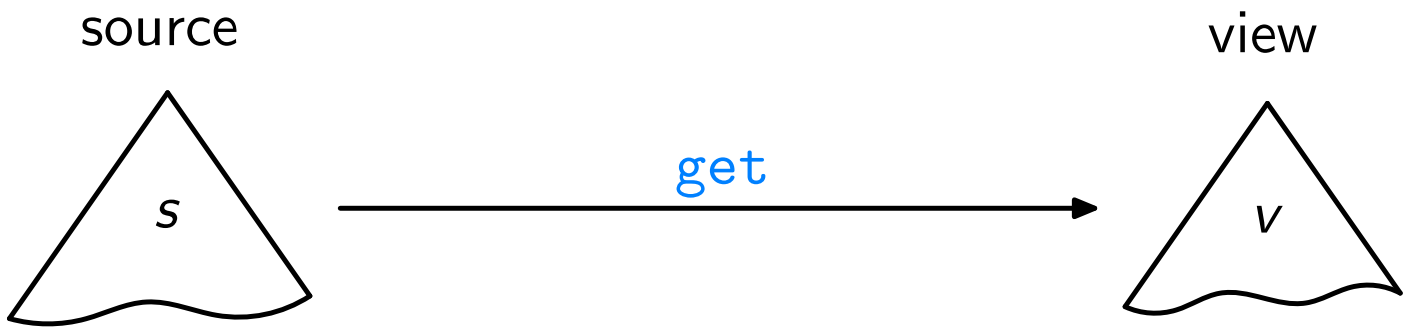
View Update: Programmiersprachliche Techniken für Bidirektionale Transformation

Janis Voigtländer

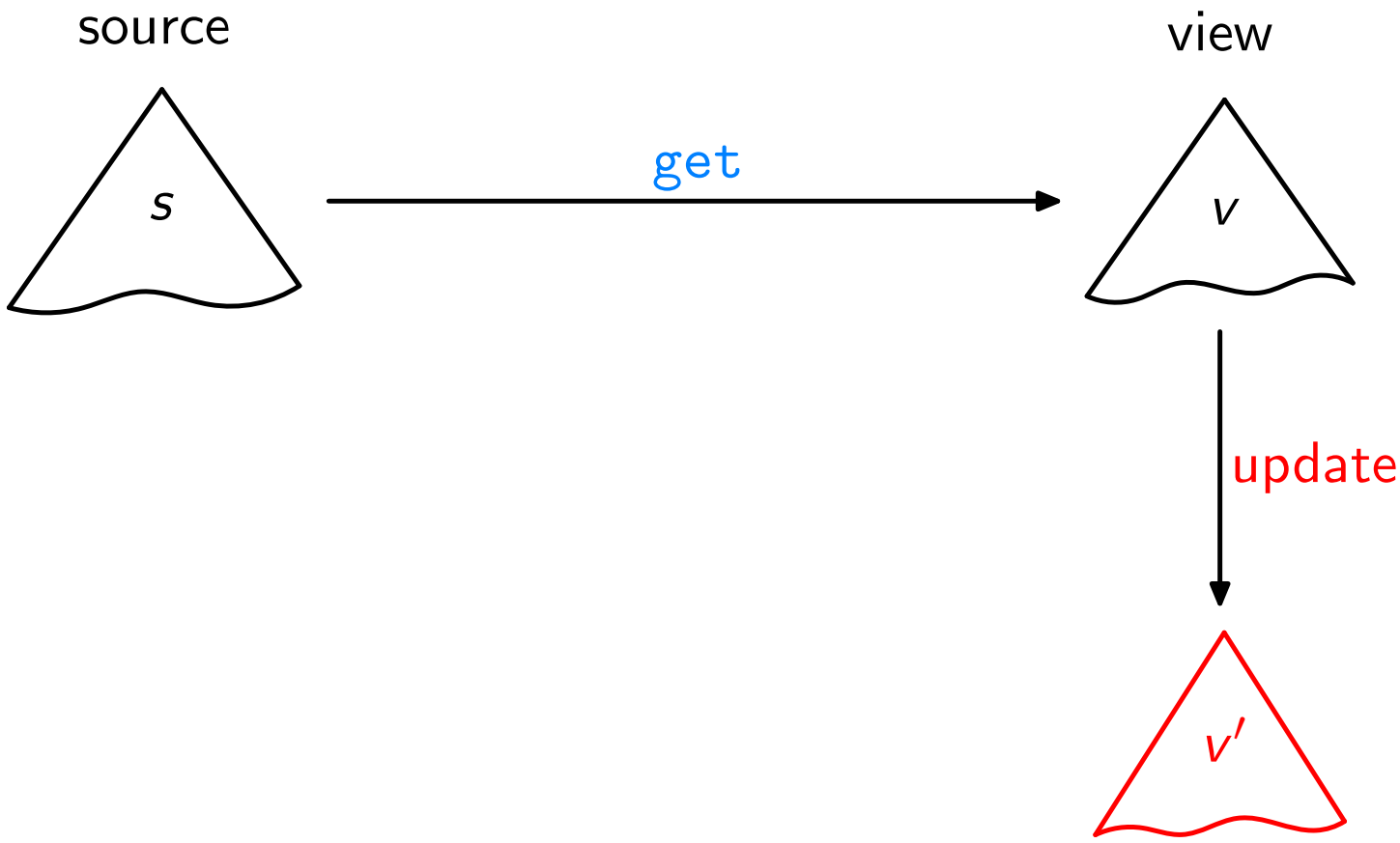
Universität Bonn

16. Juni 2011

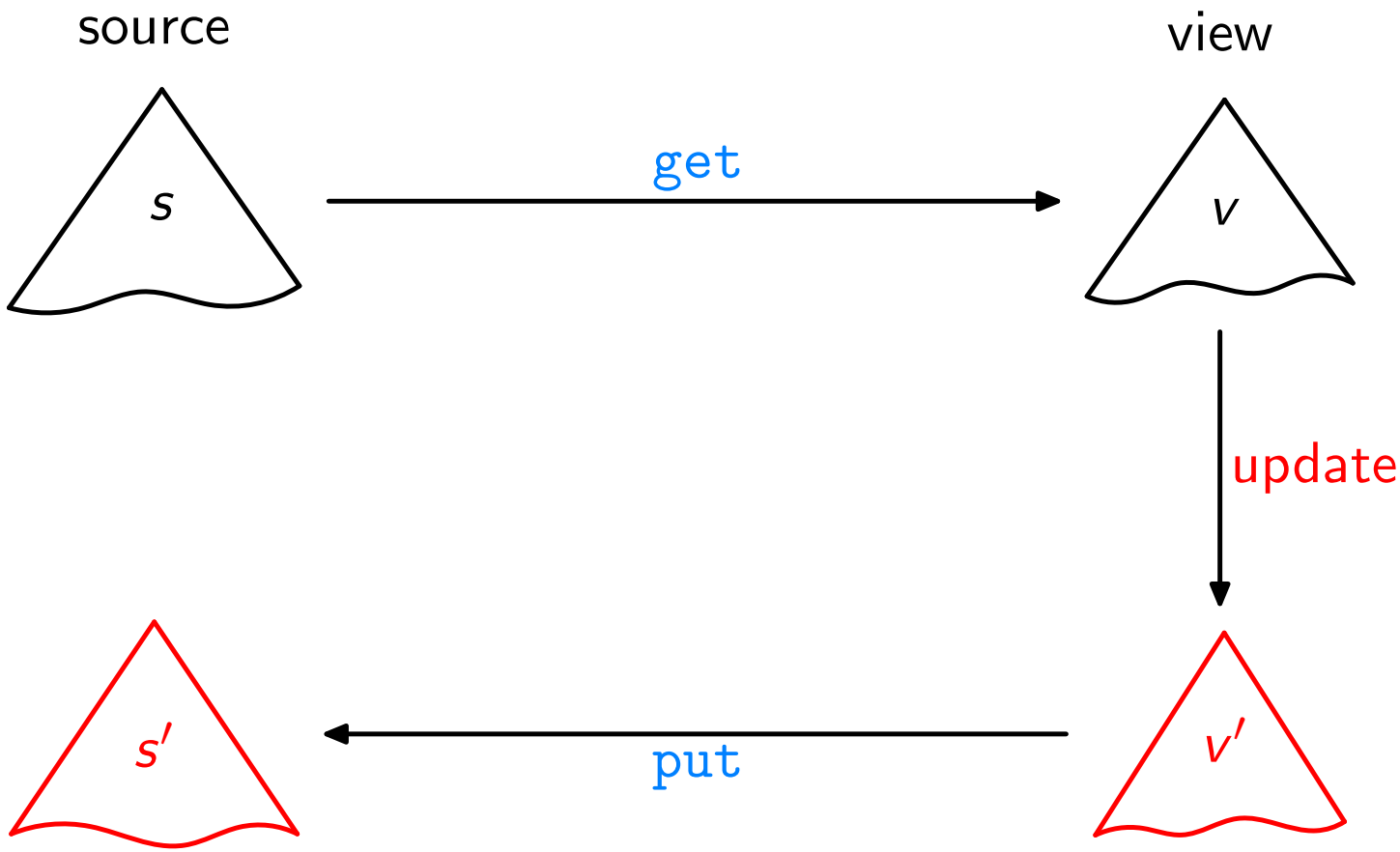
View Update, Bidirektionale Transformation



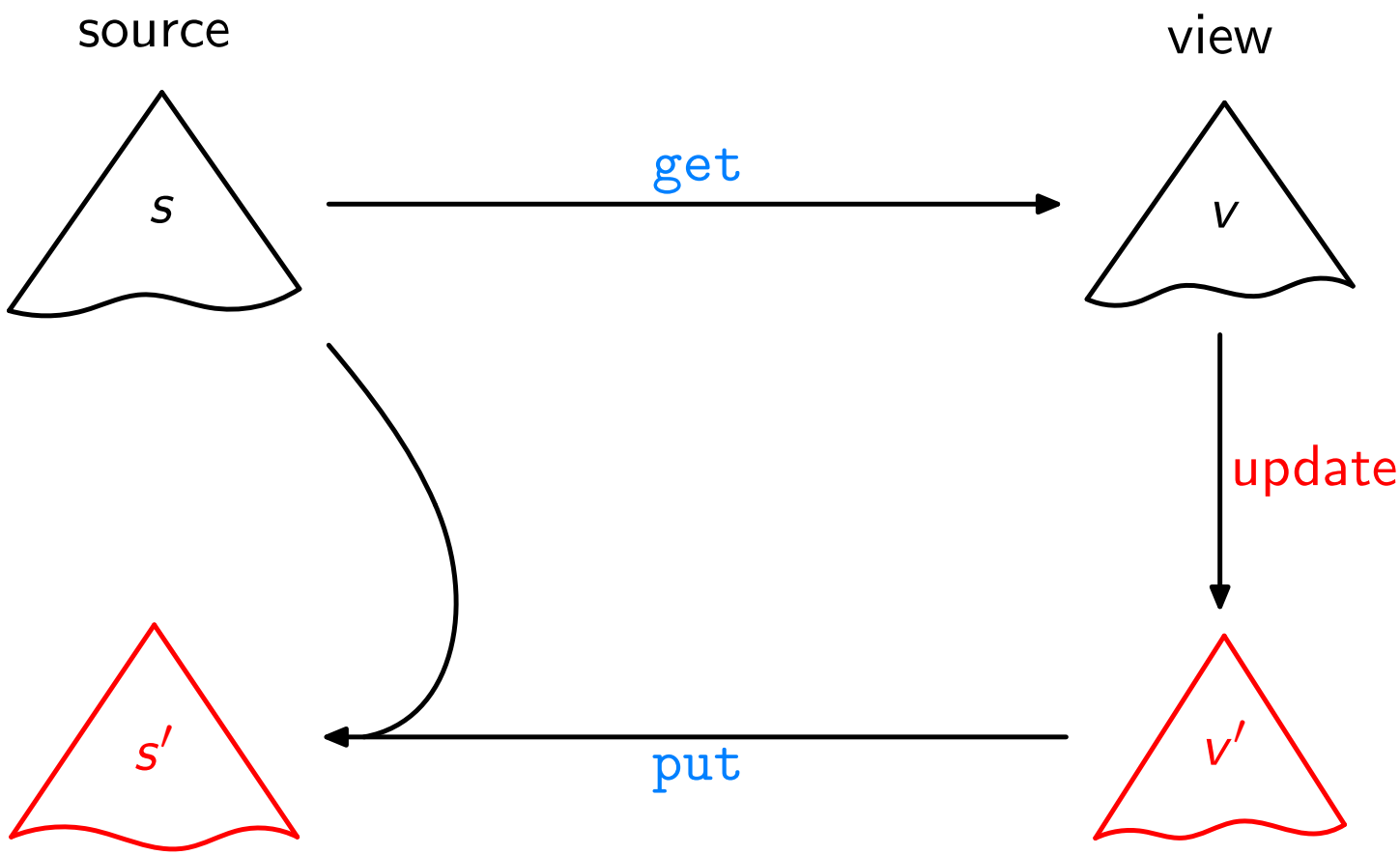
View Update, Bidirektionale Transformation



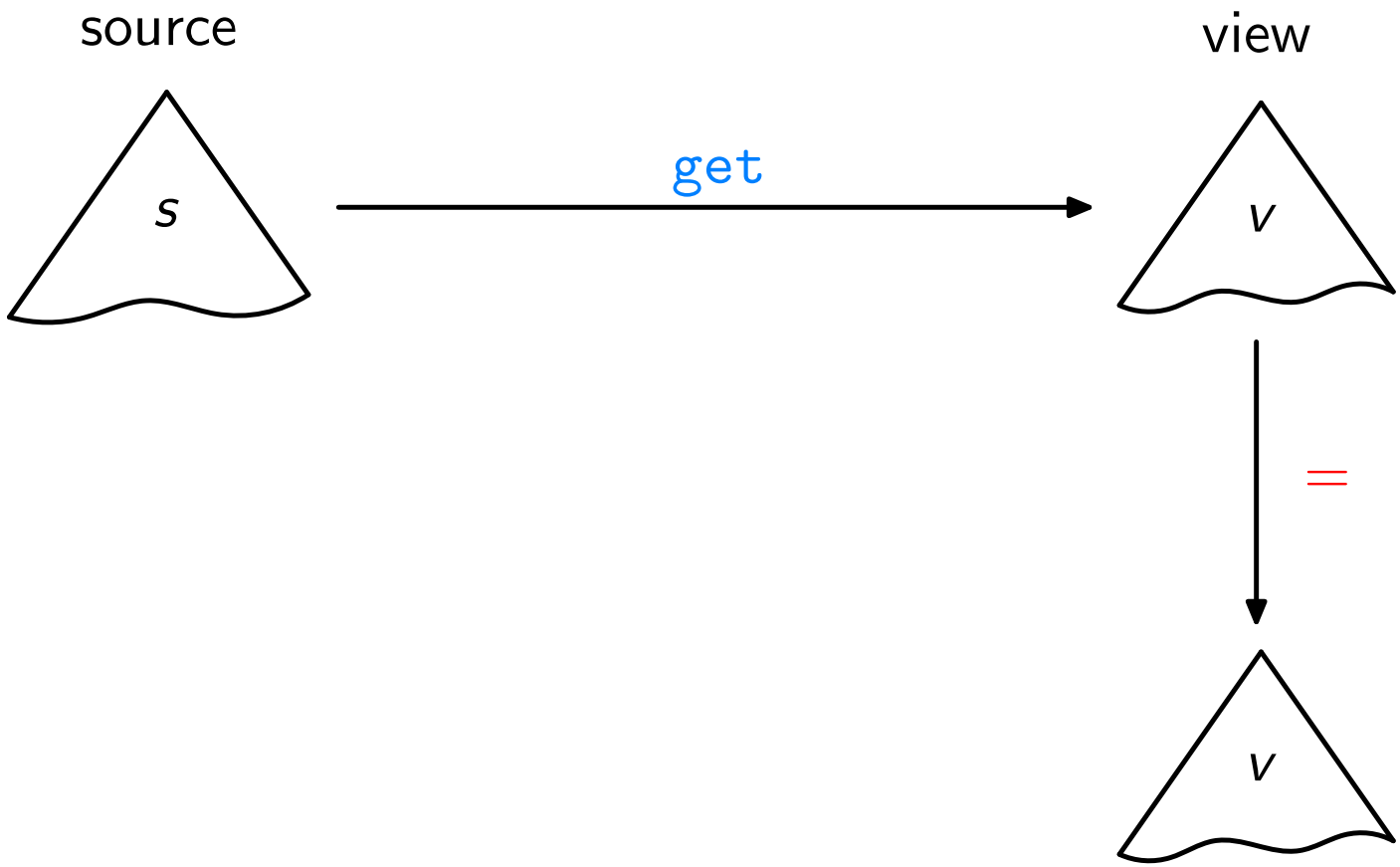
View Update, Bidirektionale Transformation



