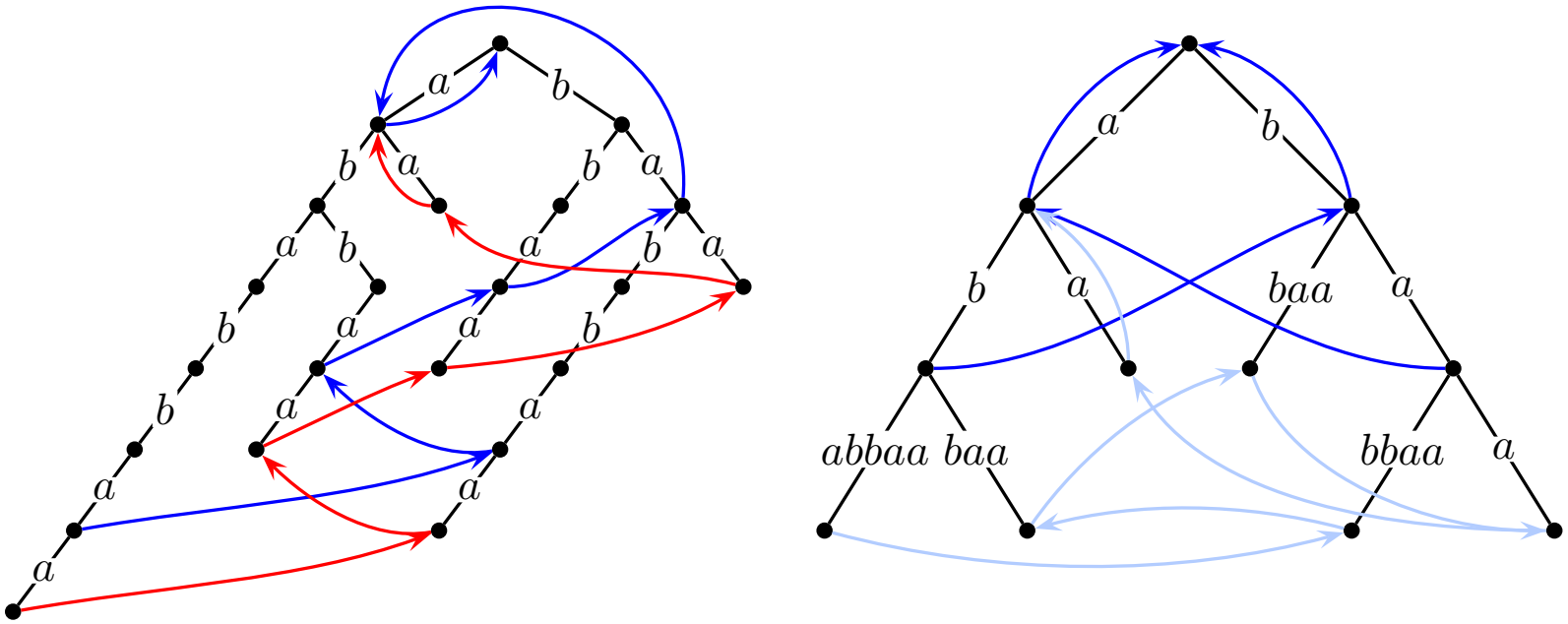
Skizze: Vom Endknoten von T^{i-1} zum aktiven Knoten von T^i

Referenz Endknoten / aktiver Knoten

Beweis.

- Bezeichnen wir mit $\overline{y}$ den Endknoten von T^{i-1}, der damit die Referenz $(s, (k, i-1))$ besitzt.
- Da $\overline{y}$ bereits in T^{i-1} die ausgehende Kante t_i besitzt, muss $y \cdot t_i$ ein Teilwort von $t_1 \cdots t_{i-1}$ sein.
- Damit muss $y \cdot t_i$ auch ein Teilwort von $t_1 \cdots t_i$ sein, das darüber hinaus mitten im Wort und nicht nur als Suffix auftauchen muss.
- Somit kann das Kind über die Kante t_i von $\overline{y}$ kein Blatt sein.

