Repräsentation von Datenstrukturen im π-Kalkül

Simon Pinkel,

Betreuer: Guido Tack

Seminar: Theorie kommunizierender Systeme: Der π-Kalkül

Gliederung

1. Motivation
2. Darstellung von Daten im π-Kalkül
3. Vergleich mit Alice
4. persistente Daten
5. Zusammenfassung

Motivation

$$Buff^{(n)} := \sum_u in_u.Buff_u^{(n)}$$

$$Buff_{\vec{v},w}^{(n)} := \begin{cases} \sum_u in_u.Buff_{u,\vec{v},w}^{(n)} + \overline{out_{\vec{w}}}.Buff_{\vec{v}}^{(n)} & (|\vec{v}|<n-1) \\ \overline{out_w}.Buff_{\vec{v}}^{(n)} & (|\vec{v}|=n-1) \end{cases}$$

- CCS-Buffer beliebiger Größe

Motivation

$$Buff^{(n)} := \sum_u in_u.Buff_u^{(n)}$$

$$Buff_{\vec{v},w}^{(n)} := \begin{cases} \sum_u in_u.Buff_{u,\vec{v},w}^{(n)} + \overline{out_{\vec{w}}}.Buff_{\vec{v}}^{(n)} & (|\vec{v}|<n-1) \\ \overline{out_w}.Buff_{\vec{v}}^{(n)} & (|\vec{v}|=n-1) \end{cases}$$

- CCS-Buffer beliebiger Größe
- 2 Kompromisse:
  - Erweiterung des Kalküls (Parameter)
  - Wertemenge V endlich

Motivation

$$Buff^{(n)} := \sum_u in_u.Buff_u^{(n)}$$

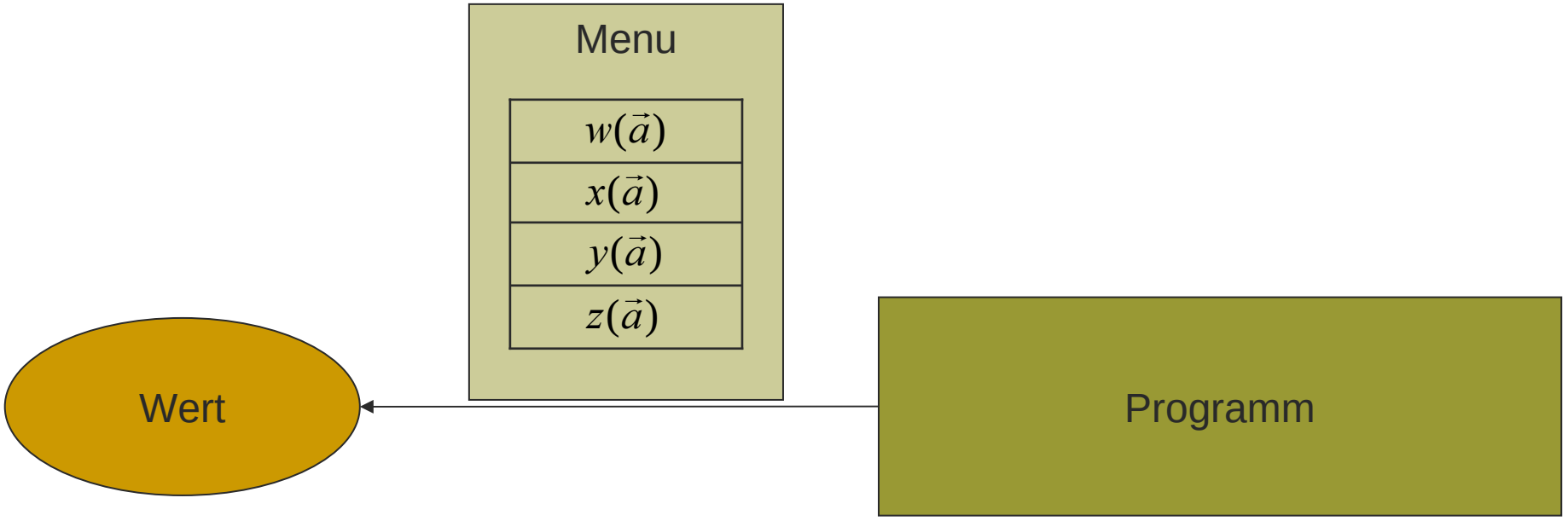
$$Buff_{\vec{v},w}^{(n)} := \begin{cases} \sum_u in_u.Buff_{u,\vec{v},w}^{(n)} + \overline{out_{\vec{w}}}.Buff_{\vec{v}}^{(n)} & (|\vec{v}|<n-1) \\ \overline{out_w}.Buff_{\vec{v}}^{(n)} & (|\vec{v}|=n-1) \end{cases}$$

- CCS-Buffer beliebiger Größe
- 2 Kompromisse:
  - Erweiterung des Kalküls (Parameter)
  - Wertemenge V endlich

→ Darstellung im π-Kalkül kompromisslos möglich!

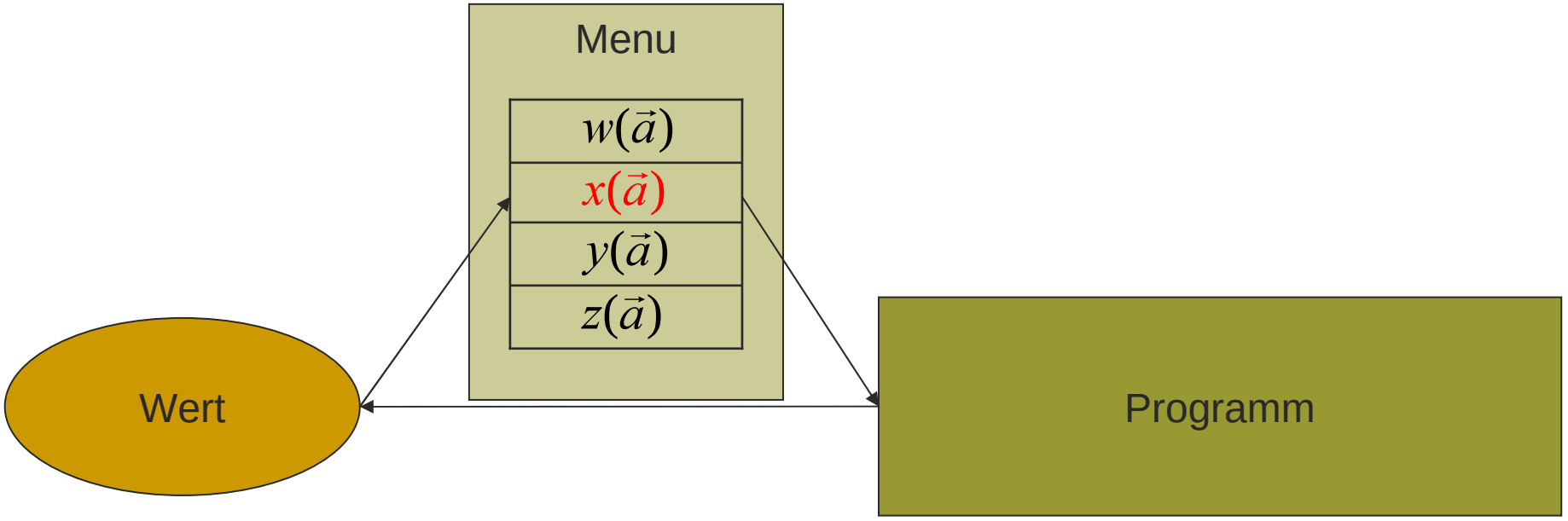
Darstellung

- Idee: Werte selektieren aus einem Menu!