View Update, Bidirektionale Transformation

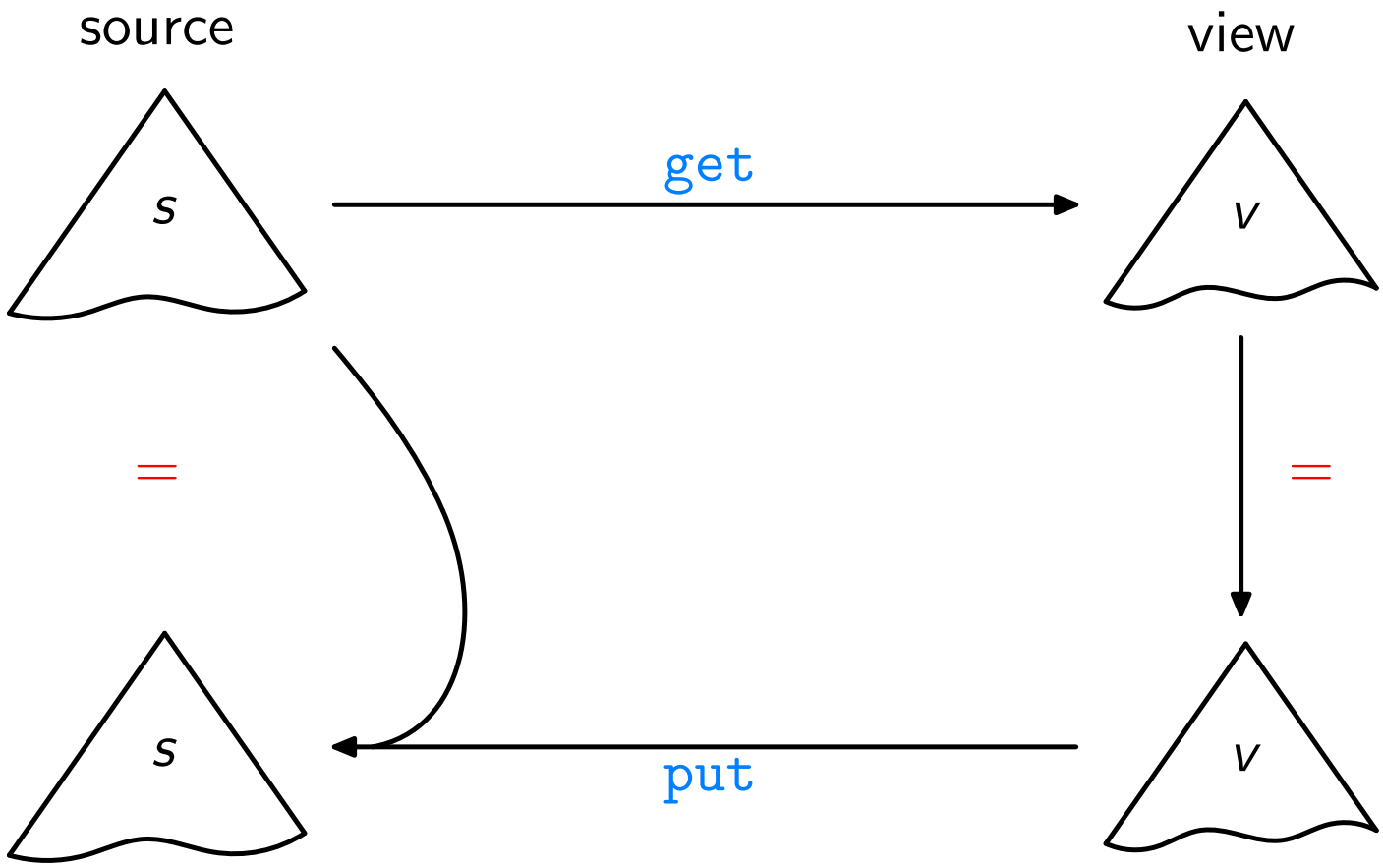


View Update, Bidirektionale Transformation



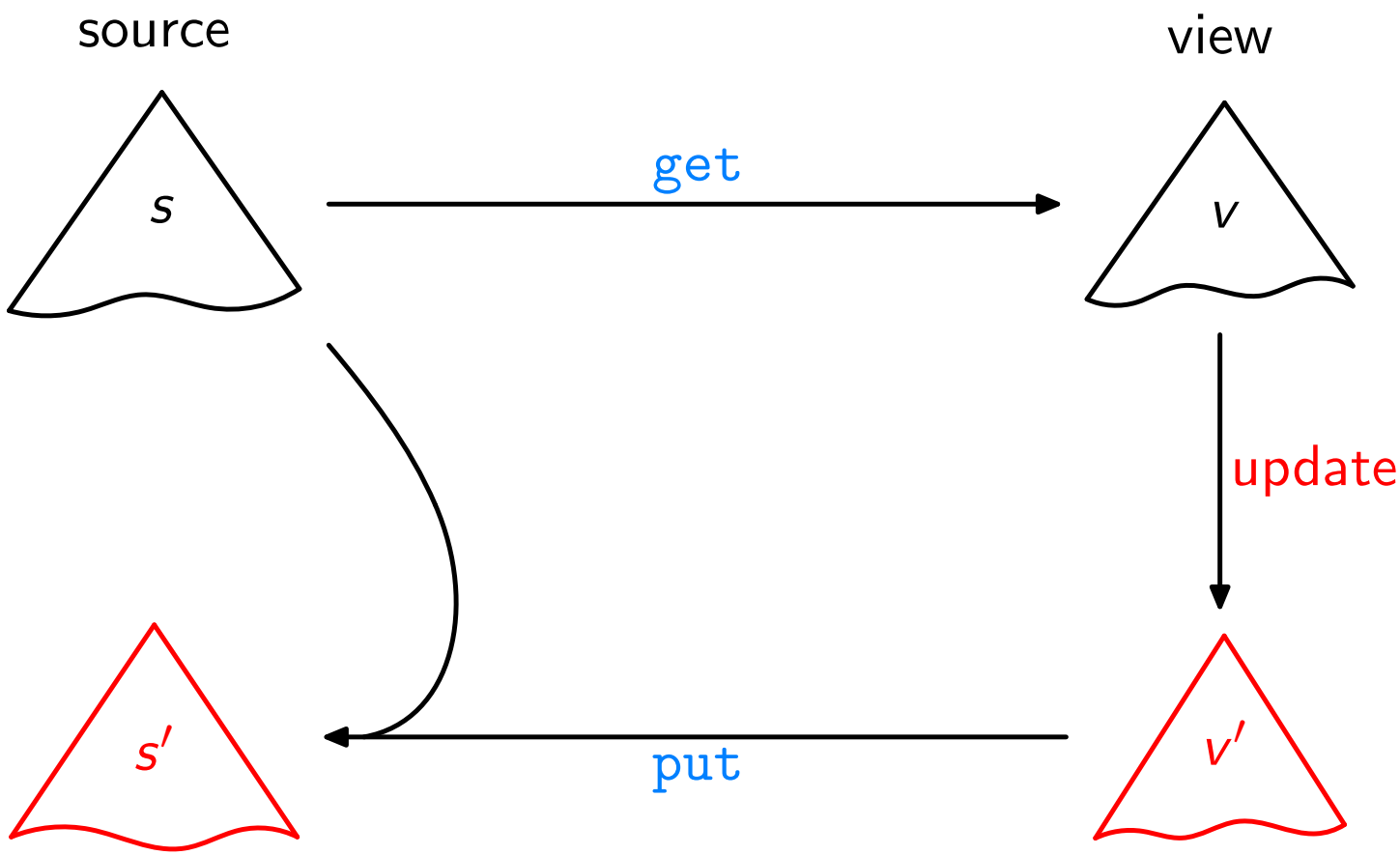
Acceptability / GetPut

View Update, Bidirektionale Transformation



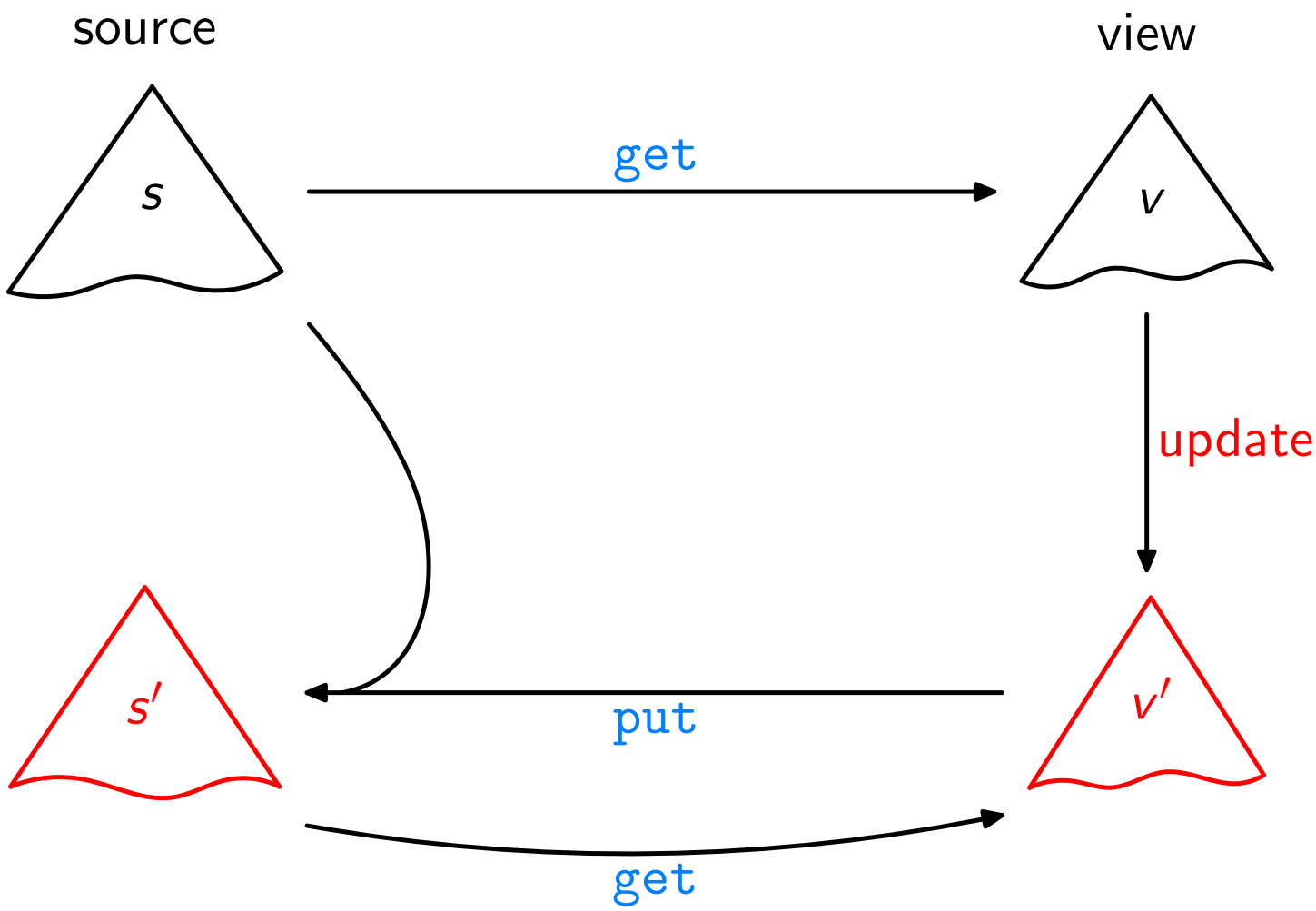
Acceptability / GetPut

View Update, Bidirektionale Transformation



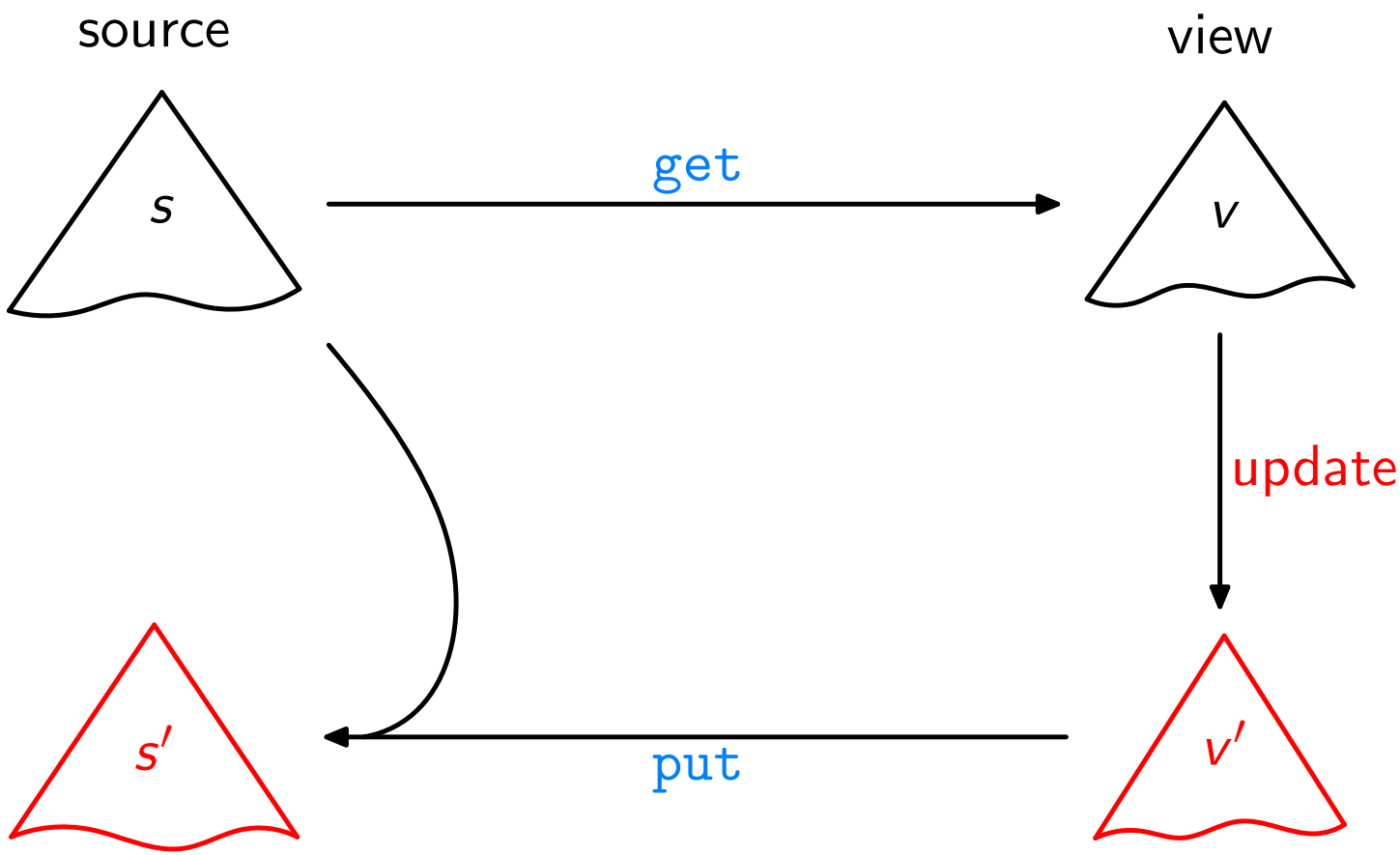
Consistency / PutGet

View Update, Bidirektionale Transformation

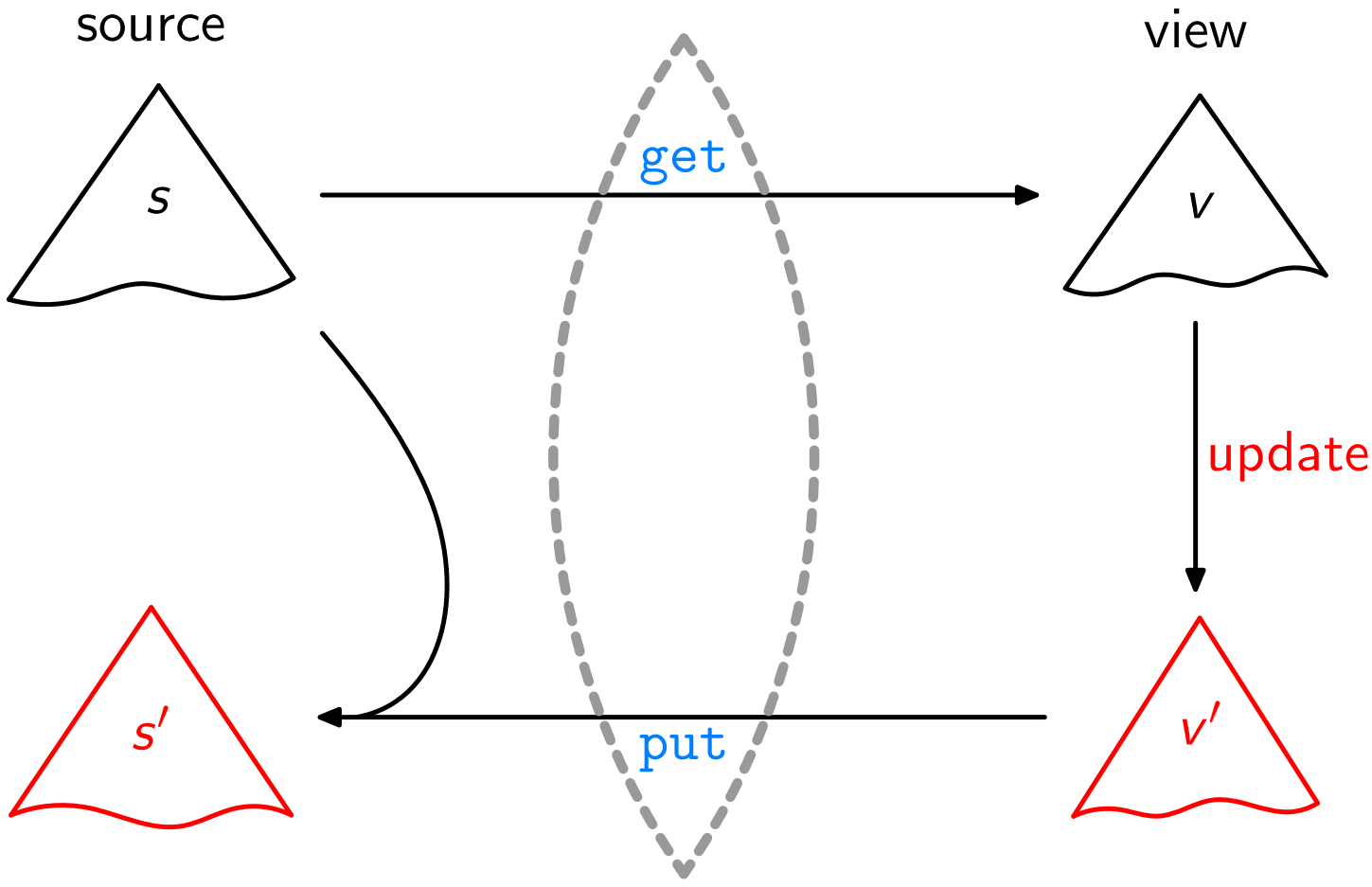


Consistency / PutGet

View Update, Bidirektionale Transformation



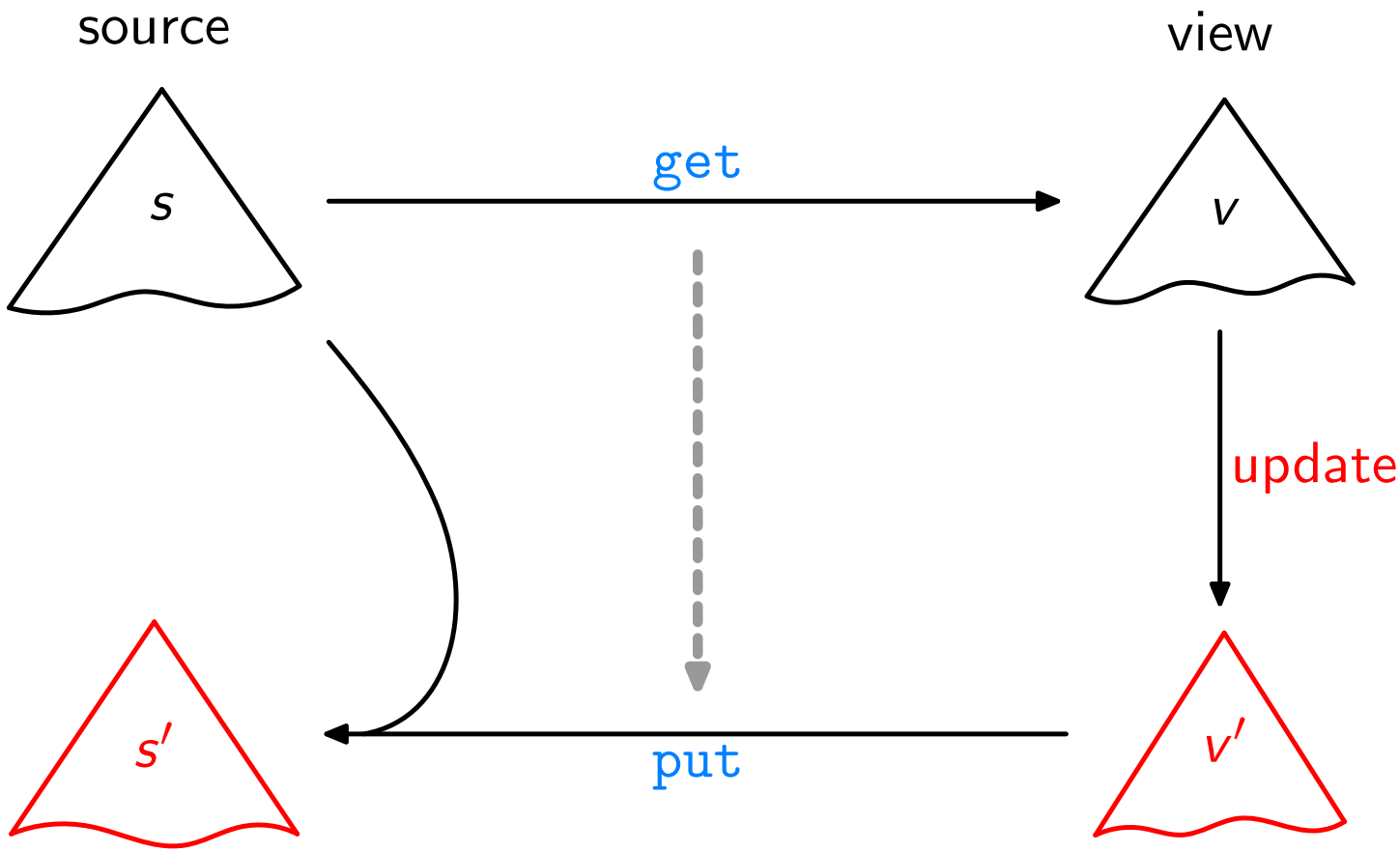
View Update, Bidirektionale Transformation



Lenses, DSLs

[Foster et al., ACM TOPLAS'07, . . .]