Beispiel



Beispiel: Suffix-Trie und Suffix-Baum für $t = ababbaa$

Innere Knoten

Lemma

Sei $t = t_1 \cdots t_n \in \Sigma^*$ und sei aw ein Teilwort von t mit $a \in \Sigma$, so dass $\overline{aw}$ ein innerer Knoten im Suffixbaum T für t ist. Dann ist auch $\overline{w}$ ein innerer Knoten in T.

Beweis.

- Wenn $\overline{aw}$ ein innerer Knoten in T ist, dann gibt es zwei ausgehende Kanten von $\overline{aw}$, deren Kantenlabel mit x bzw. y ($x \neq y \in \Sigma$) beginnt.
- Also ist sowohl awx als auch awy ein Teilwort von t.
- Somit sind auch wx und wy Teilwörter von t.
- Da im Suffix-Baum T alle Teilwörter von t dargestellt sind, muss $\overline{w}$ ein innerer Knoten sein, von dem mindestens zwei Kanten ausgehen, deren Kantenlabel mit x bzw. y beginnen.

Innere Knoten

- Der obige Satz gilt nicht notwendigerweise für Blätter des Suffix-Baumes.
- Daher sind Suffix-Links im Suffix-Baum in der Regel nur für innere Knoten und nicht für Blätter definiert.

Ukkonens Algorithmus

Algorithmus 12 : BuildSuffixTree(char $t[]$, int n)

$T := (\{root, \bot, v\}, \{root \xrightarrow{t_1} v\} \cup \{\bot \xrightarrow{x} root : x \in \Sigma\})$;
$suffix\_link(root) := \bot$;
$s := root$; $k := 2$; // $(s,(k,1))$ is a reference to $\overline{\epsilon}$
for $(i := 2; i \leq n; i{+}{+})$ **do**
  // Constructing $\hat{T}^i$ from $\hat{T}^{i-1}$
  // $(s,(k,i-1))$ is active point in $\hat{T}^{i-1}$
  $(s,k) := \text{Update}(s,(k,i-1),i)$;
  // Now $(s,(k,i-1))$ is endpoint of $\hat{T}^{i-1}$

Ukkonens Algorithmus

- In der for-Schleife des Algorithmus wird jeweils $\hat{T}^i$ aus $\hat{T}^{i-1}$ mit Hilfe der Prozedur Update konstruiert.

- $(s, (k, i-1))$ ist jeweils der aktive Knoten.
  Zu Beginn für $i = 2$ ist dies für $(\overline{\epsilon}, (1, 0)) \hat{=} \overline{\epsilon}$ klar.

- Die Wurzel besitzt hier entgegen der Definition von Suffix-Bäumen nur ein Kind
  (dies gilt aber für alle Suffix-Bäume zu Wörtern, die nur aus einem Buchstaben bestehen).

Ukkonens Algorithmus

- Zum Ende liefert die Prozedur Update, die aus $\hat{T}^{i-1}$ den Suffix-Baum $\hat{T}^i$ konstruiert, eine Referenz in $\hat{T}^{i-1}$ auf den Endknoten zurück, nämlich $(s,(k,i-1))$.
- Nach dem vorletzten Lemma ist dann die Referenz auf den aktive Knoten von $\hat{T}^i$ gerade $(s,(k,i))$.
- Da in der for-Schleife i um eins erhöht wird, erfolgt der nächste Aufruf von Update wieder korrekt mit der Referenz auf den aktiven Knoten von $\hat{T}^i$, der jetzt ja $\hat{T}^{i-1}$ ist, wieder mit der korrekten Referenz $(s,(k,i-1))$.

Canonize

- konstruiert aus einer übergebenen Referenz eine kanonische Referenz
- Man will ja eigentlich nur mit kanonischen Referenzen arbeiten.