Darstellung

- Idee: Werte selektieren aus einem Menu!

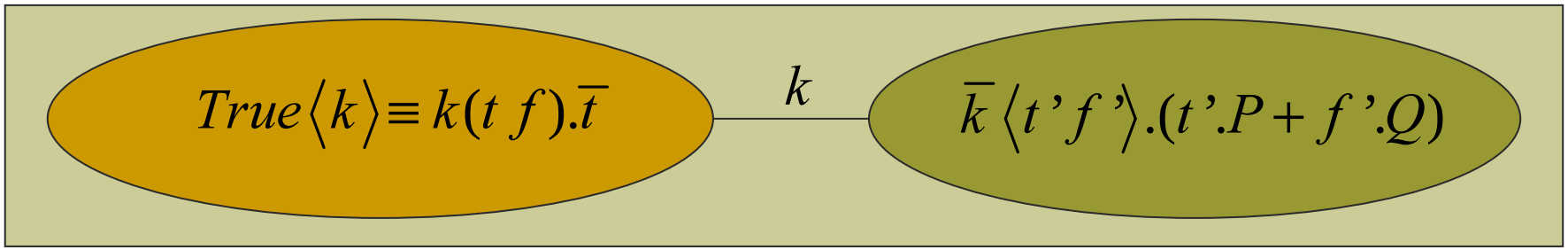


Wahrheitswerte

Beispiel: Wahrheitswerte

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

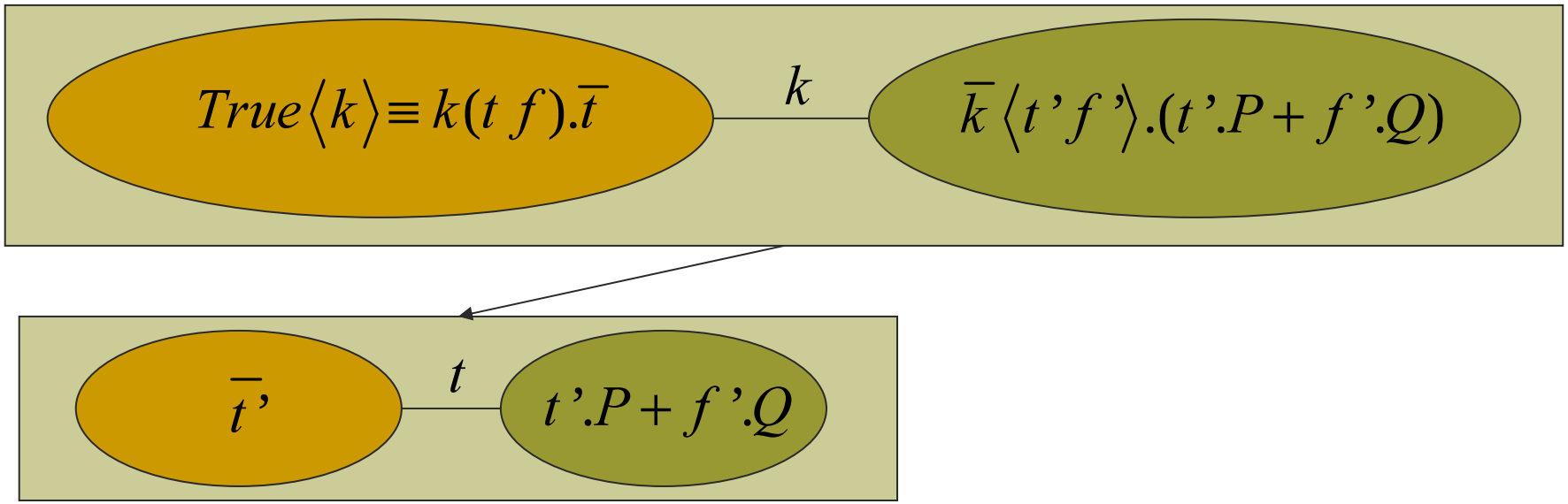


Wahrheitswerte

Beispiel: Wahrheitswerte

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

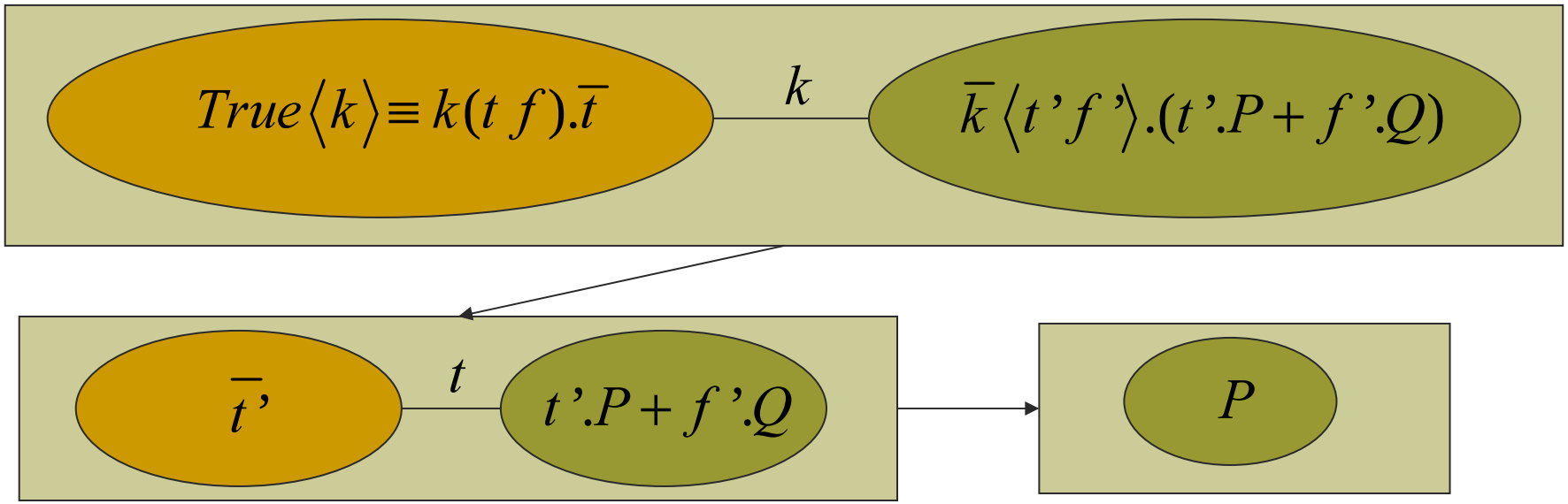


Wahrheitswerte

Beispiel: Wahrheitswerte

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$



flüchtige Daten

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

flüchtige Daten

- Daten = unäre Abstraktionen

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

flüchtige Daten

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

- Daten = unäre Abstraktionen
- Argument = *Lokation* k

flüchtige Daten

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

- Daten = unäre Abstraktionen
- Argument = *Lokation* k
- Rechnung = Selektion aus gesendetem *Menu*

flüchtige Daten

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

- Daten = unäre Abstraktionen
- Argument = *Lokation* k
- Rechnung = Selektion aus gesendetem *Menu*
- Rechenoperationen = abgeleitete Operatoren:

$$Cond(P,Q)(k) := (\text{new}\ t\ f)\overline{k}\left\langle t\ f\right\rangle.(t.P + f.Q)$$

→ Berechnungen = Reaktionen(τ)

flüchtige Daten

$$True(k) := k(t\ f).\overline{t}$$

$$False(k) := k(t\ f).\overline{f}$$

- Daten = unäre Abstraktionen
- Argument = *Lokation* k
- Rechnung = Selektion aus gesendetem *Menu*
- Rechenoperationen = abgeleitete Operatoren:

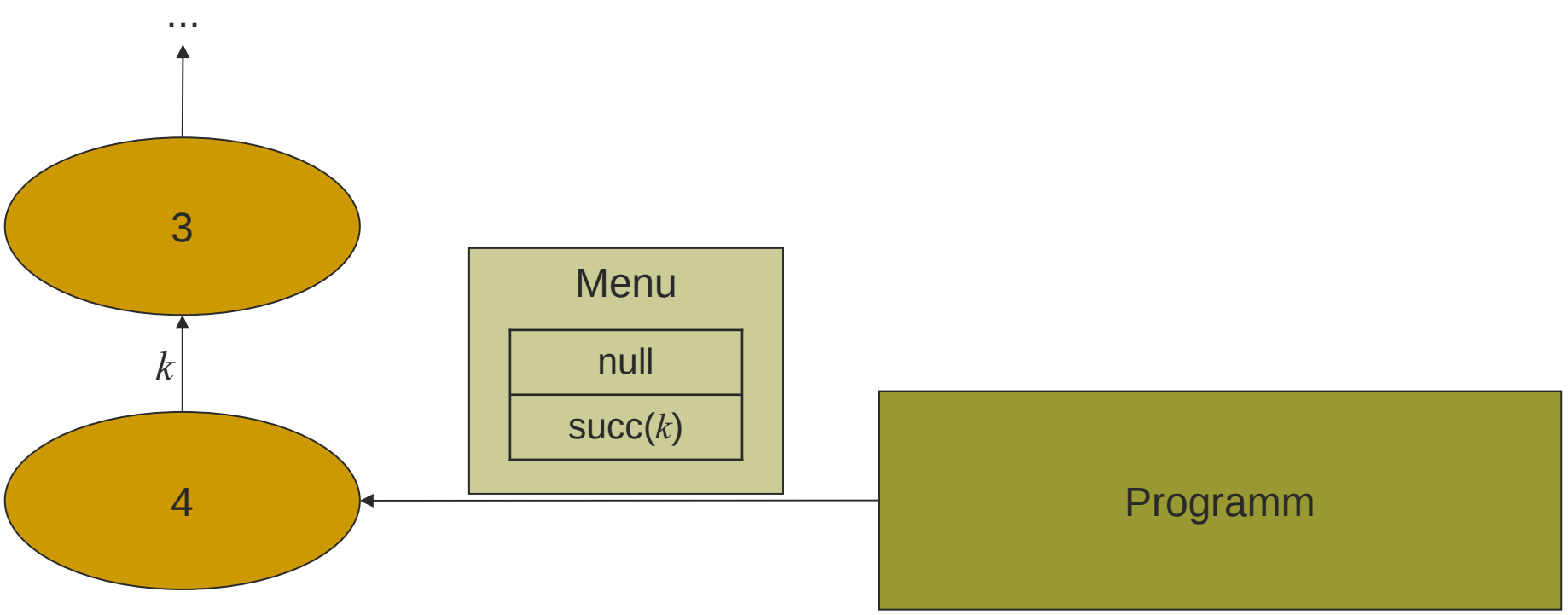
$$Cond(P,Q)(k) := (\text{new}\ t\ f)\overline{k}\left\langle t\ f\right\rangle.(t.P + f.Q)$$

→ Berechnungen = Reaktionen(τ)

- Rechnen zerstört die Datenstruktur

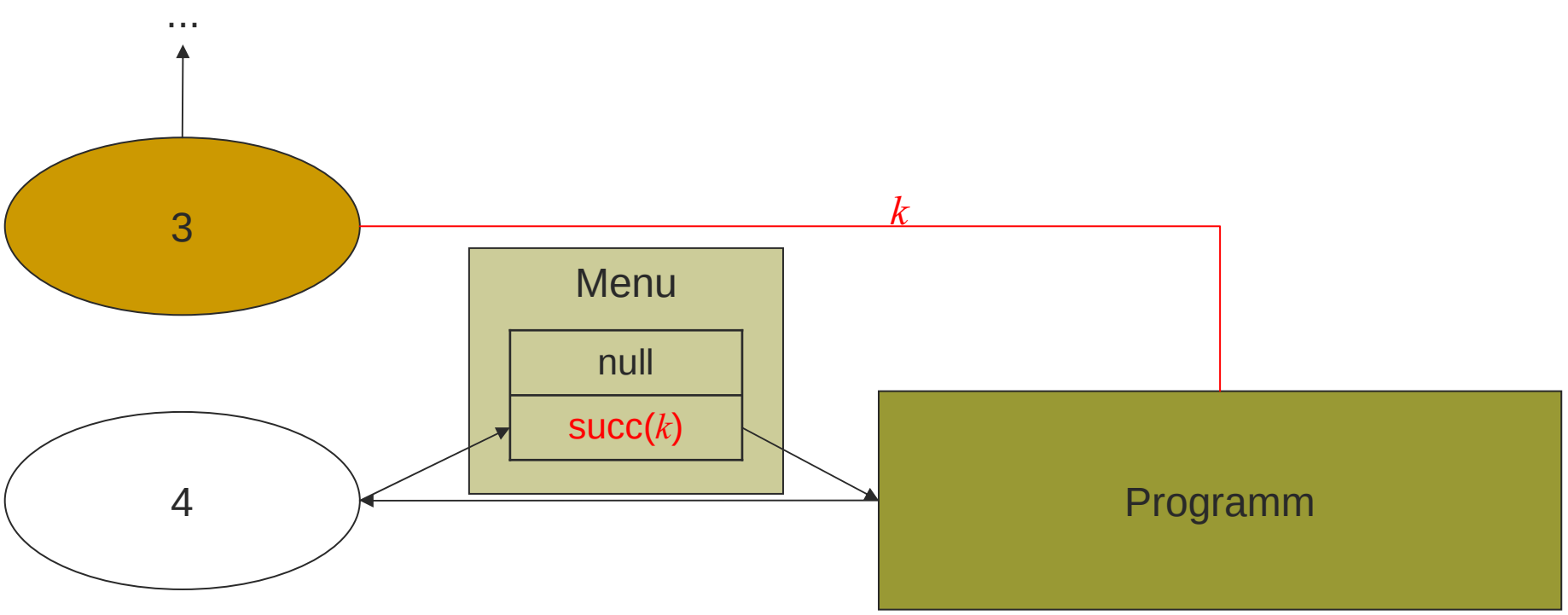
Zahlen

- parametrisierte Menueinträge



Zahlen

- parametrisierte Menueinträge



Zahlen

- Konstruktoren:

$$Zero := (k).k(n\,s).\overline{n}$$

$$Succ(N) := (k).\,\text{new}\,l\,(k(n\,s).\overline{s}\,\langle l\rangle \mid N\,\langle l\rangle)$$

- Dekonstruktor:

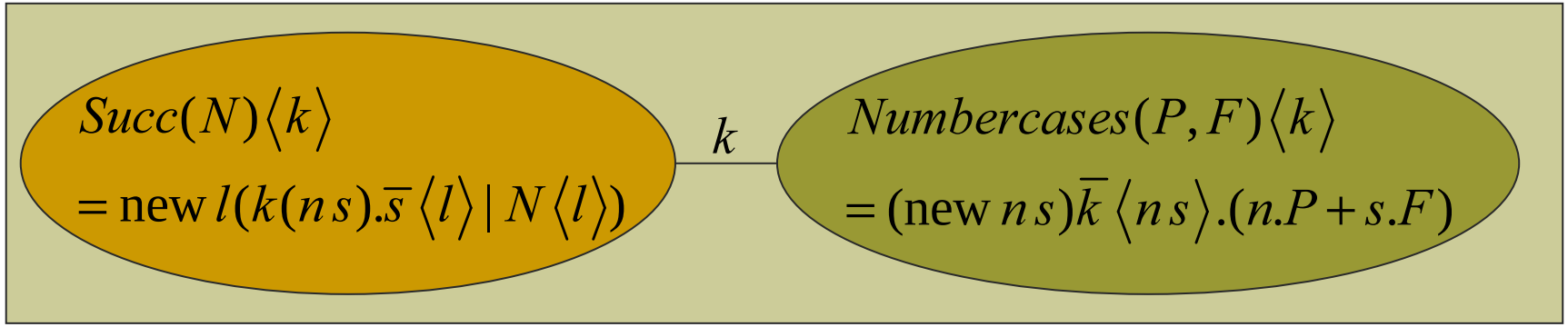
$$Numbercases(P,F)(k) := (\text{new}\,n\,s)\overline{k}\,\langle n\,s\rangle.(n.P + s.F)$$

- Beispiel:

$$4 = Succ(Succ(Succ(Succ(Null))))$$

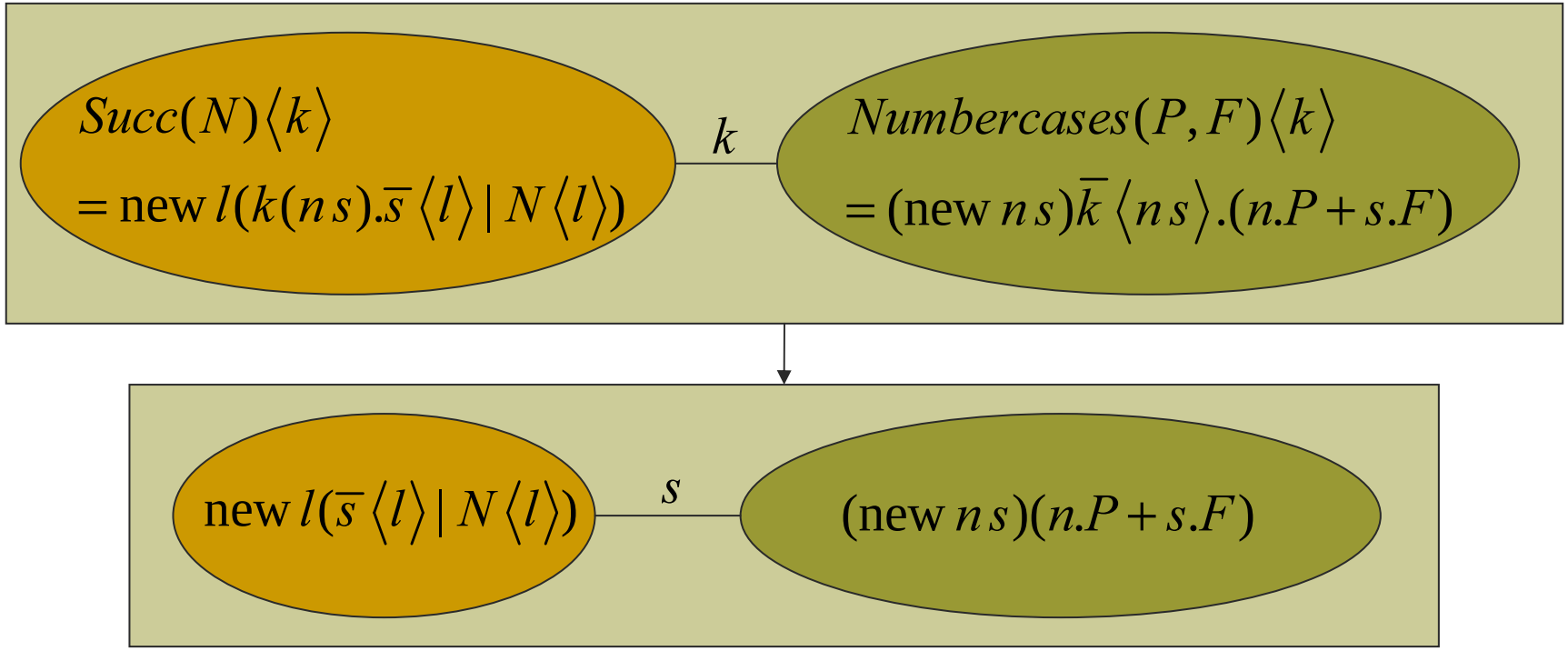
Korrektheit von Numbercases

$$Succ(N)\langle k\rangle \mid Numbercases(P,F)\langle k\rangle \rightarrow^{*} \text{new}\, l\,(N\langle l\rangle \mid F\langle l\rangle)$$