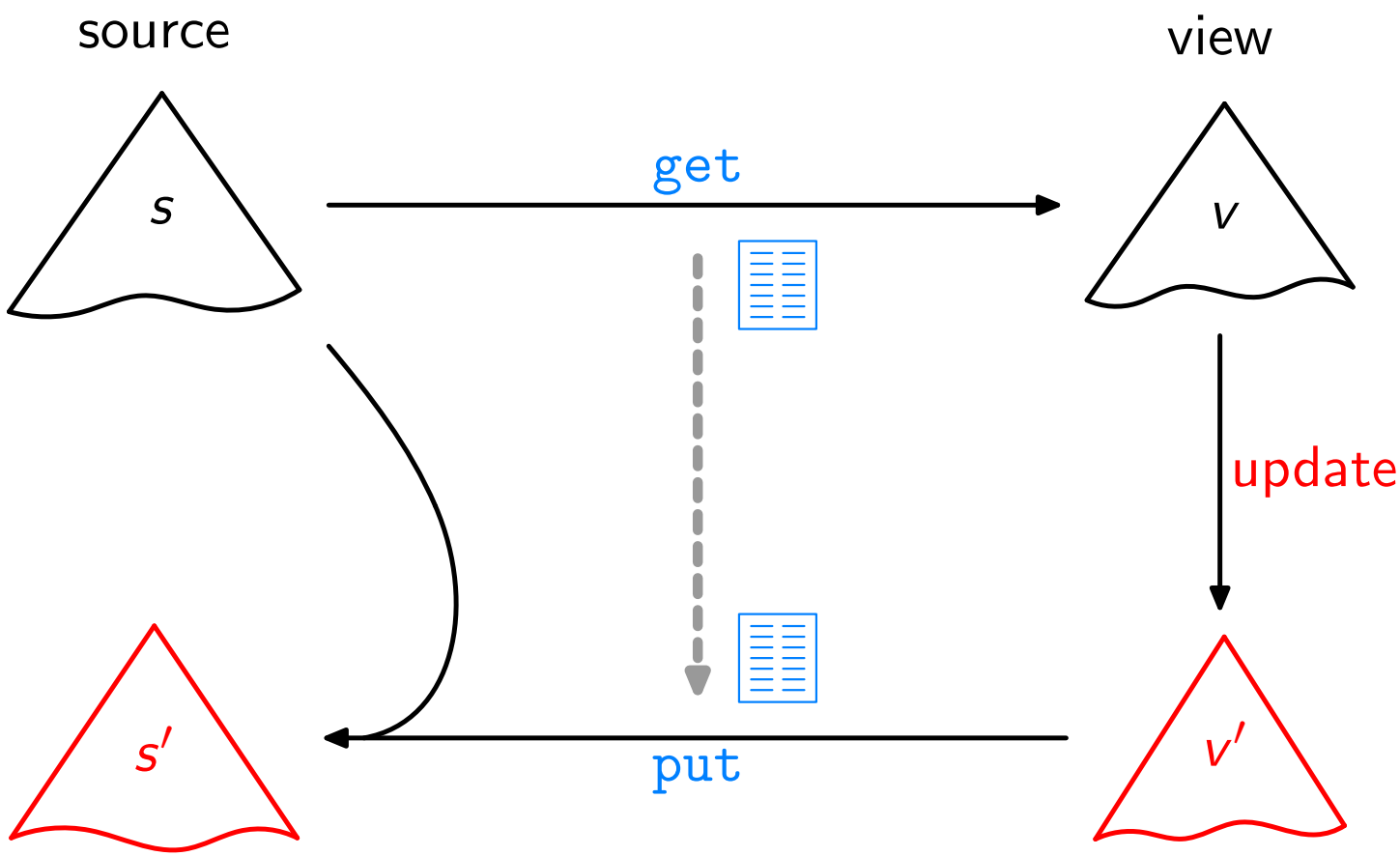
View Update, Bidirektionale Transformation



Bidirektionalisierung

[Matsuda et al., ICFP'07]

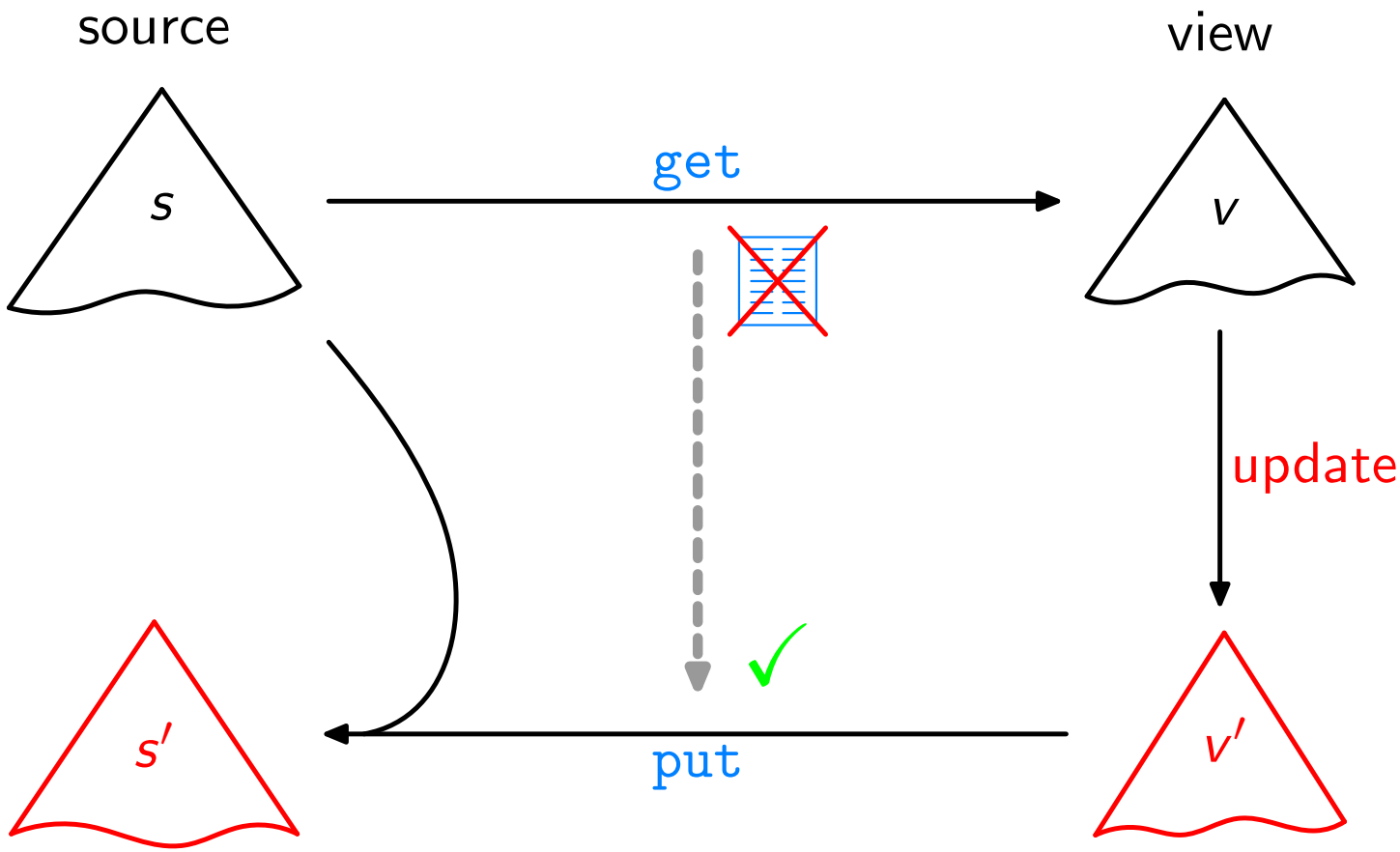
View Update, Bidirektionale Transformation



Syntaktische Bidirektionalisierung

[Matsuda et al., ICFP'07]

View Update, Bidirektionale Transformation



Semantische Bidirektionalisierung

[V., POPL'09]

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

[†] "Bidirectionalization for free!"

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

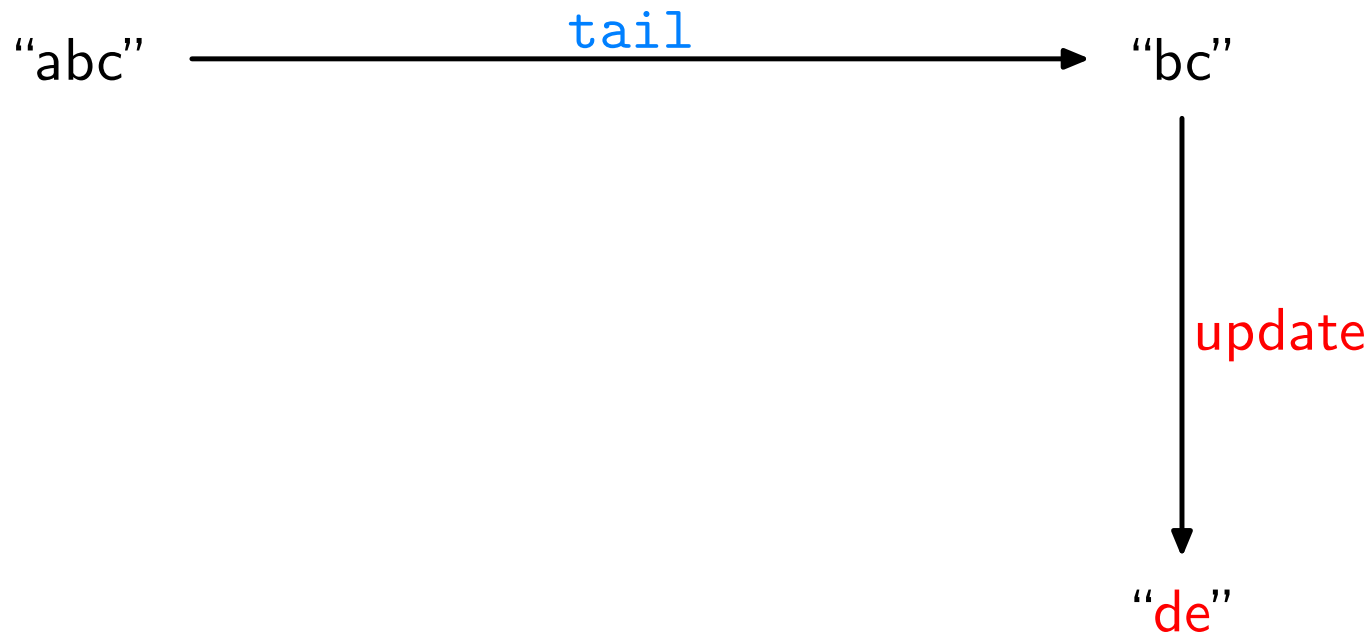
Beispiele:

[†] "Bidirectionalization for free!"

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:

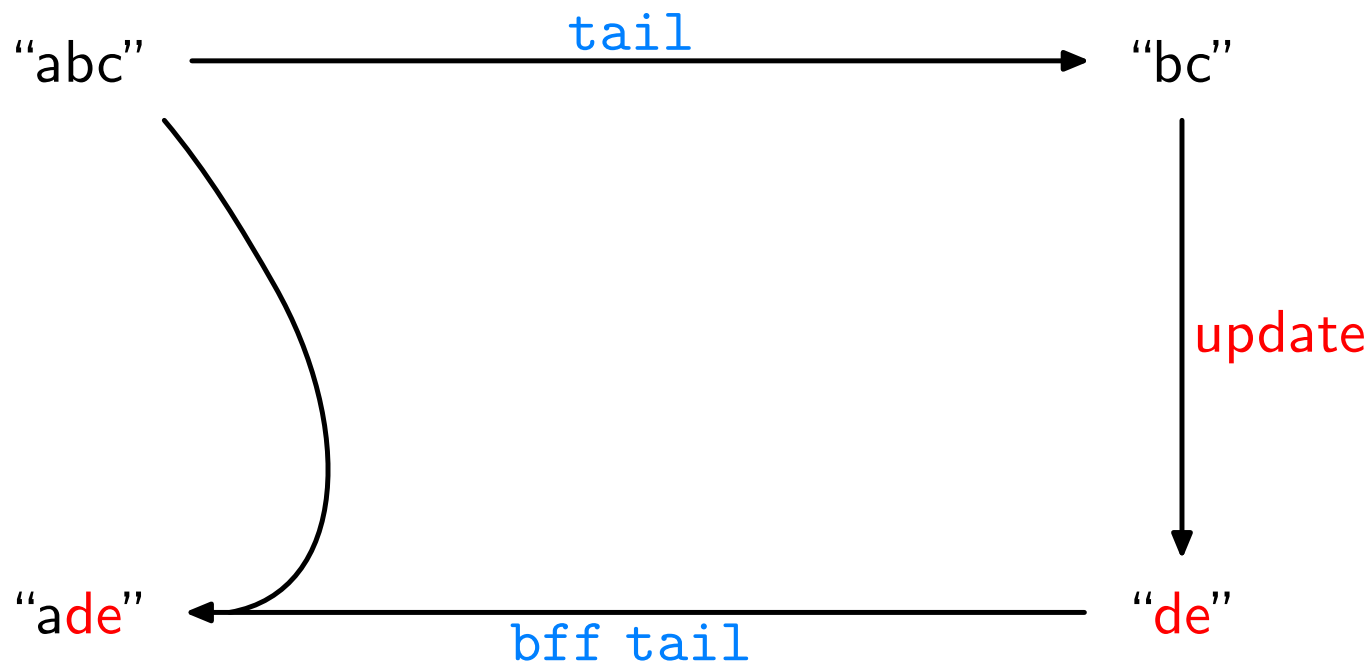


[†] “Bidirectionalization for free!”

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:

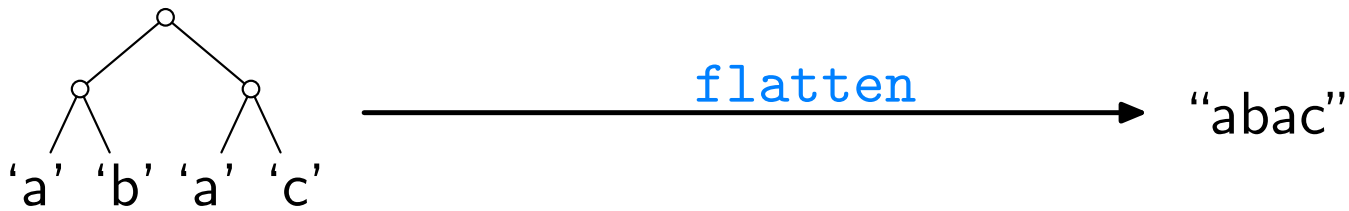


[†] “Bidirectionalization for free!”

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:

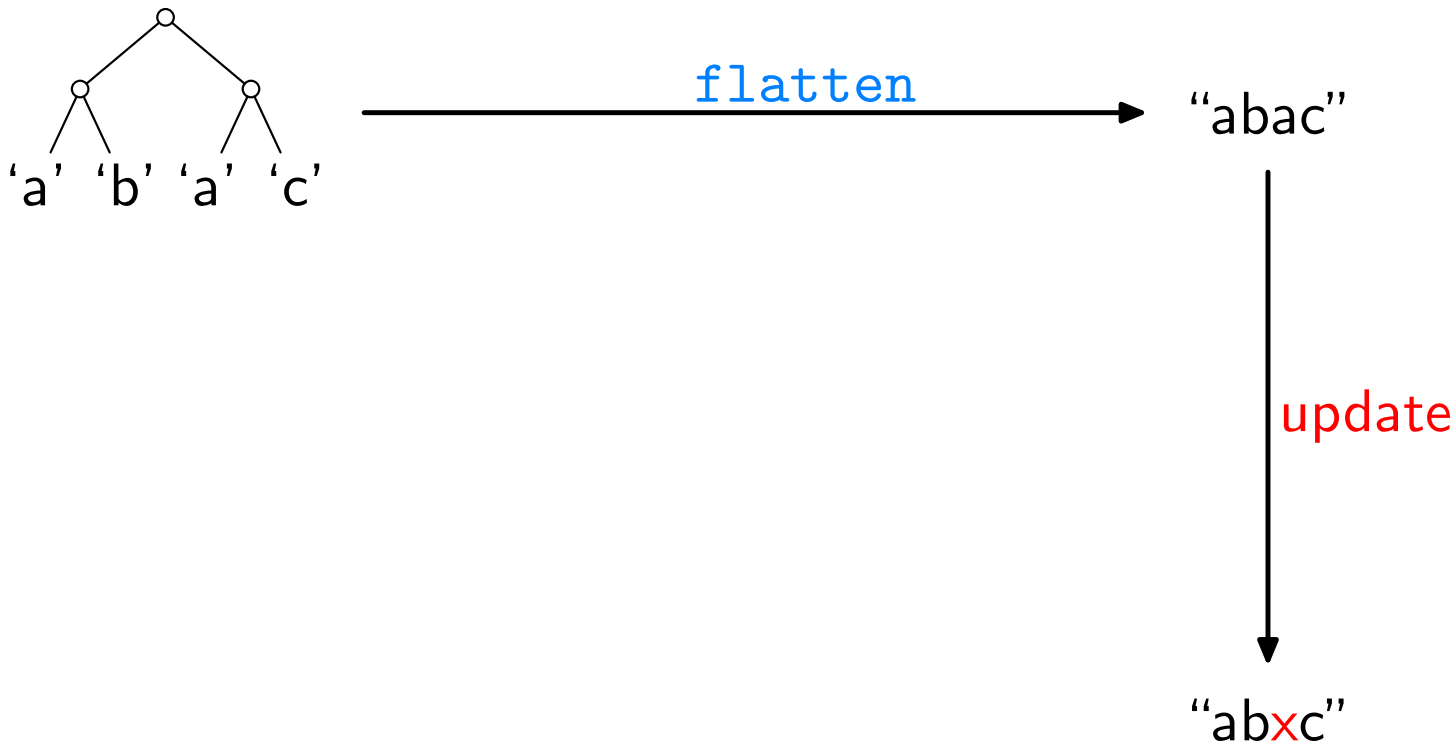


[†] "Bidirectionalization for free!"

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:

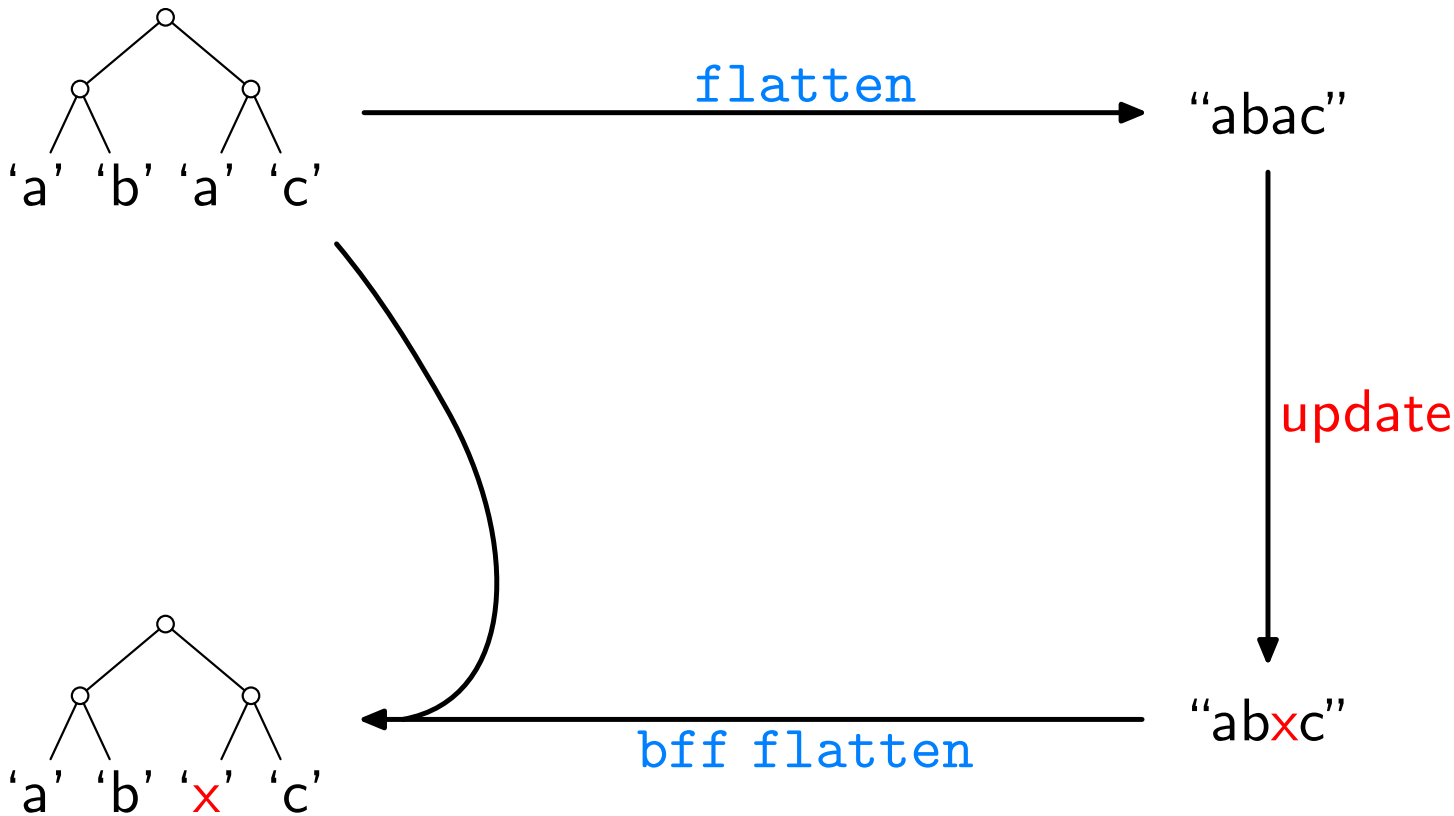


[†] "Bidirectionalization for free!"

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:

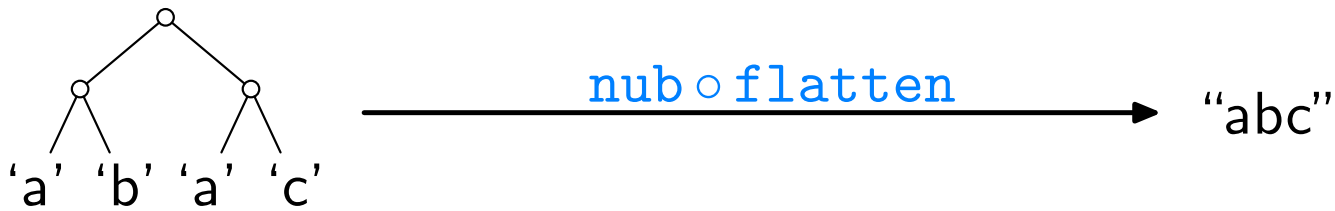


[†] “Bidirectionalization for free!”