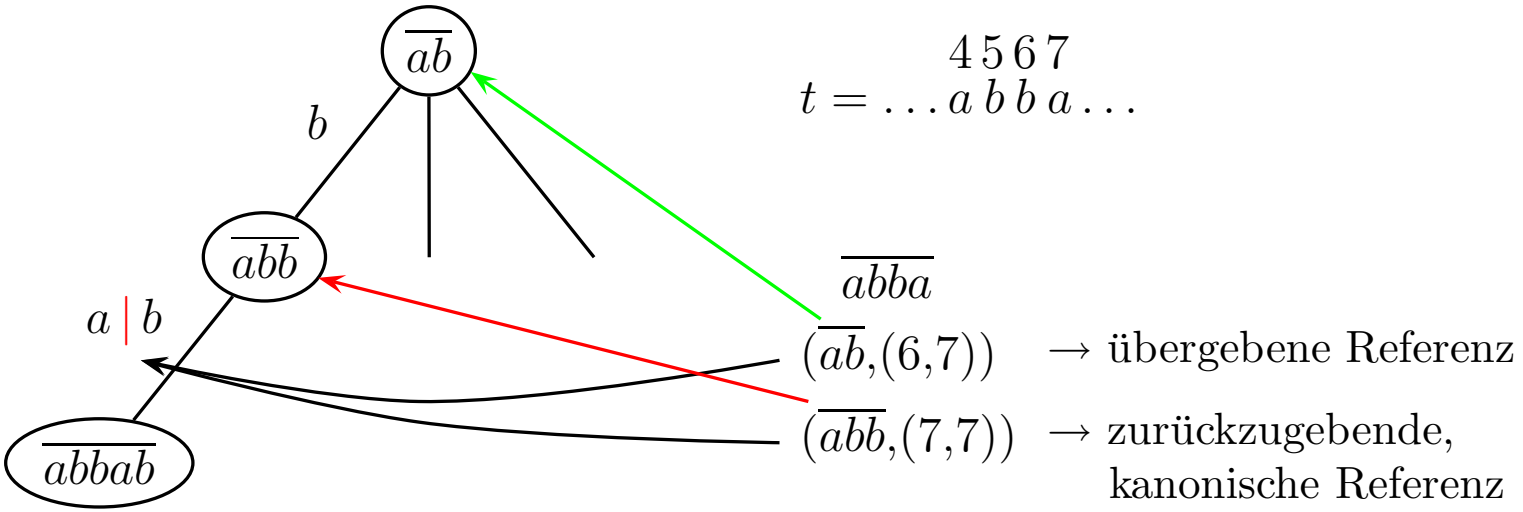
Algorithmus 13 : Canonize(node s, ref (k, p))

while $(|t_k \cdots t_p| > 0)$ **do**
  let $e := s \xrightarrow{w} s'$ s.t. $w_1 = t_k$;
  if $(|w| > |t_k \cdots t_p|)$ **then** break;
  $k := k + |w|$;
  $s := s'$
return (s, k);

Canonize

- Der Prozedur Canonize wird eine Referenz $(\overline{s}, (k, p))$ übergeben.
- Die Prozedur durchläuft dann den Suffix-Baum ab dem Knoten $\overline{s}$ entlang der Zeichenreihe $t_k \cdots t_p$.
- Kommt man nun mitten auf einer Kante zum Stehen, entspricht der zuletzt besuchte Knoten dem Knoten, der für die kanonische Referenz verwendet wird.
- Dies ist in der folgenden Abbildung am Beispiel für die Referenz $(\overline{ab}, (6, 7))$ illustriert.

Canonize



Beispiel: Erstellung kanonischer Referenzen mittels Canonize

Update

Algorithmus 14 : Update(node s, ref (k, p), int i)

```
// (s,(k,p)) is active point
old_r := root;
(s,k) := Canonize(s,(k,p));
(done,r) := TestAndSplit(s,(k,p),t_i);
while ( ! done ) do
    let m be a new node and add r --(i,∞)--> m;
    if (old_r ≠ root) then
        suffix_link(old_r) = r;
    old_r := r;
    (s,k) := Canonize(suffix_link(s),(k,p));
    (done,r) := TestAndSplit(s,(k,p),t_i);
if (old_r ≠ root) then
    suffix_link(old_r) := s;
return (s, k);
```

Update

- In Update wird nun der Suffix-Baum $\hat{T}^i$ aus dem $\hat{T}^{i-1}$ aufgebaut.
- Update bekommt eine (nicht unbedingt kanonische) Referenz des Knotens übergeben, an welchen das neue Zeichen t_i, dessen Index i ebenfalls übergeben wird, angehängt werden muss.
- Dabei hilft die Prozedur Canonize, welche die übergebene Referenz kanonisch macht, und die Prozedur TestAndSplit, die den existierenden Knoten im Suffix-Baum zurückgibt, an welchen die neue Kante mit Kantenlabel t_i angehängt werden muss, sofern diese noch nicht vorhanden ist.

