Valid But Readable: Writing *D2d* For Constructing XML

Markus Lepper[a] Baltasar Trancón y Widemann[a,b]

[a]<semantics/> gGmbH, Berlin

[b]Nordakademie Elmshorn

Joint MEC and TEI conference, Paderborn, 7. 9. 2023

1. Goals and Principles of d2d
2. Input and Parsing
3. Modules and Parametrization
4. Error Recovery and Diagnosis
5. Documentation and Subsequent Processing
6. Comprehension
7. Prototypical Applications

1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

4 Error Recovery and Diagnosis

5 Documentation and Subsequent Processing

6 Comprehension

7 Prototypical Applications

#d2d

(= directly to document
or Direct Document Denotation
or "triple-dee")

<> **Goals:**

- <> let domain experts easily write down XML encoded texts
- <> do it "in the flow" of writing:
  no clicking, no menus, no light show
- <> use any editor you like, use pen and pencil, use voice input
- <> readable not only by computers (but also by humans !-)

<> Predecessors and paradigms:

- <> LaTeX
- <> M4
- <> Iout
- <> SGML (but *not* its tag omission !-)

<> Means:

- <> compiler based
- <> only one escape character
  (plus one comment lead-in, both configurable)
  d2d: # / and /
  xml: < > / " ' = <!-- -- --> [ ]] <? &
  ; . . . (plus DTD %)
- <> tag inference (of nearly all closing and some opening tags)
- <> totally implicit tags for short but highly structured data items
- <> explicitly addressing computer laymen:
  robust parsing,
  acceptance of incomplete sources,
  generation of bug fix proposals,
  graphic documentation (WIRTH diagrams)

<> Means:

- <> compiler based
- <> only one escape character
  (plus one comment lead-in, both configurable)
  d2d: # / and /
  xml: < > / " ' = <!-- -- --> [ ]] <? &
  ; . . . (plus DTD %)
- <> tag inference (of nearly all closing and some opening tags)
- <> totally implicit tags for short but highly structured data items
- <> explicitly addressing computer laymen:
  robust parsing,
  acceptance of incomplete sources,
  generation of bug fix proposals,
  graphic documentation (WIRTH diagrams)

<> Means:

- <> compiler based
- <> only one escape character
  (plus one comment lead-in, both configurable)
  d2d: `#` `/` and `/`
  xml: `<` `>` `/` `"` `'` `=` `<!--` `--` `-->` `[` `]]` `<?` `&`
  `;` . . . (plus DTD `%`)
- <> tag inference (of nearly all closing and some opening tags)
- <> totally implicit tags for short but highly structured data items
- <> explicitly addressing computer laymen:
  robust parsing,
  acceptance of incomplete sources,
  generation of bug fix proposals,
  graphic documentation (WIRTH diagrams)

<> Specially suited for ...

- <> Poets, Essayists, Novelists, and domain experts
- <> technical descriptions and multiple-use source texts
- <> XSL-T powered content management
- <> semi-structured data bases, technical configuration data
- <> data migration, incremental formalisation of legacy texts, grammar refinement
- <> voice input
- <> semantic driven document types, data structures and mark-up

<> Successfully applied for ...

- <> book-keeping
- <> music theory analysis
- <> roman à clef
- <> textual data base

- **(Simple ideas but complicated consequences !-)**
- **Combination of advanced technologies:**
  - Two levels of parsing:
    Top-level explicit open tags
    (XML style LL(1) parsing)
  - plus tag inference:
    nearly all closing tags; some opening tags
  - Lower level of tag-less character parsers
    (Non-deterministic parsing)
  - Parametrization by rewriting
  - Input synthesis for error recovery
  - Multi-lingual documentation infrastructure
  - Special mode for XSLT sources

1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

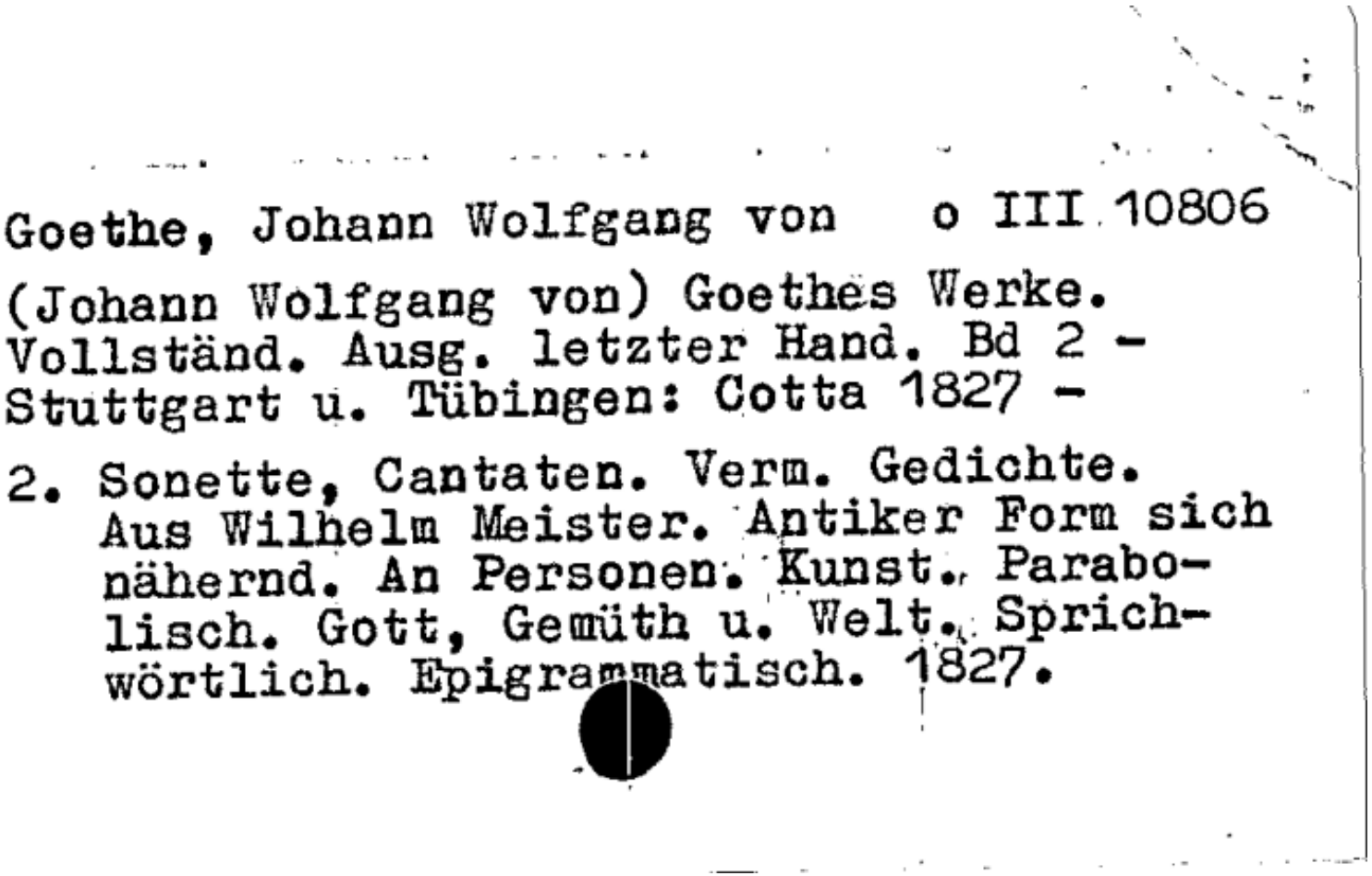
4 Error Recovery and Diagnosis

5 Documentation and Subsequent Processing

6 Comprehension

7 Prototypical Applications

Incremental Formalization – Photo



Goethe, Johann Wolfgang von o III 10806

(Johann Wolfgang von) Goethes Werke.
Vollständ. Ausg. letzter Hand. Bd 2 -
Stuttgart u. Tübingen: Cotta 1827 -

2. Sonette, Cantaten. Verm. Gedichte.
Aus Wilhelm Meister. Antiker Form sich
nähernd. An Personen. Kunst. Parabo-
lisch. Gott, Gemüth u. Welt. Sprich-
wörtlich. Epigrammatisch. 1827.

Incremental Formalization – OCR

```
Goethe, Johann Wolfgang von o III 10806

(Johann Wolfgang von) Goethes Werke.
VOllstand. Ausg. letzter Hand. Bd 2 -
Stuttgart u. Tübingen:  Cotta 1827 -

2. Sonette, Cantaten, Verm. Gedichte.
   Aus Wilhelm Meister. Antiker Form sich
   nähernd. An Personen. Kunst. Parabo-
   lisch. Gott, Gemüth u. Welt. Sprich-
   wörtlich. EpigraMMatisch. 1827.
```

Incremental Formalization – Text Type Declared

```
#d2d text using katalog:kartei_vor_1946
Goethe, Johann Wolfgang von o III 10806

(Johann Wolfgang von) Goethes Werke.
VOllstand. Ausg. letzter Hand. Bd 2 -
Stuttgart u. Tübingen:  Cotta 1827 -

2. Sonette, Cantaten, Verm. Gedichte.
   Aus Wilhelm Meister. Antiker Form sich
   nähernd. An Personen. Kunst. Parabo-
   lisch. Gott, Gemüth u. Welt. Sprich-
   wörtlich. EpigraMMatisch. 1827.
#eof
```

Incremental Formalization – First Mark-Up

```
#d2d text using katalog:kartei_vor_1946
#author Goethe, Johann Wolfgang von #sign o III 10806#/

(Johann Wolfgang von) Goethes Werke.
VOllstand. Ausg. letzter Hand. Bd 2 -
Stuttgart u. Tübingen:  Cotta 1827 -

2. Sonette, Cantaten, Verm. Gedichte.
   Aus Wilhelm Meister. Antiker Form sich
   nähernd. An Personen. Kunst. Parabo-
   lisch. Gott, Gemüth u. Welt. Sprich-
   wörtlich. EpigraMMatisch. 1827.
#eof
```

Incremental Formalization – More Mark-Up

```
#d2d text using katalog:kartei_vor_1946
#author Goethe, Johann Wolfgang von #sign o III 10806#/

(Johann Wolfgang von) #title Goethes Werke.
VOllstand. Ausg. letzter Hand. #vol 2 // -
#place Stuttgart u. Tübingen #publisher Cotta #year 1827 #/

2. Sonette, Cantaten, Verm. Gedichte.
   Aus Wilhelm Meister. Antiker Form sich
   nähernd. An Personen. Kunst. Parabo-
   lisch. Gott, Gemüth u. Welt. Sprich-
   wörtlich. EpigraMMatisch. 1827.
#eof
```

Incremental Formalization by Character Parsers

```
#d2d text using katalog:kartei_vor_1946
#author Goethe, Johann Wolfgang von #sign o III 10806#/

(Johann Wolfgang von) #title Goethes Werke.
VOllstand. Ausg. letzter Hand. #vol 2 // -
#place Stuttgart u. Tübingen #publisher Cotta #year 1827 #/
-
```

tags kartei_vor_1946 = (author & title & sign), vol?,
(place? & publisher? & year?)
tags author, title, place, publisher = #chars
tags sign, vol, year = #chars

Incremental Formalization by Character Parsers

```
#d2d text using katalog:kartei_vor_1946
#author Goethe, Johann Wolfgang von #sign o III 10806#/

(Johann Wolfgang von) #title Goethes Werke.
VOllstand. Ausg. letzter Hand. #vol 2 // -
#place Stuttgart u. Tübingen #publisher Cotta #year 1827 #/
-
```

tags kartei_vor_1946 = (author & title & sign), vol?,
(place? & publisher? & year?)
tags author, title, place, publisher = #chars
chars year = ('0'..'9')~('0'..'9')~('0'..'9')~('0'..'9')
chars vol = ('1'..'9') ~ ('0'..'9')* ~ ('a'..'z')?

