Parsing with First-Class Derivatives

OOPSLA 2016

Jonathan Brachthäuser
Tillmann Rendel
Klaus Ostermann

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

```
Some *text* that
~~~
function f() {
~~~
interrupts the code.
~~~
}
~~~
```

```
while x < 10:
··y += x
··x += 1
```

```
+-----+------+
|Hello|ASCII |
|World|Tables|
+-----+------+
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Some *text* that

~~~

```
function f() {
```

~~~

interrupts the code.

~~~

```
}
```

~~~

```
while x < 10:
··y += x
··x += 1
```

```
+-----+------+
|Hello|ASCII |
|World|Tables|
+-----+------+
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

```
Some *text* that
~~~
function f() {
~~~
interrupts the code.
~~~
}
~~~
```

```
+-----+------+
|Hello|ASCII |
|World|Tables|
+-----+------+
```

```
while x < 10:
··y += x
··x += 1
```

➡ Input to content parsers do not form a **continuous substream**!

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

The quest for more **modular** and **reusable** parser components.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Agenda

- Parsing with derivatives

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Agenda

- Parsing with derivatives
- Parser combinators

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Agenda

- Parsing with derivatives
- Parser combinators
- Derivation as a new parser combinator

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Agenda

- Parsing with derivatives
- Parser combinators
- Derivation as a new parser combinator
- Examples of gained expressivity

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with Derivatives

$$D_a(a \cdot a \cdot b + b) = a \cdot b$$

$$\mathcal{L}(D_c(\mathrm{r})) = \{w \mid cw \in \mathcal{L}(\mathrm{r})\}$$

1964

Brzozowski
Derivatives of Regular Expressions

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with Derivatives

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with Derivatives

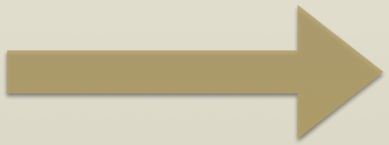
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with Derivatives

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with First-Class Derivatives

$D_c(\mathrm{p})$  p « c

Semantics *Syntax*

„Make the (Brzozowski) derivative of a parser available to parser developers.“

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parser Combinators

```
AS ::= ['a' AS]
```

```
def as: Parser[Int] = …
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parser Combinators

```
AS ::= ['a' AS]
```

```
def as: Parser[Int] =
  ( succeed(0)
  | …
  )
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parser Combinators

```
AS ::= ['a' AS]
```

```
def as: Parser[Int] =
  ( succeed(0)
  | 'a' ~ as
  )
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parser Combinators

```
AS ::= ['a' AS]
```

```
def as: Parser[Int] =
  ( succeed(0)
  | ('a' ~> as) map { n => n + 1 }
  )
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parser Combinators

```
AS ::= ['a' AS]
```

```
def as: Parser[Int] =
  ( succeed(0)
  | ('a' ~> as) map { n => n + 1 }
  )
```

```
> as parse "aaa"
```

➥ 3

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Derivation as Parser Combinator


```
def « [R](p: Parser[R], c: Char) : Parser[R]
```

```
> (as « 'a' « 'a' « 'a')  parse ""
```
➥ 3

```
> (as « 'a' « 'a' « 'a')  parse "aaa"
```
➥ 6

```
> (as «« "aaa") parse "aaa"
```
➥ 6

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Derivation as Parser Combinator

"derive" / "feed"

```
def « [R](p: Parser[R], c: Char) : Parser[R]
```

$$\mathcal{L}(D_c(\mathrm{r})) = \{w \mid cw \in \mathcal{L}(\mathrm{r})\}$$

```
(p « c) parse w = p parse (c + w)
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Reuse by Selection: An Application

Defined in some other module:

```
val whileStmt: Parser[WhileStmt] = …
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Reuse by Selection: An Application