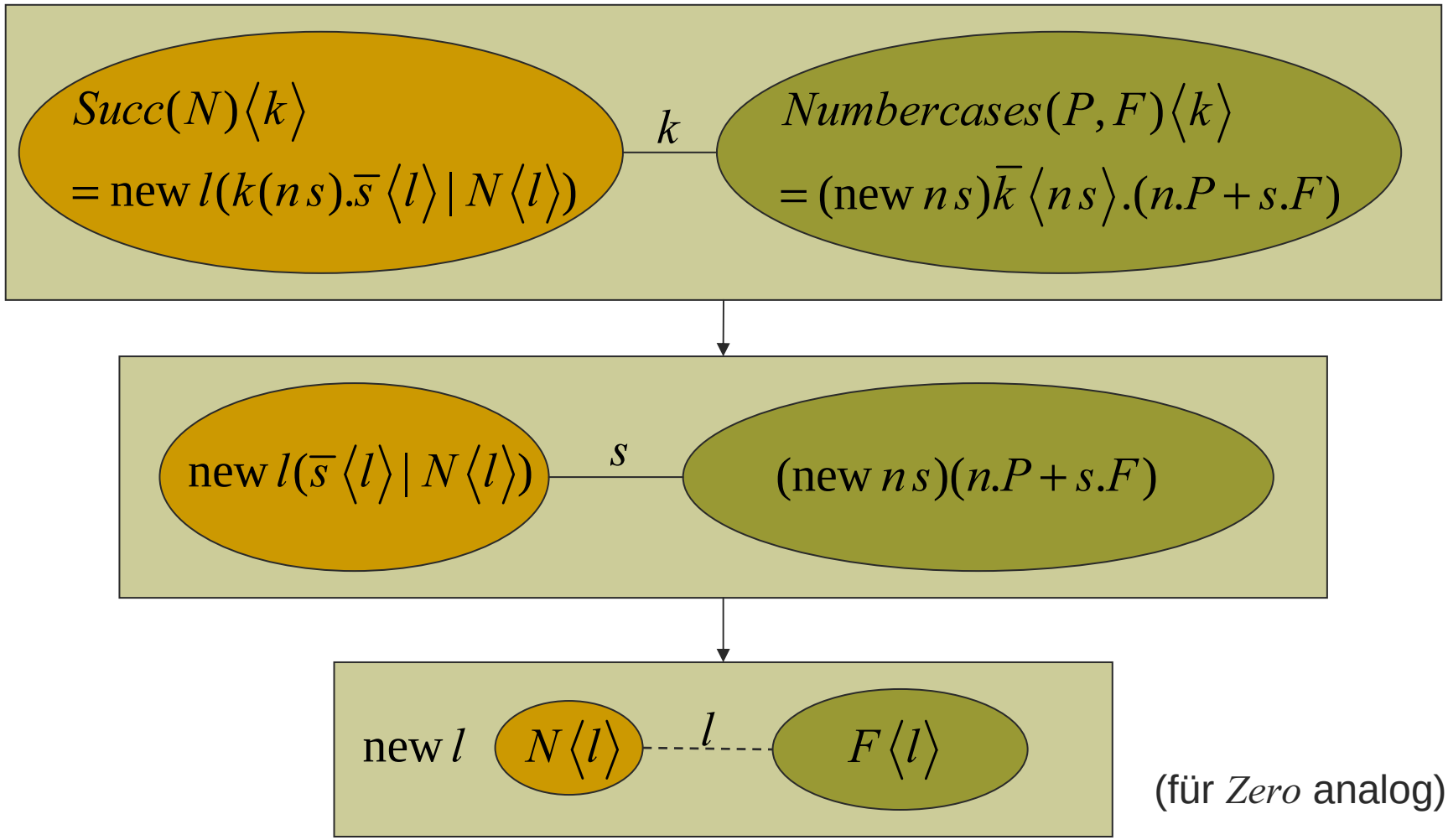
Korrektheit von Numbercases

$$Succ(N)\langle k\rangle \mid Numbercases(P,F)\langle k\rangle \rightarrow^{*} \text{new}\, l\,(N\langle l\rangle \mid F\langle l\rangle)$$



Korrektheit von Numbercases

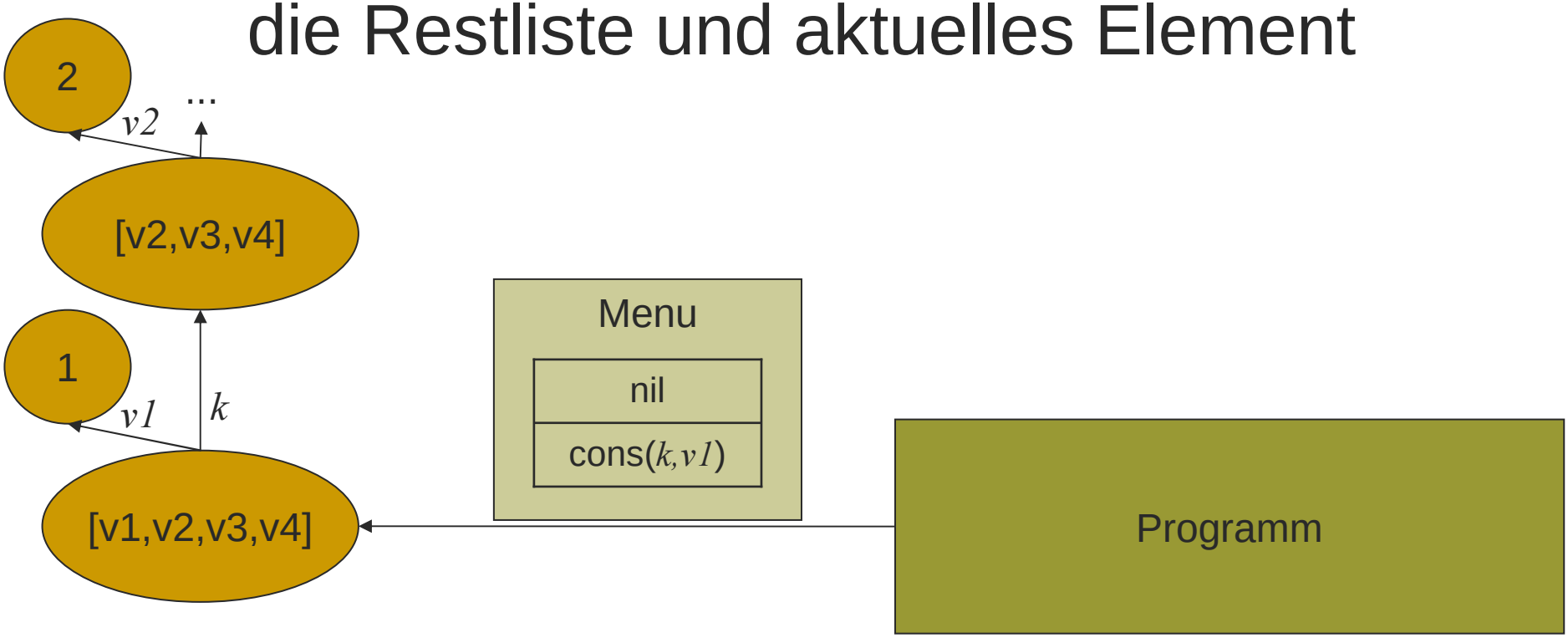
$$Succ(N)\langle k\rangle \mid Numbercases(P,F)\langle k\rangle \rightarrow^{*} \text{new}\, l\,(N\langle l\rangle \mid F\langle l\rangle)$$



(für *Zero* analog)

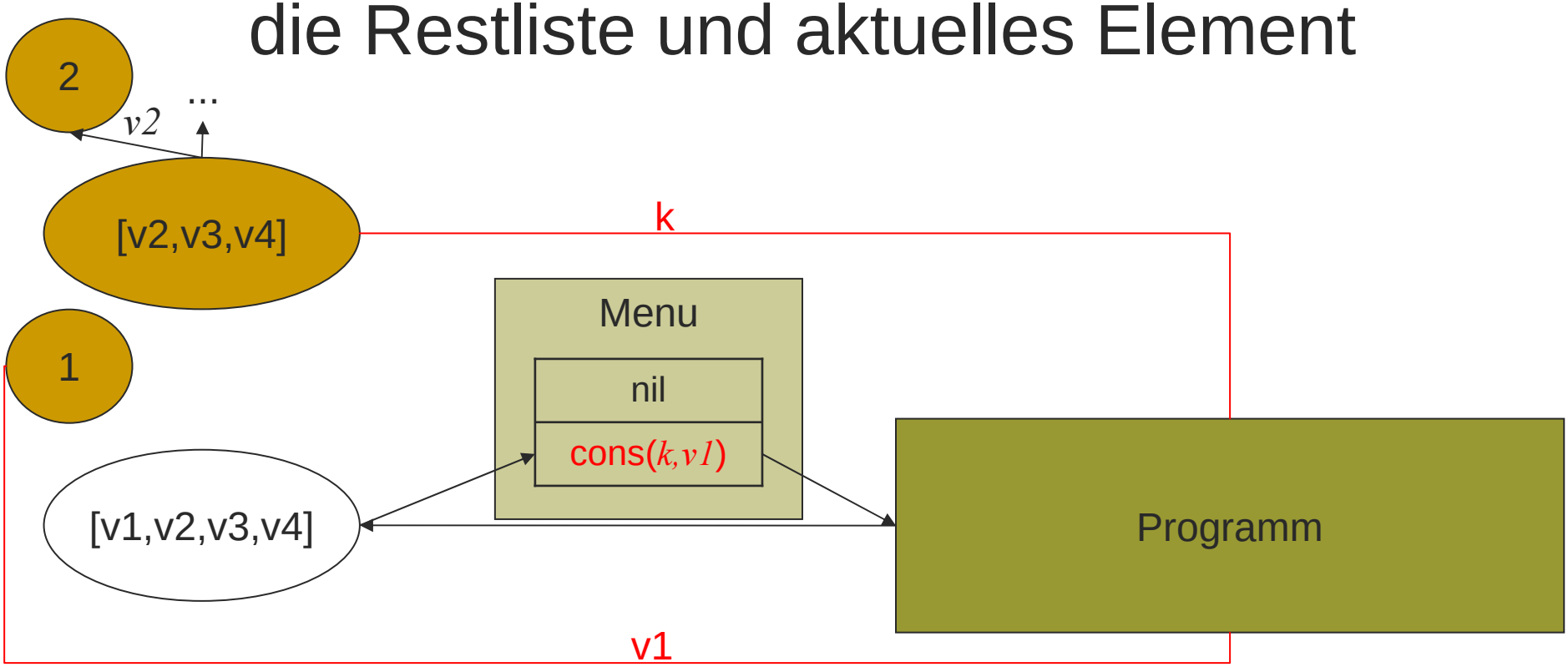
Listen

- gleiche Idee wie bei Zahlen: Zeiger auf die Restliste und aktuelles Element



Listen

- gleiche Idee wie bei Zahlen: Zeiger auf die Restliste und aktuelles Element



Listen

- Konstruktor:

$$Nil := (k).k(n\,c).\overline{n}$$

$$Cons(V, L) := (k).new\ v\ l\ (k(n\,c).\overline{c} < v\ l > | V < v > | L < l >)$$