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:



[†] "Bidirectionalization for free!"

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

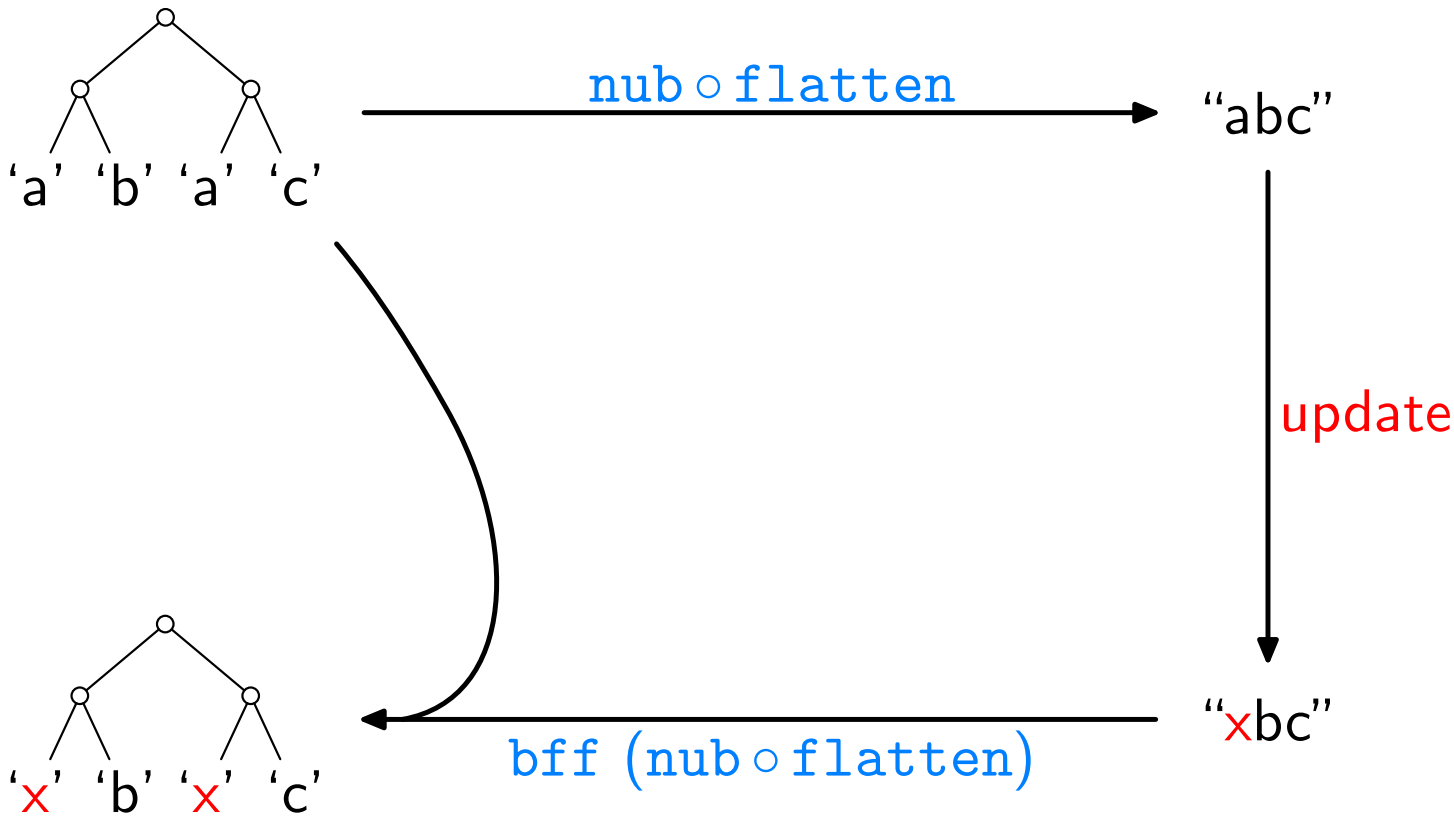
Beispiele:

[†] "Bidirectionalization for free!"

Semantische Bidirektionalisierung

Ziel: Eine Higher-Order Funktion `bff`[†], so dass jeweils für `get` und `bff get` gelten: GetPut, PutGet, ...

Beispiele:



[†] "Bidirectionalization for free!"

Untersuchung Spezifischer Instanzen

Gegeben sei eine Funktion

$$\texttt{get} :: [\alpha] \to [\alpha]$$

Wie können wir, oder `bff`, sie ohne Zugriff auf den Quellcode analysieren?

Untersuchung Spezifischer Instanzen

Gegeben sei eine Funktion

$$\texttt{get} :: [\alpha] \to [\alpha]$$

Wie können wir, oder `bff`, sie ohne Zugriff auf den Quellcode analysieren?

Idee: Anwendung von `get` auf ausgewählte Eingaben

Untersuchung Spezifischer Instanzen

Gegeben sei eine Funktion

$$\texttt{get} :: [\alpha] \to [\alpha]$$

Wie können wir, oder `bff`, sie ohne Zugriff auf den Quellcode analysieren?

Idee: Anwendung von `get` auf ausgewählte Eingaben

Wie etwa:

$$\texttt{get}\ [0..n] = \begin{cases} [1..n] & \text{wenn } \texttt{get} = \texttt{tail} \\ [n..0] & \text{wenn } \texttt{get} = \texttt{reverse} \\ [0..(\texttt{min}\ 4\ n)] & \text{wenn } \texttt{get} = \texttt{take}\ 5 \\ \vdots & \end{cases}$$

Untersuchung Spezifischer Instanzen

Gegeben sei eine Funktion

$$\texttt{get} :: [\alpha] \to [\alpha]$$

Wie können wir, oder `bff`, sie ohne Zugriff auf den Quellcode analysieren?

Idee: Anwendung von `get` auf ausgewählte Eingaben

Wie etwa:

$$\texttt{get}\ [0..n] = \begin{cases} [1..n] & \text{wenn } \texttt{get} = \texttt{tail} \\ [n..0] & \text{wenn } \texttt{get} = \texttt{reverse} \\ [0..(\texttt{min}\ 4\ n)] & \text{wenn } \texttt{get} = \texttt{take}\ 5 \\ \vdots & \end{cases}$$

Dann, Übertragung der gewonnenen Einsichten auf andere Ursprungslisten als $[0..n]$!

Verwendung eines Freien Theorems [Wadler, FPCA'89]

Für jede Funktion

$$\texttt{g} :: [\alpha] \to [\alpha]$$

gilt

$$\texttt{map}\ f\ (\texttt{g}\ l) \quad = \quad \texttt{g}\ (\texttt{map}\ f\ l)$$

für beliebige f und l, wobei:

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Verwendung eines Freien Theorems [Wadler, FPCA'89]

Für jede Funktion

$$\texttt{g} :: [\alpha] \to [\alpha]$$

gilt

$$\texttt{map}\ f\ (\texttt{g}\ l) \quad = \quad \texttt{g}\ (\texttt{map}\ f\ l)$$

für beliebige f und l, wobei:

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\ = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Gegeben eine beliebige Liste s der Länge $n + 1$, setze $\texttt{g} = \texttt{get}$, $l = [0..n]$, $f = (s\,!!)$, dann:

$$\texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n]) \quad = \quad \texttt{get}\ (\texttt{map}\ (s\,!!)\ [0..n])$$

Verwendung eines Freien Theorems [Wadler, FPCA'89]

Für jede Funktion

$$\texttt{g} :: [\alpha] \to [\alpha]$$

gilt

$$\texttt{map}\ f\ (\texttt{g}\ l) \quad = \quad \texttt{g}\ (\texttt{map}\ f\ l)$$

für beliebige f und l, wobei:

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Gegeben eine beliebige Liste s der Länge $n + 1$, setze $\texttt{g} = \texttt{get}$, $l = [0..n]$, $f = (s\,!!)$, dann:

$$\begin{aligned}
\texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n]) \quad &= \quad \texttt{get}\ (\underbrace{\texttt{map}\ (s\,!!)\ [0..n]}_{s}) \\
&= \quad \texttt{get}\ s
\end{aligned}$$

Verwendung eines Freien Theorems [Wadler, FPCA'89]

Für jede Funktion

$$\texttt{g} :: [\alpha] \to [\alpha]$$

gilt

$$\texttt{map}\ f\ (\texttt{g}\ l) \quad = \quad \texttt{g}\ (\texttt{map}\ f\ l)$$

für beliebige f und l, wobei:

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\ = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Gegeben eine beliebige Liste s der Länge $n + 1$,

$$\texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n]) \quad = \quad \texttt{get}\ s$$

Verwendung eines Freien Theorems [Wadler, FPCA'89]

Für jede Funktion

$$\texttt{g} :: [\alpha] \to [\alpha]$$

gilt

$$\texttt{map}\ f\ (\texttt{g}\ l) \quad = \quad \texttt{g}\ (\texttt{map}\ f\ l)$$

für beliebige f und l, wobei:

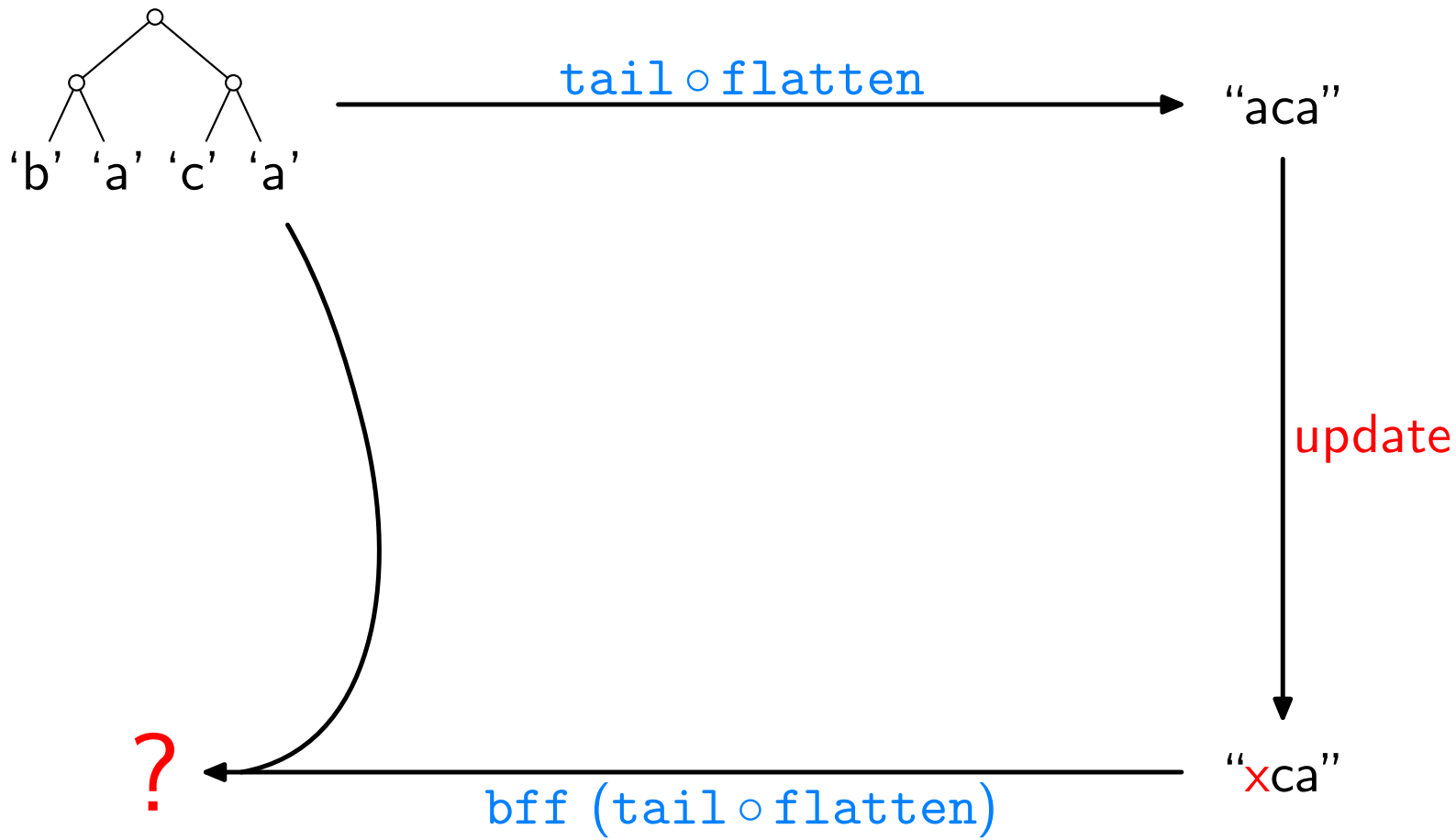
$$\begin{aligned} &\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\ &\texttt{map}\ f\ [\,] \qquad\ = [\,] \\ &\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as) \end{aligned}$$

Gegeben eine beliebige Liste s der Länge $n + 1$,

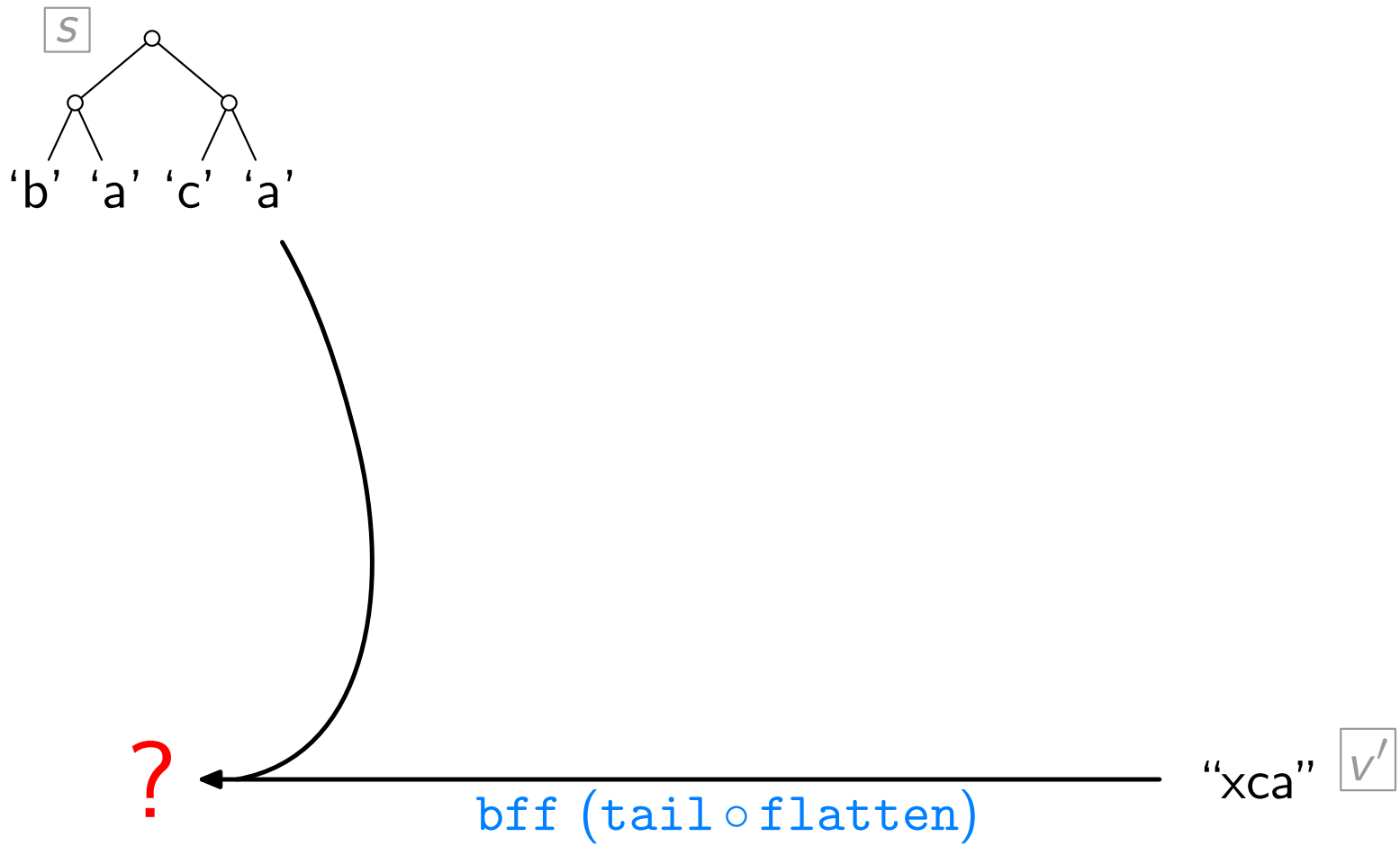
$$\texttt{get}\ s \quad = \quad \texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n])$$

für jedes $\texttt{get} :: [\alpha] \to [\alpha]$.

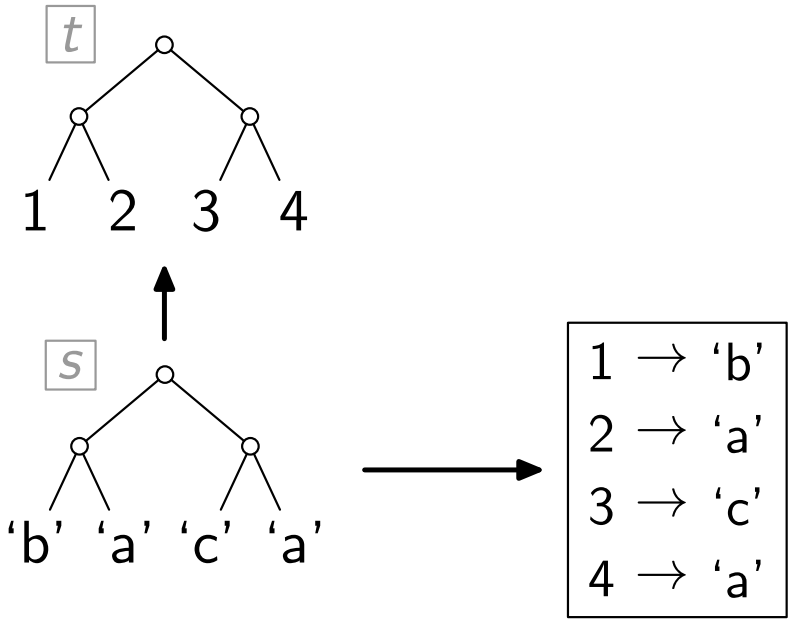
Der Semantische Ansatz am Beispiel



Der Semantische Ansatz am Beispiel

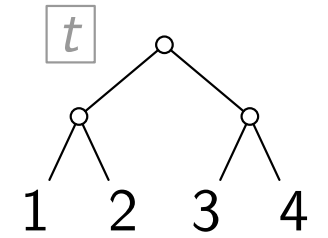


Der Semantische Ansatz am Beispiel

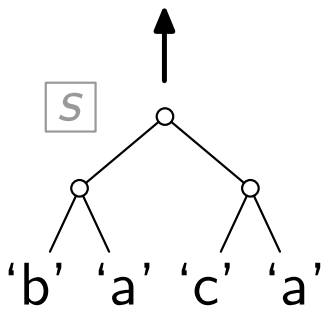


“xca” v'