Defined in some other module:

```
val whileStmt: Parser[WhileStmt] = …
```

Reuse by selection:

```
val untilStmt: Parser[UntilStmt] =
  "until" ~> (whileStmt <~ "while") map { … }
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with First-Class Derivatives enables **Reuse by Selection**

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

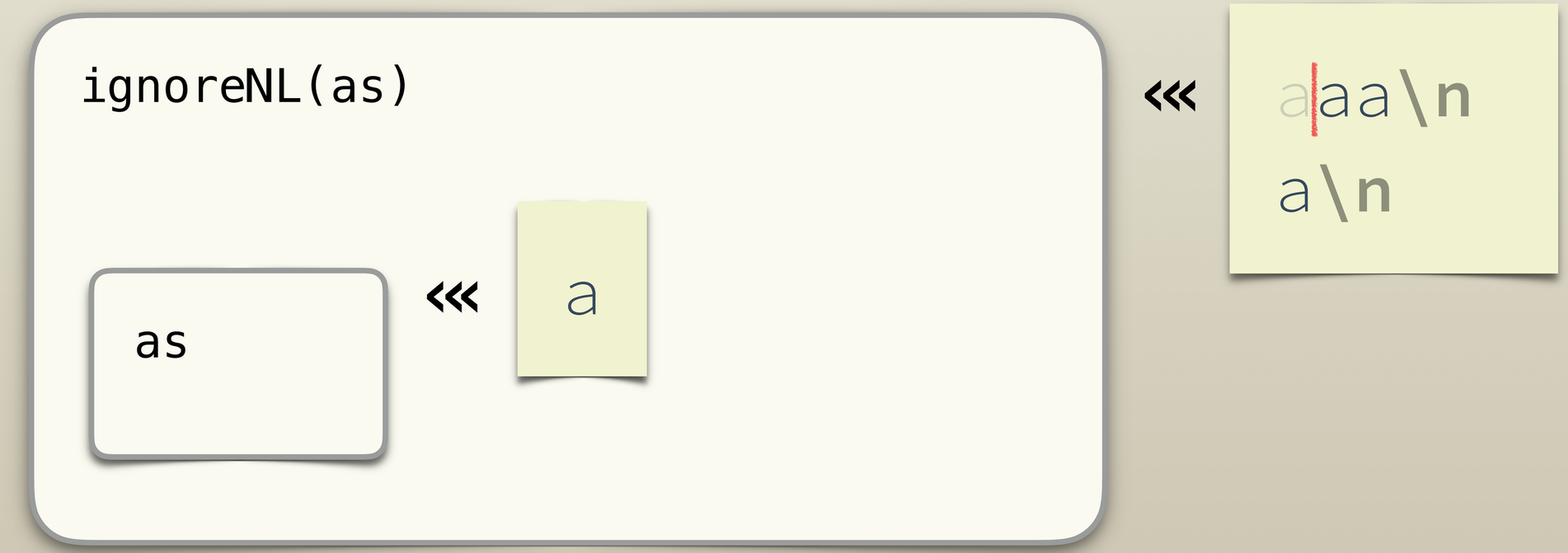