TestAndSplit

Algorithmus 15 : TestAndSplit(node s, ref (k, p), char x)

if $(|t_k \cdots t_p| = 0)$ **then**

  `// explizite Referenz`

  if $(\exists\ s \xrightarrow{x\cdots})$ **then return** *(TRUE, s)*;

  else return *(FALSE, s)*;

else

  `// Implizite Referenz`

  let $e := s \xrightarrow{w} s'$ s.t. $w_1 = t_k$;

  if $(x = w_{|t_k \cdots t_p|+1})$ **then return** *(TRUE, s)*;

  else

    split $e := s \xrightarrow{w} s'$ s.t.: $s \xrightarrow{w_1 \cdots w_{|t_k \cdots t_p|}} m \xrightarrow{w_{|t_k \cdots t_p|+1} \cdots w_{|w|}} s'$

    return *(FALSE, m)*

TestAndSplit

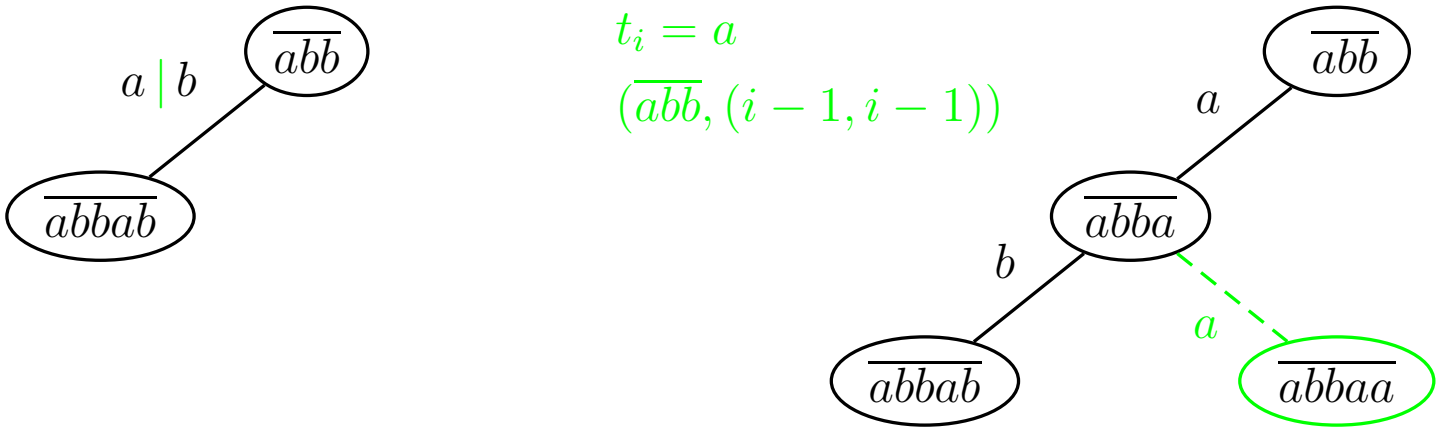
- TestAndSplit überprüft zunächst, ob die betreffende Kante im korrespondierenden Suffix-Trie bereits vorhanden ist oder nicht.
- Falls dies der Fall ist, liefert TestAndSplit TRUE und liefert einen beliebigen Knoten des Suffix-Baumes zurück.
- Ist die Kante jedoch noch nicht vorhanden, aber der Knoten, von welchem diese ausgehen sollte, existiert bereits im Suffix-Baum, wird einfach dieser explizite Knoten zurückgegeben.
- Nun kann noch der Fall auftreten, dass der Knoten, an welchen die neue Kante angehängt werden muss, nur implizit im Suffix-Baum vorhanden ist.
- In diesem Fall muss die betreffende Kante aufgespalten werden, und der benötigte Knoten als expliziter Knoten eingefügt werden.
- Anschließend wird dieser dann ausgegeben, und in der Prozedur Update wird die fehlende Kante eingefügt.

