Parsing Natural Language

Parsing Without Grammar: Shift-Reduce Dependency Parsing

Parsing: Schematics

- So far, we considered algorithms that used a grammar to only produce „valid“ trees
- In this lecture, we are using machine-learning to directly produce a parse-tree, without relying on a grammar
  - This means our output can be super unintuitive, since it is not guide by a grammar, but we are able to incorporate strong features into the process!
- Two famous approaches:
  1. Shift-Reduce-Parsing
  2. Graph-based parsing

Shift-Reduce-Parsing

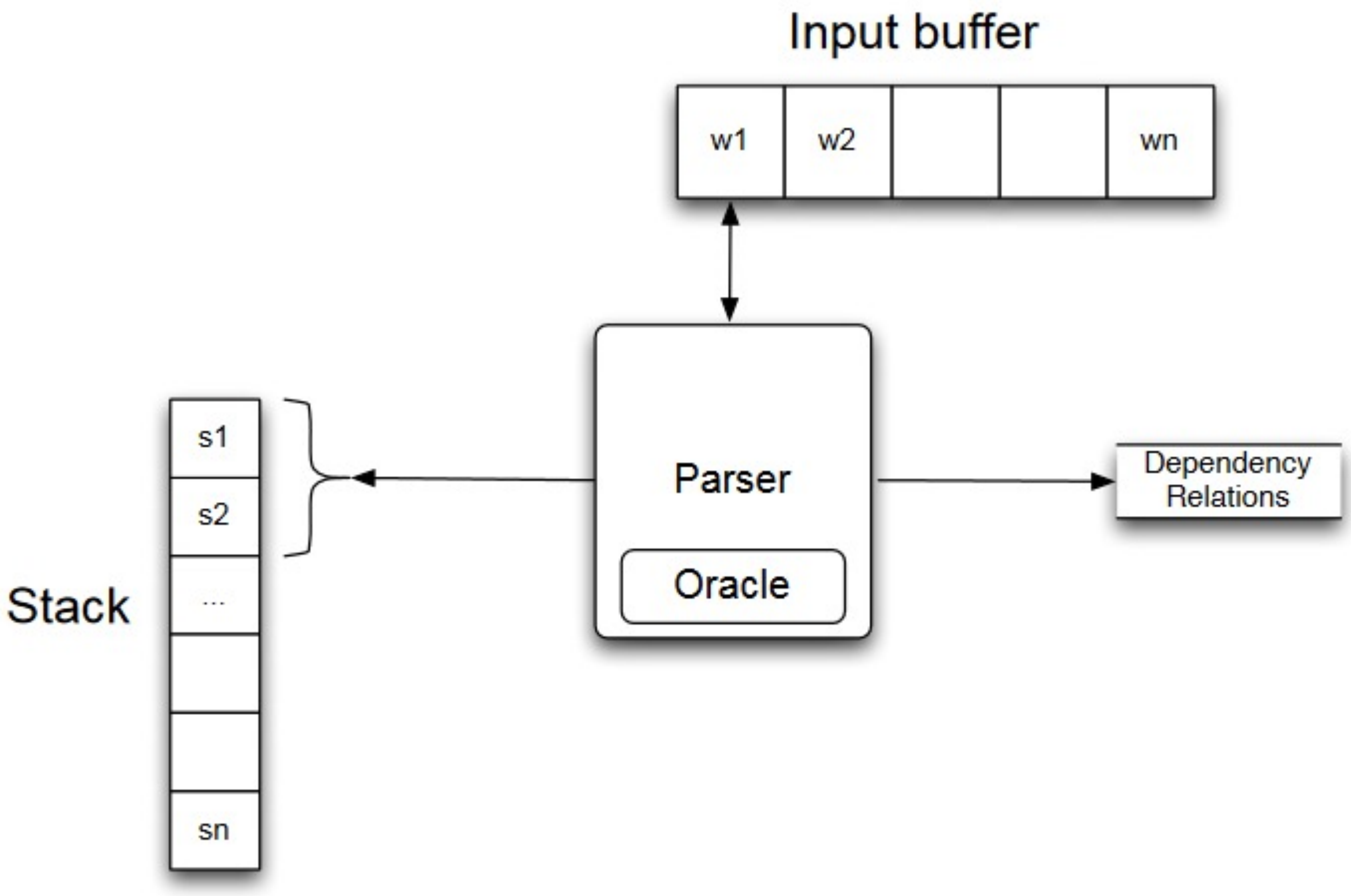
- Origins in parsing computer languages
- Makes use of a very clever data structure to deal with this problem
- Only difference to a classical parser for computer language is the addition of an „oracle“

- Usually used for Dependency Parsing problems!

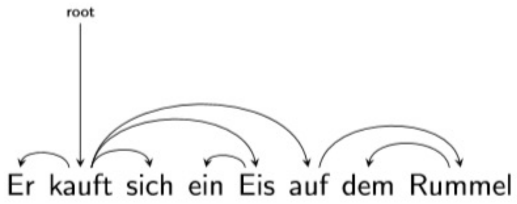
Shift-Reduce-Parsing

- Makes use of a very clever data structure to deal with this problem
- The clever idea, is that we can convert the problem of producing a tree into a classification problem with just 3 classes!
- This works by the introduction of a Buffer and a Stack! (see next slide)