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```

«« aaa\n
a\n

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

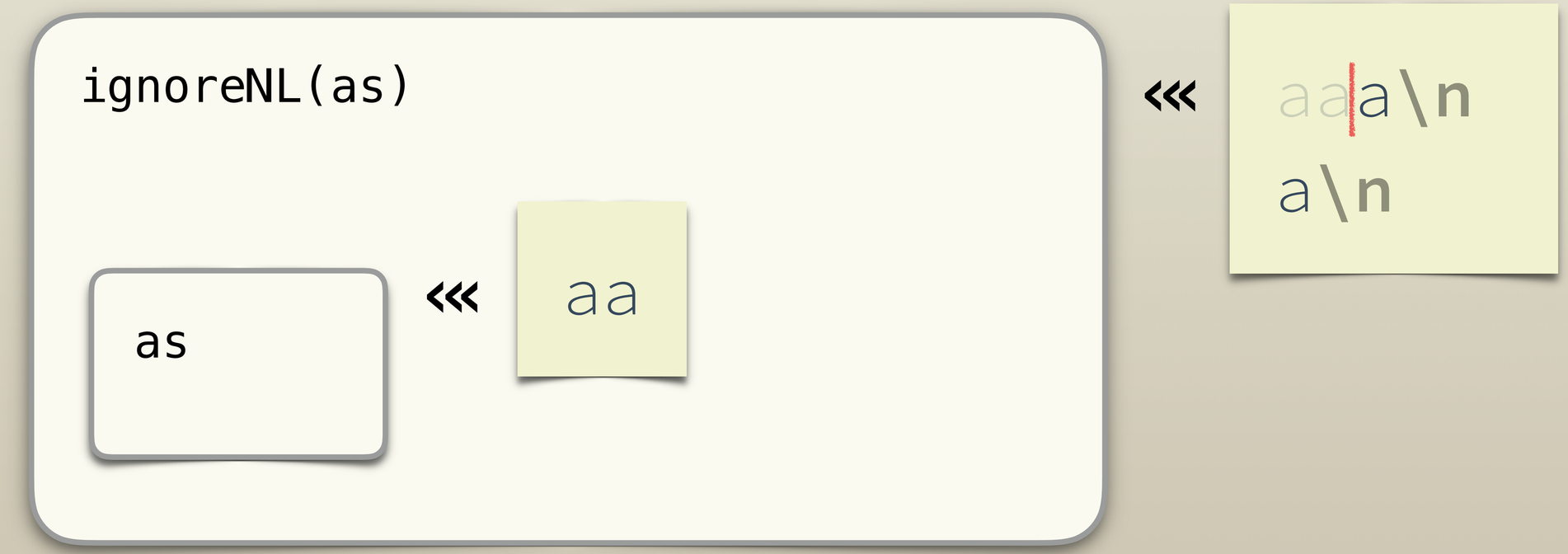
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

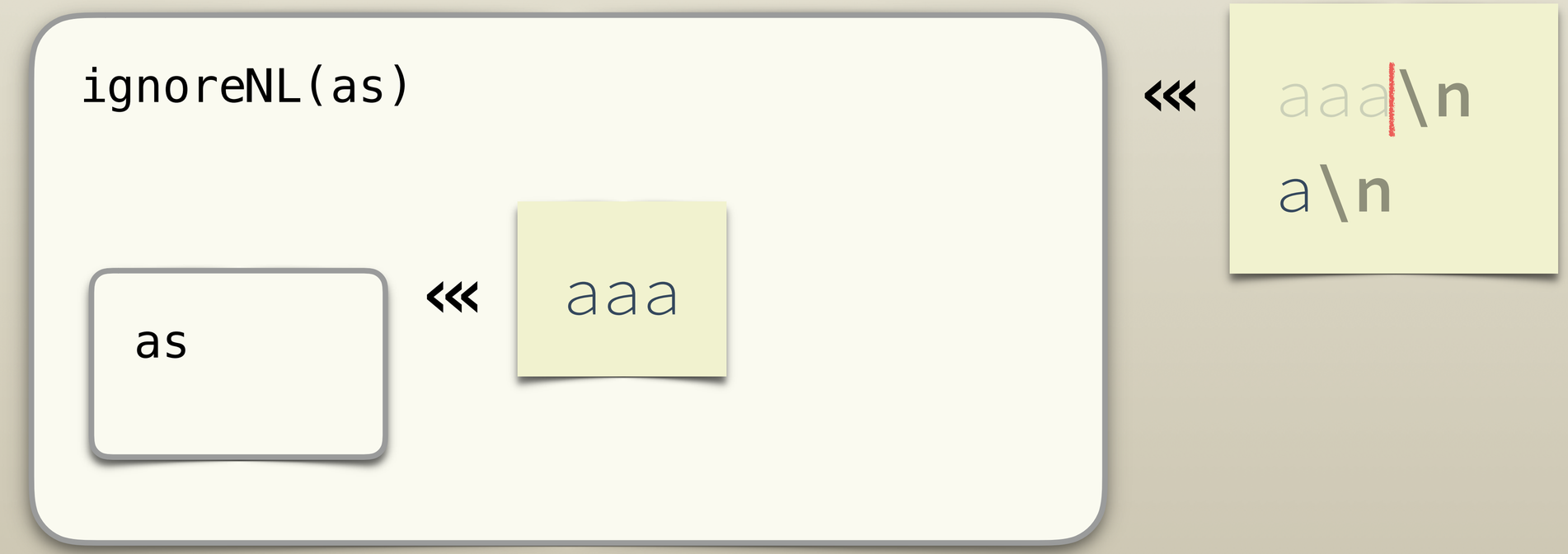
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

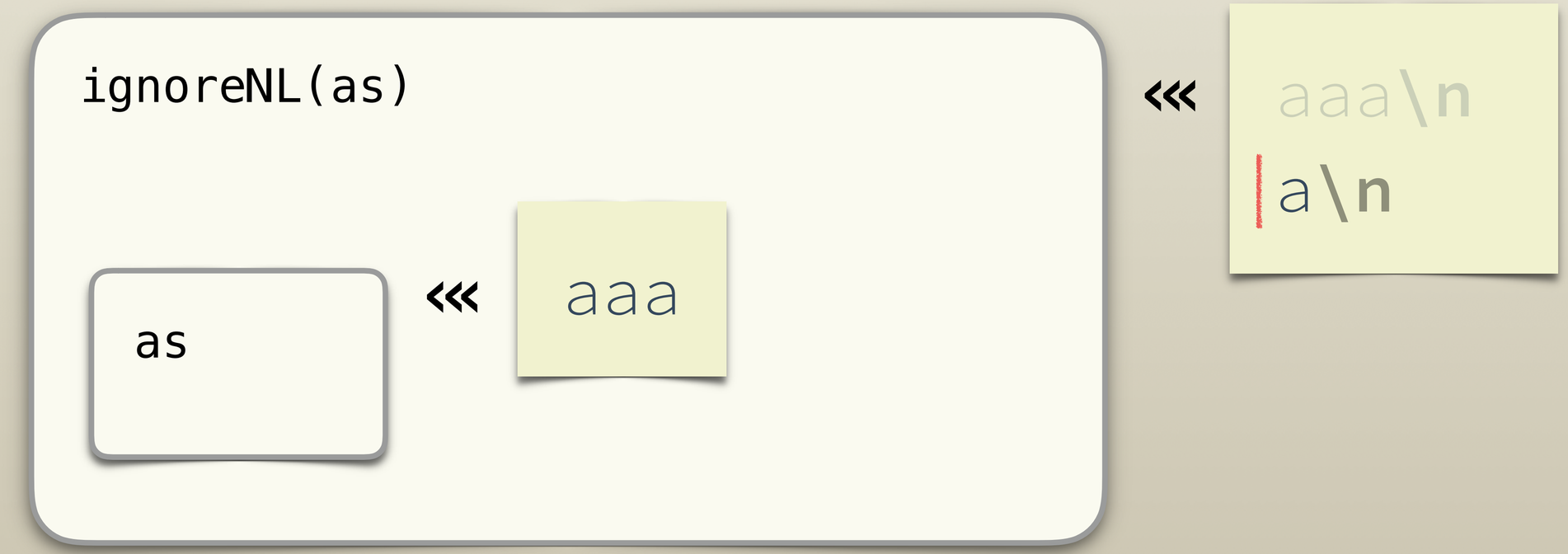
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