TestAndSplit

- In TestAndSplit ist die erste Fallunterscheidung, ob die kanonische Referenz auf einen expliziten oder impliziten Knoten zeigt ($|t_k \cdots t_p| = 0$).
- Ist der Knoten explizit, muss in jedem Falle kein neuer innerer Knoten generiert werden und die Kante mit Label t_i kann an den expliziten Knoten s im Suffix-Baum angehängt werden.

- Ist andernfalls der referenzierte Knoten implizit, so wird zuerst getestet, ob an diesem eine 'Kante' mit Label t_i hängt ($t_i = w_{|t_k \cdots t_p|+1}$).
- Falls ja, ist nichts zu tun.
  Ansonsten muss diese Kante mit dem langen Label aufgebrochen werden (wie im folgenden Beispiel).

TestAndSplit



Beispiel: Vorgehensweise von TestAndSplit

- Dann wird der neue Knoten r, der in die lange Kante eingefügt wurde, von TestAndSplit zurückgegeben, damit die Prozedur Update daran die neue Kante mit Label t_i anhängen kann.

Fortsetzung Update

- zurück zur Prozedur Update
- Auch hier folgt man wieder vom aktiven Knoten aus den Suffix-Links und hängt eine neue Kante mit Label t_i ein.
- Im Suffix-Baum werden nur für interne Knoten die Suffix-Links berechnet und nicht für die Blätter.
- Mit *old_r* merkt man sich immer den zuletzt neu eingefügten internen Knoten, für den der Suffix-Link noch zu konstruieren ist.
- Zu Beginn setzt man *old_r* auf *root*, um damit anzuzeigen, dass man noch keinen neuen Knoten eingefügt hat.

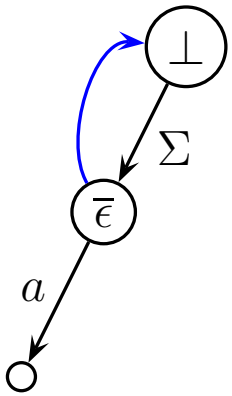
Fortsetzung Update

- Wann immer man dem Suffix-Link gefolgt ist und im Schritt vorher einen neuen Knoten eingefügt hat, setzt man für diesen zuvor eingefügten Knoten den Suffix-Link jetzt mittels $\text{Suffix-Link}(old\_r) = r$.

- Der Knoten r ist gerade der Knoten, an den wir jetzt aktuell die Kante mit Label t_i anhängen wollen und somit der Vater des neuen Blattes.

- Somit wird der Suffix-Link von den korrespondierenden Eltern korrekt gesetzt.

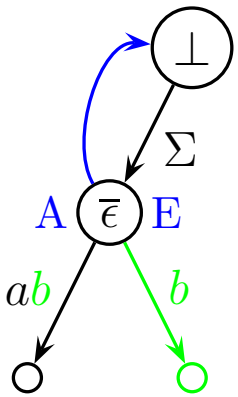
Ukkonen-Algorithmus: Beispiel



$(\overline{\epsilon}, (2, 1)); i = 1; t_i = a$

Beispiel: Ukkonens Algorithmus: $\hat{T}^1$ für $t = ababbaa$

Ukkonen-Algorithmus: Beispiel



$(\overline{\epsilon}, (2,1))$; $i = 2$; $t_i = b$

↓ Canonize

$(\overline{\epsilon}, (2,1))$

↯ TestAndSplit = (FALSE,$\overline{\epsilon}$)

$(\overline{\epsilon}, (2,1))$

↓ Suffix-Link

$(\bot, (2,1))$

↓ Canonize

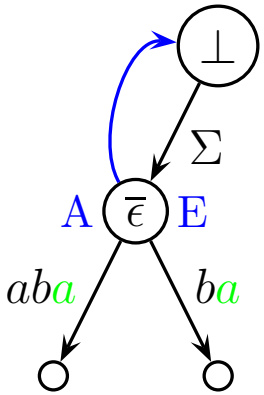
$(\bot, (2,1))$

↯ TestAndSplit = (TRUE,$\bot$)