Incremental Formalization by Character Parsers

```
#d2d text using katalog:kartei_vor_1946
#author Goethe, Johann Wolfgang von #sign o III 10806#/

(Johann Wolfgang von) #title Goethes Werke.
VOllstand. Ausg. letzter Hand. #vol 2 // -
#place Stuttgart u. Tübingen #publisher Cotta #year 1827 #/
-
```

tags kartei_vor_1946 = (author & title & sign), vol?,
(place? & publisher? & year?)
tags author, title, place, publisher = #chars
chars year = ('0'..'9')~('0'..'9')~('0'..'9')~('0'..'9')
chars vol = ('1'..'9') ~ ('0'..'9')* ~ ('a'..'z')?
chars sign = [series ('o' | 'p' | "Coll")],
[group ('I'|'V'|'X'|'L'|'C')+]?, [num ('0'..'9')+]

Incremental Formalization by Character Parsers

```
#d2d text using katalog:kartei_vor_1946
#author Goethe, Johann Wolfgang von #sign o III 10806#/

(Johann Wolfgang von) #title Goethes Werke.
VOllstand. Ausg. letzter Hand. #vol 2 // -
#place Stuttgart u. Tübingen #publisher Cotta #year 1827 #/
-
```

tags kartei_vor_1946 = (author & title & sign), vol?,
(place? & publisher? & year?)
tags author, title, place, publisher = #chars
chars year = ('0'..'9')~('0'..'9')~('0'..'9')~('0'..'9')
chars vol = ('1'..'9') ~ ('0'..'9')* ~ ('a'..'z')?
chars sign = [series ('o' | 'p' | "Coll")],
[group ('I'|'V'|'X'|'L'|'C')+]?, [num ('0'..'9')+]

<sign><series>o</series><group>III</group><num>10806</num></sign>

1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

4 Error Recovery and Diagnosis

5 Documentation and Subsequent Processing

6 Comprehension

7 Prototypical Applications

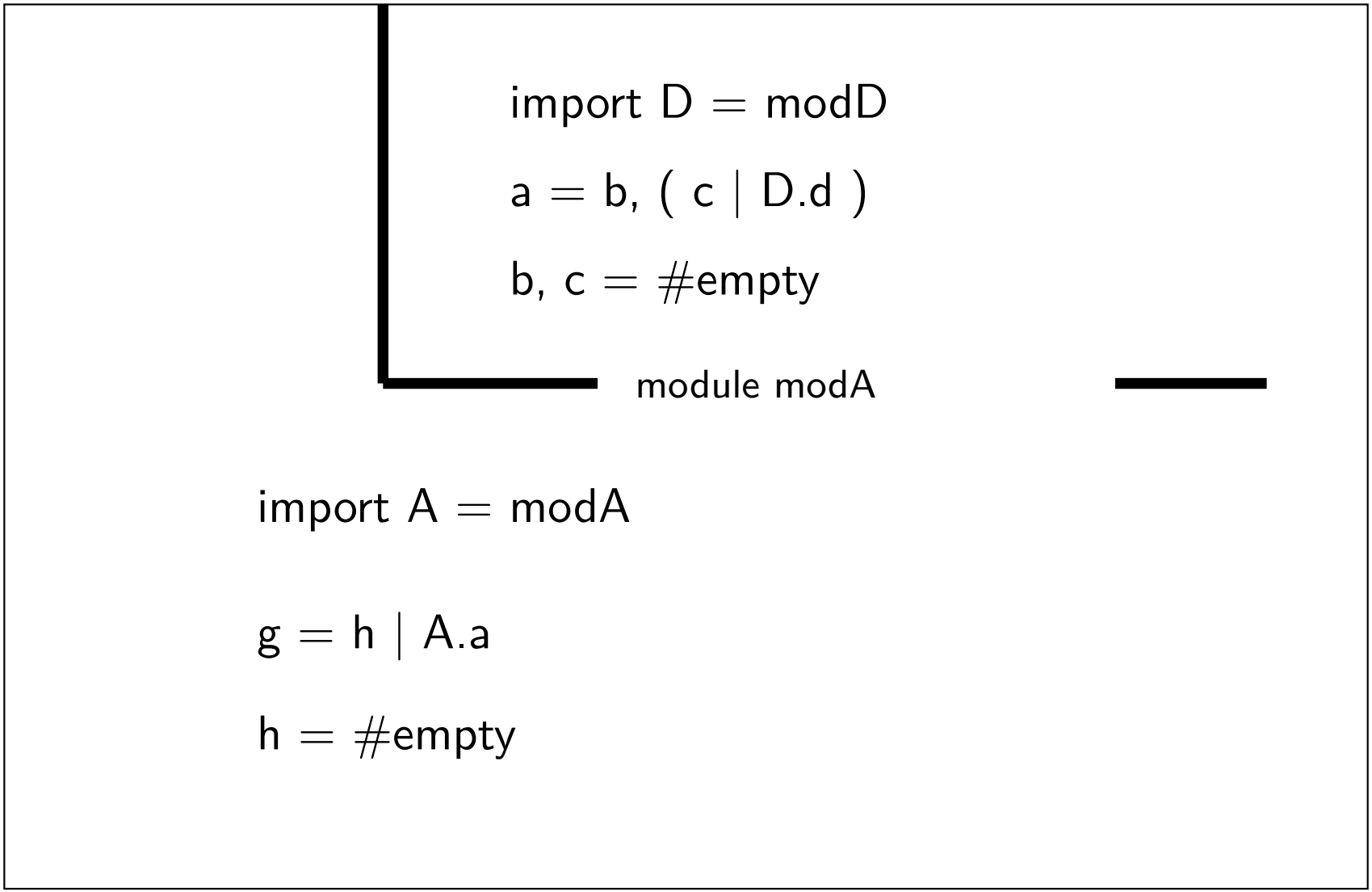
- d2d works immediately on w3c DTDs.
- But its own definition format adds ...
  - character parsers
  - unification of child elements and attributes
  - multi-lingual documentation
  - parametrization by rewriting

Rewriting for Parametrization

<> Operator ^(.../...) means:
every reference can be seen as a parameter
= "spontaneous rewriting"
= no "pre-wiring" is necessary
(which is often not sensibly *possible* !-)

<> Operator @ means:
All element definitions can serve as content types
= no special definition pool necessary

Parametrization I



Parametrization I

Inject local expression into imported expression:

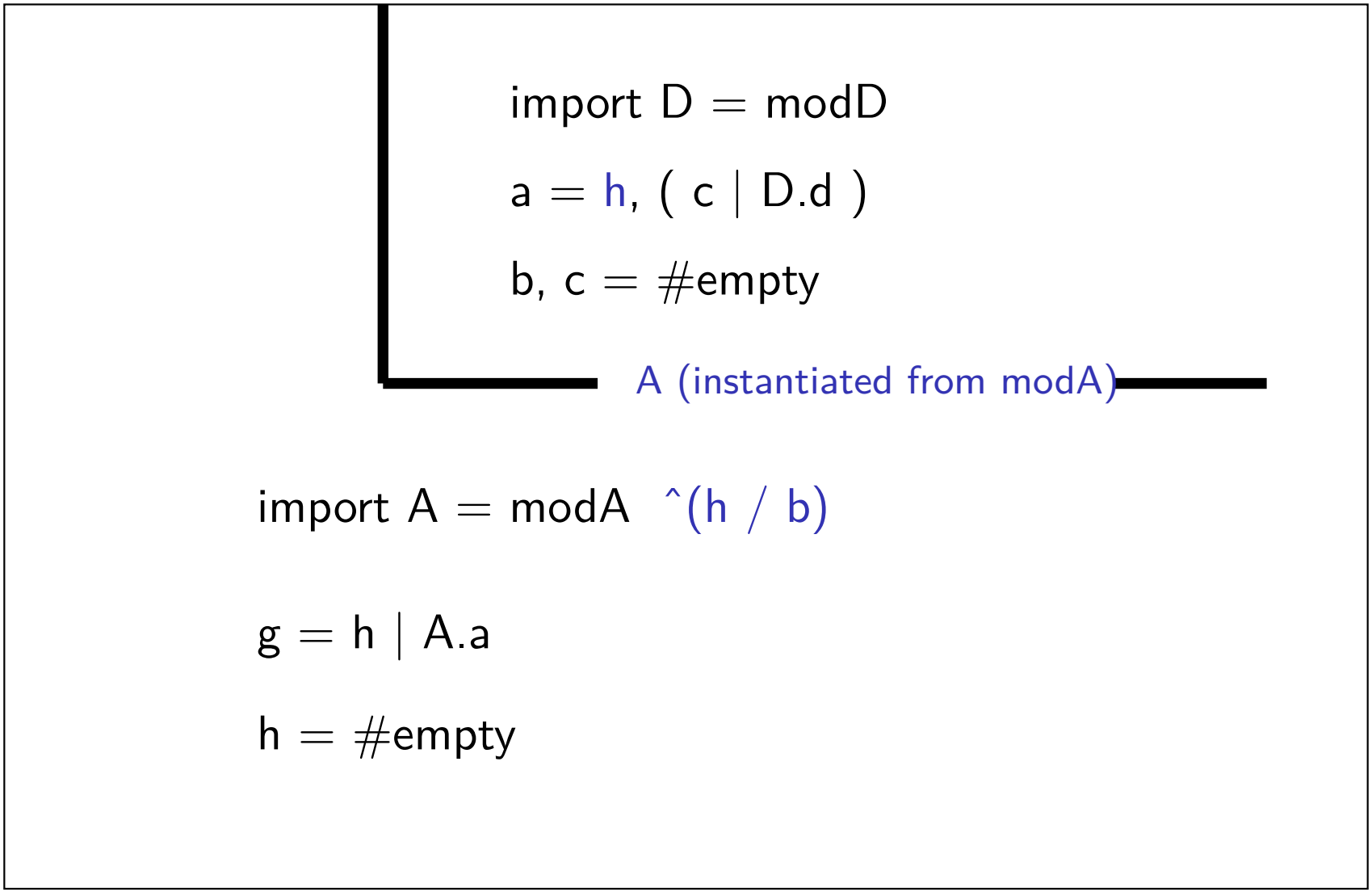
```
import D = modD

a = b, ( c | D.d )

b, c = #empty
```

module modA

```
import A = modA  ^(h / b)

g = h | A.a

h = #empty
```

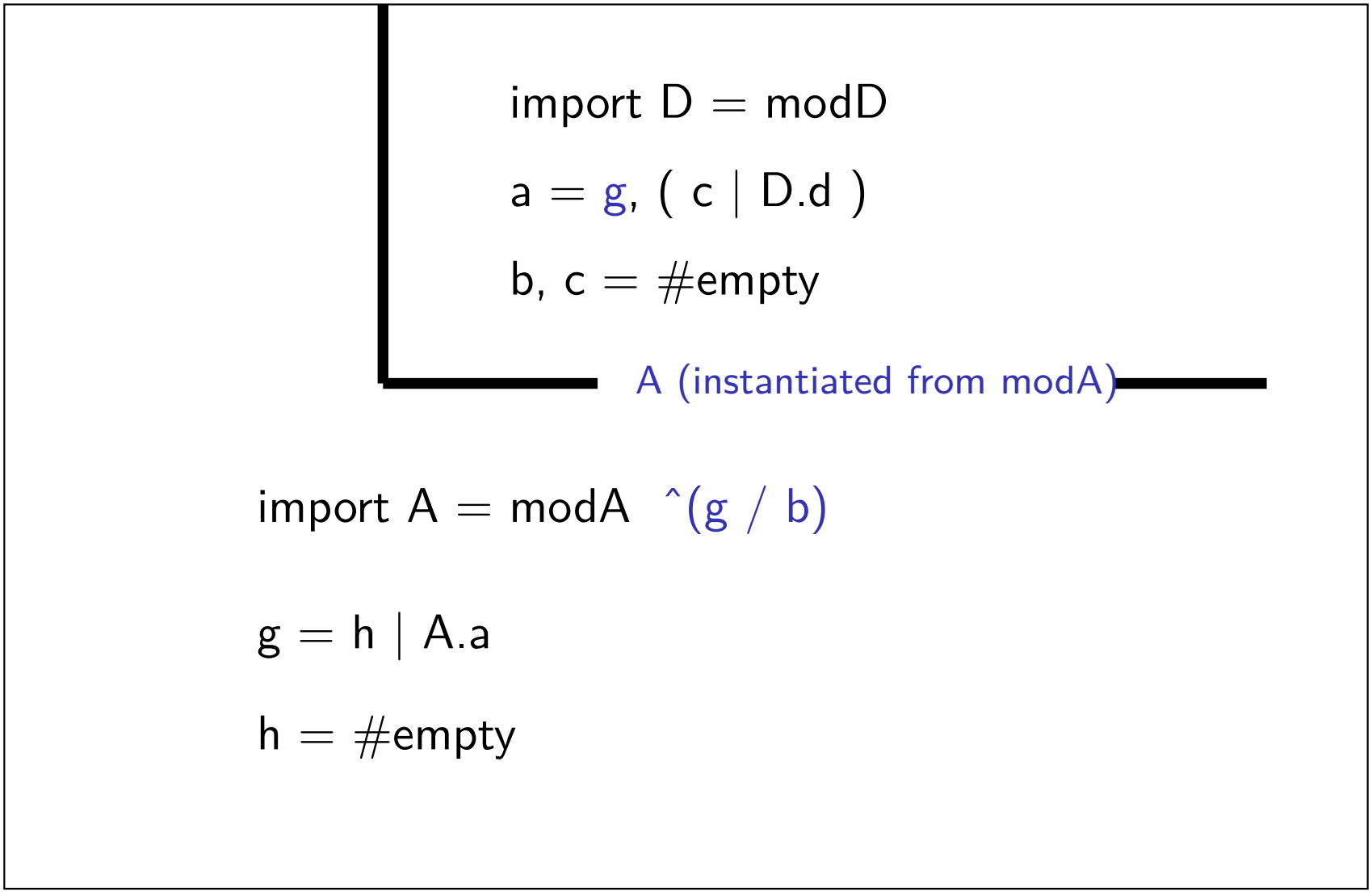
Parametrization I

Inject local expression into imported expression:



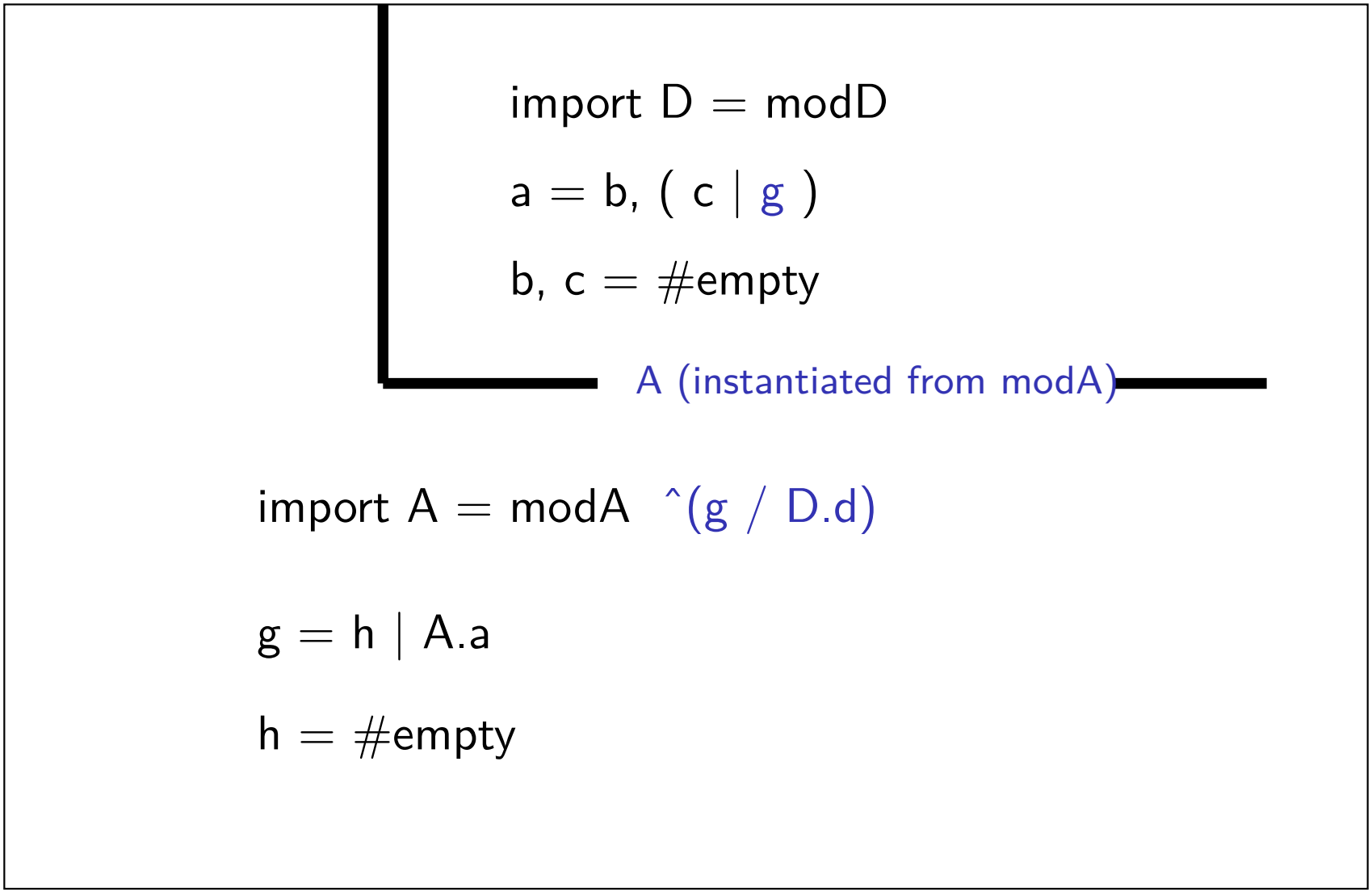
Parametrization I

Inject local expression into imported expression, making cycle:



Parametrization I

Inject local expression into imported expression, making cycle:



Parametrization I

Inject imported expression into imported expression:

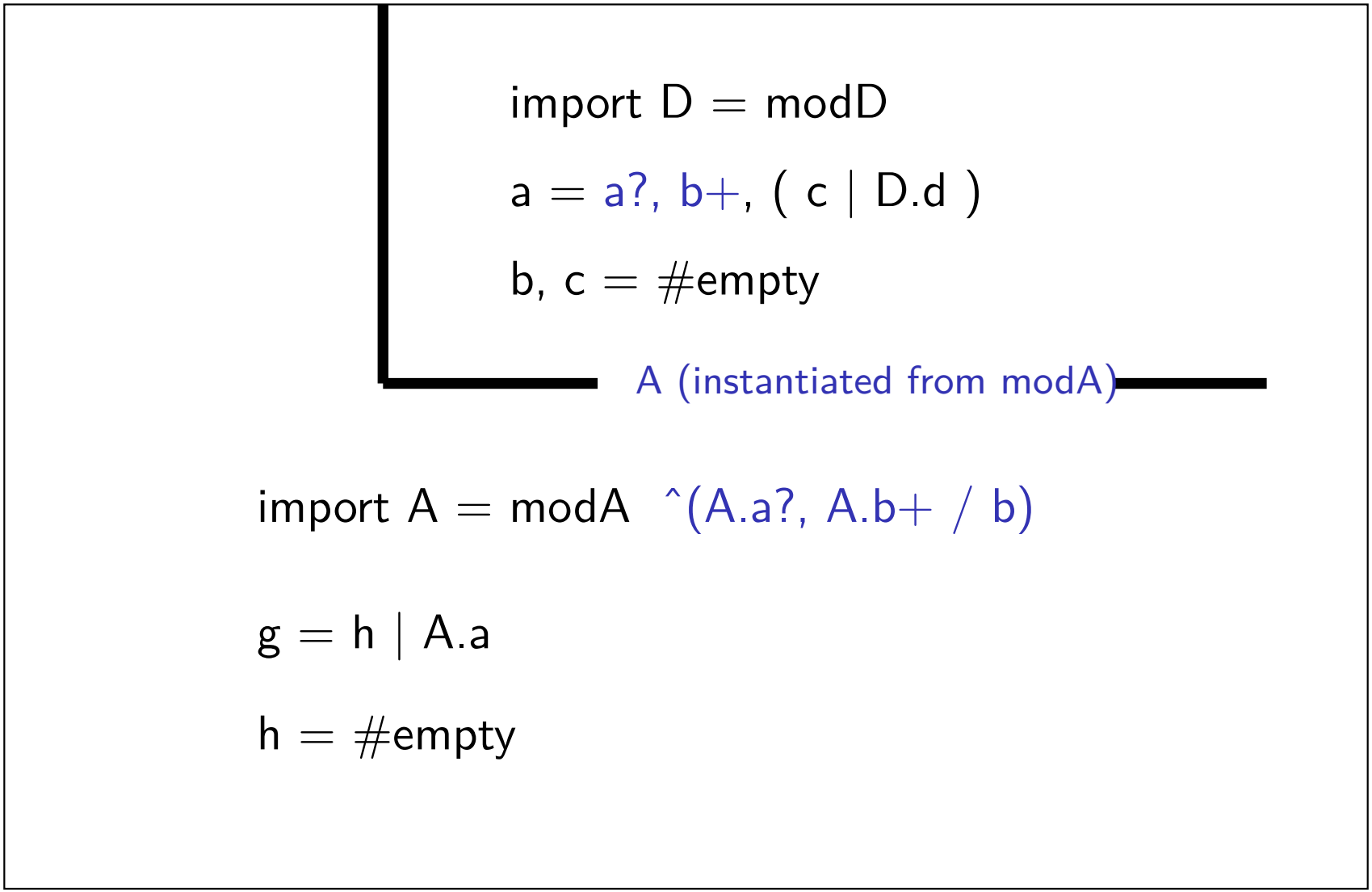
```
import D = modD

a = b, ( c | D.d )

b, c = #empty
```

module modA

```
import A = modA  ^(A.a?, A.b+ / b)

g = h | A.a

h = #empty
```

Parametrization I

Inject imported expression into imported expression:



Parametrization I

Use indirect import / imports are re-exported:

```
import D = modD ^( b | c / x)

a = b, ( c | D.d )

b, c = #empty
```

A (instantiated from modA)

```
import A = modA

g = h | A.a | A.D.d

h = #empty
```

Parametrization I

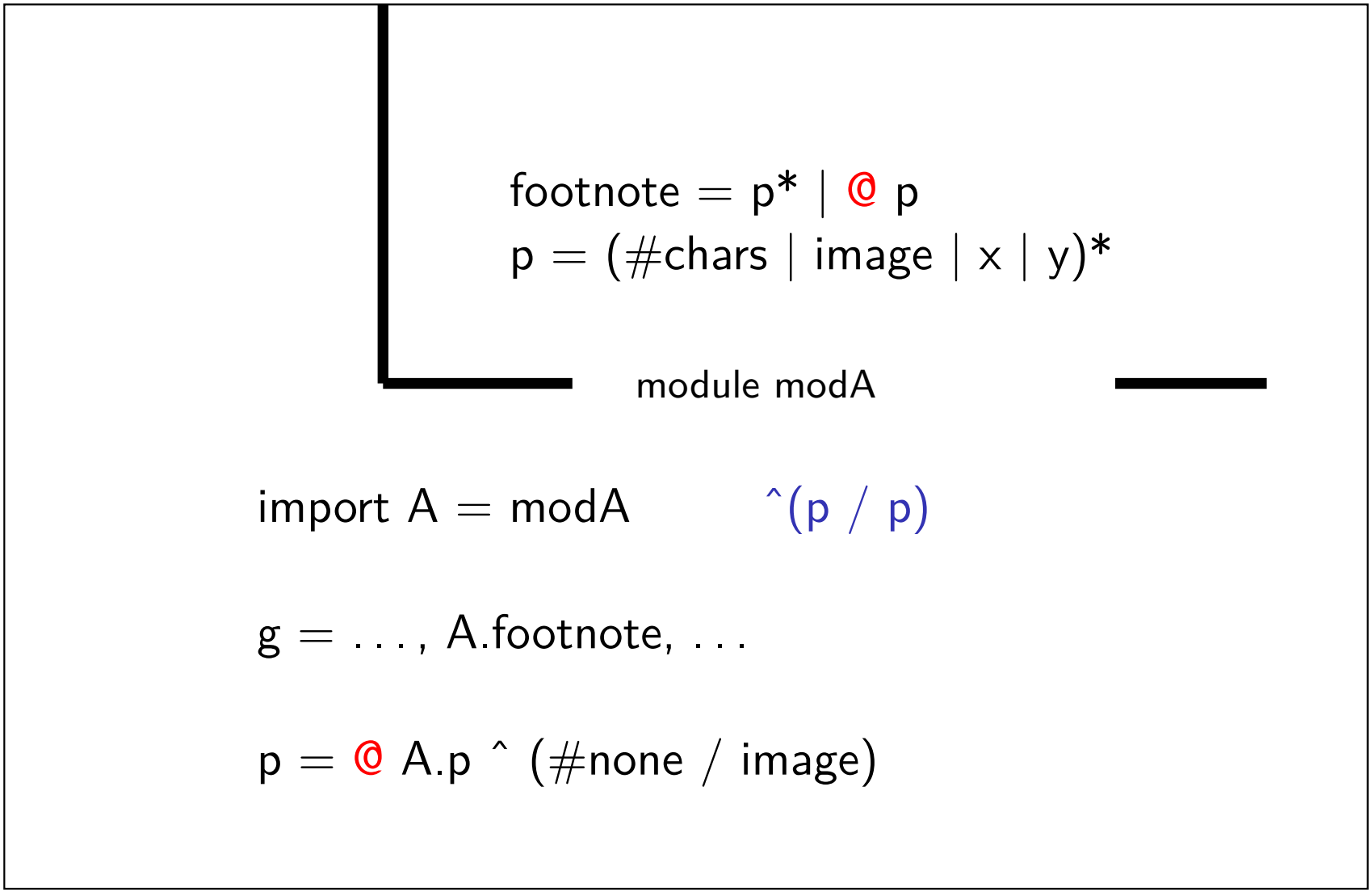
Replace whole module import, e.g. with locale:

```
module modA

  import D = modD modE

  a = b, ( c | D.d )

  b, c = #empty
```

```
import A = modA ^(E / D)
import E = modE
g = h | A.a

h = #empty
```