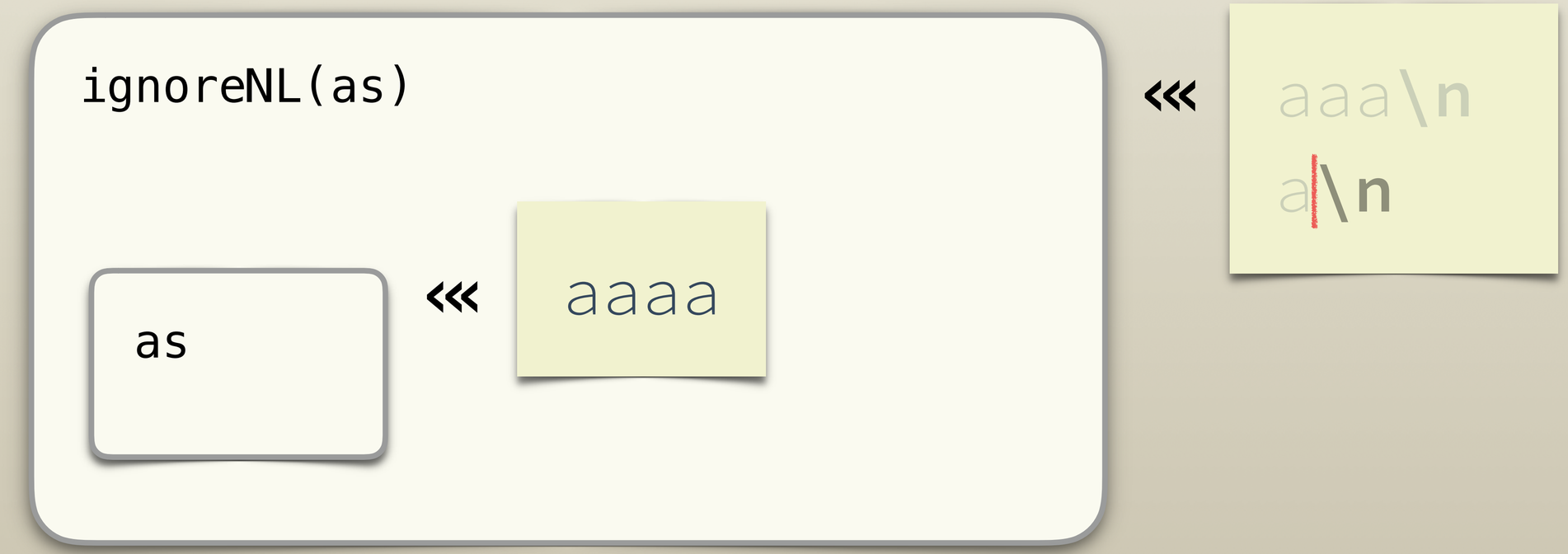
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

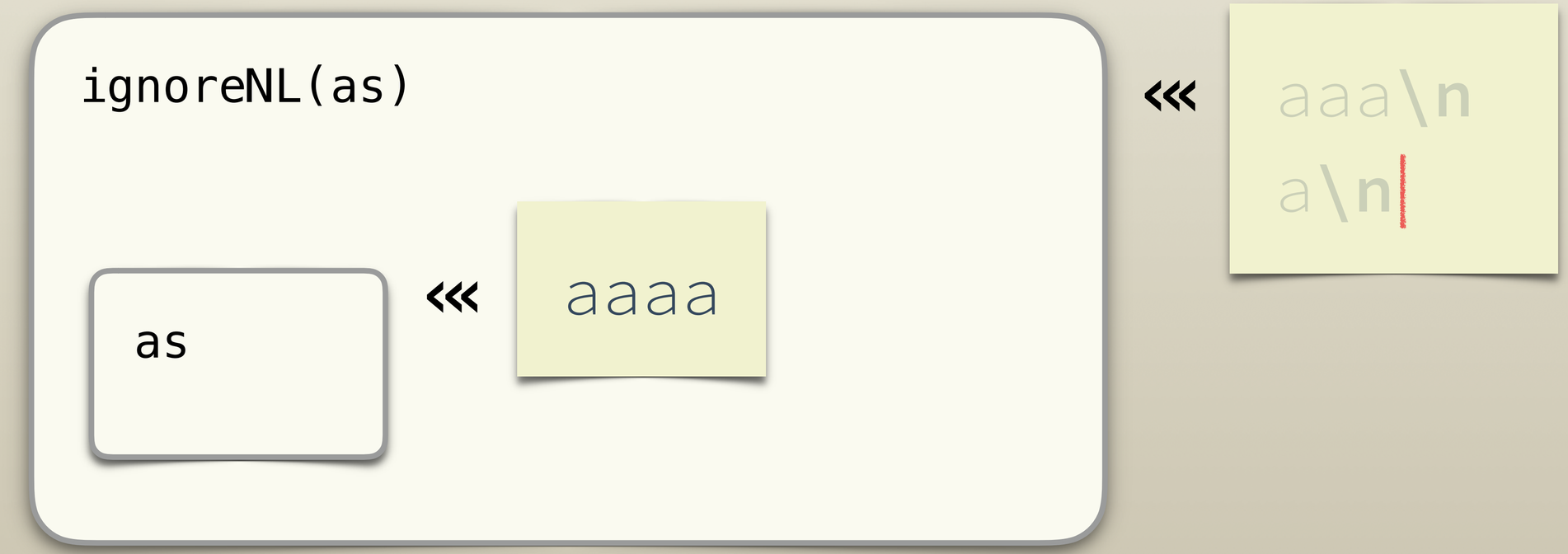
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

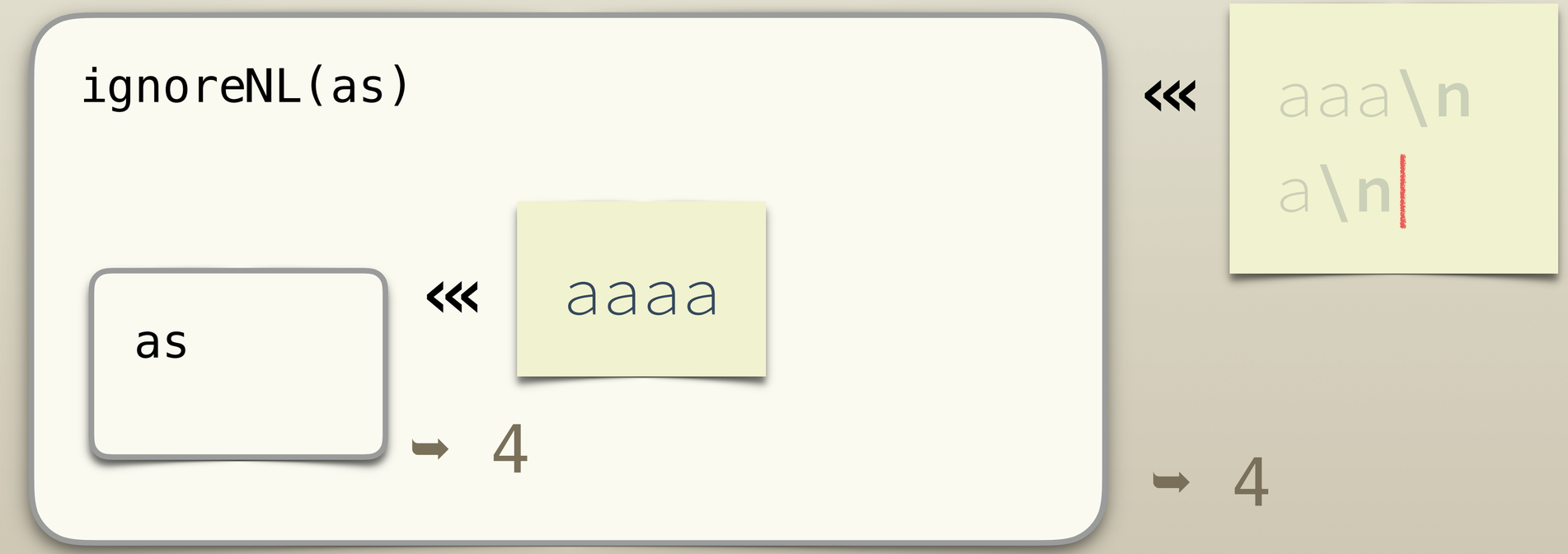
```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator

```
def ignoreNL[R](p: Parser[R]): Parser[R]
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Termination as Parser Combinator