$(\bot, (2,1))$

Beispiel: Ukkonens Algorithmus: $\hat{T}^2$ für $t = ababbaa$

Ukkonen-Algorithmus: Beispiel



$(\bot, (2, 2)); i = 3; t_i = a$

↓ Canonize

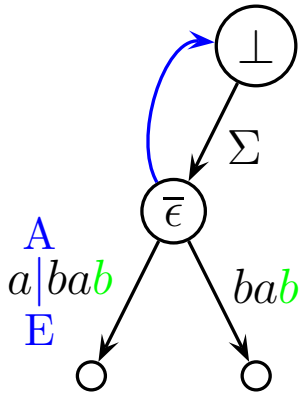
$(\overline{\epsilon}, (3, 2))$

↯ TestAndSplit = (TRUE,$\overline{\epsilon}$)

$(\overline{\epsilon}, (3, 2))$

Beispiel: Ukkonens Algorithmus: $\hat{T}^3$ für $t = ababbaa$

Ukkonen-Algorithmus: Beispiel



$(\overline{\epsilon}, (3,3))$; $i = 4$; $t_i = b$

↓ Canonize

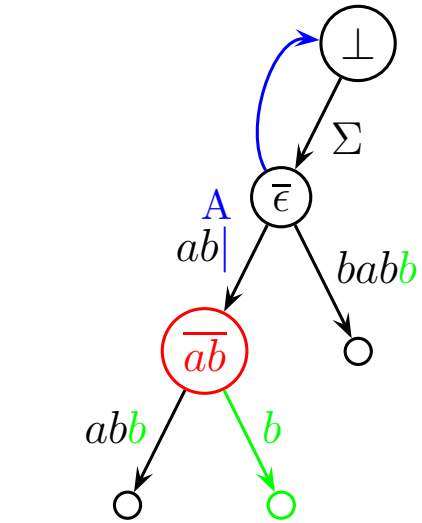
$(\overline{\epsilon}, (3,3))$

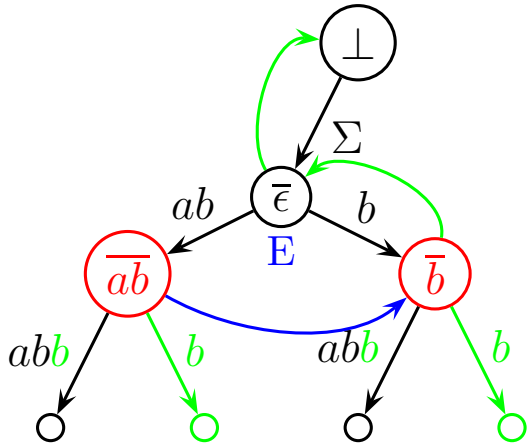
↯ TestAndSplit = (TRUE,$\overline{\epsilon}$)

$(\overline{\epsilon}, (3,3))$

Beispiel: Ukkonens Algorithmus: $\hat{T}^4$ für $t = ababbaa$

Ukkonen-Algorithmus: Beispiel





$(\overline{\epsilon}, (3,4))$; $i = 5$; $t_i = b$

↓ Canonize

$(\overline{\epsilon}, (3,4))$

↯ TestAndSplit = (FALSE,$\overline{ab}$)

$(\overline{\epsilon}, (3,4))$

↓ Suffix-Link

$(\bot, (3,4))$

↓ Canonize

$(\overline{\epsilon}, (4,4))$

↯ TestAndSplit = (FALSE,$\overline{b}$)