Der Semantische Ansatz am Beispiel



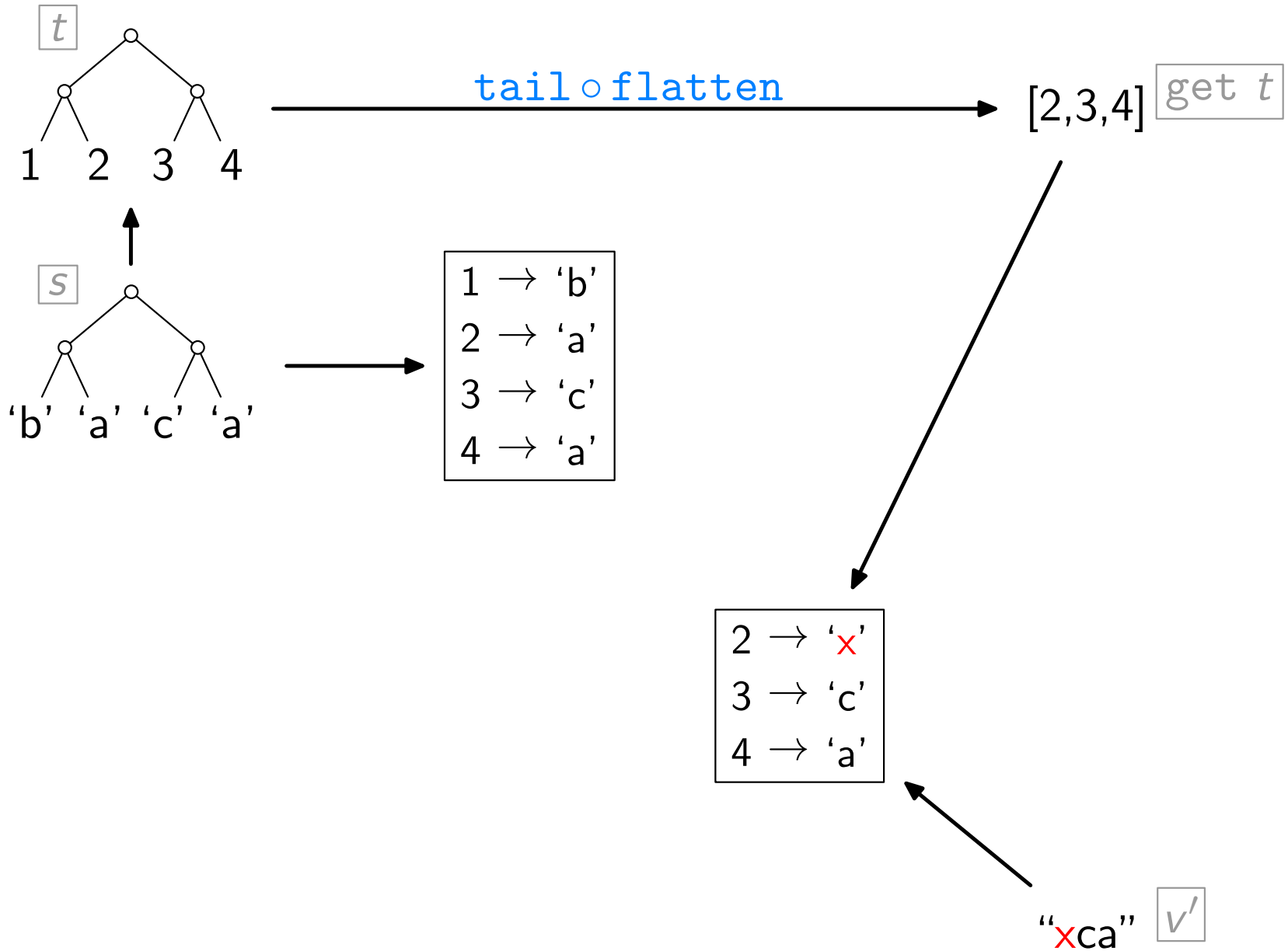
[2,3,4] get t



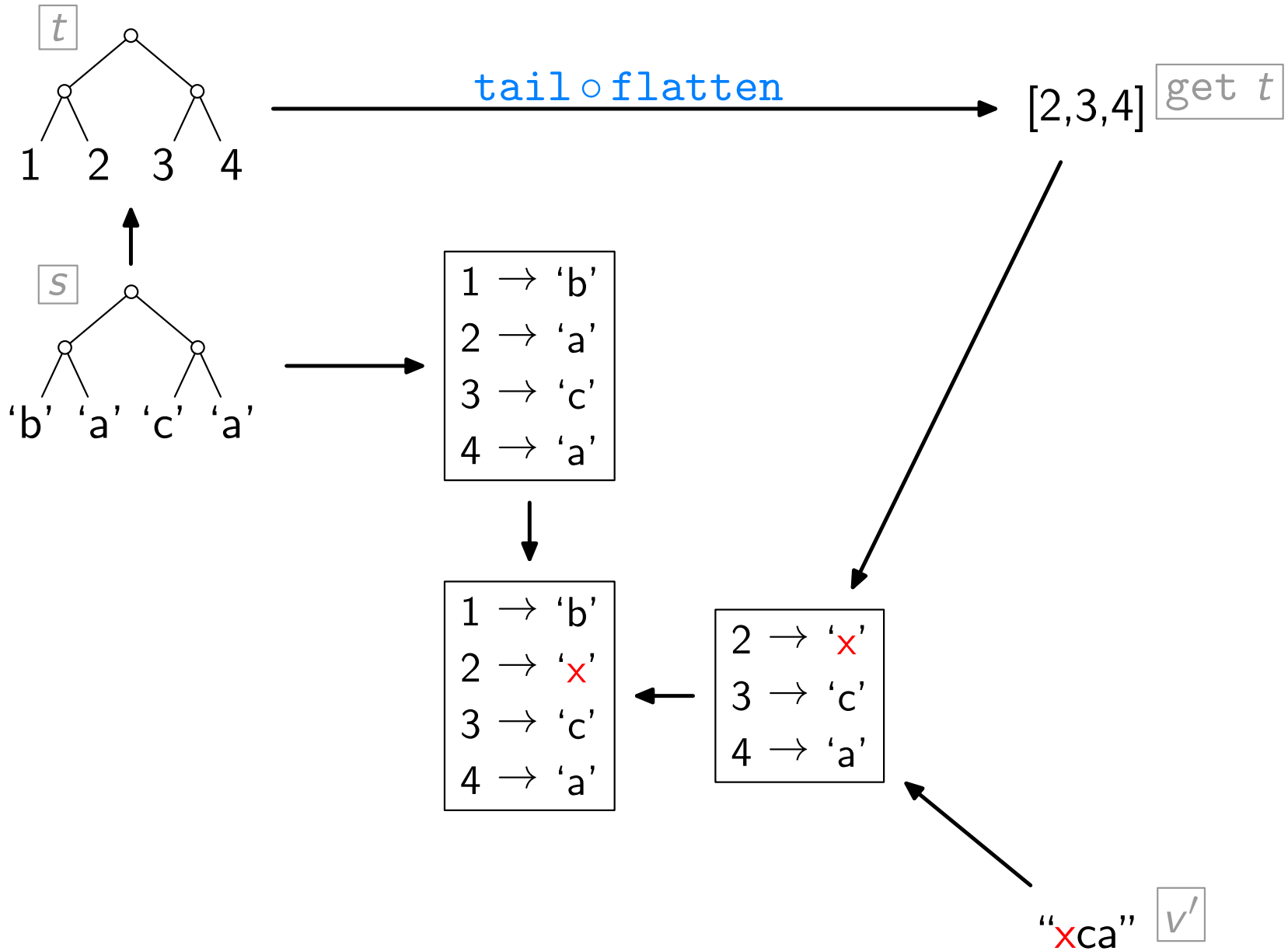
$1 \to$ 'b'
$2 \to$ 'a'
$3 \to$ 'c'
$4 \to$ 'a'

"xca" v'

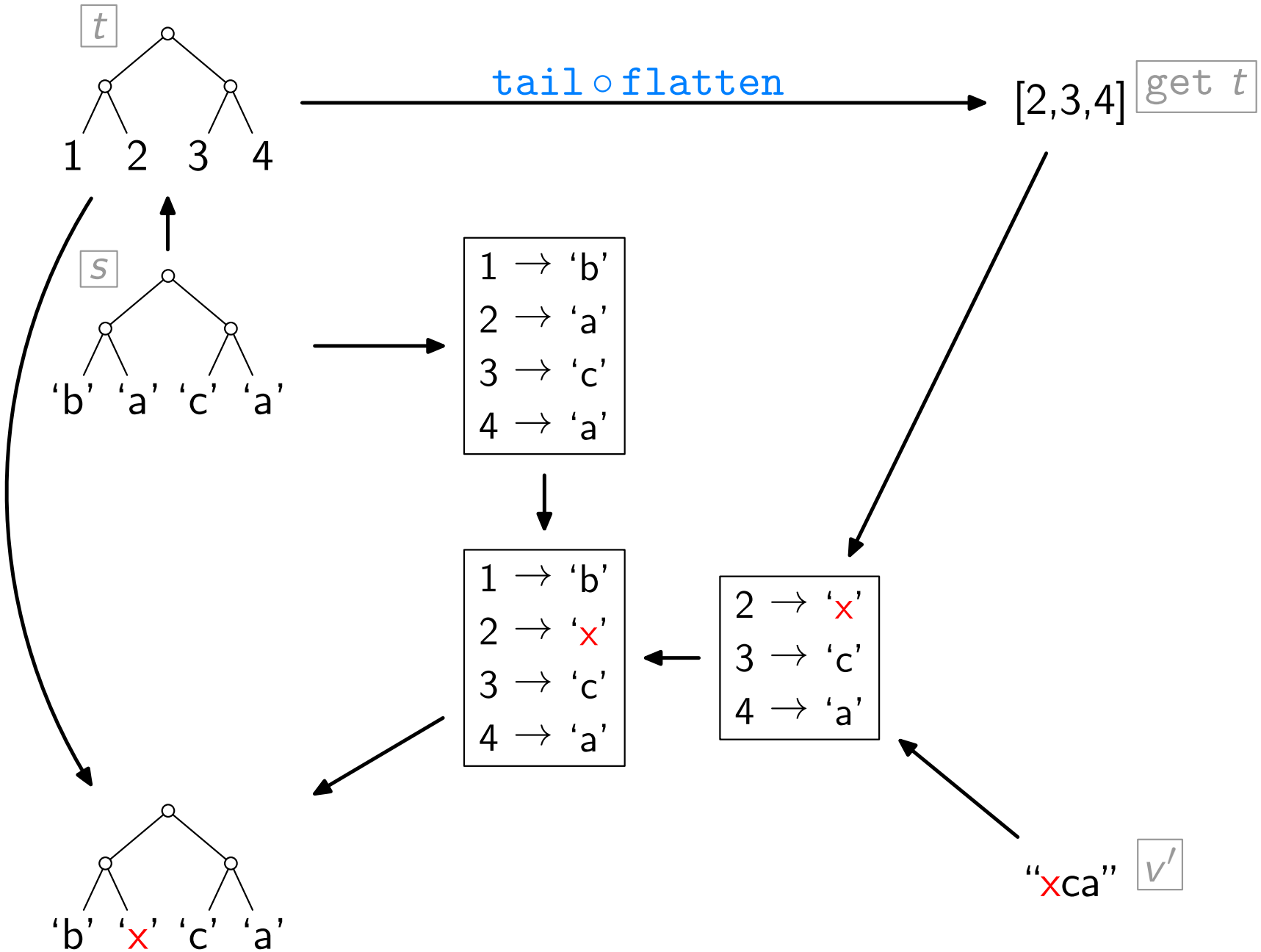
Der Semantische Ansatz am Beispiel



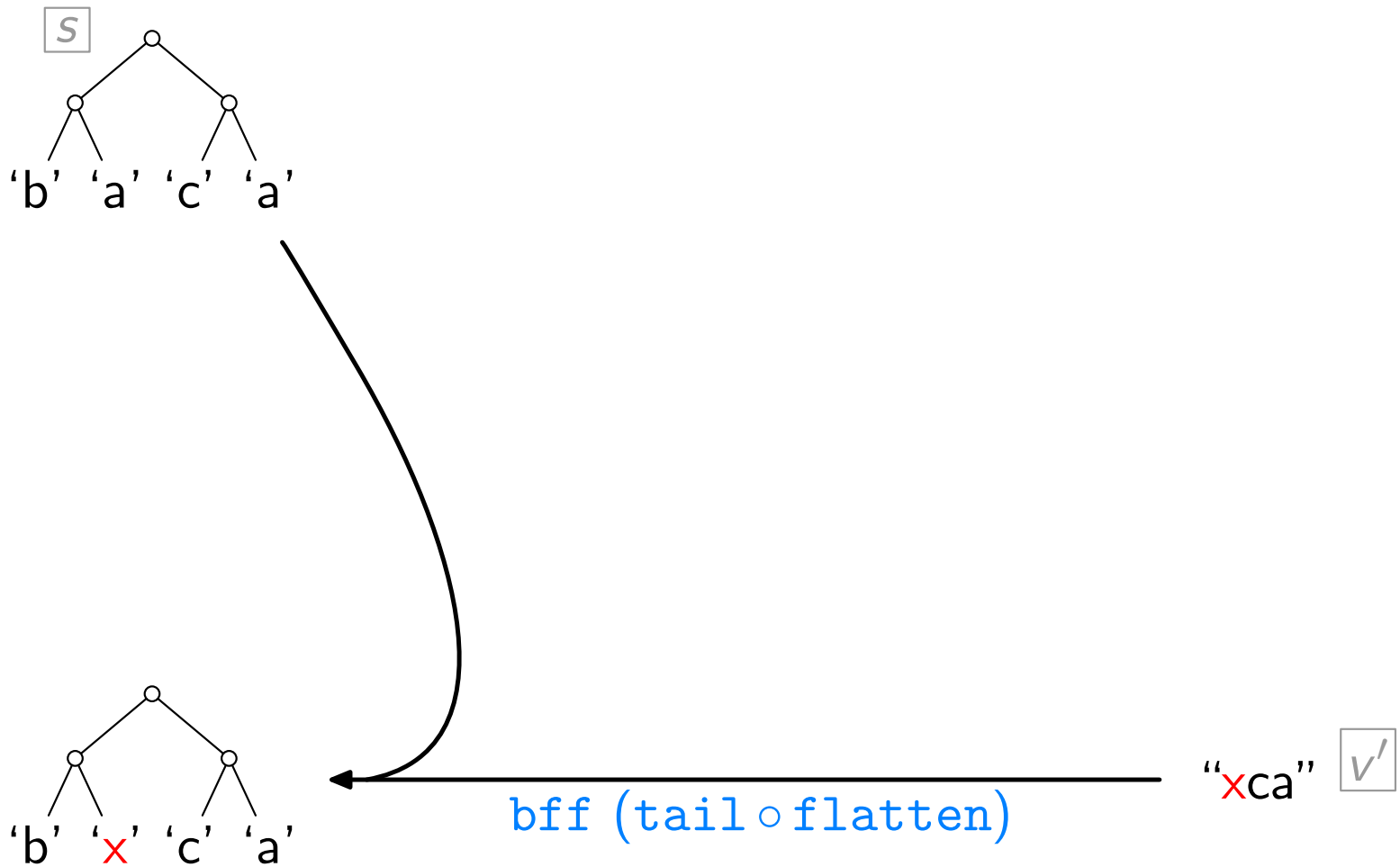
Der Semantische Ansatz am Beispiel



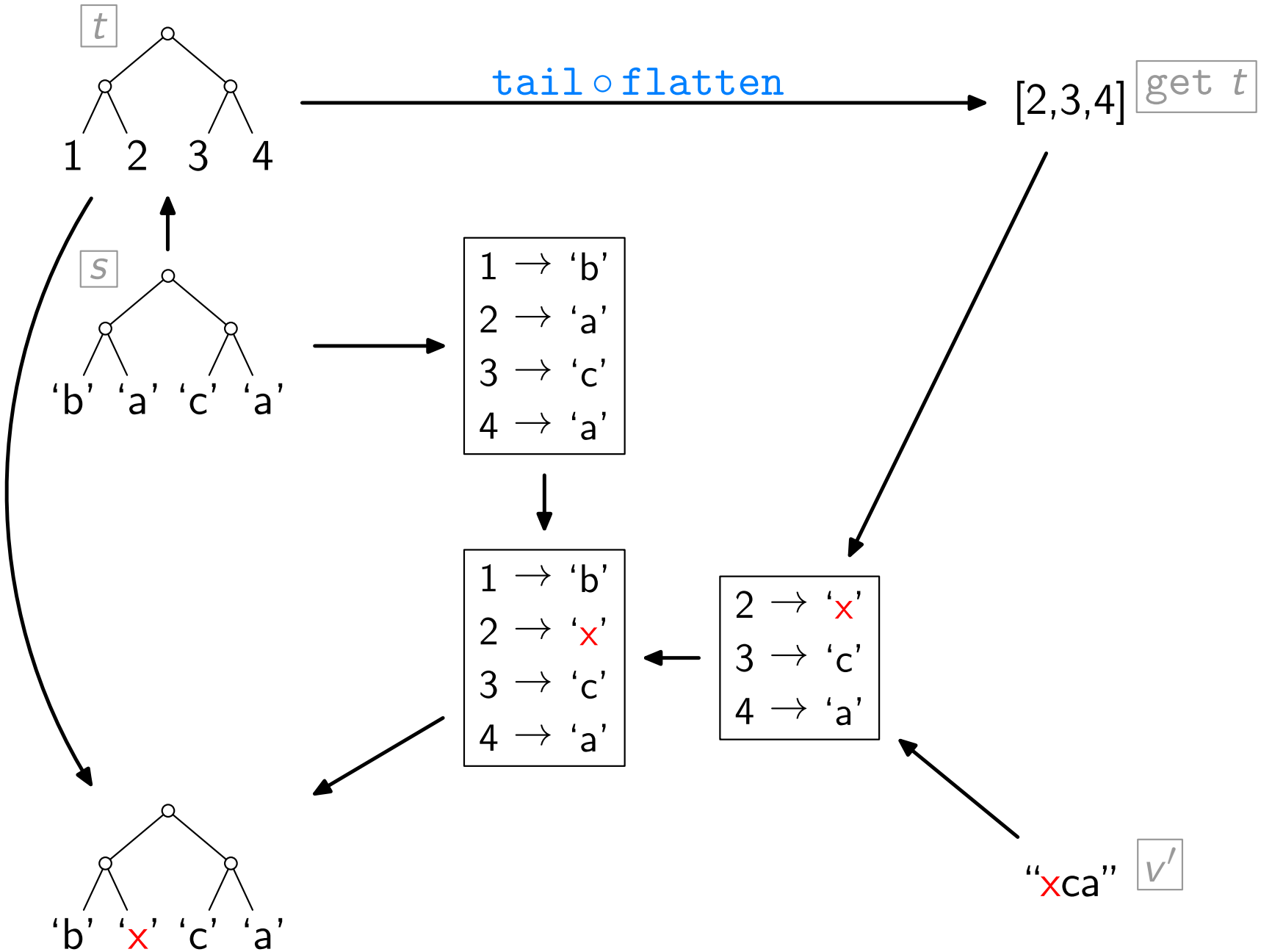
Der Semantische Ansatz am Beispiel



Der Semantische Ansatz am Beispiel



Der Semantische Ansatz am Beispiel



„Zwischenstand“

[V., POPL’09]:

- sehr einfacher, leichtgewichtiger Zugang zu Bidirektionalität
- Beweise mittels „freier Theoreme“ [Wadler, FPCA’89]
- Nachteil: generelle Zurückweisung formverändernder Updates

„Zwischenstand“

[V., POPL'09]:

- sehr einfacher, leichtgewichtiger Zugang zu Bidirektionalität
- Beweise mittels „freier Theoreme“ [Wadler, FPCA'89]
- Nachteil: generelle Zurückweisung formverändernder Updates

[Matsuda et al., ICFP'07]:

- stark abhängig von syntaktischen Bedingungen
- erlaubt (ad-hoc) eine Reihe formverändernder Updates

„Zwischenstand“

[V., POPL’09]:

- sehr einfacher, leichtgewichtiger Zugang zu Bidirektionalität
- Beweise mittels „freier Theoreme“ [Wadler, FPCA’89]
- Nachteil: generelle Zurückweisung formverändernder Updates

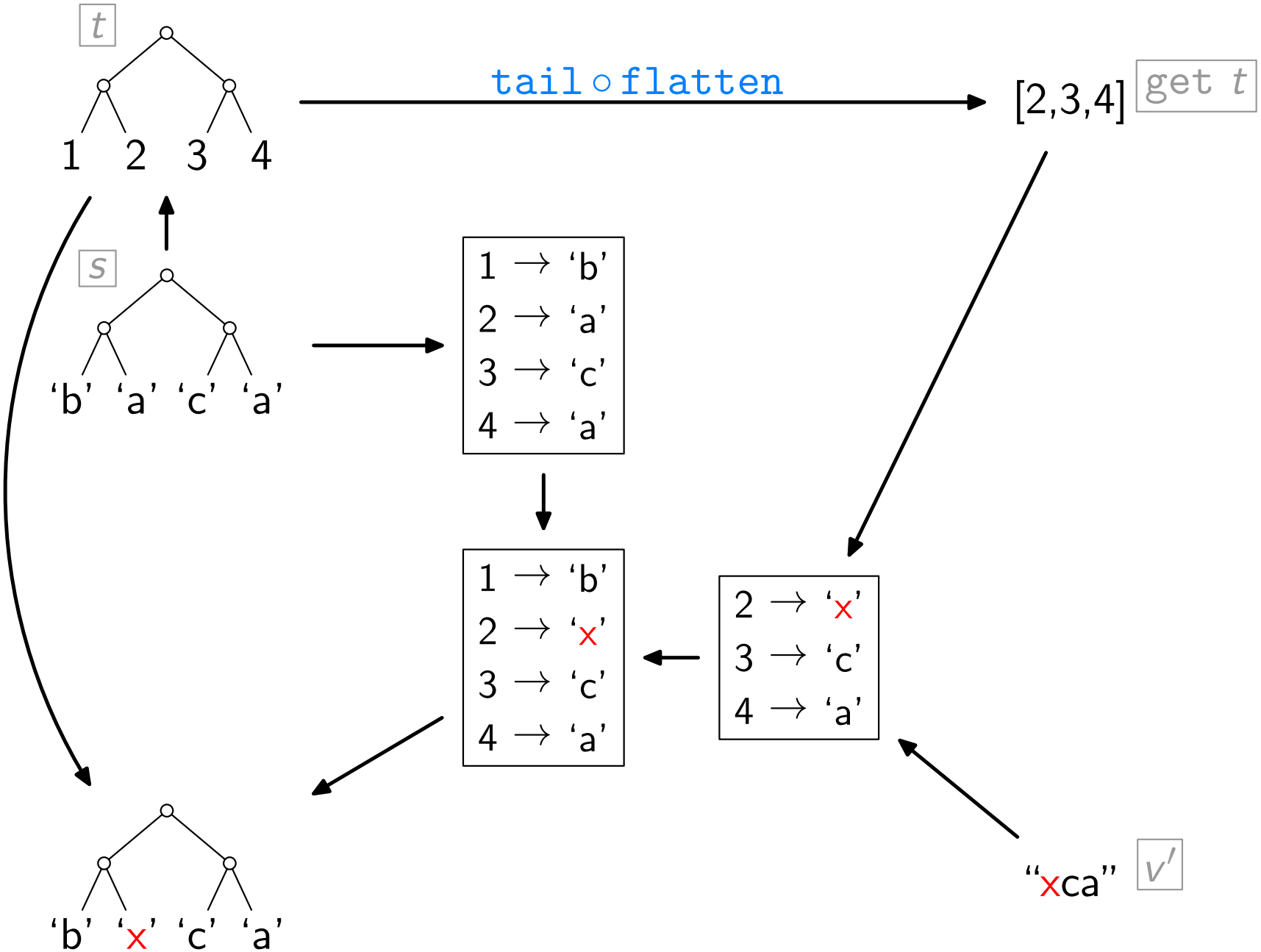
[Matsuda et al., ICFP’07]:

- stark abhängig von syntaktischen Bedingungen
- erlaubt (ad-hoc) eine Reihe formverändernder Updates

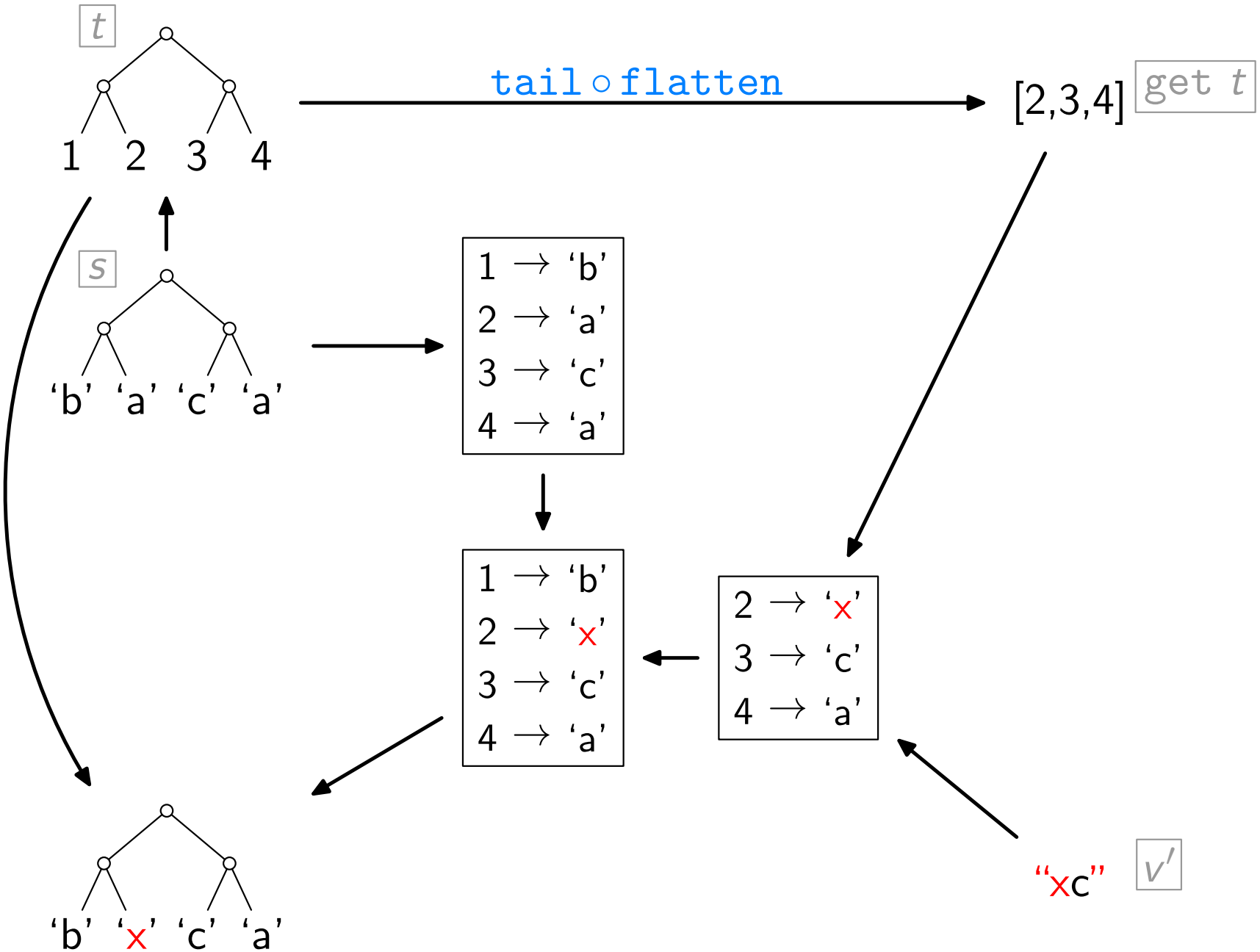
[V. et al., ICFP’10]:

- Synthese der beiden Techniken
- erbt sprachliche Einschränkungen beider
- erlaubt mehr Updates als sonst die jeweils bessere

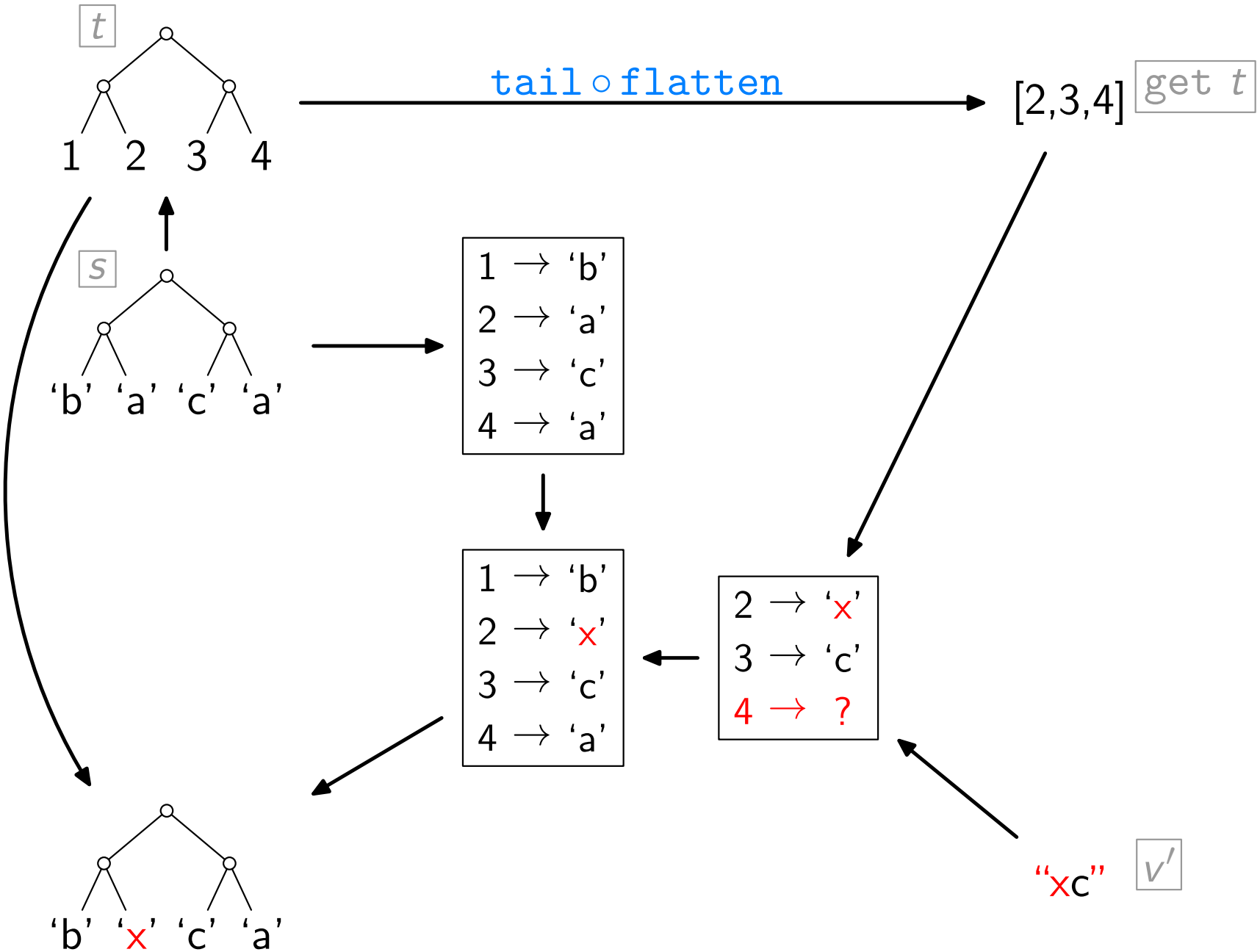
Mehr Form-Flexibilität



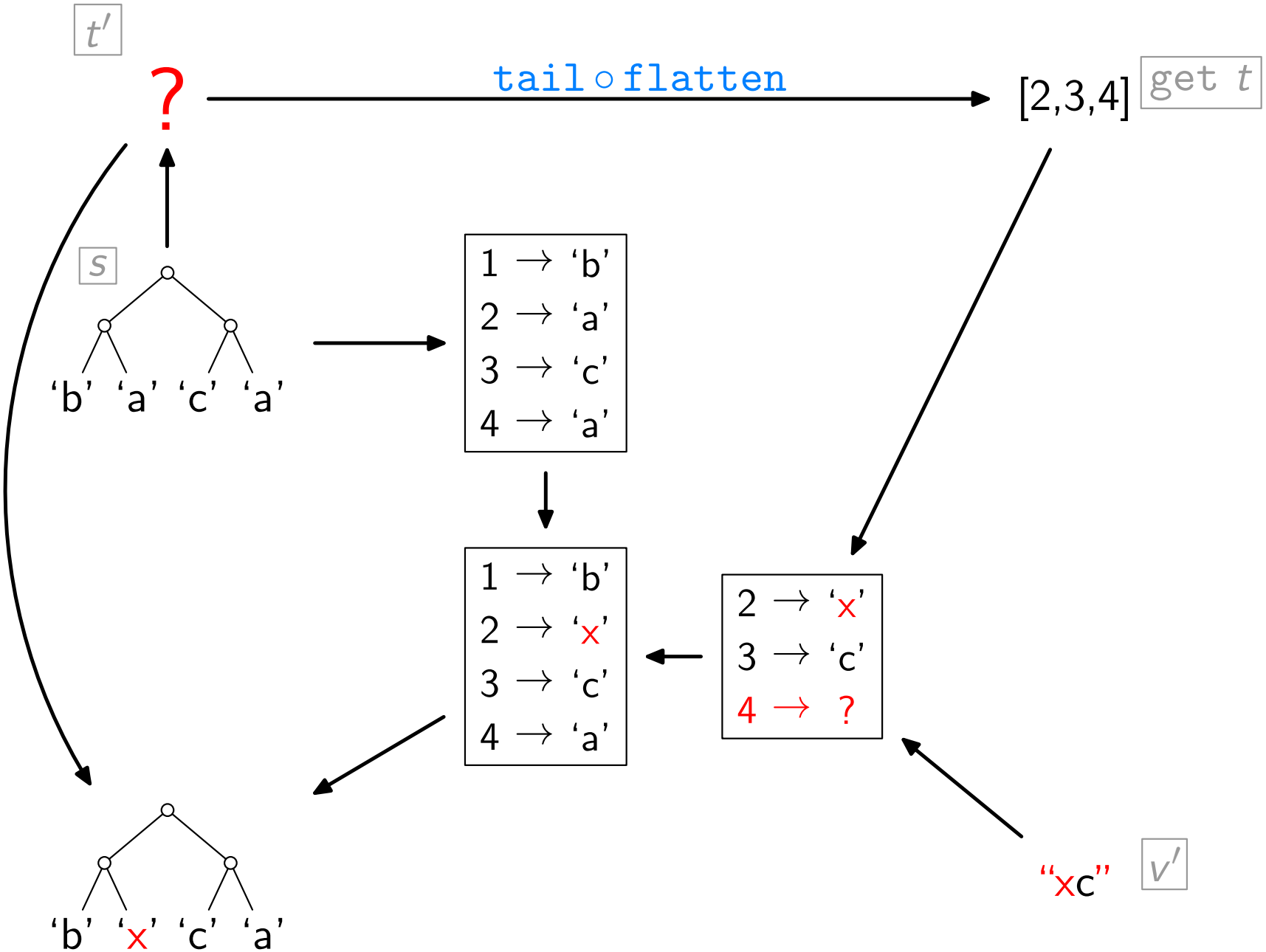
Mehr Form-Flexibilität



Mehr Form-Flexibilität



Mehr Form-Flexibilität



Forderungen an t'

Es liefere σ zu einer Datenstruktur jeweils eine Repräsentation der äußeren Form (Shape).

Dann streben wir an:

1. $\sigma(\texttt{get}\ t') = \sigma(v')$
2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$

Forderungen an t'

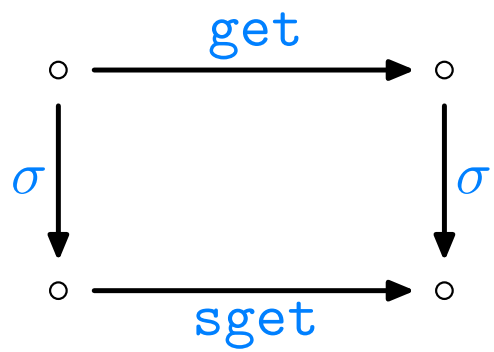
Es liefere σ zu einer Datenstruktur jeweils eine Repräsentation der äußeren Form (Shape).

Dann streben wir an:

1. $\sigma(\texttt{get}\ t') = \sigma(v')$
2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$

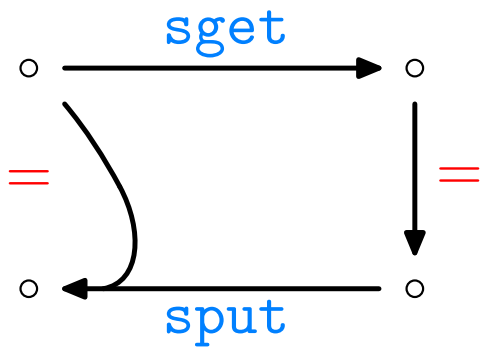
Entscheidende Idee: Abstraktion!

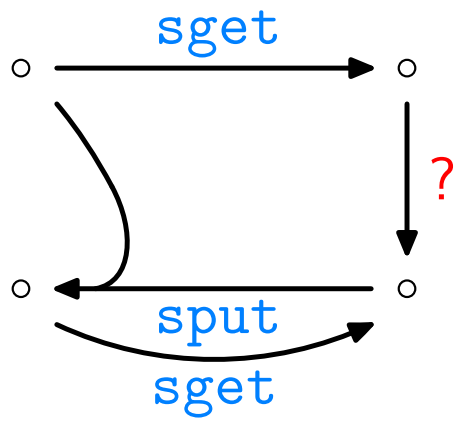
Finde `sget`, so dass:



„Bootstrapping“

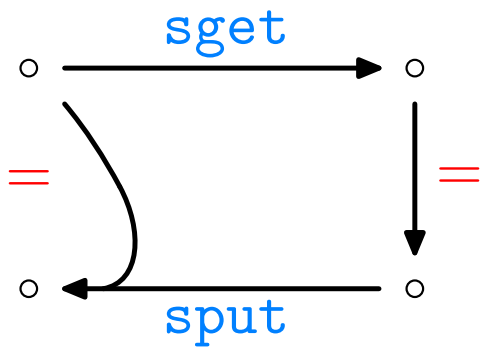
Zu `sget`, finde `sput`, so dass die GetPut- und PutGet-Gesetze erfüllt sind:

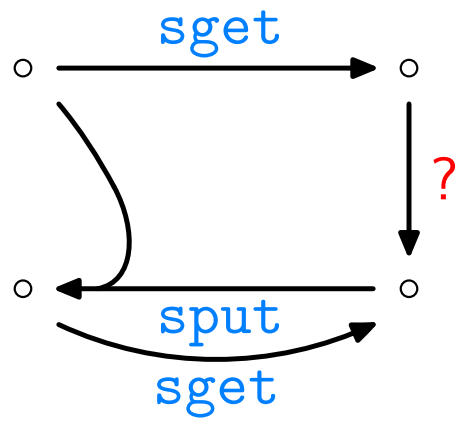




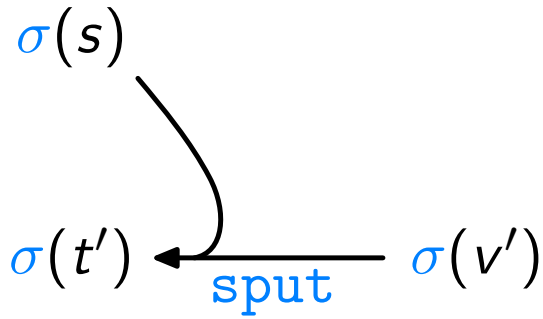
„Bootstrapping“

Zu `sget`, finde `sput`, so dass die GetPut- und PutGet-Gesetze erfüllt sind:





Dann, setze t' so dass:



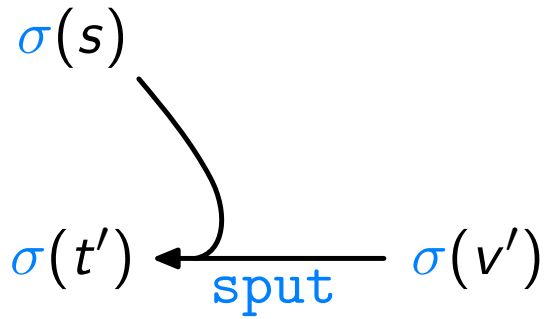
Forderungen an t'

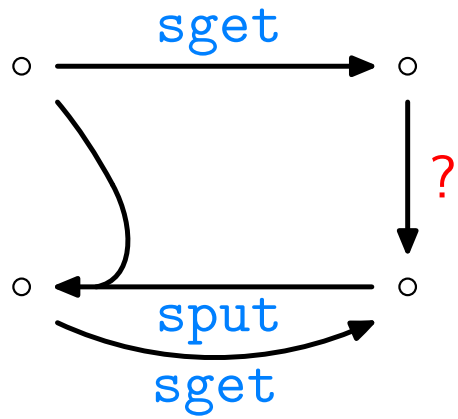
1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

Forderungen an t'

1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

Aus:

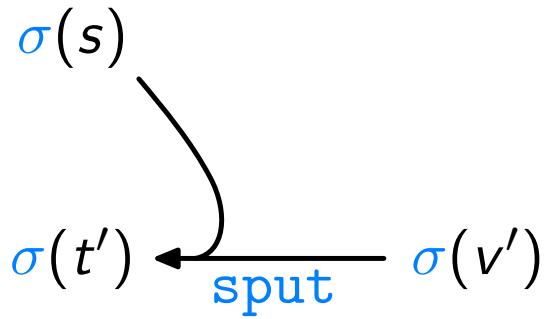


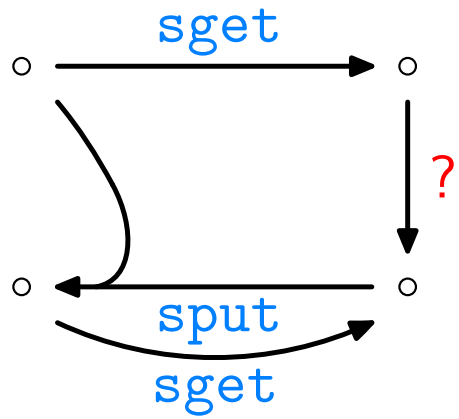


Forderungen an t'

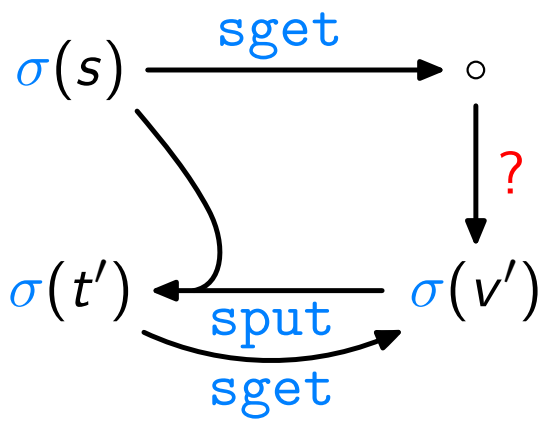
1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

Aus:





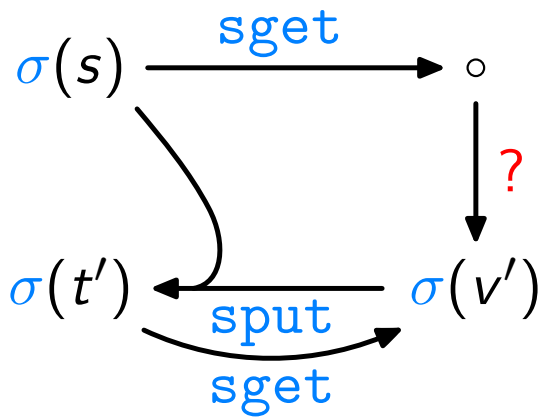
folgt:



Forderungen an t'

1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

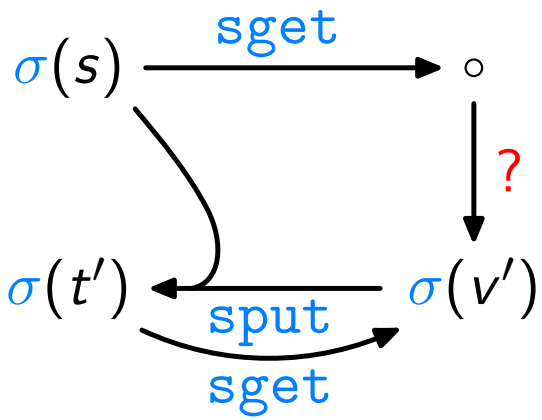
Aus:

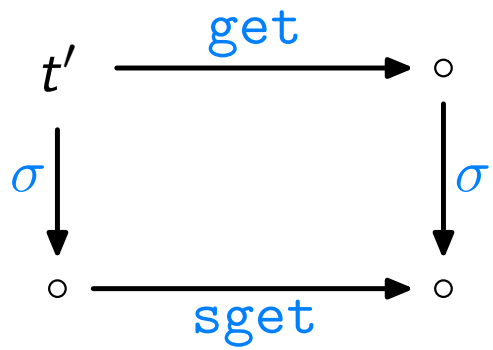


Forderungen an t'

1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

Aus:





folgt: $\sigma(\texttt{get}\ t') = \sigma(v')$.