```
def done[R](p: Parser[R]) : Parser[R]
```

```
> done(as «« "aaa") parse ""
```
➥ 3

```
> done(as «« "aaa") parse "a"
```
➥ parse error

```
> (done(as) «« "aaa") parse ""
```
➥ parse error

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator: Implementation

```
def ignoreNL[R](p: Parser[R]): Parser[R] =
  ( done(p)
  | …
  | …
  )
```

```
val aLines: Parser[Int] = ignoreNL(as)
```

```
> aLines parse ""
```

➥ 0

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator: Implementation

```
def ignoreNL[R](p: Parser[R]): Parser[R] =
  ( done(p)
  | '\n' ~> ignoreNL(p)
  | …
  )
```

```
> aLines parse "\n"
```

➥ 0

```
> aLines parse "\n\n"
```

➥ 0

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator: Implementation

```
def ignoreNL[R](p: Parser[R]): Parser[R] =
  ( done(p)
  | '\n' ~> ignoreNL(p)
  | no('\n') flatMap { c ⇒ ignoreNL(p « c) }
  )
```

no: Char => Parser[Char]

Here the input delegation takes place!

EBERHARD KARLS UNIVERSITÄT TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator: Implementation

```
def ignoreNL[R](p: Parser[R]): Parser[R] =
  ( done(p)
  | '\n' ~> ignoreNL(p)
  | no('\n') flatMap { c ⇒ ignoreNL(p « c) }
  )
```

```
> aLines parse "a"
```

➥ 1

```
> aLines parse "\na\na\na"
```

➥ 3

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator: An Application

Unescape derives the delegatee by unescaped characters

```
def unescape[R](p: Parser[R]): Parser[R] =
  done(p) | "\\n" ~> unescape(p « '\n') | …
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Preprocessing as a Combinator: An Application

Unescape derives the delegatee by unescaped characters

```
def unescape[R](p: Parser[R]): Parser[R] =
  done(p) | "\\n" ~> unescape(p « '\n') | …
```

```
val regexString = '"' ~> unescape(regex) <~ '"'
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with First-Class Derivatives enables **Local Stream-Preprocessing**

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator

Simple variant of indentation sensitivity:

```
while·x·<·10:↵
··y·+=·x↵
··x·+=·1↵
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator

Simple variant of indentation sensitivity:

```
··y·+=·x↵
··x·+=·1↵
```

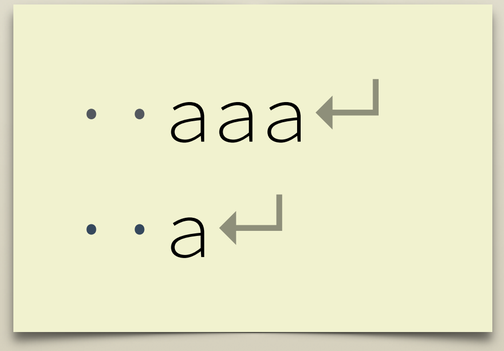

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator

Simple variant of indentation sensitivity:

```
indented(aLines)  parse
```

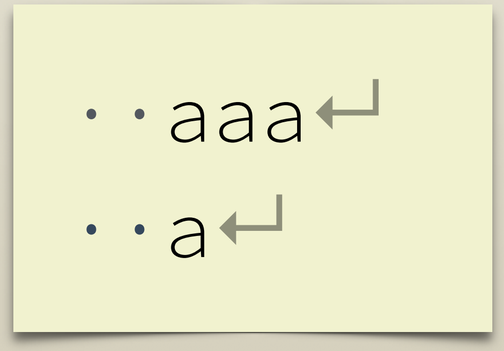


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

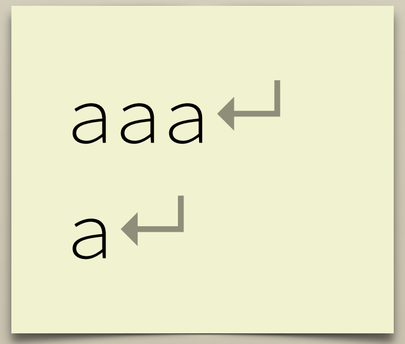
Indentation as a Combinator

Simple variant of indentation sensitivity:

```
indented(aLines)  parse
```



```
aLines            parse
```



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator

Simple variant of indentation sensitivity:

```
aLines        parse
```

aaa↵
a↵

≡

```
as            parse
```

aaaa

➥ 4

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator: Implementation

Simple variant of indentation sensitivity:

```
def indented[T](p: Parser[T]): Parser[T] =
  done(p) | (space ~ space) ~› readLine(p)
```

```
··aaa\n
··a\n
```

```
def readLine[T](p: Parser[T]): Parser[T] =
  ( no('\n') flatMap { c ⇒ readLine(p « c) }
  | '\n'     flatMap { c ⇒ indented(p « c) }
  )
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator: Implementation

Simple variant of indentation sensitivity:

```
def indented[T](p: Parser[T]): Parser[T] =
  done(p) | (space ~ space) ~› readLine(p)
```

```
··aaa\n
··a\n
```

```
def readLine[T](p: Parser[T]): Parser[T] =
  ( no('\n') flatMap { c ⇒ readLine(p « c) }
  | '\n'     flatMap { c ⇒ indented(p « c) }
  )
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator: Implementation

Simple variant of indentation sensitivity:

```
··aaa\n
··a\n
```

```
def indented[T](p: Parser[T]): Parser[T] =
  done(p) | (space ~ space) ~› readLine(p)
```

```
def readLine[T](p: Parser[T]): Parser[T] =
  ( no('\n') flatMap { c ⇒ readLine(p « c) }
  | '\n'     flatMap { c ⇒ indented(p « c) }
  )
```

```
> indented(aLines) parse "  a\n  a\n"
```

➥ 2

EBERHARD KARLS UNIVERSITÄT TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation as a Combinator: Implementation

Simple variant of indentation sensitivity:

```
··aaa\n
··a\n
```

```
def indented[T](p: Parser[T]): Parser[T] =
  done(p) | (space ~ space) ~› readLine(p)
```

```
def readLine[T](p: Parser[T]): Parser[T] =
  ( no('\n') flatMap { c ⇒ readLine(p « c) }
  | '\n'     flatMap { c ⇒ indented(p « c) }
  )
```

```
> indented(indented(aLines))
    parse "    a\n    a\n"
```

➥ 2

EBERHARD KARLS UNIVERSITÄT TÜBINGEN
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing with First-Class Derivatives enables **Suspending Delegation**

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

In the Paper

Parsing with First-Class Derivatives

Jonathan Immanuel Brachthäuser Tillmann Rendel Klaus Ostermann
University of Tübingen, Germany
{brachthaeuser, rendel, ostermann}@informatik.uni-tuebingen.de