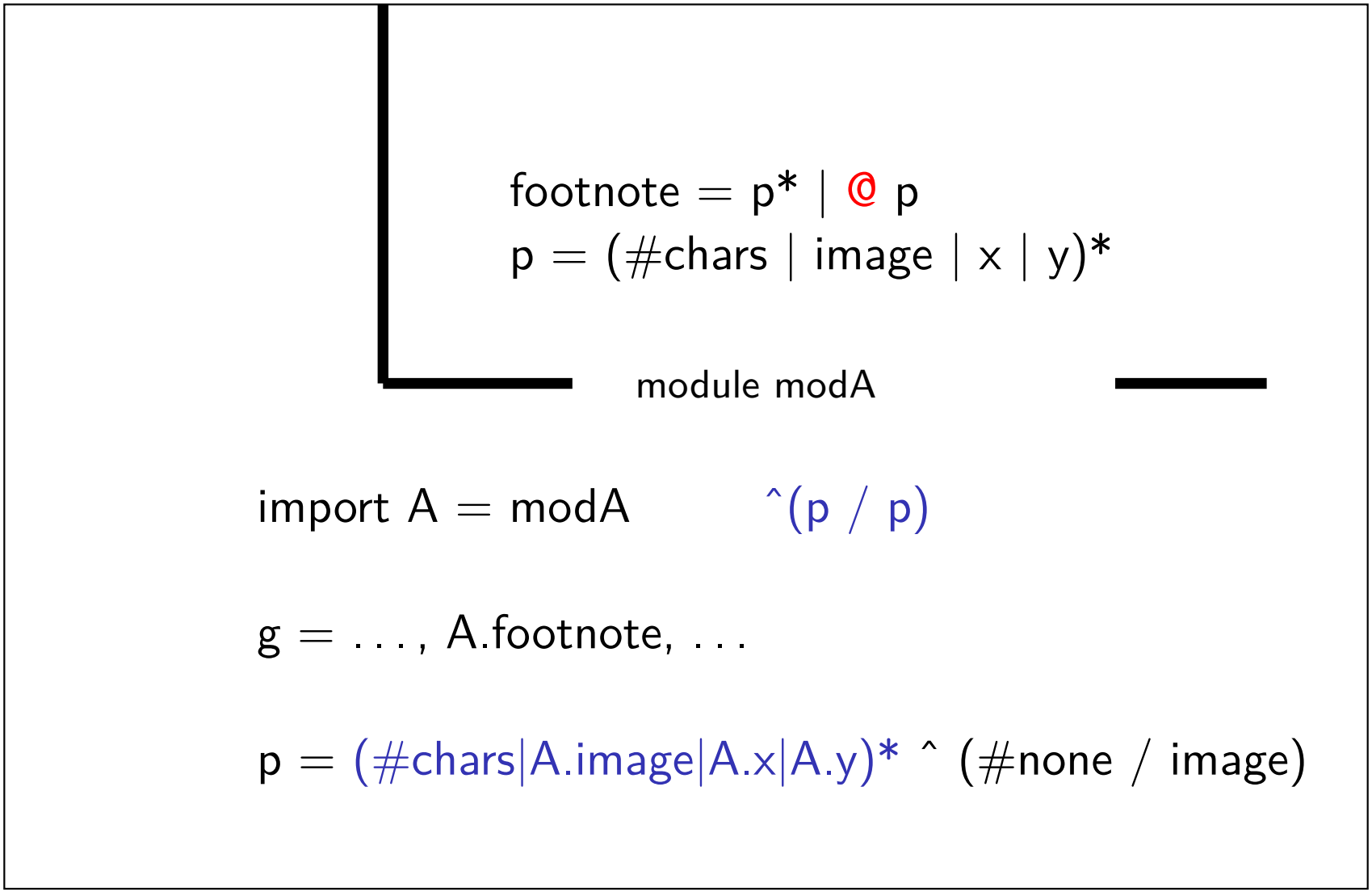
Parametrization II

Rewrite reference to rewritten expression:



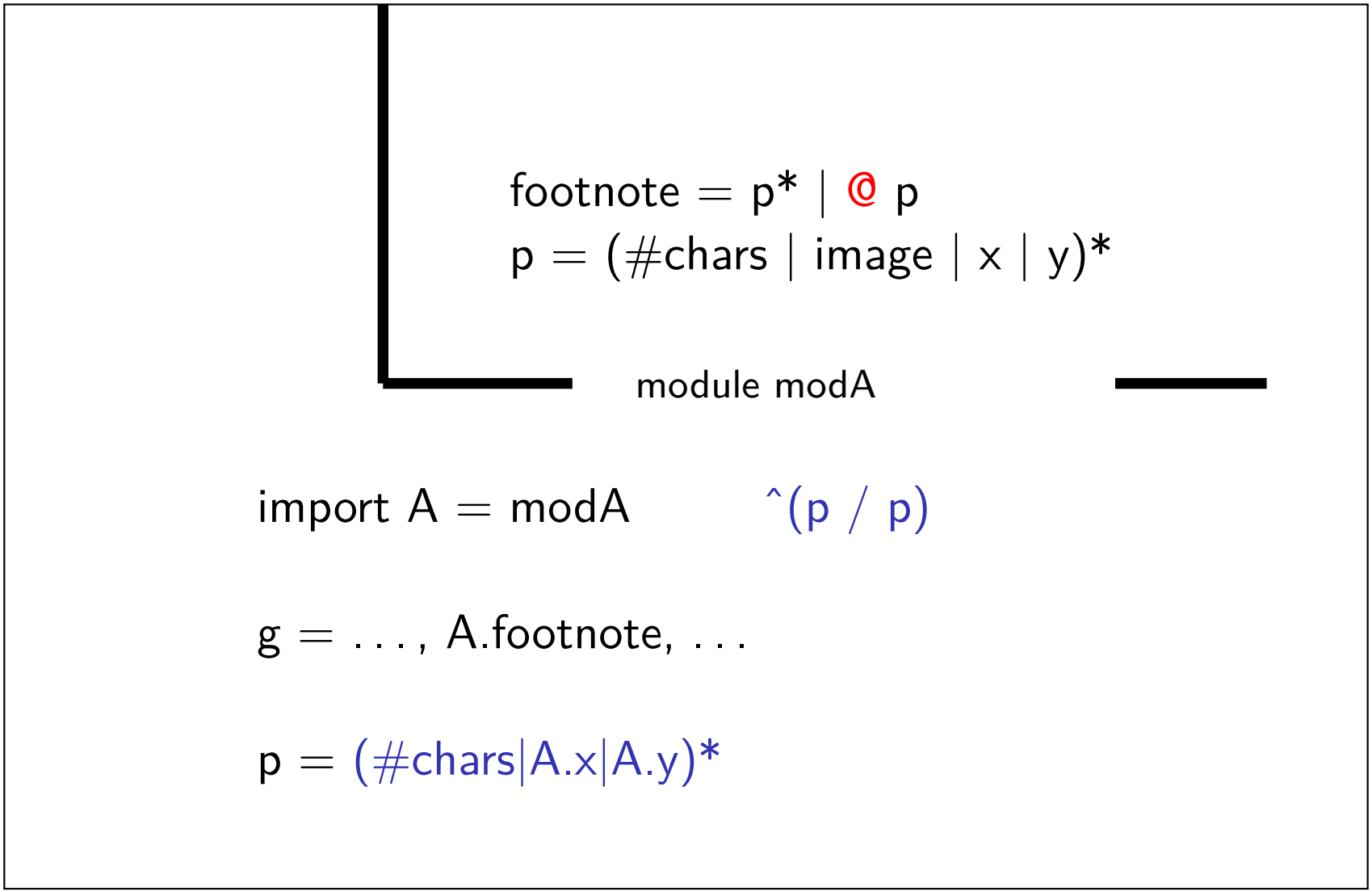
Parametrization II

Operator "@" means contents:

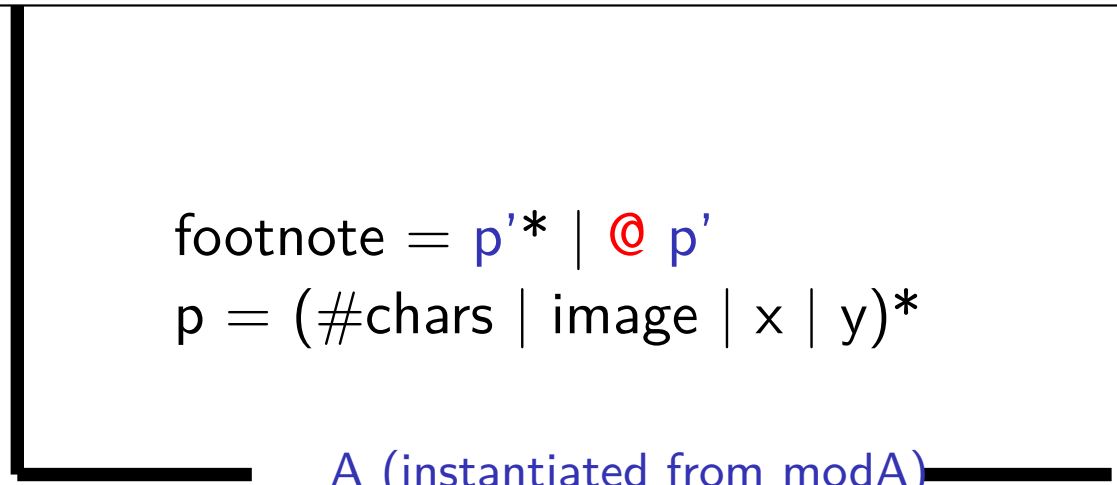


Parametrization II

Rewriting to "#none" means elimination:



Parametrization II



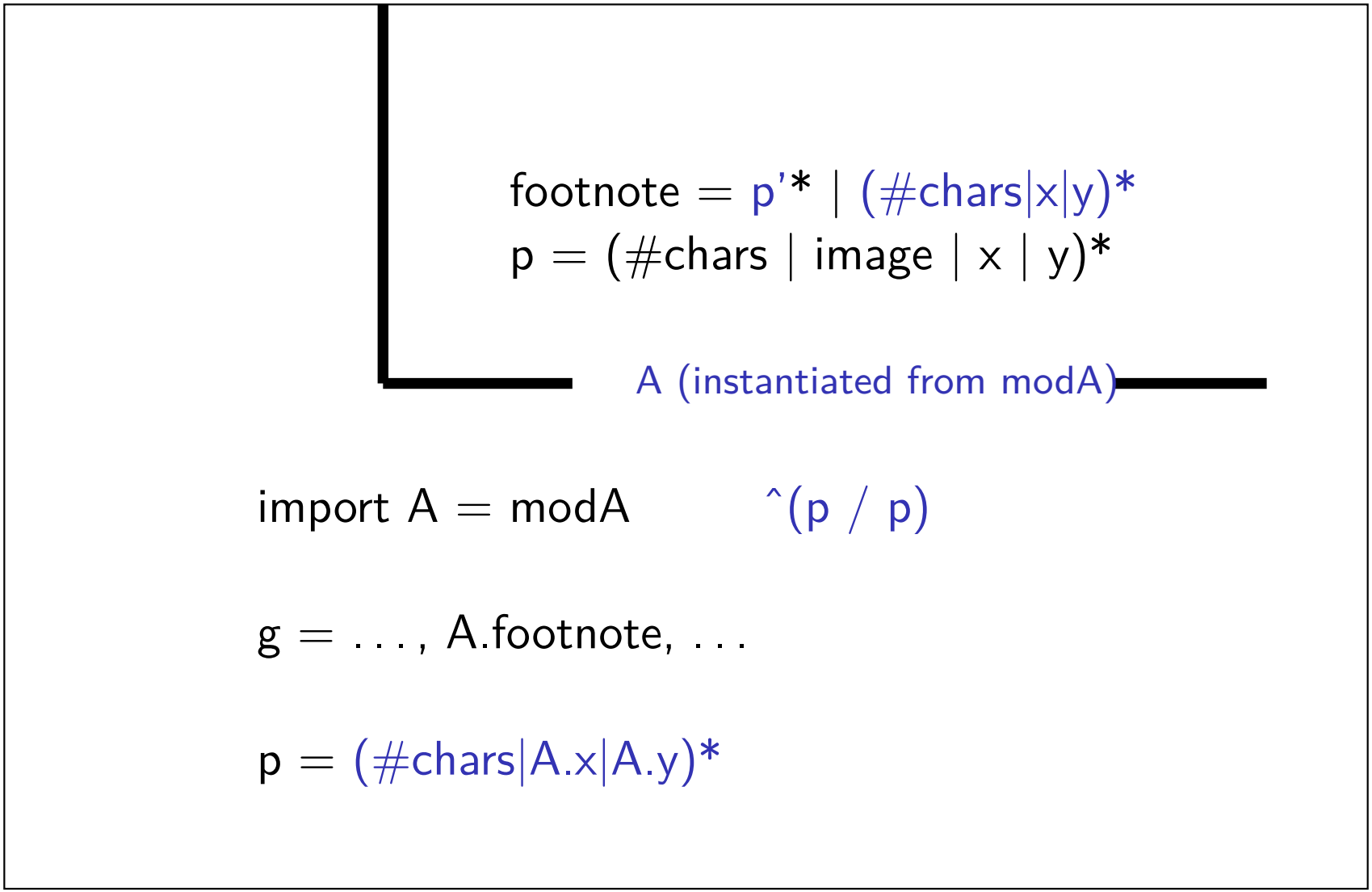
import A = modA ^(p / p)

g = . . . , A.footnote, . . .

p = (#chars|A.x|A.y)*

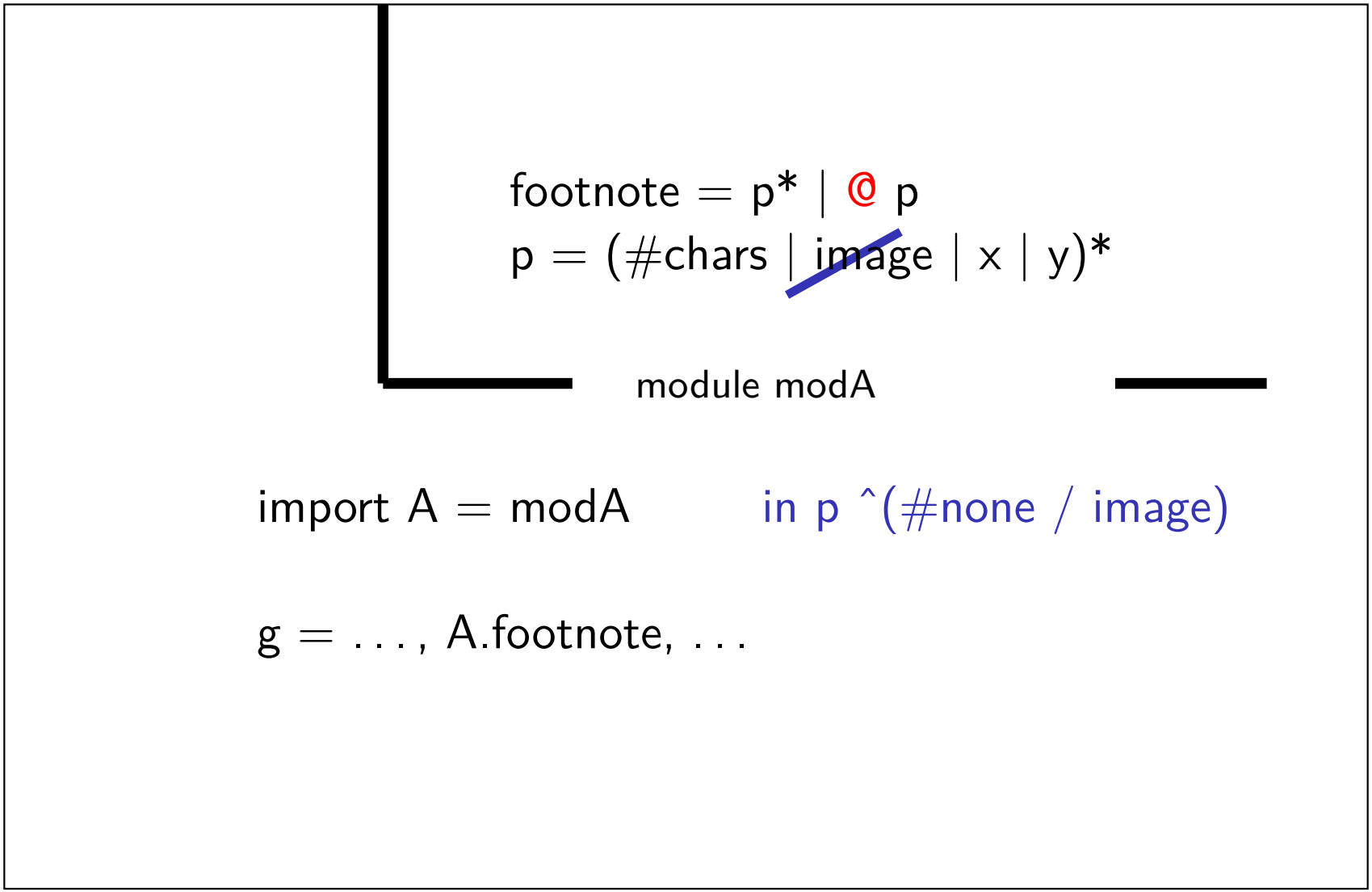
Parametrization II

Operator "@" means contents:



Parametrization II

For convenience: Local module rewriting:



1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

4 **Error Recovery and Diagnosis**

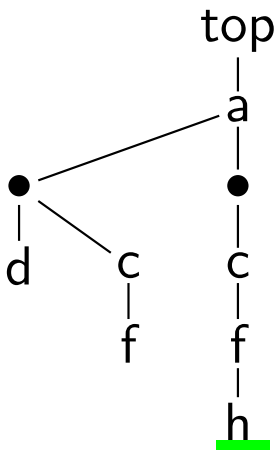
5 Documentation and Subsequent Processing

6 Comprehension

7 Prototypical Applications

Signalling of missing content before an open tag

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c}\ \&\ d)^*, e^+$

$\boxed{f}, g?$

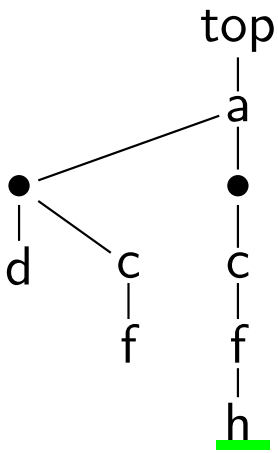
$\boxed{h}$

#a #d #c #f #c #f #h

$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before an open tag

result tree



#d
#/c
#g
#/f
#/h

regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c}\ \&\ d)^*, e^+$

$\boxed{f}, g?$

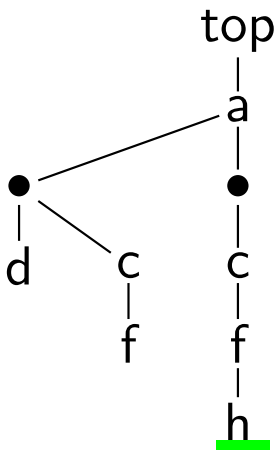
$\boxed{h}$

#a #d #c #f #c #f #h

$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before an open tag

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c} \,\&\, d)^*, e^+$

$\boxed{f}, g?$

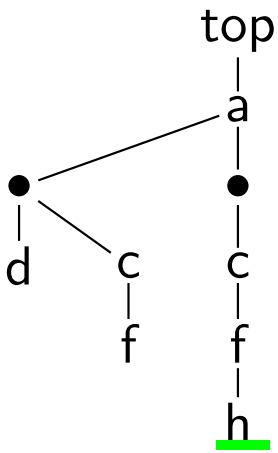
$\boxed{h}$ ▮

#a #d #c #f #c #f #h ▮ #b

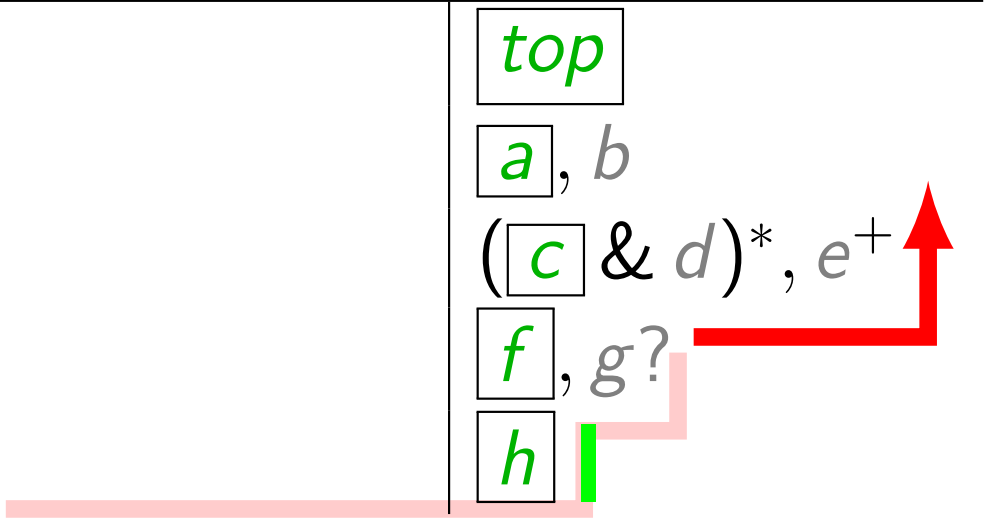
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before an open tag

result tree



regexp stack

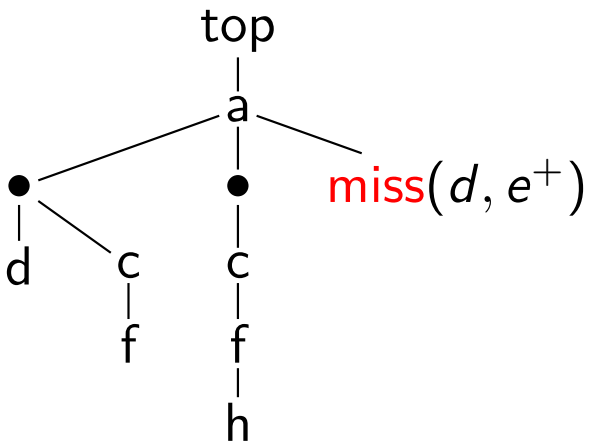


#a #d #c #f #c #f #h | #b

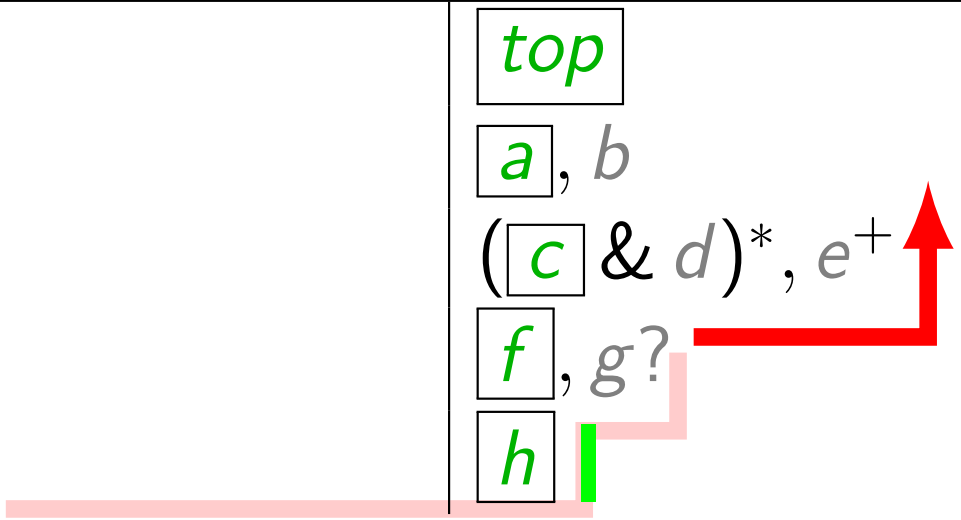
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before an open tag