$(\overline{\epsilon}, (4,4))$

↓ Suffix-Link

$(\bot, (4,4))$

↓ Canonize

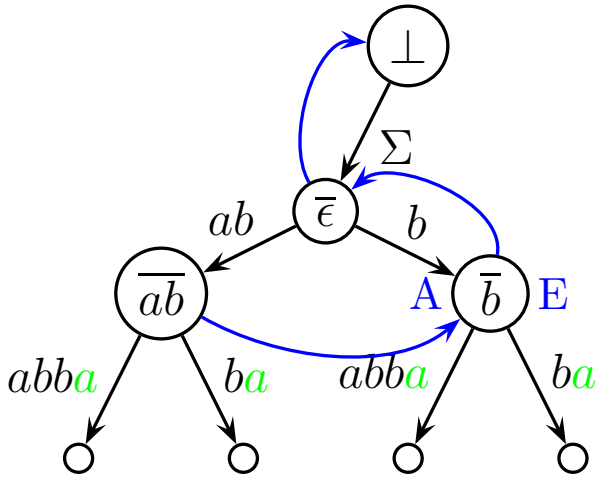
$(\overline{\epsilon}, (5,4))$

↯ TestAndSplit = (TRUE,$\overline{\epsilon}$)

$(\overline{\epsilon}, (5,4))$

Beispiel: Ukkonens Algorithmus: $\hat{T}^5$ für $t = ababbaa$

Ukkonen-Algorithmus: Beispiel



$(\overline{\epsilon}, (5,5))$; $i = 6$; $t_i = a$

↓ Canonize

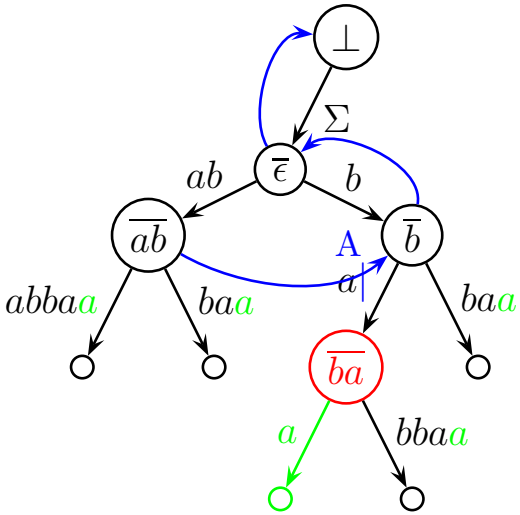
$(\overline{b}, (6,5))$

↯ TestAndSplit = (TRUE,$\overline{b}$)

$(\overline{b}, (6,5))$

Beispiel: Ukkonens Algorithmus: $\hat{T}^6$ für $t = ababbaa$

Ukkonen-Algorithmus: Beispiel



$(\overline{b}, (6,6))$; $i = 7$; $t_i = a$

↓ Canonize

$(\overline{b}, (6,6))$

↯ TestAndSplit = (FALSE,$\overline{ba}$)

$(\overline{b}, (6,6))$

↓ Suffix-Link

$(\overline{\epsilon}, (6,6))$

↓ Canonize

$(\overline{\epsilon}, (6,6))$

↯ TestAndSplit = (FALSE,$\overline{a}$)

$(\overline{\epsilon}, (6,6))$

↓ Suffix-Link

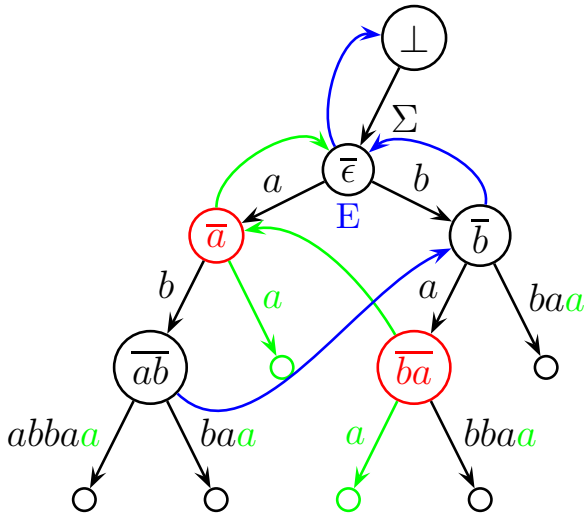
$(\bot, (6,6))$

↓ Canonize

$(\overline{\epsilon}, (7,6))$

↯ TestAndSplit = (TRUE,$\overline{\epsilon}$)

$(\overline{\epsilon}, (7,6))$



Beispiel: Ukkonens Algorithmus: $\hat{T}^7$ für $t = ababbaa$