Abstract

Brzozowski derivatives, well known in the context of regular expressions, have recently been rediscovered to give a simplified explanation to parsers of context-free languages. We add derivatives as a novel first-class feature to a standard parser combinator language. First-class derivatives enable an inversion of the control flow, allowing to implement modular parsers for languages that previously required separate preprocessing steps or cross-cutting modifications of the parsers. We show that our framework offers new opportunities for reuse and supports a modular definition of interesting use cases of layout-sensitive parsing.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Syntax; D.3.4 [*Programming Languages*]: Processors—Parsing; F.4.2 [*Mathematical Logic and Formal Languages*]: Grammars and Other Rewriting Systems—Parsing; F.4.3 [*Mathematical Logic and Formal Languages*]: Formal Languages—Operations on languages

General Terms Algorithms, Languages

Keywords Parsing; Modularity; Derivative; Left Quotient; Parser Combinators

1. Introduction

The theory and practice of context-free grammars is well-developed, and they form the foundation for most approaches to parsing of computer languages. Unfortunately, some syntactic features of practical interest are inherently not context-free. Examples include indentation-sensitivity as can be found in Haskell or Python or two-dimensional grid tables as in some Markdown dialects. To parse languages with such inherently non-context-free features, language implementors have to resort to ad-hoc additions to their parsing approach, often reducing the modularity of their parser because these ad-hoc additions are cross-cutting with respect to the otherwise context-free syntactic structure of the language.

Parser combinators [10, 10, 15, 21, 22] are a parsing approach that is well-suited for such ad-hoc additions. With parser combinators, a parser is described as a first-class entity in some host language, and parsers for smaller subsets of a language are combined into parsers of larger subsets using built-in or user-defined combinators such as sequence, alternative, or iteration. The fact that parsers are first-class entities at runtime allows an elegant way of structuring parsers, simply by reusing the modularization features and abstraction mechanisms of the host-language. Despite this potential, the reuse of carefully crafted parsers is often limited: Necessary extension points in terms of nonterminals or exported functions might be missing. At the same time, cross-cutting syntactical features like two-dimensional layout can often not be separated into reusable modules.

We propose to increase the modularity of parser combinators by adding a combinator to compute a parser's Brzozowski derivative. Derivatives, well known in the context of regular expressions, have recently been rediscovered to give a simple explanation to parsers of context-free languages [1, 6, 17, 18]. A parser p derived by a token c is again a parser p' characterized by $\mathcal{L}(p') = \{w \mid cw \in \mathcal{L}(p)\}$. For example, deriving the parser $p =$ `"while"` by the token `'w'` thus yields a parser recognizing the word `"hile"`. Deriving p by any other token yields the empty parser. So far, to the best of our knowledge, Brzozowski derivatives have only been used to describe the semantics of grammars or to implement parsers, but not to extend the language of grammars itself. Similar to prior work we base the definition of our parsers on derivatives. In addition, we introduce first-class derivatives as parser combinators by exposing the derivation operator to the user. First-class derivatives effectively allow user-defined combinators to filter or reorder the token stream, which turns out to increase the modularity of parsers written with the library.

Overall, we make the following contributions:

- We identify the *substream property*, shared by many traditional parser combinator libraries, as the main hurdle precluding modular support for many inherently non-context-free syntactic features (Section 2).

- we abstract over recursive delegation patterns
- we implement combinators for the three motivating examples (and more)
- we extend `indented` to also account for implicit and explicit line joining
- we describe the implementation techniques of our library

Limitations & Future Work

- Improve performance for practicality

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Limitations & Future Work

- Improve performance for practicality
- Determine the language class of CFG extended with FCD

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Limitations & Future Work

- Improve performance for practicality
- Determine the language class of CFG extended with FCD
- Investigate other forms of deriatives

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Limitations & Future Work

- Improve performance for practicality
- Determine the language class of CFG extended with FCD
- Investigate other forms of derivatives
- Combine derivatives with other formalisms, such as PEG

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Conclusion

- FCD enable the **reuse** of a parser also in a context, where the input stream to the parser is not a substream of the physical input stream.
- With FCD, some layout features can be **modularly** described as parser combinators.
- Future work could address mentioned limitations and explore other kinds of derivatives

Thank You!

http://github.com/b-studios/fcd

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

End Of Slides

Paradigm: Delegating to Parsers

We abstract over the process of delegating all inputs to a parser

```
def delegate[T](p: P[T]): P[P[T]]
```

```
delegate(p) flatMap done  = p
delegate(p) « c           = delegate(p « c)
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation revisited

With delegation we can reimplement the indentation combinator

```
val line = many(no('\n')) ~ '\n'
```

```
def indented[T](p: P[T]): P[T] =
  ( done(p)
  | (space ~ space) ~> (line &› delegate(p)) >> indented
  )
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Indentation revisited

With delegation we can reimplement the indentation combinator

```
val line = many(no('\n')) ~ '\n'
```

```
def indented[T](p: P[T]): P[T] =
  ( done(p)
  | (space ~ space) ~> (line &› delegate(p)) >> indented
  )
```

"Delimited Delegation"

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Parsing ASCII tables

```
+-----+------+
|Hello|ASCII |
|World|Tables|
+-----+------+
```

```
type Layout = List[Int] // list of column sizes
```

…

```
def delCells[T](l: Layout, cells: List[P[T]]): List[P[P[T]]] =
  l.zip(cells).map {
    case (n, p) ⇒ delegateN(n, p).map(p ⇒ p « '\n') ‹~ '|'
  }
```

Delegating to two interleaved Parsers

```
val marker = "~~~\n"
```

```
def inText[R, S](code: P[R]): P[S] ⇒ P[(R, S
  ( done(code) & done(text)
  | marker                                  ~›
  | not(marker) & line &› delegate(text) >>
  )
```

```
def inCode[R, S](text: P[S]): P[R] ⇒ P[(R, S)] = code ⇒
  ( marker                                  ~› inText(code)(text)
  | not(marker) & line &› delegate(code) >> inCode(text)
  )
```

```
Some text
~~~
function () {
~~~
Some more text
~~~
}
~~~
```