result tree



regexp stack

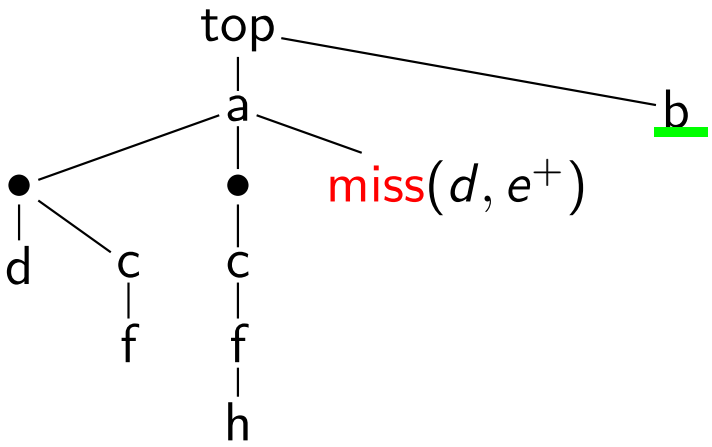


#a #d #c #f #c #f #h | #b

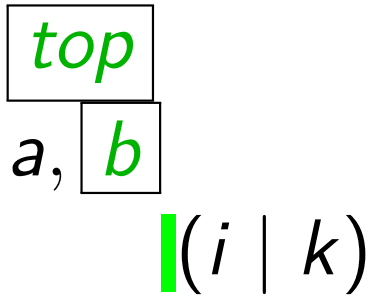
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before an open tag

result tree



regexp stack

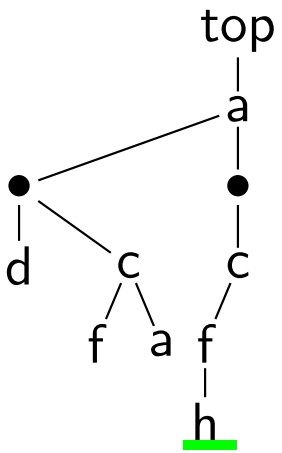


#a #d #c #f #c #f #h #b ...

$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before a close tag

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c} \,\&\, d)^*, e^+$

$\boxed{f}, a$

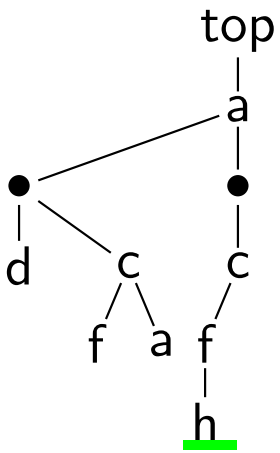
$\boxed{h}$

#a #d #c #f #a #c #f #h

$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, a \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before a close tag

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c} \,\&\, d)^*, e^+$

$\boxed{f}, a$

$\boxed{h}$

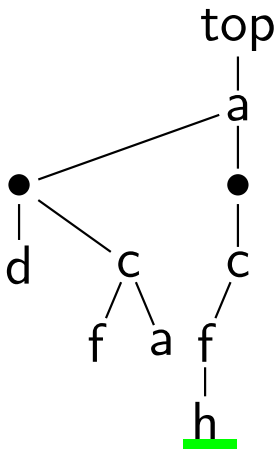
#a
#/f
#/h

#a #d #c #f #a #c #f #h

$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, a \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before a close tag

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c} \,\&\, d)^*, e^+$

$\boxed{f}, a$

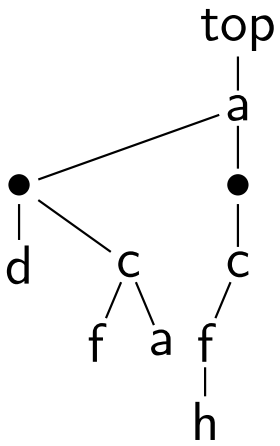
$\boxed{h}$ ▮

#a #d #c #f #a #c #f #h ▮ #/a

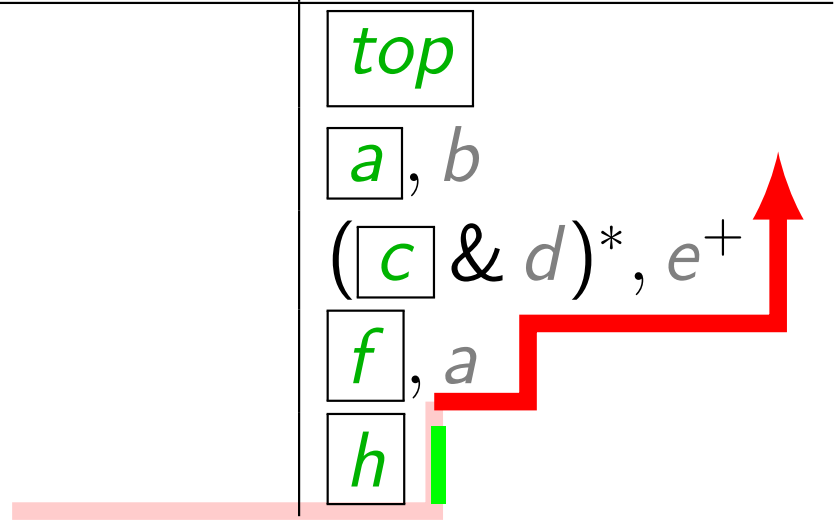
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, a \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before a close tag

result tree



regexp stack

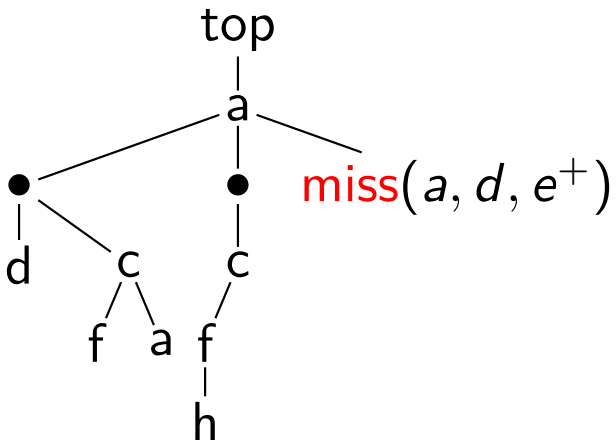


#a #d #c #f #a #c #f #h | #/a

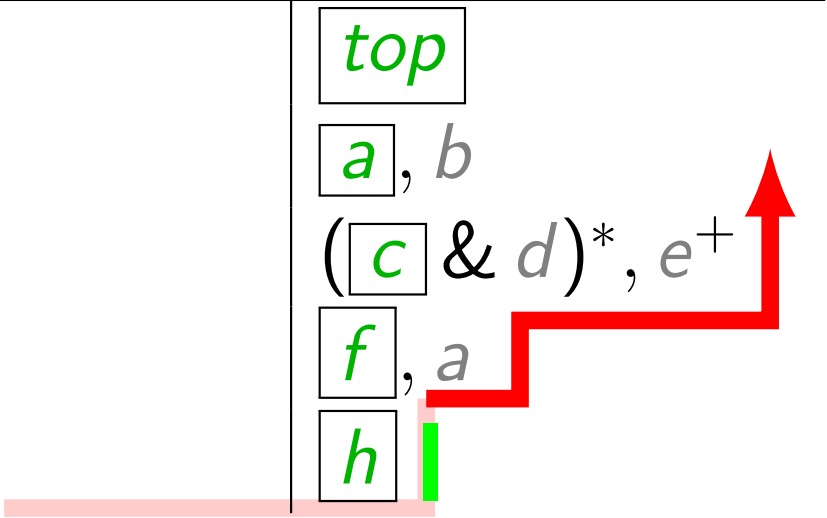
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, a \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before a close tag

result tree



regexp stack

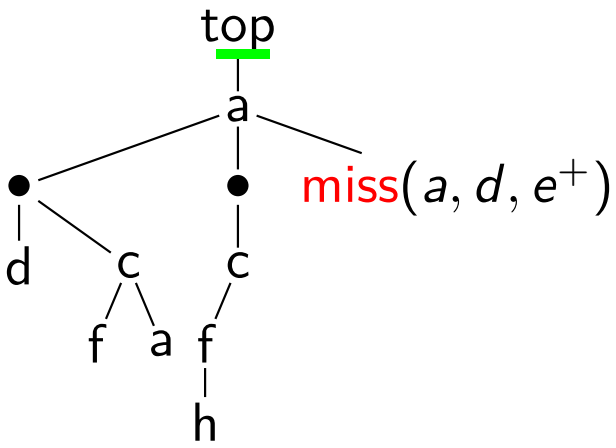


#a #d #c #f #a #c #f #h | #/a

$$\begin{aligned} \text{top} &= a, b \\ a &= (c\&d)^*, e^+ \qquad b = i \mid k \\ c &= f, a \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling of missing content before a close tag

result tree



regexp stack

top

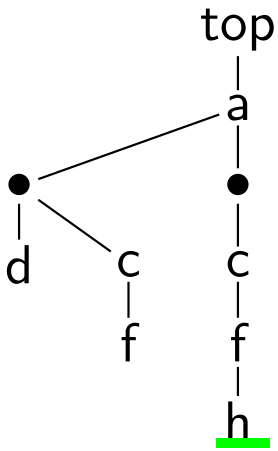
a | *b*

#a #d #c #f #a #c #f #h #/a | ...

$$\begin{aligned} \text{top} &= a, b \\ a &= (c\&d)^*, e^+ \qquad b = i \mid k \\ c &= f, a \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c} \,\&\, d)^*, e^+$