Dependency-Parsing: Data structure



Shift Reduce Parsing



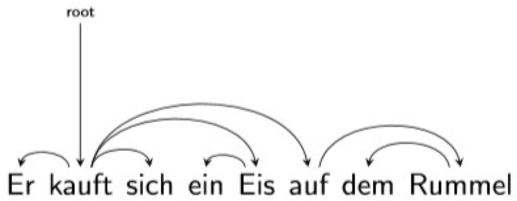
He buys an ice cream at the carnival

Stack

Buffer

root Er kauft sich ein Eis auf dem Rummel

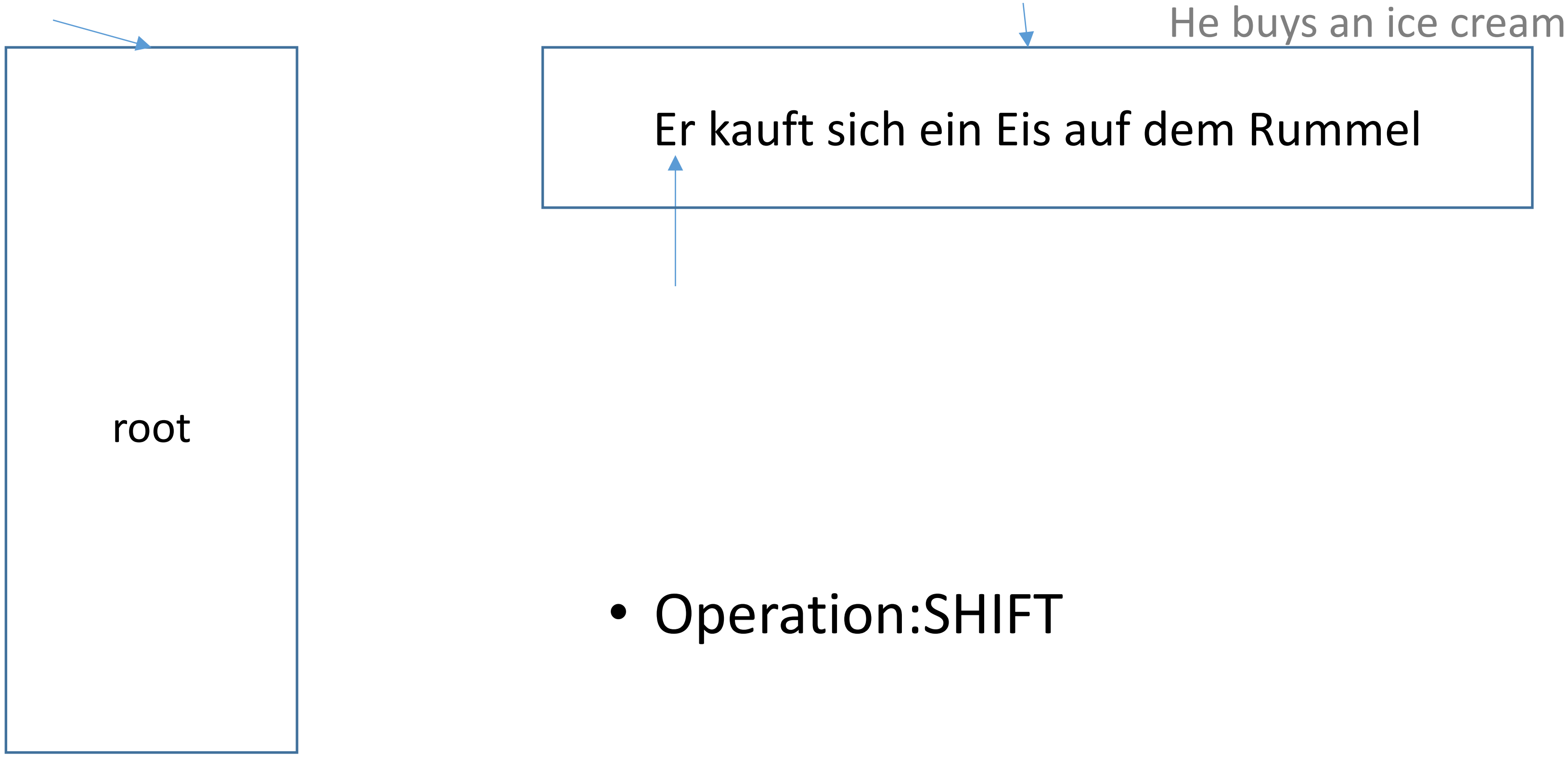
Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

Buffer

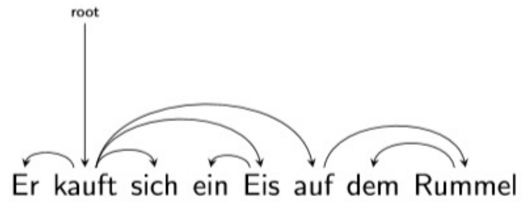


root

Er kauft sich ein Eis auf dem Rummel

- Operation:SHIFT

Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

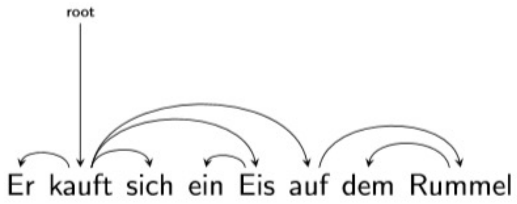
Er
root

Buffer

kauft sich ein Eis auf dem Rummel

- Operation:SHIFT

Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

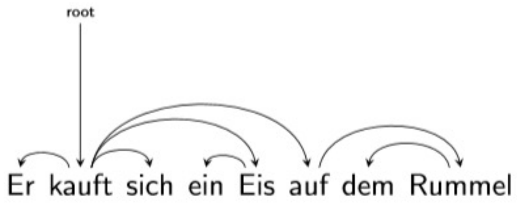
Buffer

sich ein Eis auf dem Rummel

kauft
Er
root

- Operation:SHIFT

Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

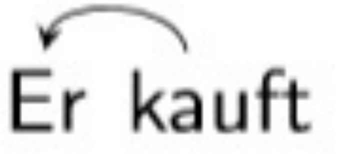
kauft
~~Er~~
root

Buffer

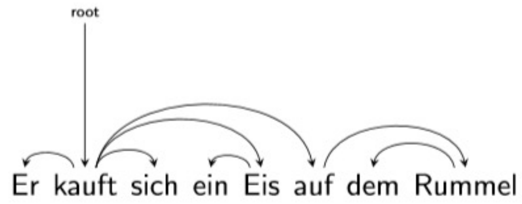
sich ein Eis auf dem Rummel

- Operation:LEFTARC

- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

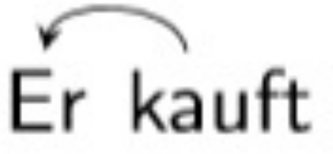
sich
kauft
root

Buffer

ein Eis auf dem Rummel

- Operation:SHIFT

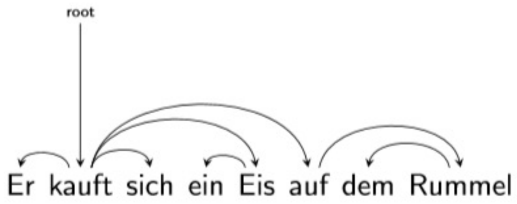
- Result:



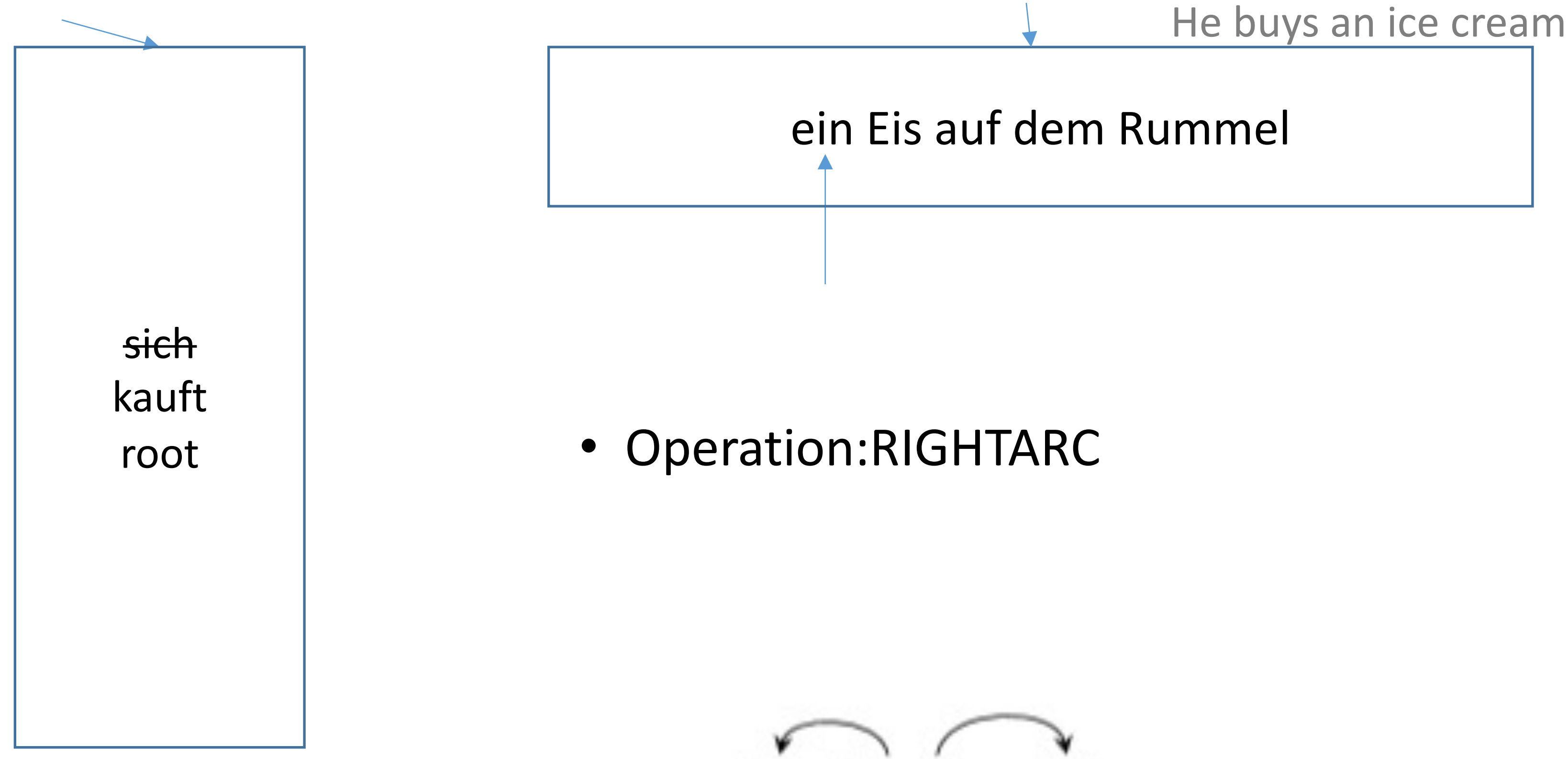
Shift Reduce Parsing

Stack

Buffer



He buys an ice cream at the carnival

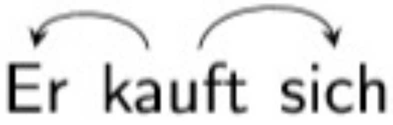


~~sich~~
kauft
root

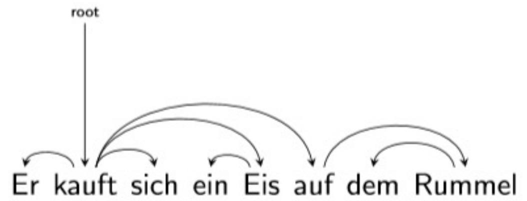
ein Eis auf dem Rummel

- Operation:RIGHTARC

- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

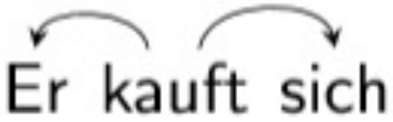
ein
kauft
root

Buffer

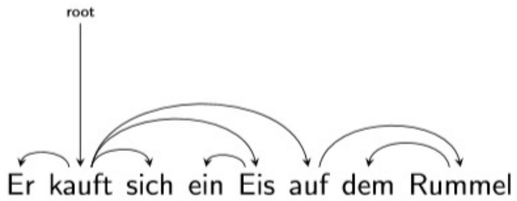
Eis auf dem Rummel

- Operation:SHIFT

- Result:



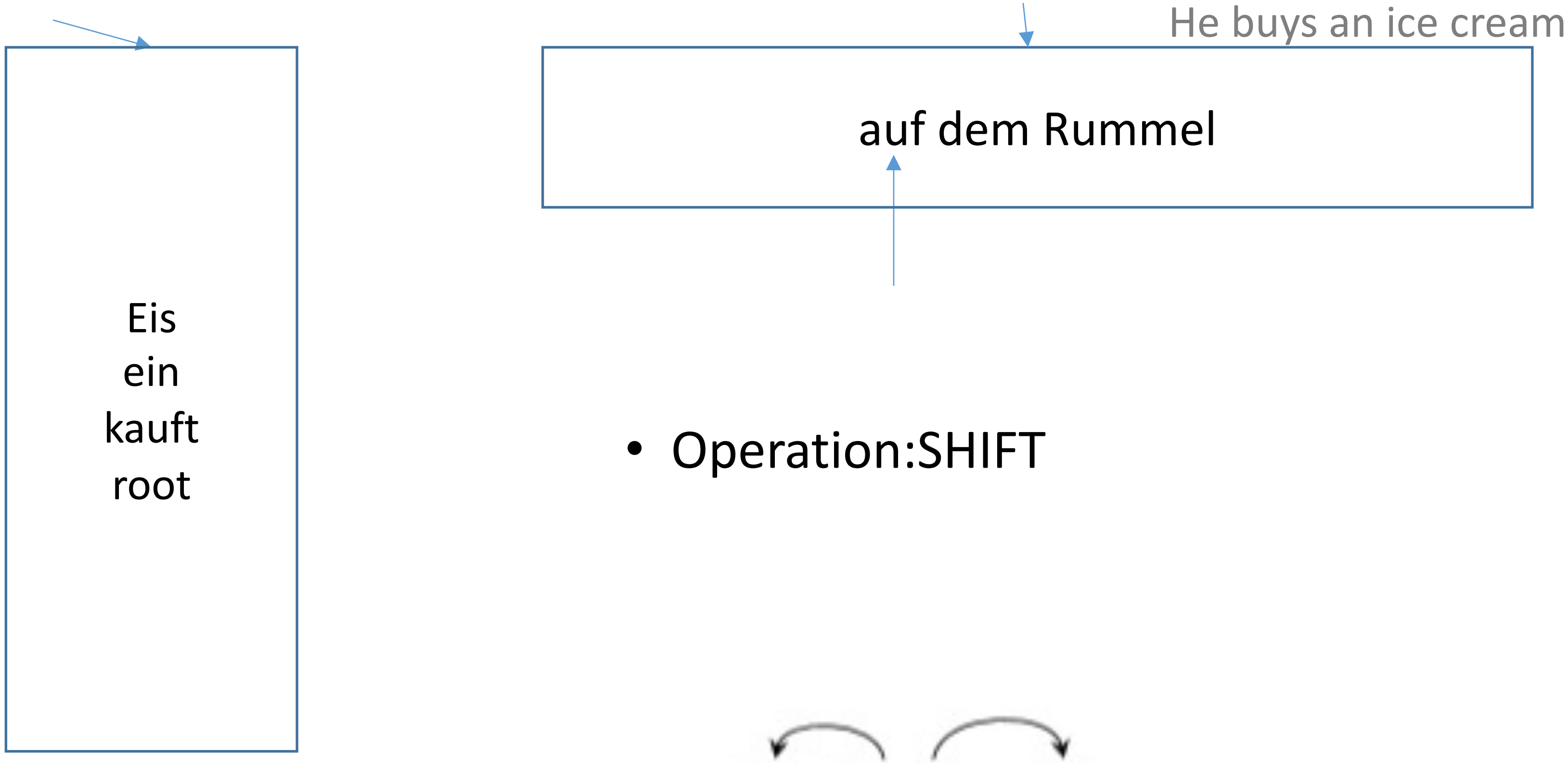
Shift Reduce Parsing



He buys an ice cream at the carnival

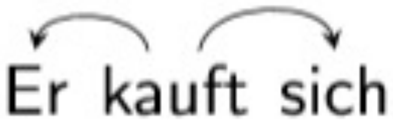
Stack

Buffer

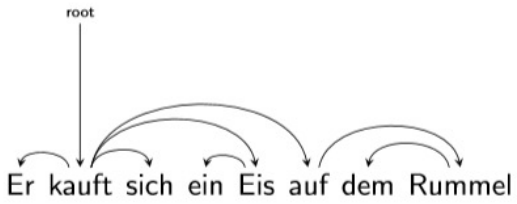


- Operation:SHIFT

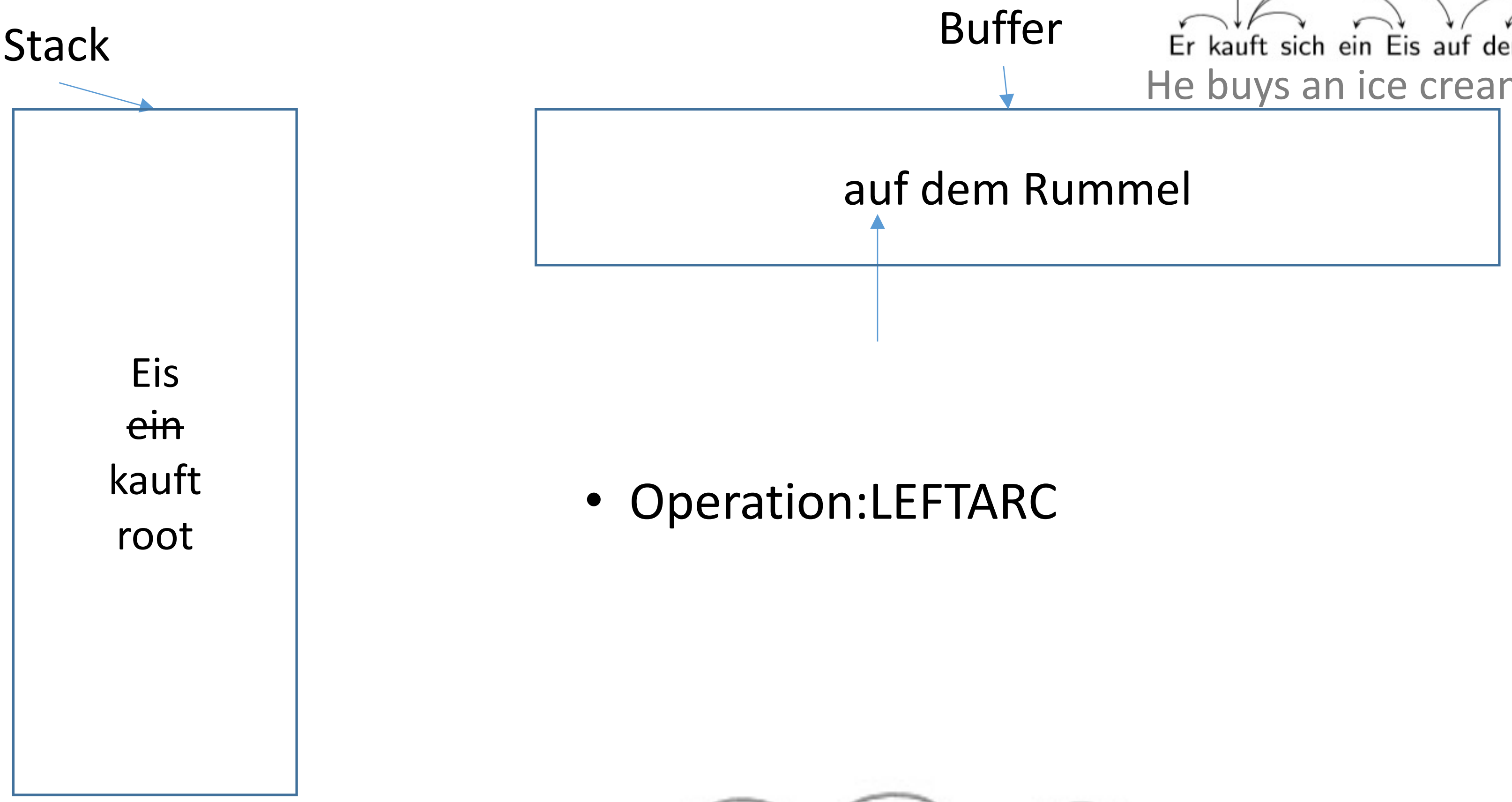
- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