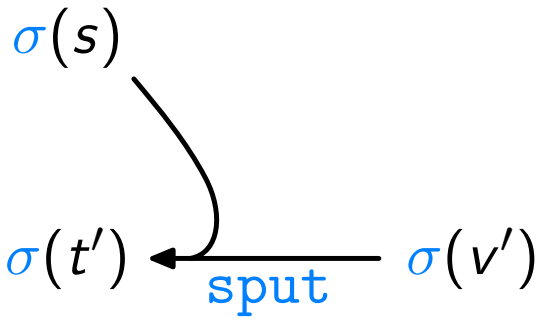
Forderungen an t'

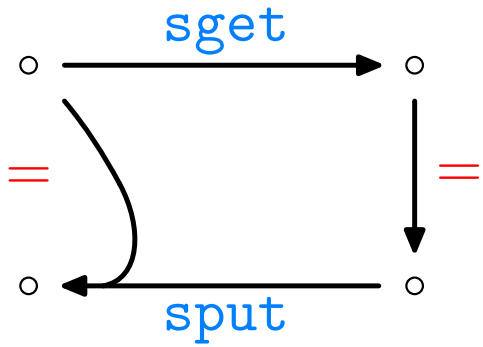
2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$?

Forderungen an t'

2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$?

Aus:

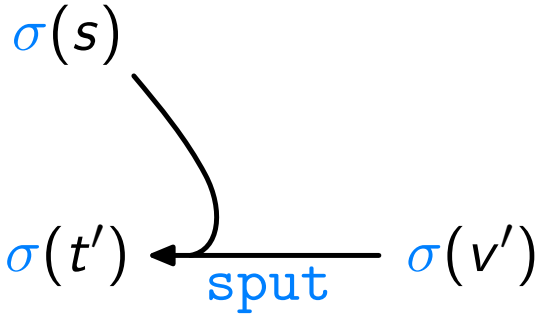


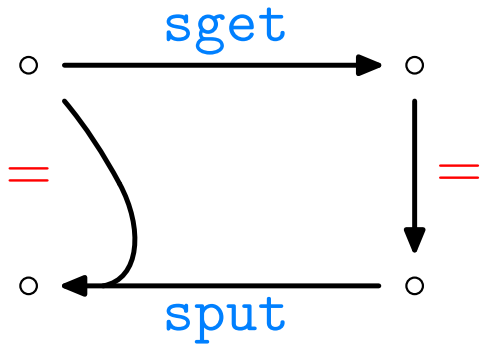


Forderungen an t'

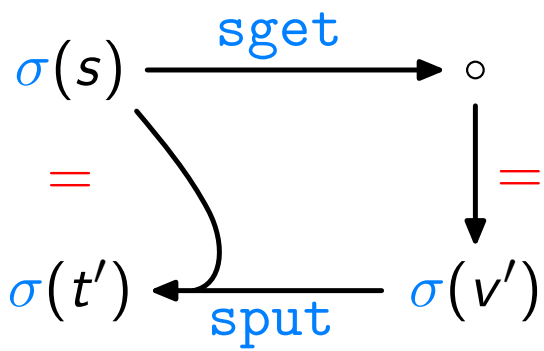
2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$?

Aus:





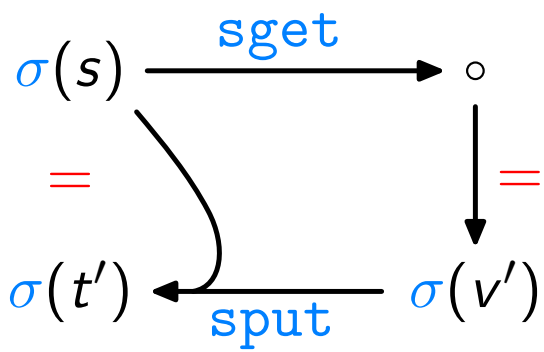
folgt:



Forderungen an t'

2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$?

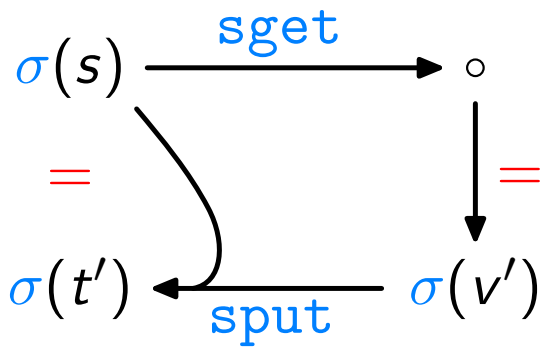
Aus:

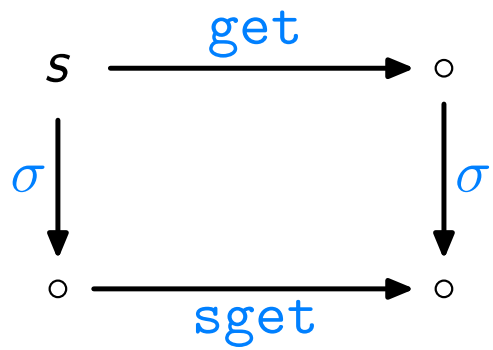


Forderungen an t'

2. wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$?

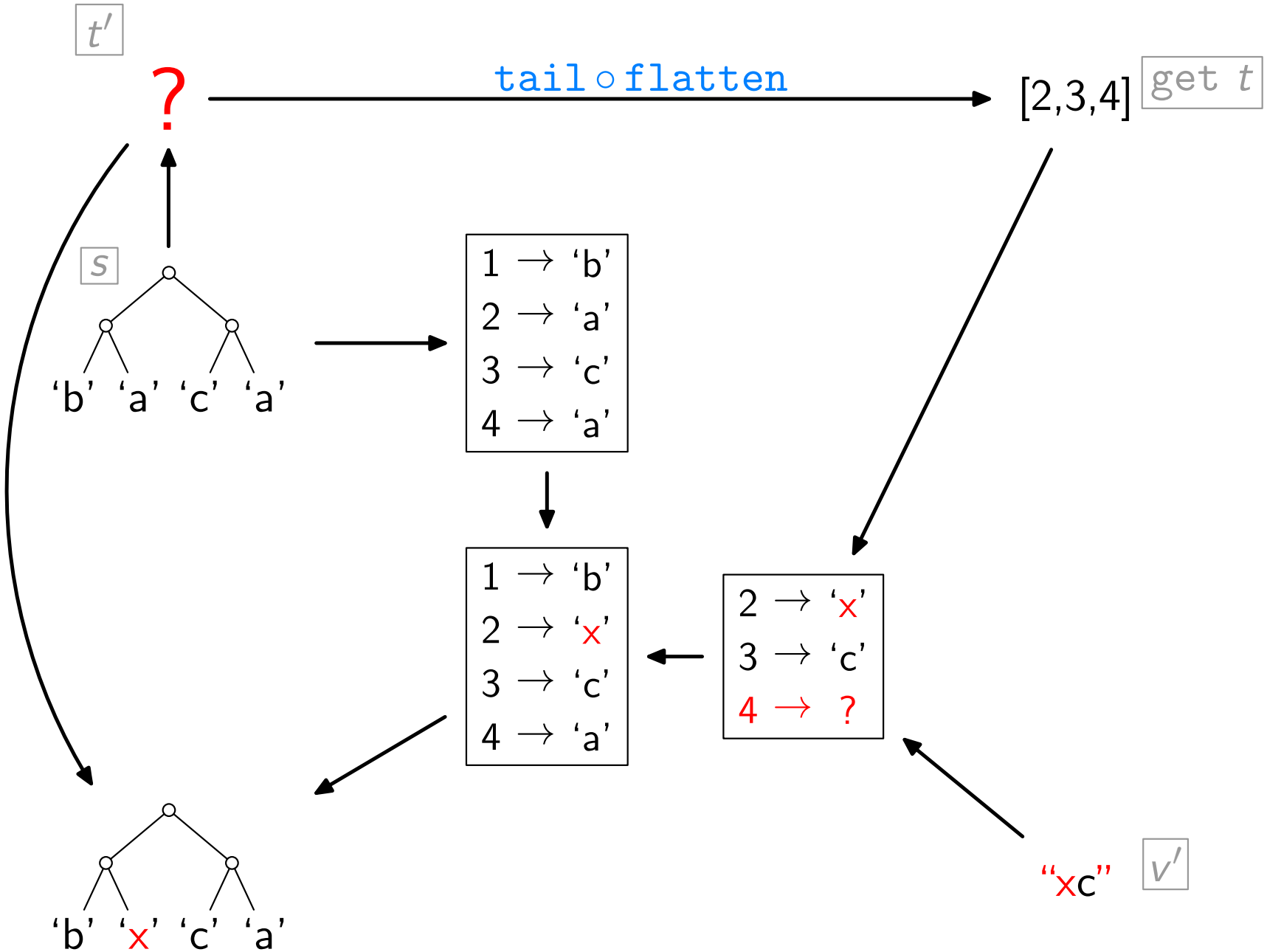
Aus:



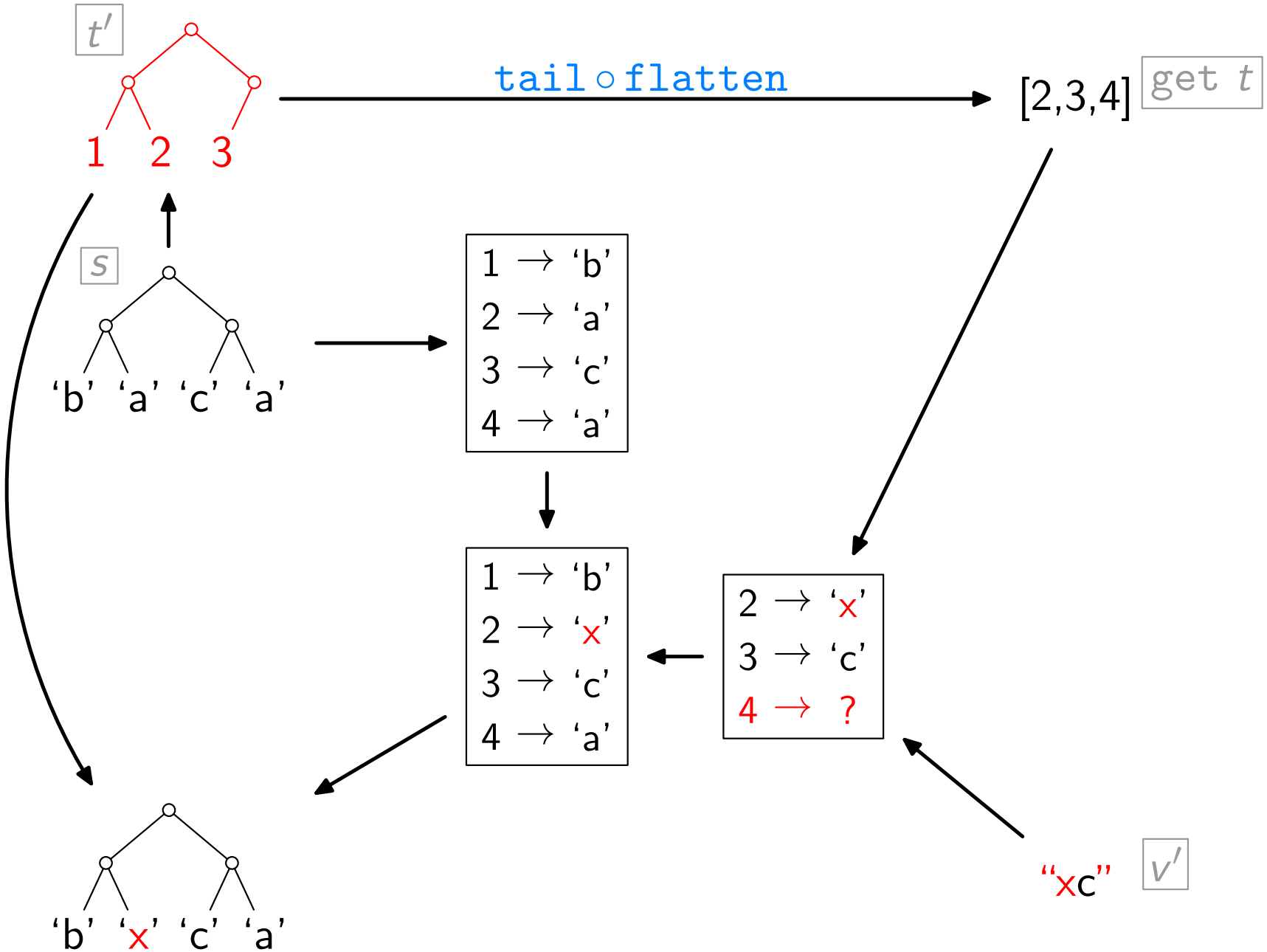


folgt, dass wenn $\sigma(v') = \sigma(\texttt{get}\ s)$, dann $\sigma(t') = \sigma(s)$.