$\boxed{f}, g?$

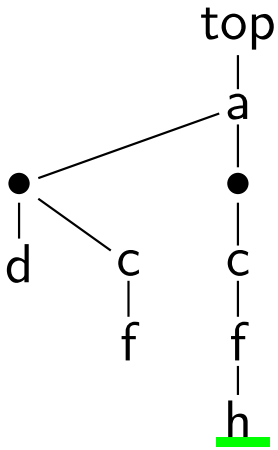
$\boxed{h}$

#a #d #c #f #c #f #h

$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

result tree



regexp stack

$\boxed{top}$

$\boxed{a}, b$

$(\boxed{c}\ \&\ d)^*, e^+$

$\boxed{f}, g?$

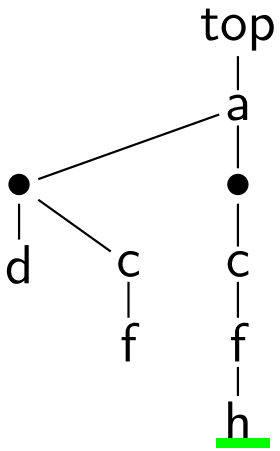
$\boxed{h}$

#a #d #c #f #c #f #h #x γ #g

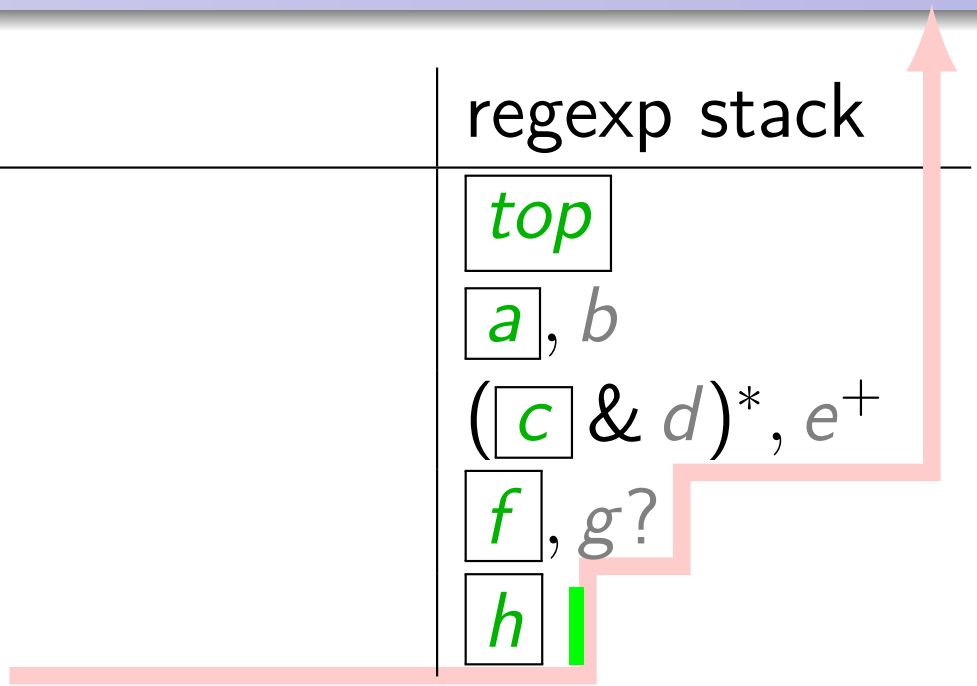
$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

result tree



regexp stack

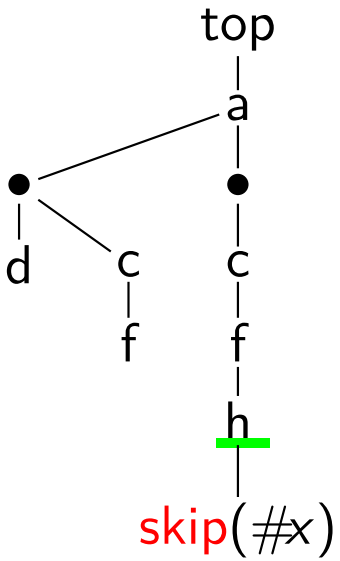


#a #d #c #f #c #f #h #x γ #g

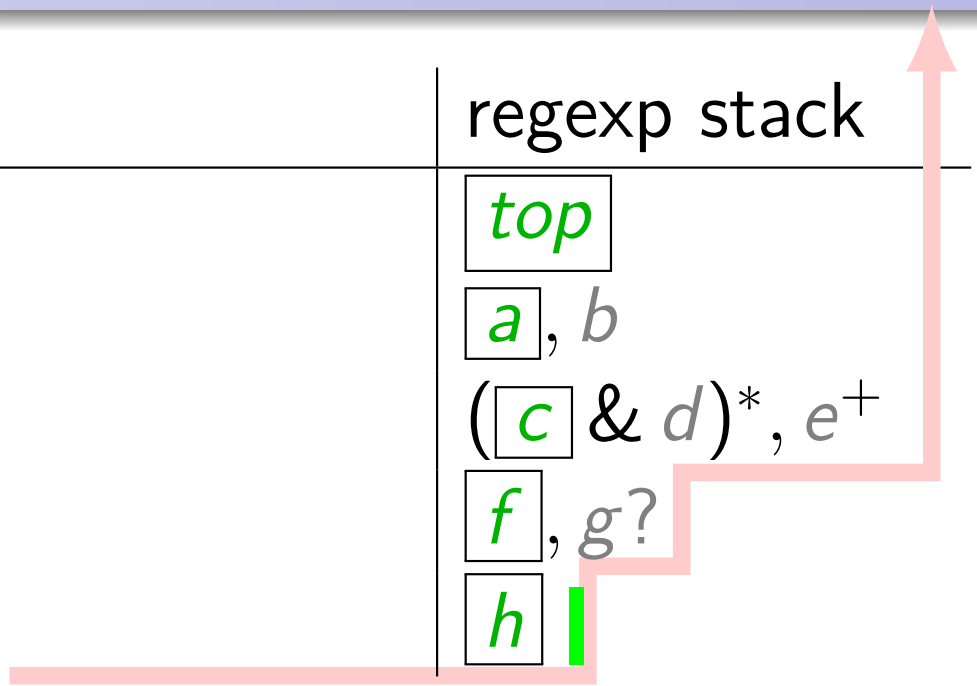
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

result tree



regexp stack

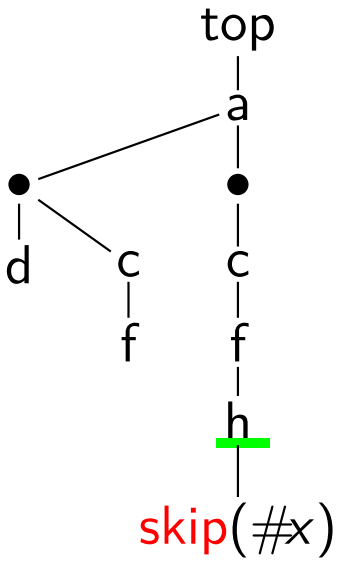


#a #d #c #f #c #f #h #x γ #g

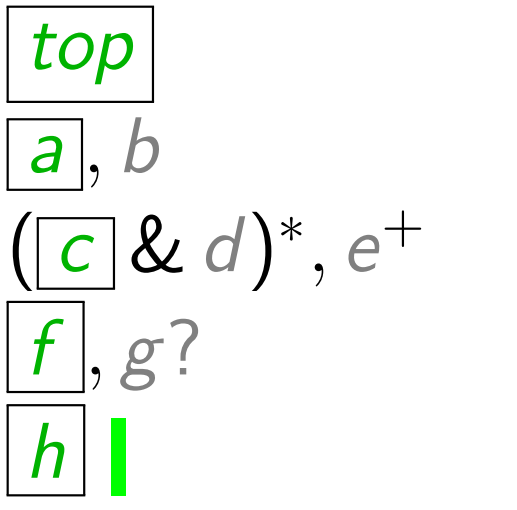
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

result tree



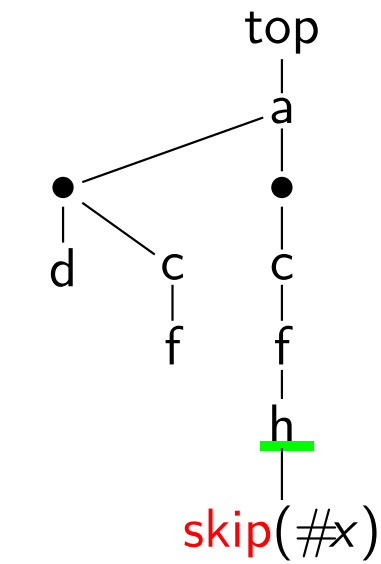
regexp stack



#a #d #c #f #c #f #h | γ #g

$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input



result tree

regexp stack

top

a, b

$(c \,\&\, d)^*, e^+$

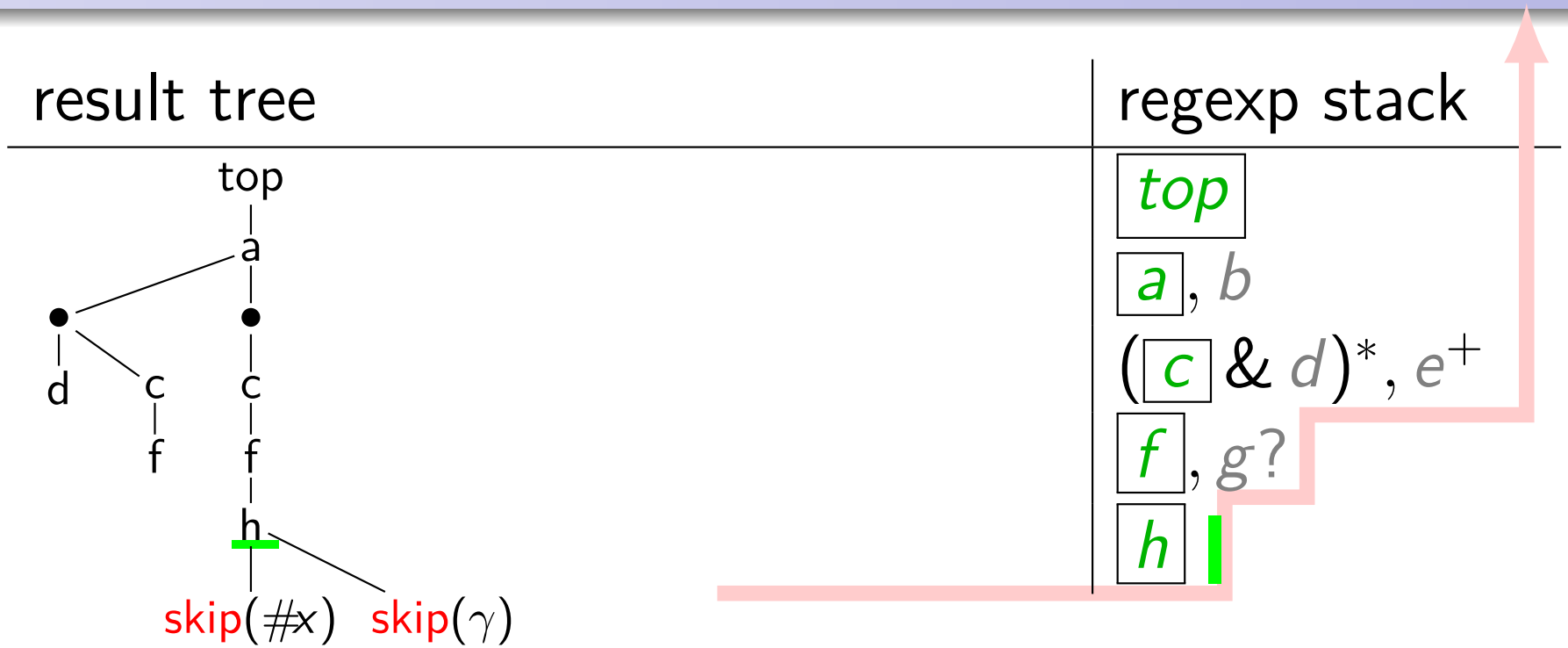
$f, g?$

h

$\#a\ \#d\ \#c\ \#f\ \#c\ \#f\ \#h \mid \gamma\ \#g$

$$\begin{aligned} top &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

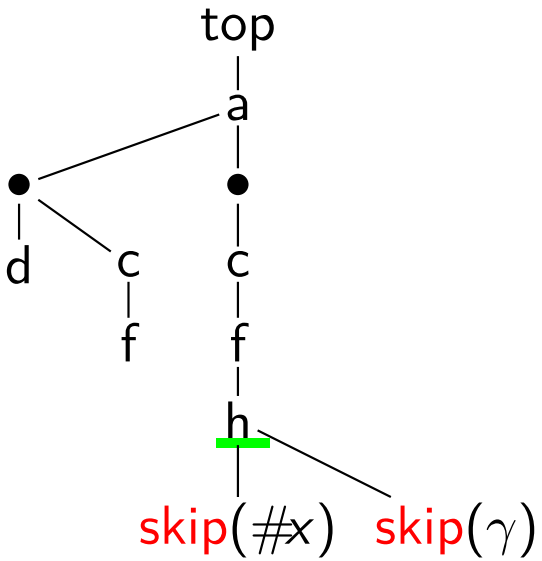


#a #d #c #f #c #f #h | γ #g

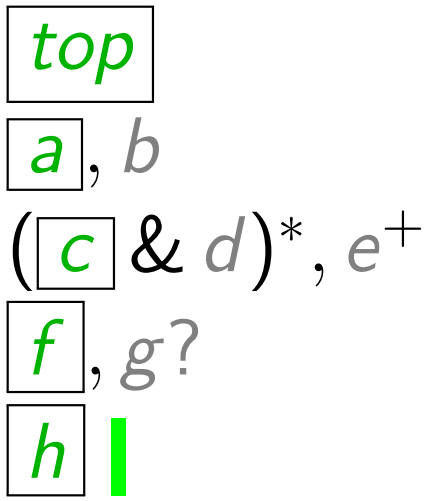
$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Signalling superfluous input

result tree



regexp stack



#a #d #c #f #c #f #h | #g ...

$$\begin{aligned} \text{top} &= a, b \\ a &= (c \& d)^*, e^+ \qquad b = i \mid k \\ c &= f, g? \\ f &= h? \\ d, e, g &= \#\text{chars} \qquad h, i, k = \#\text{empty} \end{aligned}$$

Possible rendering of embedded error messages:

In the last years, meta_tools have been successfully employed in very different medium-scale industrial, private and administrative applications.