Delegating to two interleaved Parsers

```
val marker = "~~~\n"
```

```
def inText[R, S](code: P[R]): P[S] ⇒ P[(R, S)] = text ⇒
  ( done(code) & done(text)
  | marker                                  ~› inCode(text)(code)
  | not(marker) & line &› delegate(text) >> inText(code)
  )
```

```
def inCode[R, S](text: P[S]): P[R] ⇒ P[(R, S)] = code ⇒
  ( marker                                  ~› inText(code)(text)
  | not(marker) & line &› delegate(code) >> inCode(text)
  )
```

Implementation of our Library

```
trait P[+R] { p =>
  def results : Res[R]
  def derive  : Elem ⇒ P[R]
  ...
}
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Implementation of our Library

```
trait P[+R] { p =>
  def results : Res[R]
  def derive  : Elem ⇒ P[R]

  def &[S](q: P[S]) = new P[(R, S)] {
    def results = for(r <- p.results, s <- q.results) yield (r, s)
    def derive  = el => (p derive el) & (q derive el)
  }
  ...
}
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Implementation of our Library

```
trait P[+R] { p =>
  def results : Res[R]
  def derive  : Elem ⇒ P[R]

  ...
  def «(el: Elem) = p derive el
}
```


EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Programmiersprachen und Softwaretechnik

Syntax of our Parser Combinator Library

empty lang.	`fail : ∀ R. P[R]`
empty word	`succeed(r) : ∀ R. R ⇒ P[R]`
token select.	`acceptIf(f) : (Elem ⇒ Boolean) ⇒ P[Elem]`
concat.	`p ~ q : ∀ R S. (P[R], P[S]) ⇒ P[(R, S)]`
union	`p | q : ∀ R. (P[R], P[R]) ⇒ P[R]`
map	`p ^^ f : ∀ R S. (P[R], R ⇒ S) ⇒ P[S]`

Boolean Operators

intersection

```
p & q  : ∀ R S. (P[R], P[S]) ⇒ P[(R, S)]
```

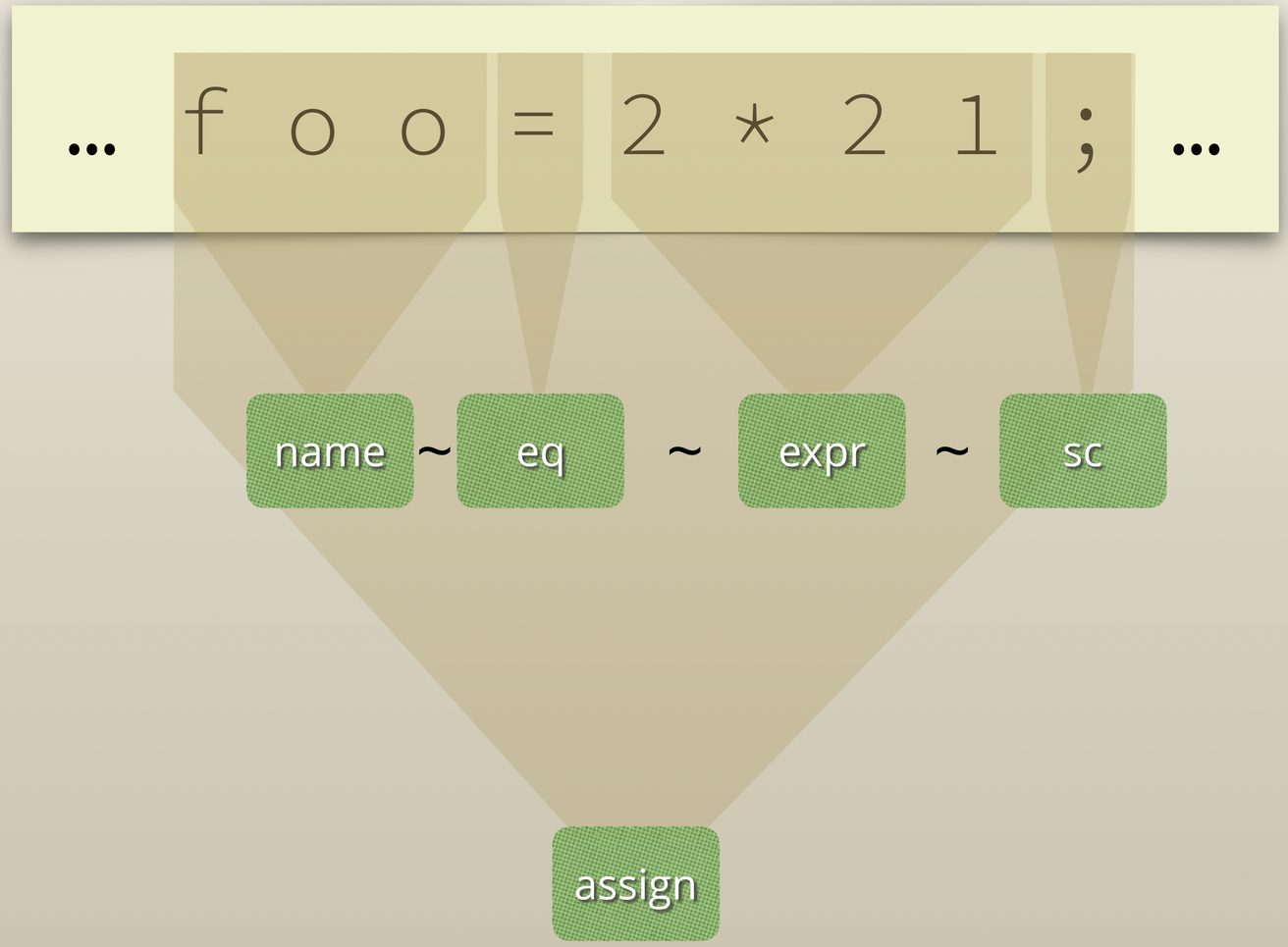
complement

```
not(p) : ∀ R. P[R] ⇒ P[Unit]
```

universal

```
always : P[Unit]
```

Substream Property



„A parsers virtual input stream corresponds to a continuous substream of the original input stream.“