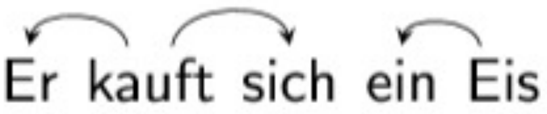


auf dem Rummel

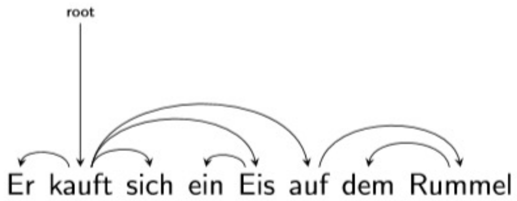
Eis
~~ein~~
kauft
root

- Operation:LEFTARC

- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

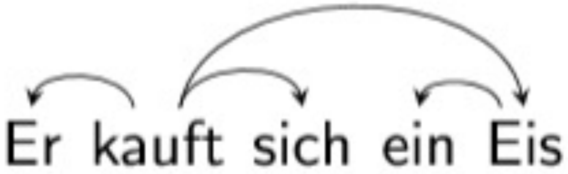
Buffer

auf dem Rummel

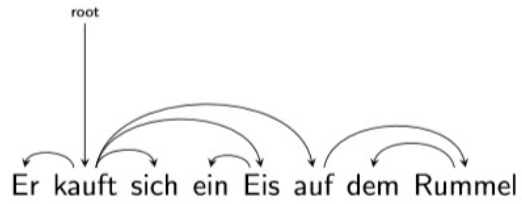
~~Eis~~
kauft
root

- Operation:RIGHTARC

- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

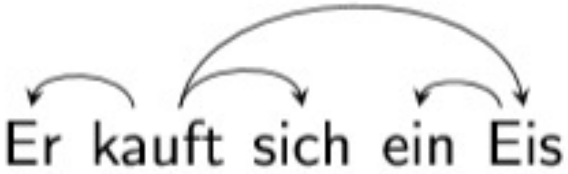
auf
kauft
root

Buffer

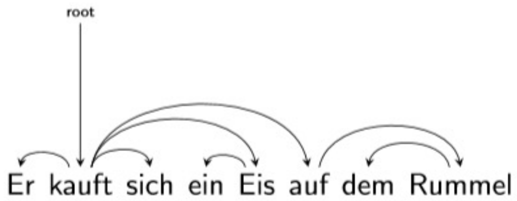
dem Rummel

- Operation:SHIFT

- Result:



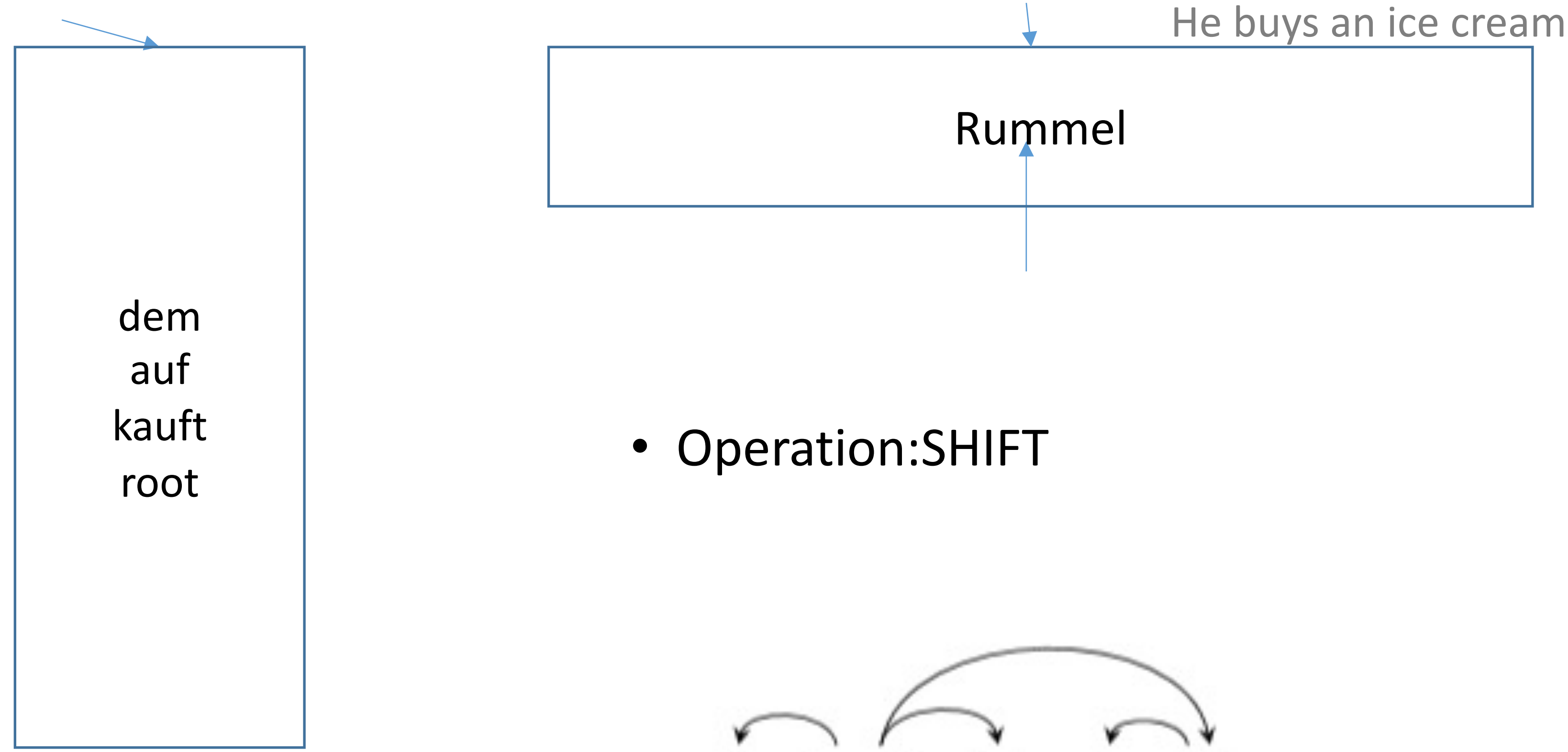
Shift Reduce Parsing



He buys an ice cream at the carnival

Stack

Buffer

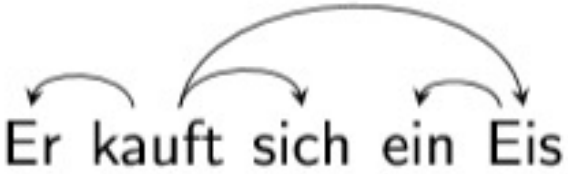


Rummel

dem
auf
kauft
root

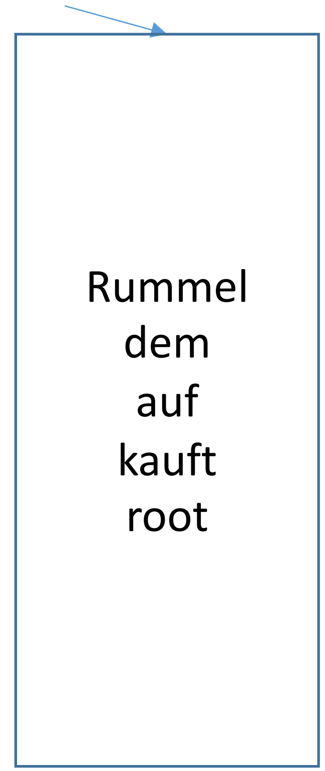
- Operation:SHIFT

- Result:



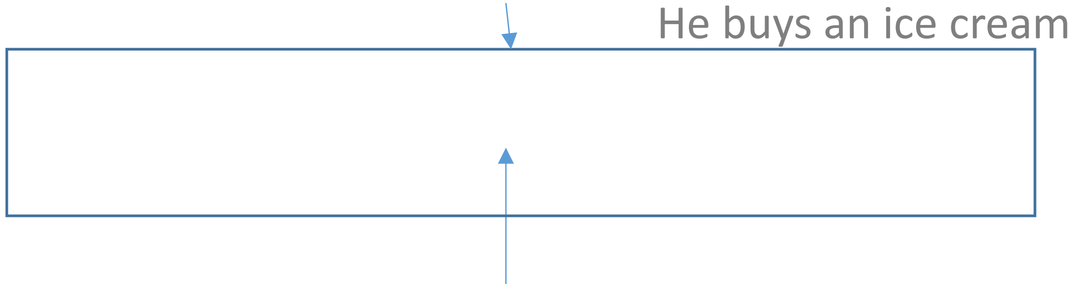
Shift Reduce Parsing

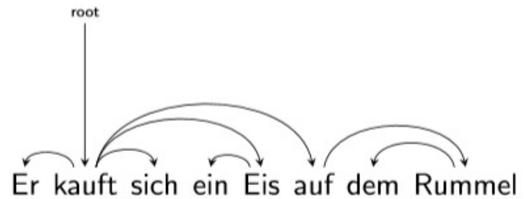
Stack



Rummel
dem
auf
kauft
root

Buffer

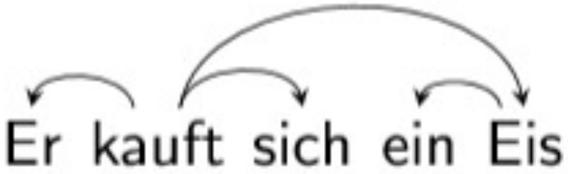




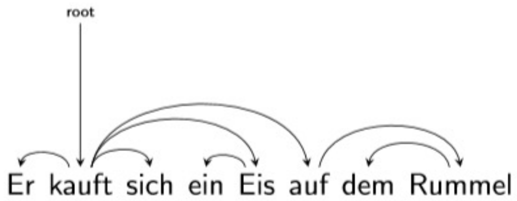
He buys an ice cream at the carnival

- Operation:SHIFT

- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

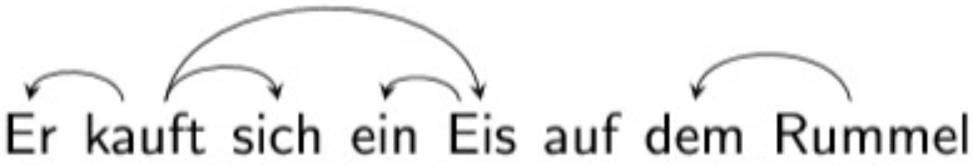
Stack

Buffer

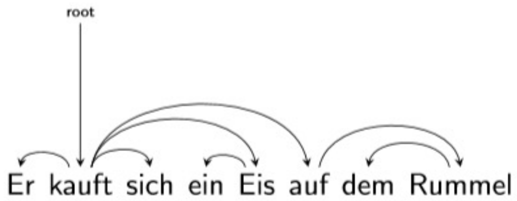
Rummel
~~dem~~
auf
kauft
root

- Operation:LEFTARC

- Result:



Shift Reduce Parsing



He buys an ice cream at the carnival

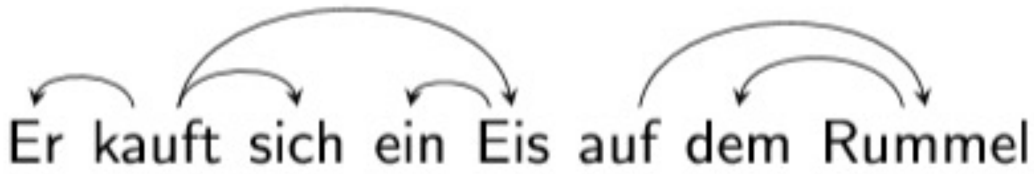
Stack

Buffer

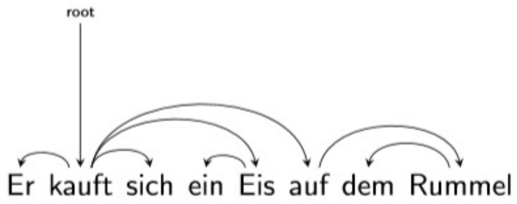
~~Rummel~~
auf
kauft
root

- Operation:RIGHTARC

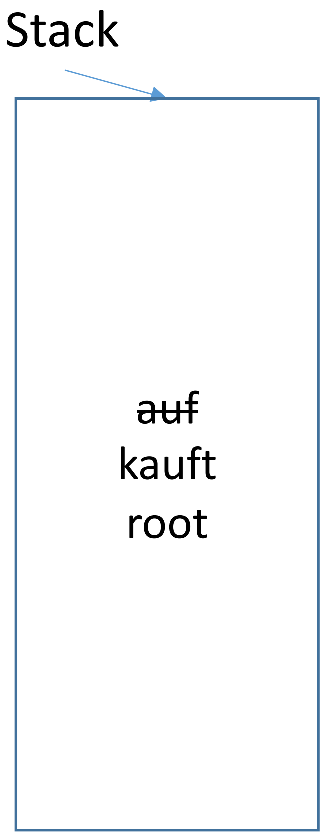
- Result:



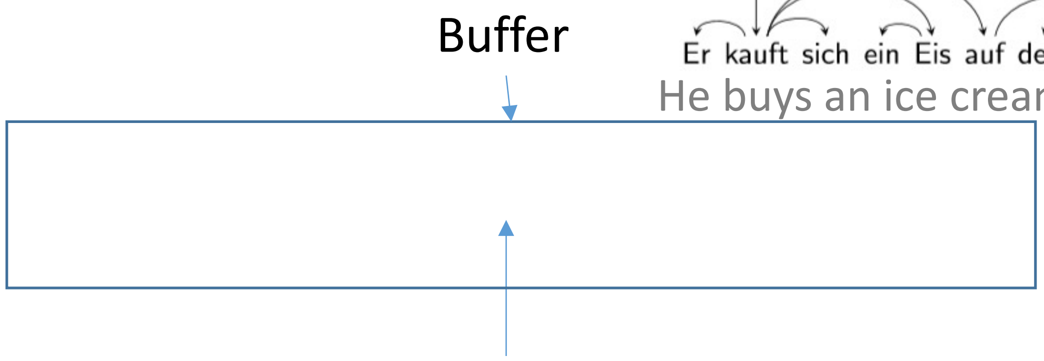
Shift Reduce Parsing



He buys an ice cream at the carnival

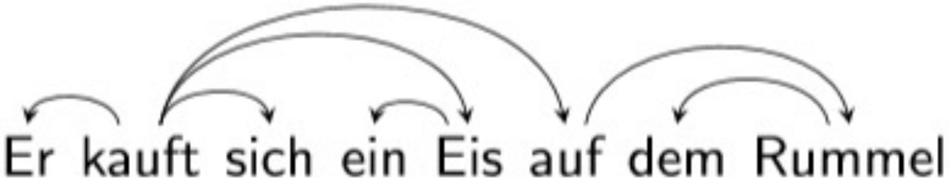


~~auf~~
kauft
root



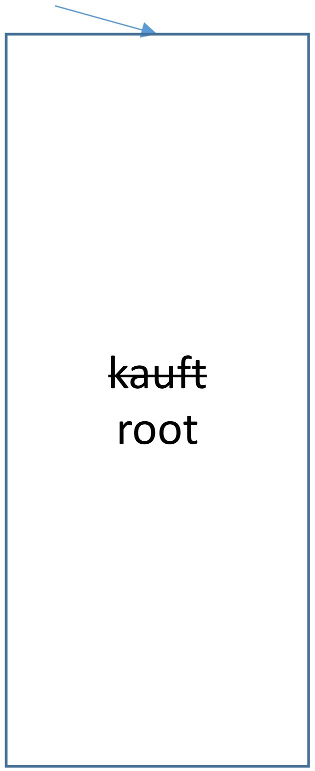
- Operation:RIGHTARC

- Result:



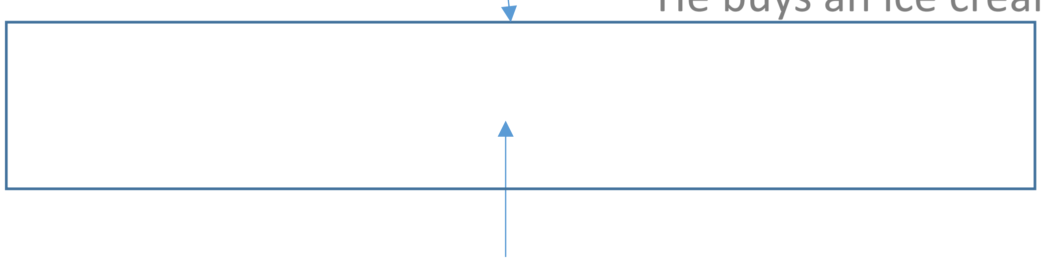
Shift Reduce Parsing

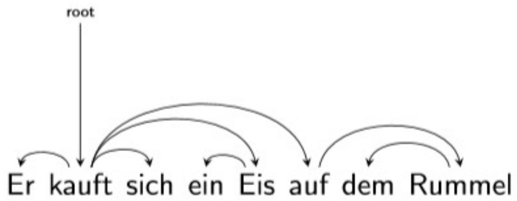
Stack



~~kauft~~
root

Buffer

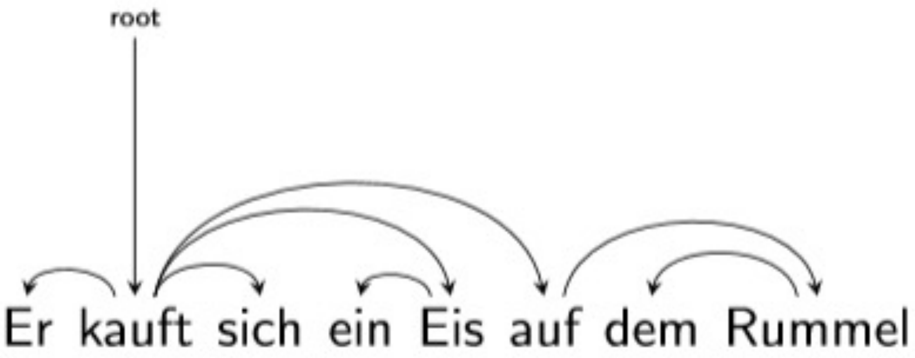




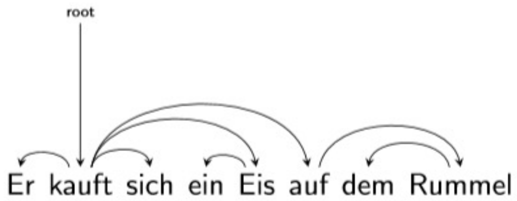
He buys an ice cream at the carnival

- Operation:RIGHTARC

- Result:



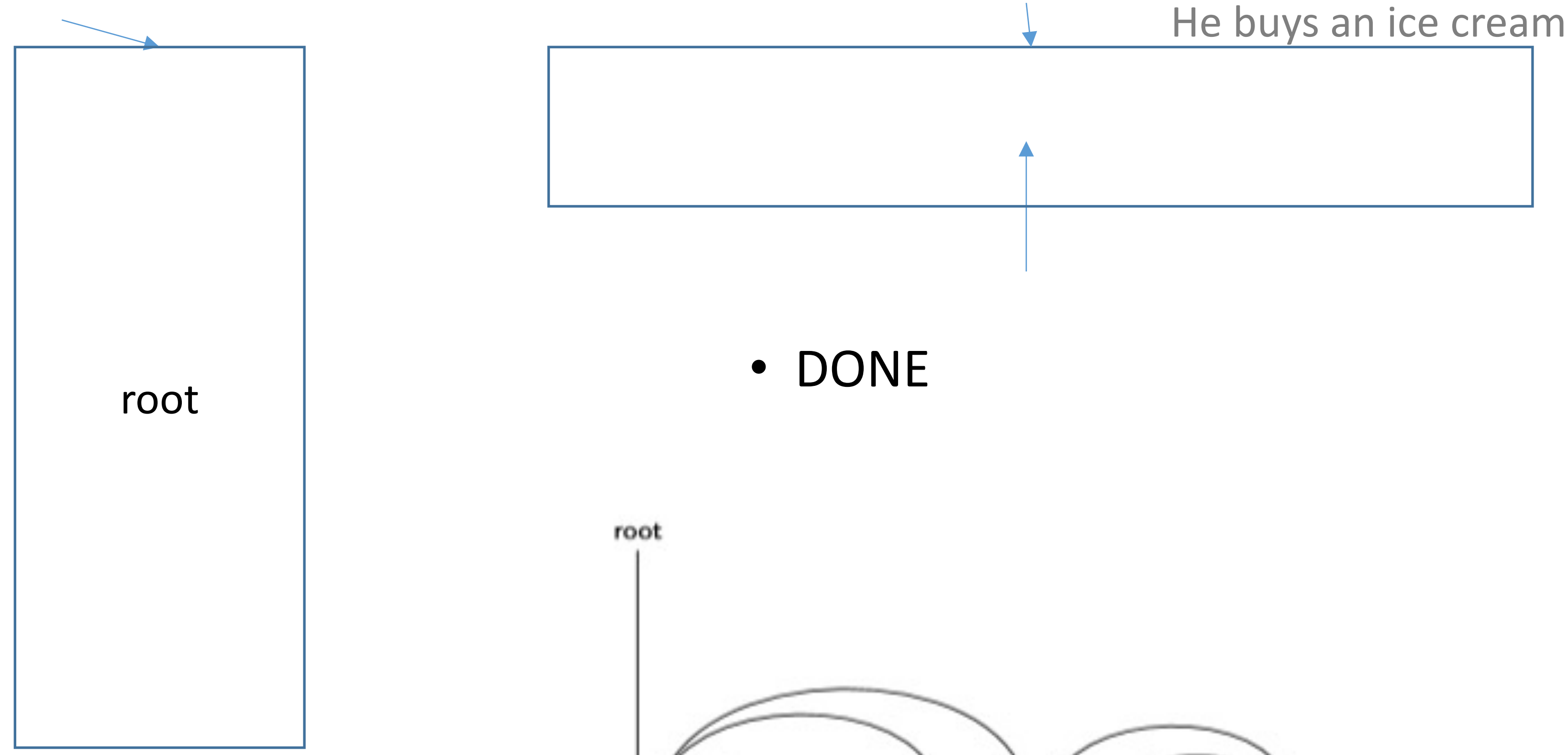
Shift Reduce Parsing



He buys an ice cream at the carnival

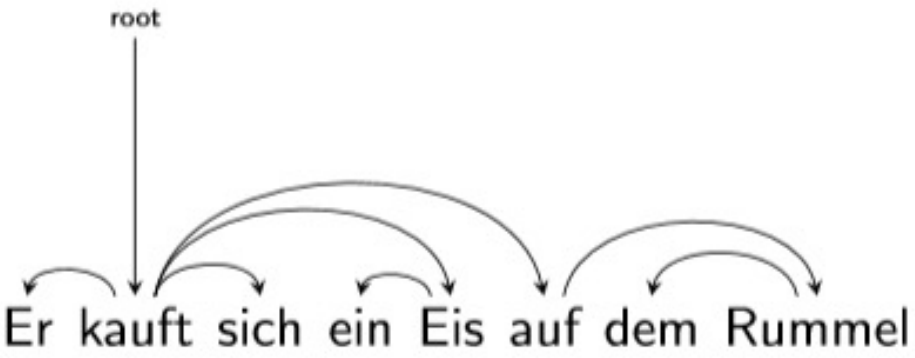
Stack

Buffer



- DONE

- Result:



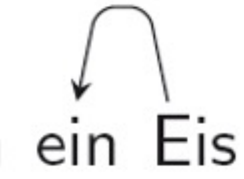
Shift Reduce Parsing: Arc Standard

- How to use this for Dependency Parsing?

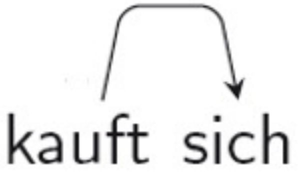
➔Use an adjusted set of operations!

➔ Arc-Standard

1. **LEFTARC**: Add an edge between the first token of the stack and the second token of the stack. Delete the second token from the stack

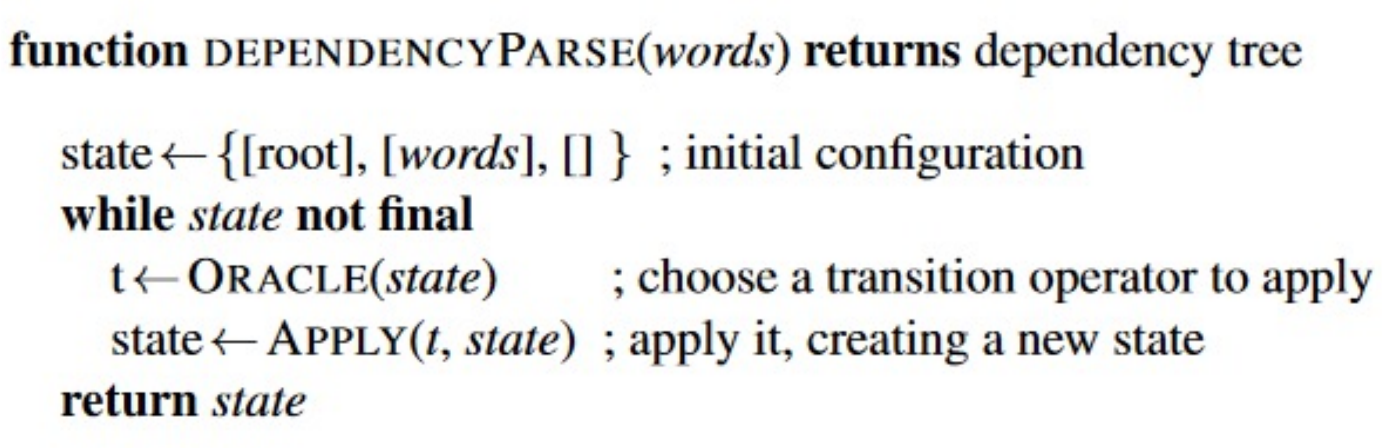


2. **RIGHTARC**: Add an edge between the second token of the stack and the first token of the stack. Delete the first token from the stack.



3. **SHIFT**: Move the first token from the buffer onto the stack

Dependency Parsing – Greedy Parse



```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)          ; choose a transition operator to apply
    state ← APPLY(t, state)  ; apply it, creating a new state
  return state
```

Dependency Parsing – Oracle

- Features:
  - Use arbitrary features from the stack, the buffer, or the edges we produced already

  - Usually features are of the sort:
    - text of Stack[0] and operation $< s_0.text, op >$
    - Text Stack[1] und operation $< s_1.text, op >$
    - POS-Tag of Stack[0] and operation $< s_0.pos, op >$
    - POS-Tag Stack[1] and operation $< s_1.pos, op >$
    - text of Buffer[0] and operation $< b_0.text, op >$
    - Text Buffer[1] and operation $< b_1.text, op >$
    - POS-Tag of Buffer[0] and operation $< b_0.pos, op >$
    - POS-Tag Buffer[1] und operation $< b_1.pos, op >$
    - …

Recap: Shift Reduce Parsing

- Grammarless way of parsing, but it relies on the quality of its oracle
- Any classifier can be used as an oracle (e.g. SVM, MaxEnt)
- Makes use of a clever data structure consisting of a stack and a buffer
- Input is the sentence, split into tokens
- Output is the set of operations:
    - Which is also the main way of adapting this kind of parser, you could add more operations (e.g. a SWAP)
- If you want to train such a classifier, you have to convert the tree bank into a suitable sequence of operations beforehand.
- If you want to predict Dependency-Relations, you just add them to the Operations: LEFTARC and RIGHTARC

➔ Greedy parsing has a linear runtime $O(n)$, with n tokens in the sentence