#titel Components of # **#mt** // == #
#p (title)

The components of meta_tools range from small utility libraries, which can be used ubiquitously, up to large source code generating systems.

---

1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

4 Error Recovery and Diagnosis

5 Documentation and Subsequent Processing

6 Comprehension

7 Prototypical Applications

Source of rendering and documentation

```
#d2d module basic
  module inlineElements

    // ...

    tags nl = #empty
    docu to_xhtml_1_0 nl = "#br/"
    docu user_en nl = "Forces a line break, but not a new paragraph."
    docu user_de nl =
       #d2d Erzwingt Zeilenumbruch, aber #emph!keinen! Absatz.#/d2d

    // etc.

  end module
end module
```

Source of rendering and documentation

```
#d2d module basic
  module inlineElements

    // ...

    tags nl = #empty
    docu to_xhtml_1_0 nl = "#br/"
    docu user_en nl = "Forces a line break, but not a new paragraph."
    docu user_de nl =
       #d2d Erzwingt Zeilenumbruch, aber #emph!keinen! Absatz.#/d2d

    // etc.

  end module
end module
```

element declaration and content model

Source of rendering and documentation

```
#d2d module basic
  module inlineElements

    // ...

    tags nl = #empty
    docu to_xhtml_1_0 nl = "#br/"
    docu user_en nl = "Forces a line break, but not a new paragraph."
    docu user_de nl =
       #d2d Erzwingt Zeilenumbruch, aber #emph!keinen! Absatz.#/d2d

    // etc.

  end module
end module
```

element declaration and content model

rendering by XSLT
written as d2d
here: to XHTML 1.0

Source of rendering and documentation

```
#d2d module basic
  module inlineElements

    // ...

    tags nl = #empty
    docu to_xhtml_1_0 nl = "#br/"
    docu user_en nl = "Forces a line break, but not a new paragraph."
    docu user_de nl =
      #d2d Erzwingt Zeilenumbruch, aber #emph!keinen! Absatz.#/d2d

    // etc.

  end module
end module
```

element declaration and content model

rendering by XSLT written as d2d here: to XHTML 1.0

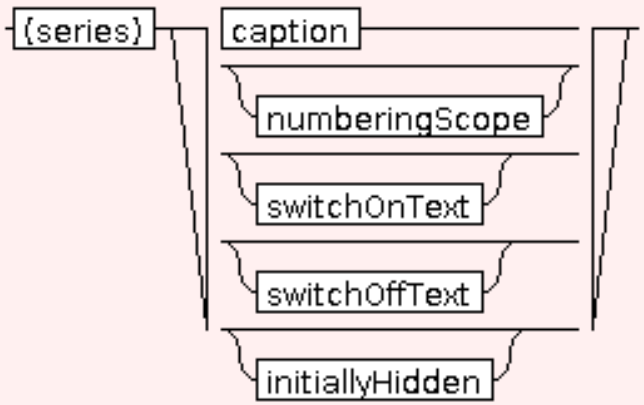
multi-lingual user documentation texts (as d2d)

Rendered Documentation, Navigable

TAGS parser `basic.floatings:` **`listOf`** via STR.FLOAT

Will be expanded to a list of the floating objects of a certain series.

Attention: The first tag is implicit, ie. may not appear in source text.




```
#implicit STR.FLOAT.series,
(STR.FLOAT.caption &
 STR.FLOAT.numberingScope? &
 STR.FLOAT.switchOnText? &
 STR.FLOAT.switchOffText? &
 STR.FLOAT.initiallyHidden?)
```

Used in : article -- webpage

Source is at SYSTEM "http://bandm.eu/doctypes/d2d_gp/basic.dd2":1878.5-1880.57. Xml tagging is `{http://bandm.eu/doctypes/d2d_gp/basic}a:listOf`

Rendered Documentation, Navigable

source text module name

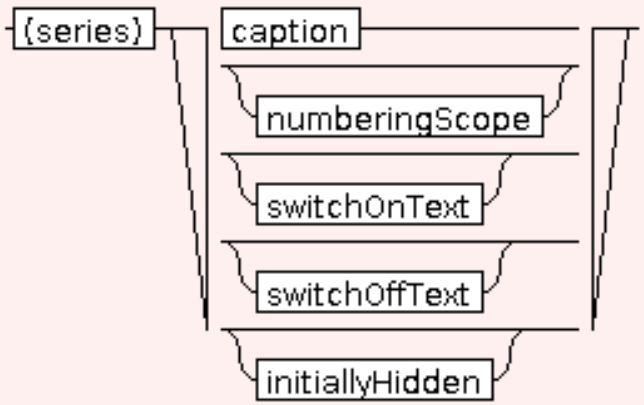
d2d tag

path of imports

TAGS parser `basic.floatings:` **`listOf`** via STR.FLOAT

Will be expanded to a list of the floating objects of a certain series.

Attention: The first tag is implicit, ie. may not appear in source text.




```
#implicit STR.FLOAT.series,
(STR.FLOAT.caption &
 STR.FLOAT.numberingScope? &
 STR.FLOAT.switchOnText? &
 STR.FLOAT.switchOffText? &
 STR.FLOAT.initiallyHidden?)
```

Used in : article -- webpage

Source is at SYSTEM "http://bandm.eu/doctypes/d2d_gp/basic.dd2":1878.5-1880.57. Xml tagging is `{http://bandm.eu/doctypes/d2d_gp/basic}a:listOf`

Rendered Documentation, Navigable

source text module name

d2d tag

path of imports

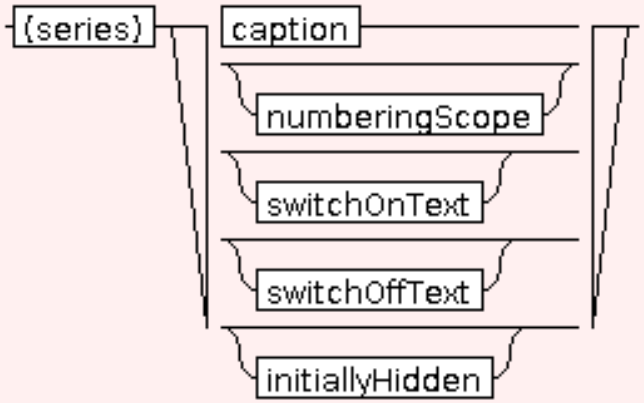
explicit doc text

automatic warning about inferred open tag

TAGS parser `basic.floatings:` **`listOf`** via STR.FLOAT

Will be expanded to a list of the floating objects of a certain series.

Attention: The first tag is implicit, ie. may not appear in source text.




```
#implicit STR.FLOAT.series,
(STR.FLOAT.caption &
 STR.FLOAT.numberingScope? &
 STR.FLOAT.switchOnText? &
 STR.FLOAT.switchOffText? &
 STR.FLOAT.initiallyHidden?)
```

Used in : article -- webpage

Source is at SYSTEM "http://bandm.eu/doctypes/d2d_gp/basic.dd2":1878.5-1880.57. Xml tagging is `{http://bandm.eu/doctypes/d2d_gp/basic}a:listOf`

Rendered Documentation, Navigable

source text module name

d2d tag

TAGS parser `basic.floatings:` **`listOf`** via STR.FLOAT

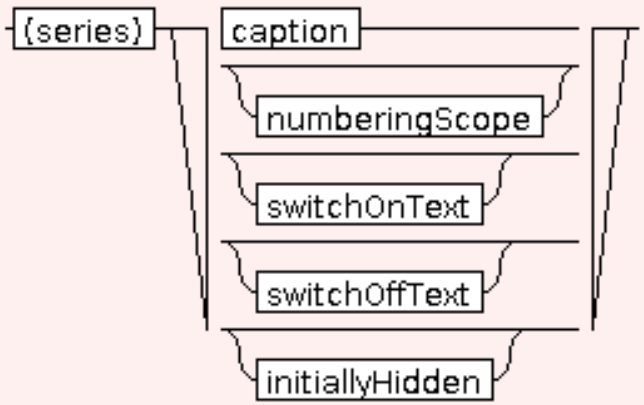
path of imports

Will be expanded to a list of the floating objects of a certain series.

explicit doc text

Attention: The first tag is implicit, ie. may not appear in source text.

automatic warning about inferred open tag



Wirth diagram (here: permutation operator)

source text (reconstructed)

```
#implicit STR.FLOAT.series,
(STR.FLOAT.caption &
 STR.FLOAT.numberingScope? &
 STR.FLOAT.switchOnText? &
 STR.FLOAT.switchOffText? &
 STR.FLOAT.initiallyHidden?)
```

Used in : article -- webpage

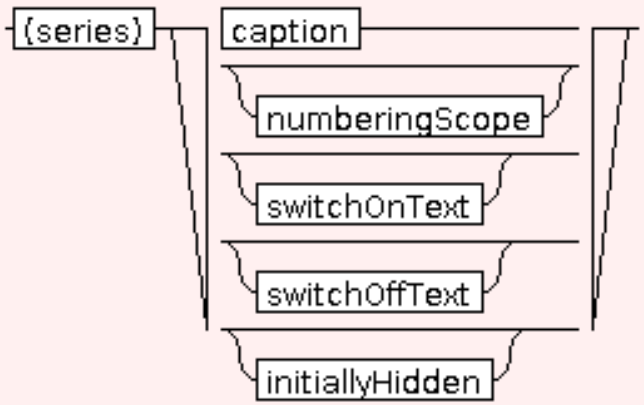
Source is at SYSTEM "http://bandm.eu/doctypes/d2d_gp/basic.dd2":1878.5-1880.57. Xml tagging is `{http://bandm.eu/doctypes/d2d_gp/basic}a:listOf`

Rendered Documentation, Navigable

TAGS parser `basic.floatings:` **`listOf`** via STR.FLOAT

Will be expanded to a list of the floating objects of a certain series.

Attention: The first tag is implicit, ie. may not appear in source text.




```
#implicit STR.FLOAT.series,
(STR.FLOAT.caption &
 STR.FLOAT.numberingScope? &
 STR.FLOAT.switchOnText? &
 STR.FLOAT.switchOffText? &
 STR.FLOAT.initiallyHidden?)
```

Used in : article -- webpage

Source is at SYSTEM "http://bandm.eu/doctypes/d2d_gp/basic.dd2":1878.5-1880.57. Xml tagging is `{http://bandm.eu/doctypes/d2d_gp/basic}a:listOf`

source text module name

d2d tag

path of imports

explicit doc text

automatic warning about inferred open tag

Wirth diagram (here: permutation operator)

source text (reconstructed)

referring elements

source file location

XML tag with namespace

1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

4 Error Recovery and Diagnosis

5 Documentation and Subsequent Processing

6 Comprehension

7 Prototypical Applications

<> Features and their Interactions:

- <> module parametrization by rewriting
- <> explicit tagging with minimal technical distraction
- <> robust parsing, synthesis of missing explicit tags
- <> implicit tagging by non-deterministic character parsing
- <> unification of attributes and element text content
- <> integration of multi-lingual documentation
- <> . . . and of subsequent processing by XSLT

1 Goals and Principles of d2d

2 Input and Parsing

3 Modules and Parametrization

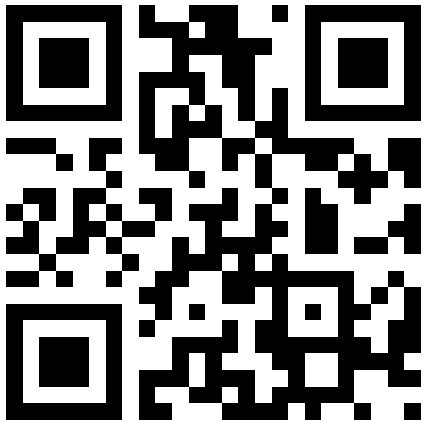
4 Error Recovery and Diagnosis

5 Documentation and Subsequent Processing

6 Comprehension

7 **Prototypical Applications**

links



Survey page at `http://bandm.eu/d2d`
in English and German, with literature and examples.