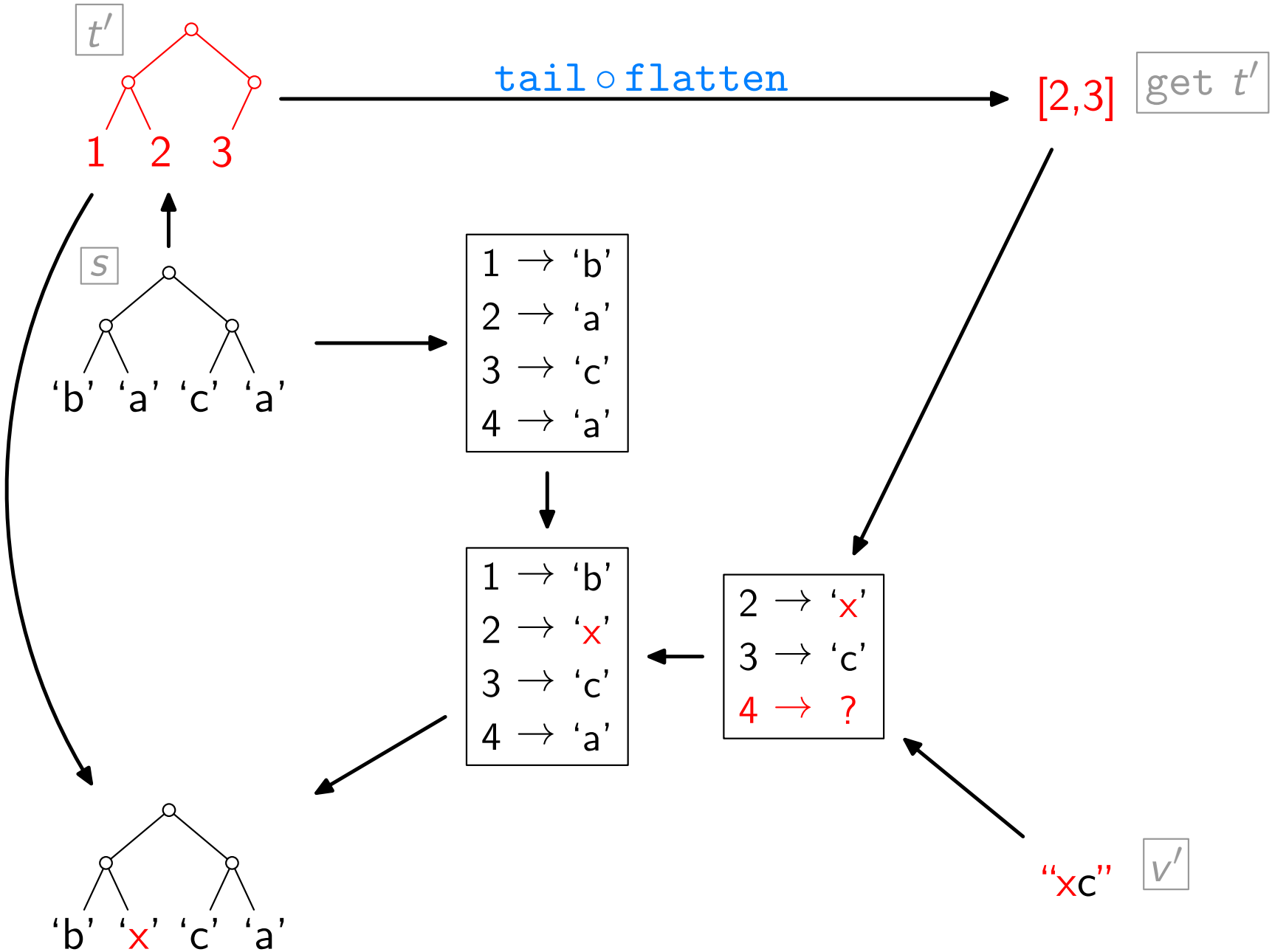
Mehr Form-Flexibilität



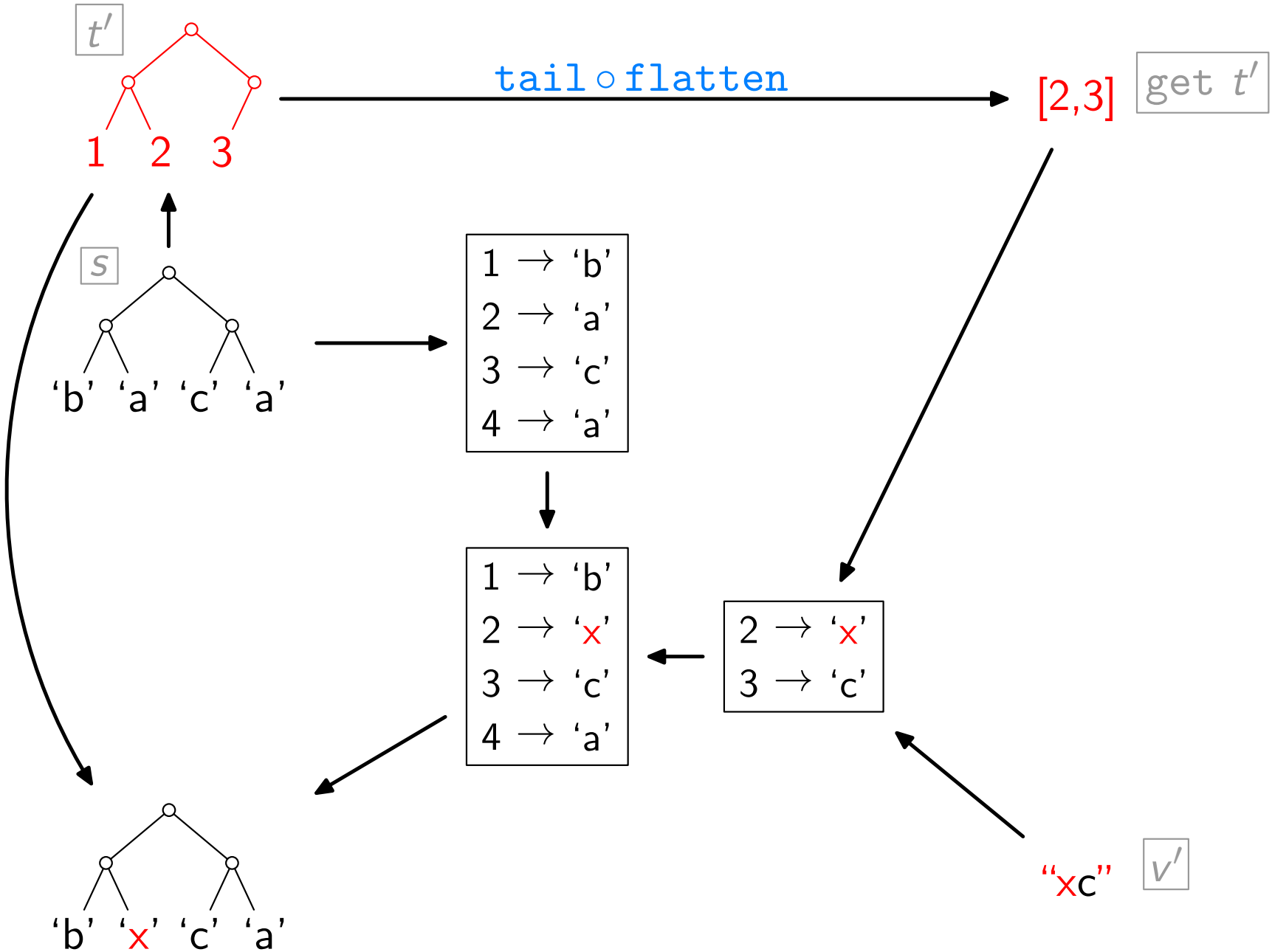
Mehr Form-Flexibilität



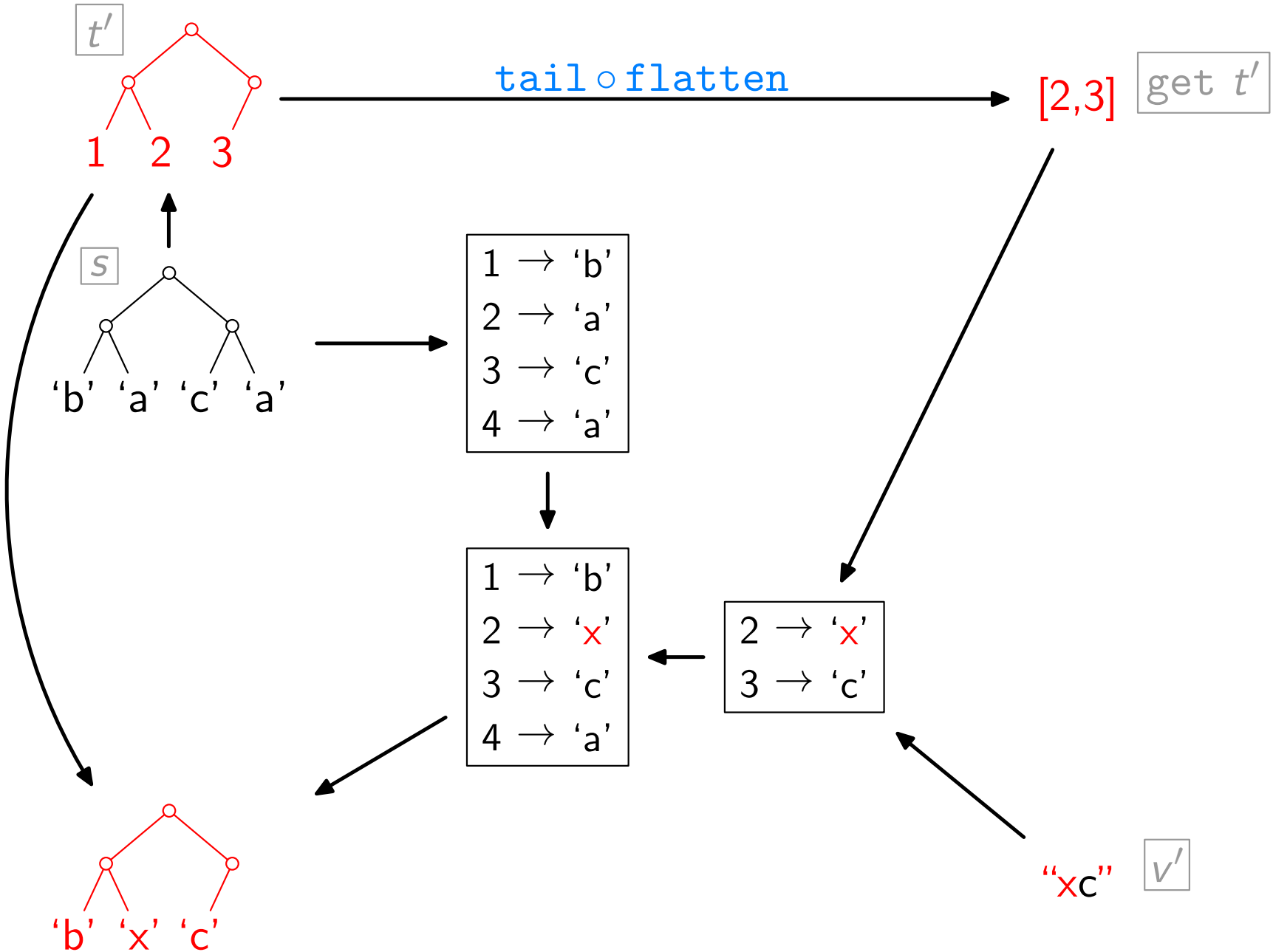
Mehr Form-Flexibilität



Mehr Form-Flexibilität

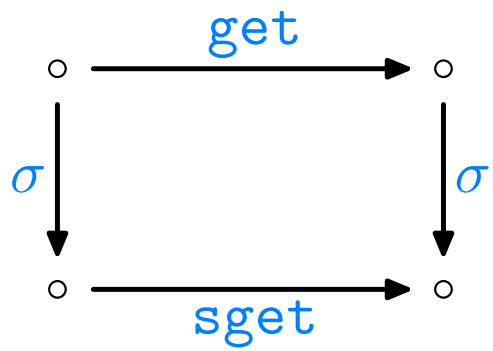


Mehr Form-Flexibilität



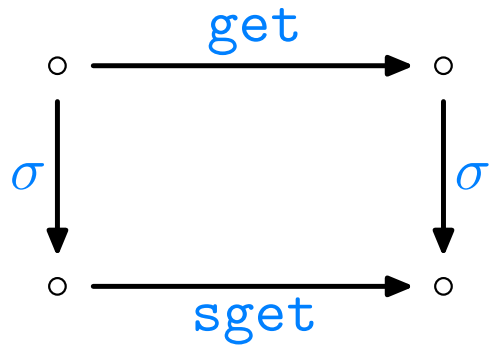
Entscheidende Zutaten

Entscheidend war, `sget` zu finden, so dass:

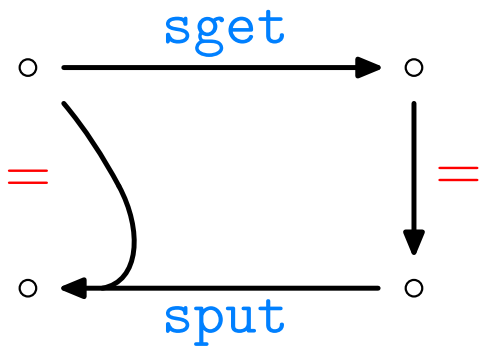


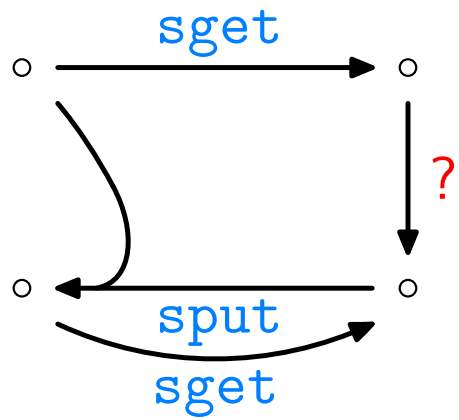
Entscheidende Zutaten

Entscheidend war, `sget` zu finden, so dass:



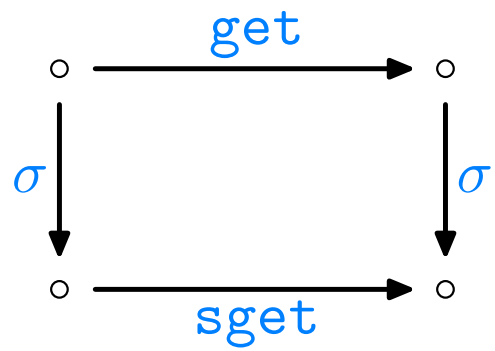
und dazu `sput`, so dass:





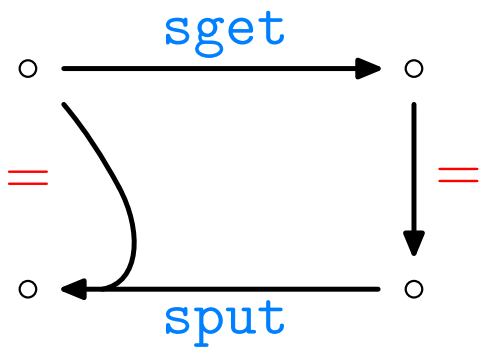
Entscheidende Zutaten

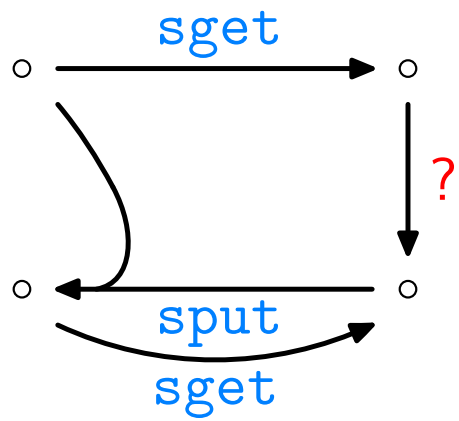
Entscheidend war, `sget` zu finden, so dass:



syntaktische Abstraktion

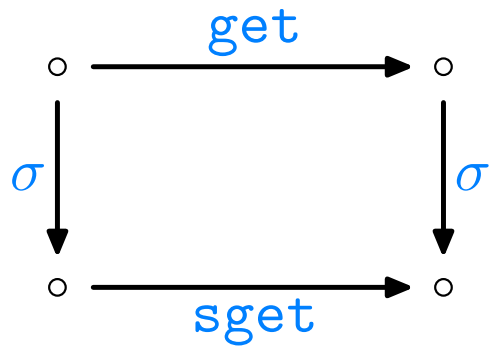
und dazu `sput`, so dass:





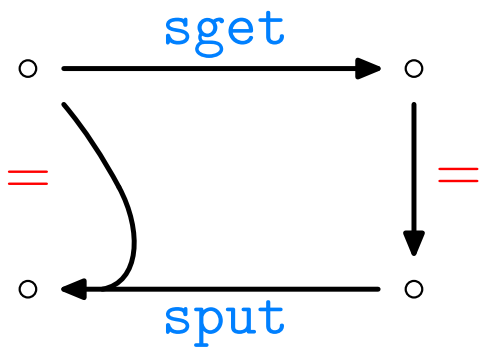
Entscheidende Zutaten

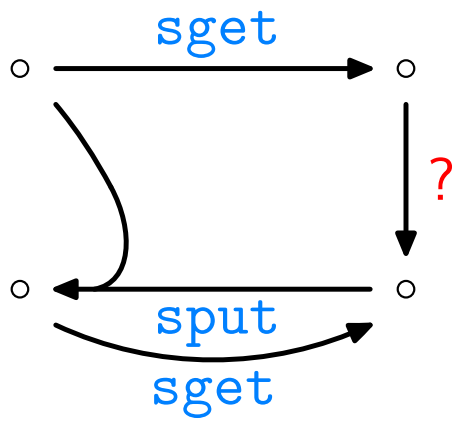
Entscheidend war, `sget` zu finden, so dass:



syntaktische Abstraktion

und dazu `sput`, so dass:





[Matsuda et al., ICFP'07]

Der Nutzen von Abstraktion

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] &&= [\,] \\
&\texttt{get}\ [x] &&= [\,] \\
&\texttt{get}\ (x : y : zs) &&= y : (\texttt{get}\ zs)
\end{aligned}
$$

Der Nutzen von Abstraktion

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] &&= [\,] \\
&\texttt{get}\ [x] &&= [\,] \\
&\texttt{get}\ (x : y : zs) &&= y : (\texttt{get}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{compl}\ [\,] &&= \mathsf{C}_1 \\
&\texttt{compl}\ [x] &&= \mathsf{C}_2\ x \\
&\texttt{compl}\ (x : y : zs) &&= \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}
$$

Der Nutzen von Abstraktion

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] &&= [\,] \\
&\texttt{get}\ [x] &&= [\,] \\
&\texttt{get}\ (x : y : zs) &&= y : (\texttt{get}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{compl}\ [\,] &&= \mathsf{C}_1 \\
&\texttt{compl}\ [x] &&= \mathsf{C}_2\ x \\
&\texttt{compl}\ (x : y : zs) &&= \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{put}\ [\,] &&[\,] &&= [\,] \\
&\texttt{put}\ [x] &&[\,] &&= [x] \\
&\texttt{put}\ (x : y : zs) &&(y' : v') &&= \cdots
\end{aligned}
$$

Der Nutzen von Abstraktion

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] &&= [\,] \\
&\texttt{get}\ [x] &&= [\,] \\
&\texttt{get}\ (x : y : zs) &&= y : (\texttt{get}\ zs)
\end{aligned}
\qquad\Longrightarrow\qquad
\begin{aligned}
&\texttt{sget} :: \mathsf{Int} \to \mathsf{Int} \\
&\texttt{sget}\ 0 &&= 0 \\
&\texttt{sget}\ 1 &&= 0 \\
&\texttt{sget}\ (n+2) &&= 1 + (\texttt{sget}\ n)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{compl}\ [\,] &&= \mathsf{C}_1 \\
&\texttt{compl}\ [x] &&= \mathsf{C}_2\ x \\
&\texttt{compl}\ (x : y : zs) &&= \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{put}\ [\,] &&[\,] &&= [\,] \\
&\texttt{put}\ [x] &&[\,] &&= [x] \\
&\texttt{put}\ (x : y : zs) &&(y' : v') &&= \cdots
\end{aligned}
$$

Der Nutzen von Abstraktion

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] &&= [\,] \\
&\texttt{get}\ [x] &&= [\,] \\
&\texttt{get}\ (x : y : zs) &&= y : (\texttt{get}\ zs)
\end{aligned}
\qquad\Longrightarrow\qquad
\begin{aligned}
&\texttt{sget} :: \mathsf{Int} \to \mathsf{Int} \\
&\texttt{sget}\ 0 &&= 0 \\
&\texttt{sget}\ 1 &&= 0 \\
&\texttt{sget}\ (n+2) &&= 1 + (\texttt{sget}\ n)
\end{aligned}
$$

$$\Downarrow \qquad\qquad\qquad \Downarrow$$

$$
\begin{aligned}
&\texttt{compl}\ [\,] &&= \mathsf{C}_1 \\
&\texttt{compl}\ [x] &&= \mathsf{C}_2\ x \\
&\texttt{compl}\ (x : y : zs) &&= \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}
\qquad\qquad
\begin{aligned}
&\texttt{compl}\ 0 &&= \mathsf{C}_1 \\
&\texttt{compl}\ 1 &&= \mathsf{C}_2 \\
&\texttt{compl}\ (n+2) &&= \texttt{compl}\ n
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{put}\ [\,] &\ [\,] &&= [\,] \\
&\texttt{put}\ [x] &\ [\,] &&= [x] \\
&\texttt{put}\ (x : y : zs) &\ (y' : v') &&= \cdots
\end{aligned}
$$

Der Nutzen von Abstraktion

$$
\begin{array}{l}
\texttt{get} :: [\alpha] \to [\alpha] \\
\texttt{get}\ [\,] \quad\quad\quad\ = [\,] \\
\texttt{get}\ [x] \quad\quad\quad\ = [\,] \\
\texttt{get}\ (x : y : zs) = y : (\texttt{get}\ zs)
\end{array}
\qquad \Longrightarrow \qquad
\begin{array}{l}
\texttt{sget} :: \mathsf{Int} \to \mathsf{Int} \\
\texttt{sget}\ 0 \quad\quad\ = 0 \\
\texttt{sget}\ 1 \quad\quad\ = 0 \\
\texttt{sget}\ (n+2) = 1 + (\texttt{sget}\ n)
\end{array}
$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$
\begin{array}{l}
\texttt{compl}\ [\,] \quad\quad\quad\ = \mathsf{C}_1 \\
\texttt{compl}\ [x] \quad\quad\quad\ = \mathsf{C}_2\ x \\
\texttt{compl}\ (x : y : zs) = \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{array}
\qquad\qquad
\begin{array}{l}
\texttt{compl}\ 0 \quad\quad\ = \mathsf{C}_1 \\
\texttt{compl}\ 1 \quad\quad\ = \mathsf{C}_2 \\
\texttt{compl}\ (n+2) = \texttt{compl}\ n
\end{array}
$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$
\begin{array}{lll}
\texttt{put}\ [\,] & [\,] & = [\,] \\
\texttt{put}\ [x] & [\,] & = [x] \\
\texttt{put}\ (x : y : zs) & (y' : v') & = \cdots
\end{array}
\qquad\qquad
\begin{array}{lll}
\texttt{sput}\ 0 & 0 & = 0 \\
\texttt{sput}\ 1 & 0 & = 1 \\
\texttt{sput}\ (n+2) & 0 & = \cdots \\
\texttt{sput}\ n & (v'+1) & = \cdots
\end{array}
$$

Fazit

- Bidirektionale Transformation:
  - „hot topic“
  - verschiedene Ansätze, hier programmiersprachlich

Fazit

- Bidirektionale Transformation:
  - „hot topic“
  - verschiedene Ansätze, hier programmiersprachlich
- Semantischer Ansatz:
  - leichtgewichtig, „as a library“
  - entscheidende Rolle: polymorphe Funktionstypen

Fazit

- Bidirektionale Transformation:
  - „hot topic“
  - verschiedene Ansätze, hier programmiersprachlich
- Semantischer Ansatz:
  - leichtgewichtig, „as a library“
  - entscheidende Rolle: polymorphe Funktionstypen
- Syntaktischer Ansatz:
  - klassische Programmtransformation
  - „constant-complement“ [Banc. & Sp., ACM TODS'81]

Fazit

- Bidirektionale Transformation:
  - „hot topic“
  - verschiedene Ansätze, hier programmiersprachlich
- Semantischer Ansatz:
  - leichtgewichtig, „as a library“
  - entscheidende Rolle: polymorphe Funktionstypen
- Syntaktischer Ansatz:
  - klassische Programmtransformation
  - „constant-complement“ [Banc. & Sp., ACM TODS'81]
- Kombination per „Separation of Concerns“:
  - Trennung der Datenstruktur in Form und Inhalt
  - Behandlung der Form durch syntaktischen Ansatz
  - Behandlung des Inhalts durch semantischen Ansatz
  - hier nicht gezeigt: Parametrisierung über „bias“

Literatur I


F. Bancilhon and N. Spyratos.
Update semantics of relational views.
ACM Transactions on Database Systems, 6(3):557–575, 1981.


J.N. Foster, M.B. Greenwald, J.T. Moore, B.C. Pierce, and A. Schmitt.
Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem.
ACM Transactions on Programming Languages and Systems, 29(3):17, 2007.


K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi.
Bidirectionalization transformation based on automatic derivation of view complement functions.
In *International Conference on Functional Programming, Proceedings*, pages 47–58. ACM Press, 2007.

Literatur II

J. Voigtländer, Z. Hu, K. Matsuda, and M. Wang.
Combining syntactic and semantic bidirectionalization.
In *International Conference on Functional Programming, Proceedings*. ACM Press, 2010.

J. Voigtländer.
Bidirectionalization for free!
In *Principles of Programming Languages, Proceedings*, pages 165–176. ACM Press, 2009.

P. Wadler.
Theorems for free!
In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.