Limitations implied by the Substream Property

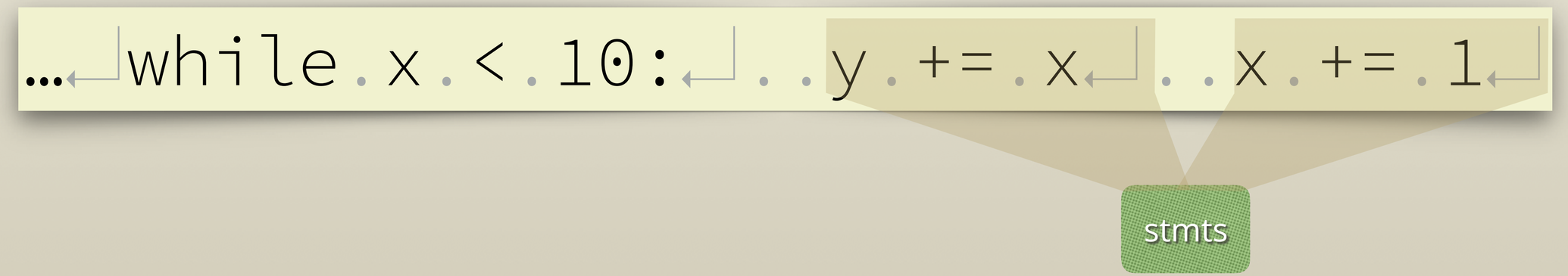
1. Removing from the middle of the stream
2. Extracting interleaved segments of a stream
3. Adding to the stream

Limitations implied by the Substream Property

1. Removing from the stream

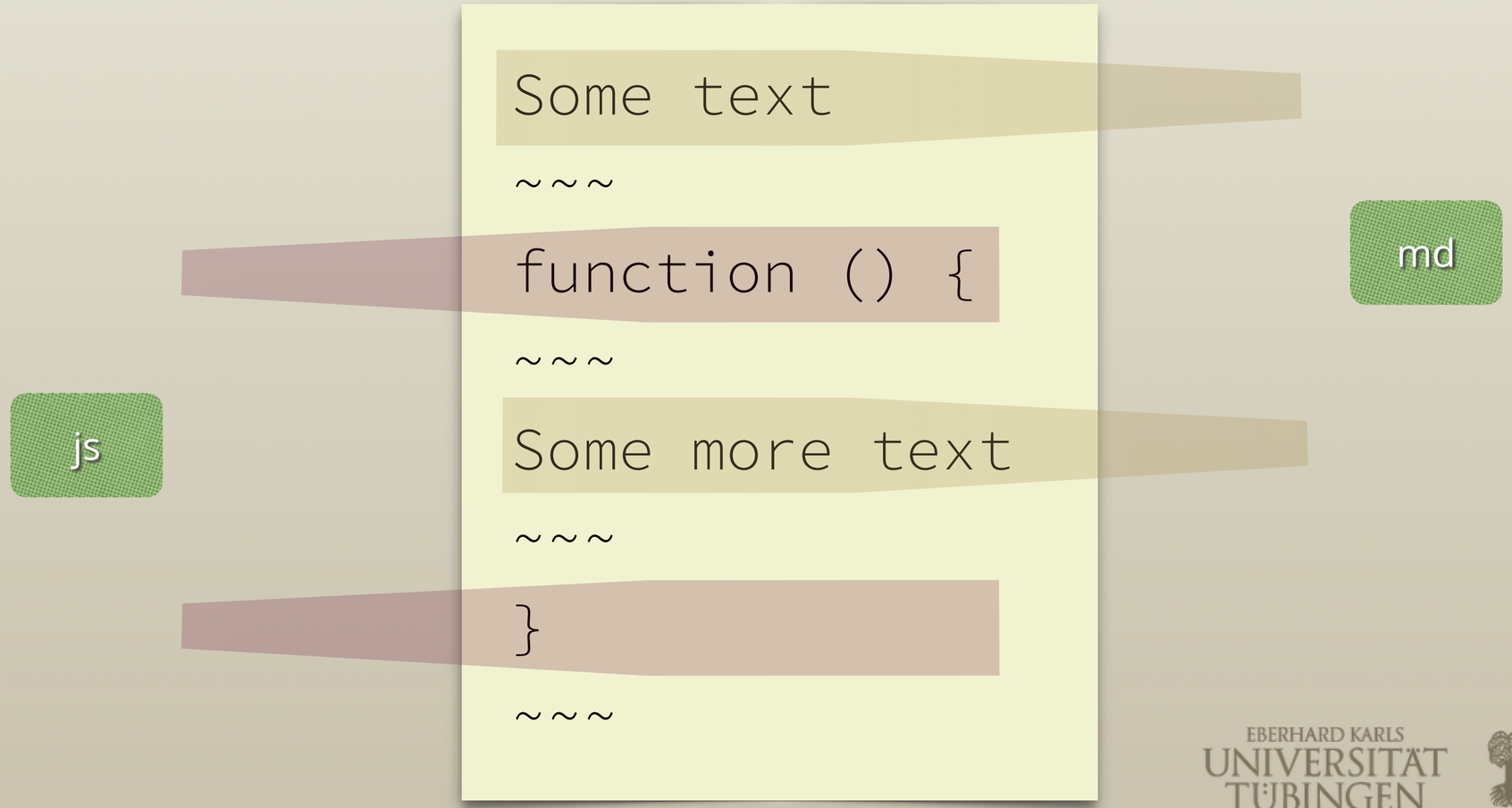
Limitations implied by the Substream Property

1. Removing from the stream



Limitations implied by the Substream Property

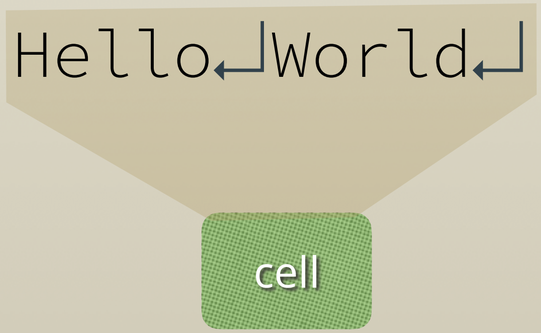
2. Extracting interleaved segments of a stream



Limitations implied by the Substream Property

3. Adding to the stream

```
+-----+------+
|Hello|ASCII |
|World|Tables|
+-----+------+
```



Limitations implied by the Substream Property

3. Adding to the stream