- Dekonstruktor:

$$Listcases(P, F)(k) := (new\ n\ c)\overline{k}\left\langle n\,c\right\rangle.(n.P + c.F)$$

- Beispiel:

$$[1,2,3,4] = Cons(1, Cons(2, Cons(3, Cons(4, Nil))))$$

Programmierung

$$decr(km) := \text{case } k \text{ of}$$

$$Zero \Rightarrow Zero\langle m\rangle$$

$$Succ(k') \Rightarrow m(n\,s).k'\langle ns\rangle$$

$$\sim> decr(km) \mid Succ(N)\langle k\rangle \rightarrow^{*} \approx (N)\langle m\rangle$$

Programmierung

$$decr(km) := \text{case}\, k\, \text{of}$$

$$Zero \Rightarrow Zero\langle m\rangle$$

$$Succ(k') \Rightarrow m(n\,s).k'\langle n\,s\rangle$$

$$\sim> decr(km) \mid Succ(N)\langle k\rangle \rightarrow^{*}\approx (N)\langle m\rangle$$

$$Ncopy(lm) := \text{case}\, l\, \text{of}$$

$$Zero \Rightarrow Zero\langle m\rangle$$

$$Succ(l') \Rightarrow \text{new}\, m'(m(n\,s).s\langle m'\rangle \mid Ncopy(l'm'))$$

$$\sim> Ncopy\langle lm\rangle \mid N\langle l\rangle \rightarrow^{*} N\langle m\rangle$$

Programmierung

$$decr(km) := \text{case}\, k \,\text{of}$$

$$\begin{aligned} &Zero && \Rightarrow Zero\langle m\rangle \\ &Succ(k') && \Rightarrow m(n\,s).k'\langle n\,s\rangle \end{aligned}$$

$$\sim\!> decr(km) \mid Succ(N)\langle k\rangle \;\rightarrow^{*}\approx\; (N)\langle m\rangle$$

$$Ncopy(lm) := \text{case}\, l \,\text{of}$$

$$\begin{aligned} &Zero && \Rightarrow Zero\langle m\rangle \\ &Succ(l') && \Rightarrow \text{new}\, m'(m(n\,s).s\langle m'\rangle \mid Ncopy(l'\,m')) \end{aligned}$$

$$\sim\!> Ncopy\langle lm\rangle \mid N\langle l\rangle \rightarrow^{*} N\langle m\rangle$$

$$plus(klm) := \text{case}\, k \,\text{of}$$

$$\begin{aligned} &Zero && \Rightarrow Ncopy\langle lm\rangle \\ &Succ(k') && \Rightarrow \text{new}\, m'(m(n\,s).s\langle m'\rangle \mid plus(k'\,l\,m')) \end{aligned}$$

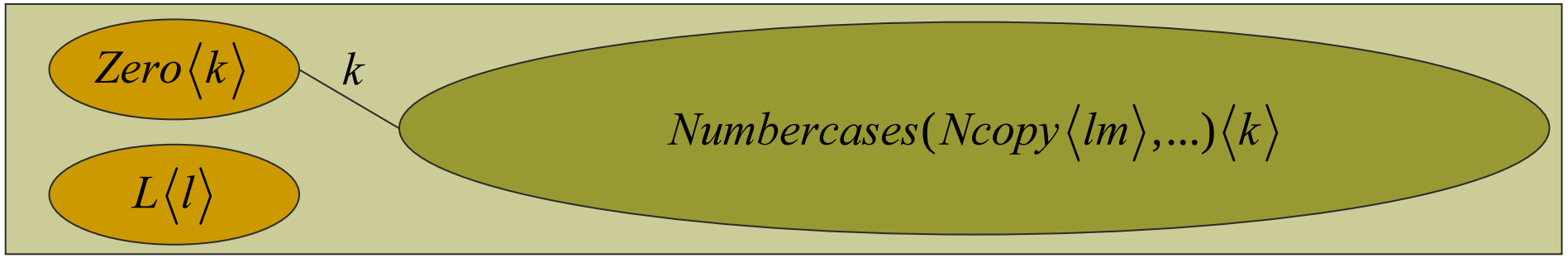
$$\sim\!> plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$$

Korrektheit von plus

$plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$:

strukturelle Induktion über K:

$K = Zero$:

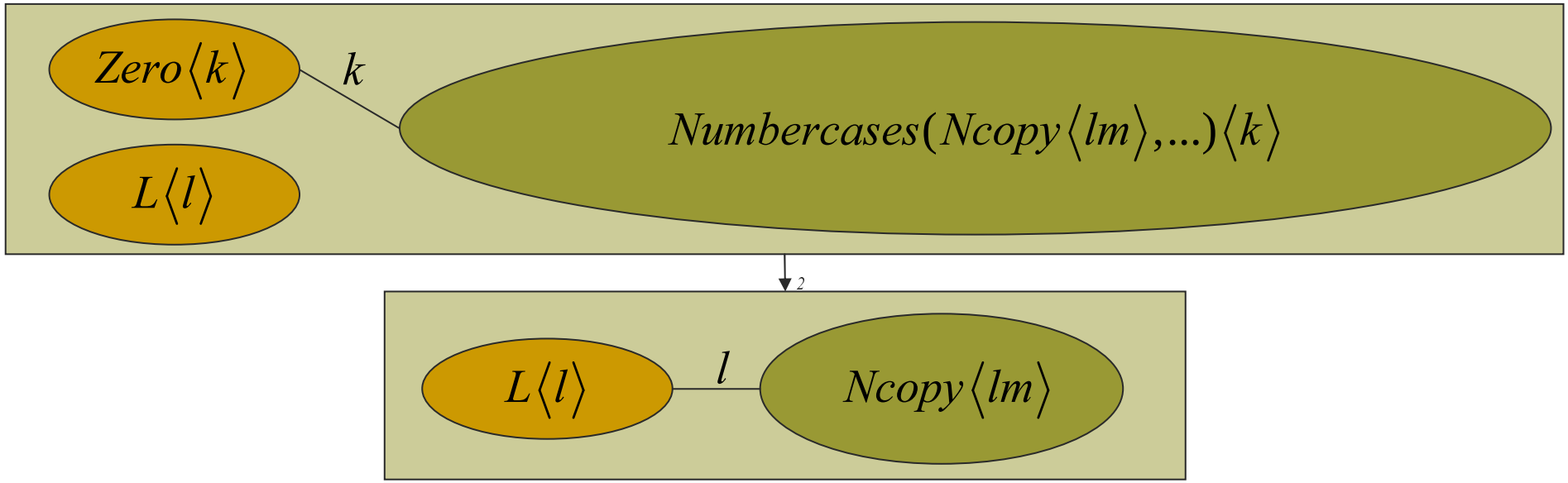


Korrektheit von plus

$plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$:

strukturelle Induktion über K:

$K = Zero$:

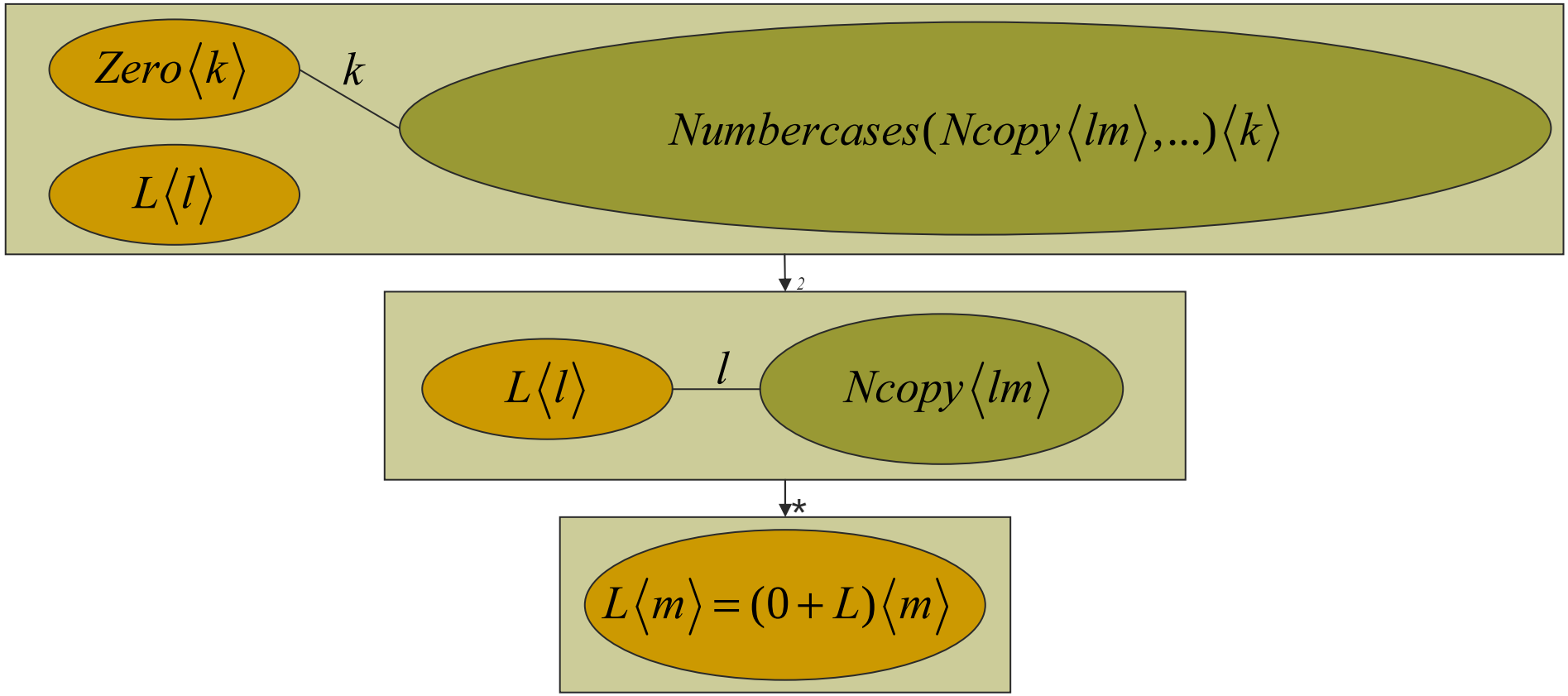


Korrektheit von plus

$plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$:

strukturelle Induktion über K:

$K = Zero$:

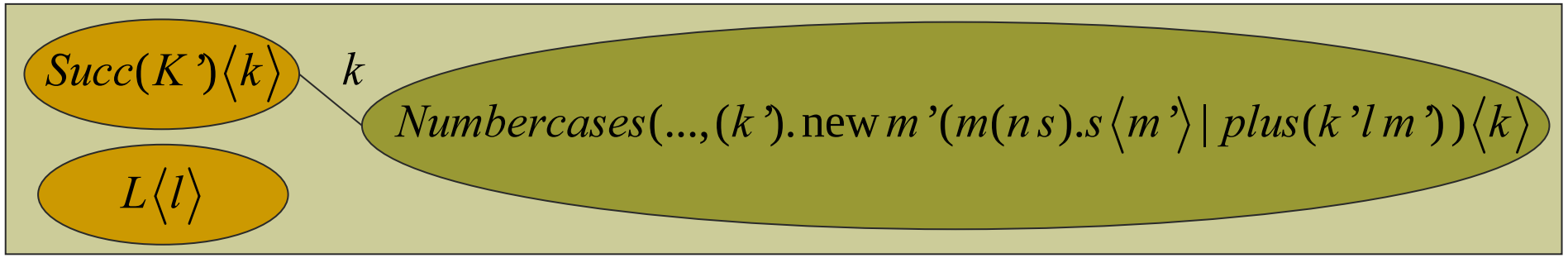


Korrektheit von plus

$plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$:

strukturelle Induktion über K:

$K = Succ(K')$:

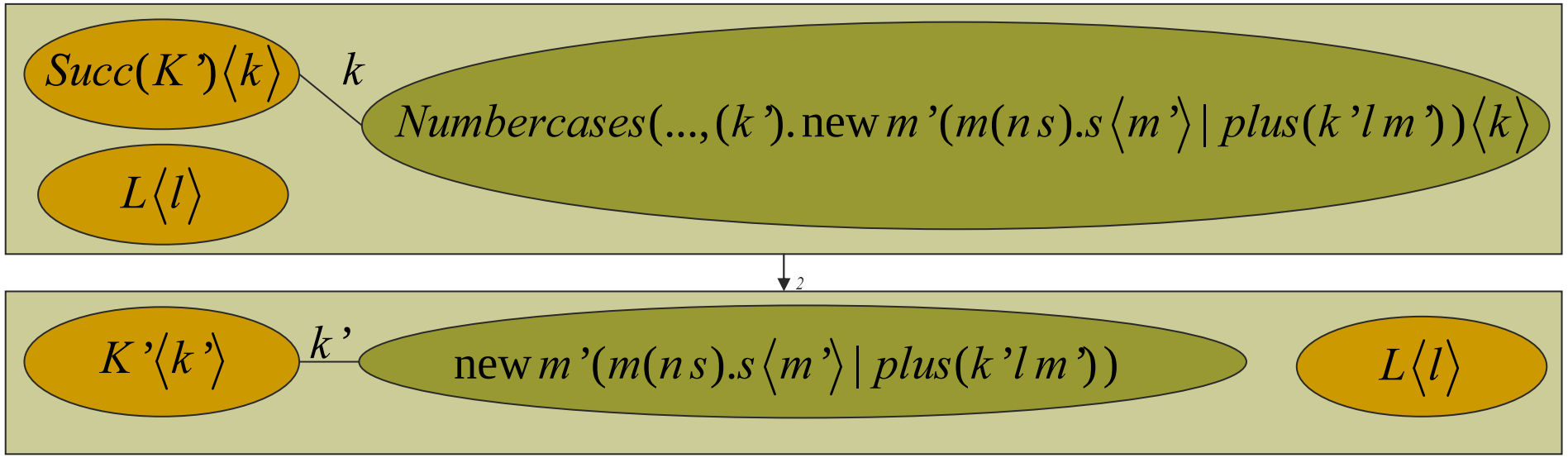


Korrektheit von plus

$plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$:

strukturelle Induktion über K:

$K = Succ(K')$:

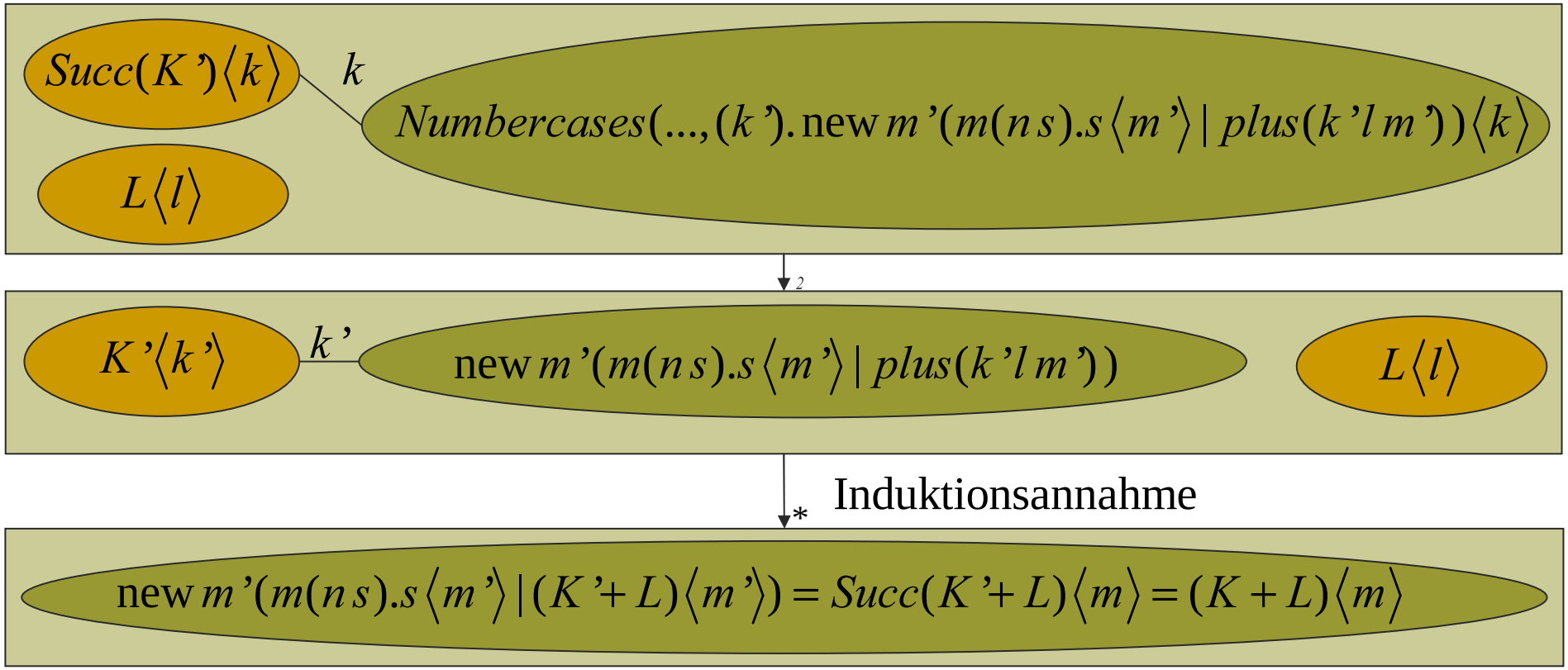


Korrektheit von plus

$plus(klm) \mid K\langle k\rangle \mid L\langle l\rangle \rightarrow^{*} (K+L)\langle m\rangle$:

strukturelle Induktion über K:

$K = Succ(K')$:



Unsichere Operatoren

- bisher definierte Operationen waren Abstraktionen
- Argumente waren die Lokationen der zu verrechnenden Werte

Unsichere Operatoren

- bisher definierte Operationen waren Abstraktionen
- Argumente waren die Lokationen der zu verrechnenden Werte → Unsicher! Beispiel:

$$P := Ncopy\langle lm \rangle \,|\, 3\langle l \rangle \,|\, 4\langle l \rangle$$

Unsichere Operatoren

- bisher definierte Operationen waren Abstraktionen
- Argumente waren die Lokationen der zu verrechnenden Werte → Unsicher! Beispiel:

$$P := Ncopy\langle lm\rangle \mid 3\langle l\rangle \mid 4\langle l\rangle$$

→ 2 mögliche Reaktionen:

$$P \rightarrow^* 3\langle m\rangle \mid 4\langle l\rangle,\ P \rightarrow^* 3\langle l\rangle \mid 4\langle m\rangle$$

Unsichere Operatoren

- bisher definierte Operationen waren Abstraktionen
- Argumente waren die Lokationen der zu verrechnenden Werte → Unsicher! Beispiel:

$$P := Ncopy\langle lm \rangle \mid 3\langle l \rangle \mid 4\langle l \rangle$$

→ 2 mögliche Reaktionen:

$$P \rightarrow^* 3\langle m \rangle \mid 4\langle l \rangle,\ P \rightarrow^* 3\langle l \rangle \mid 4\langle m \rangle$$

- Lösung: Restriktion!

$$NMove(N) := (m).\mathrm{new}\, l\, (N\langle l \rangle \mid Ncopy\langle lm \rangle)$$

Vergleich mit Alice

- Beispiel:

$$K\langle k\rangle \mid L\langle l\rangle \mid Join\langle klm\rangle \mid Length\langle m\rangle$$

Vergleich mit Alice

- Beispiel:

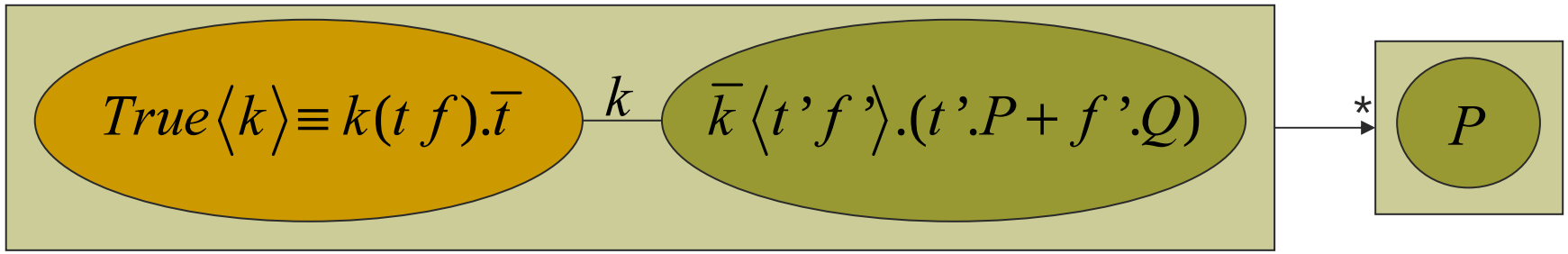
$$K\langle k\rangle \mid L\langle l\rangle \mid Join\langle klm\rangle \mid Length\langle m\rangle$$

versus

```
let
    fun ccAppend (x::xs,ys) = x::(spawn ccAppend(xs,ys))
      | ccAppend (nil,ys)   = ys
in
    spawn length (ccAppend(L,K))
end
```

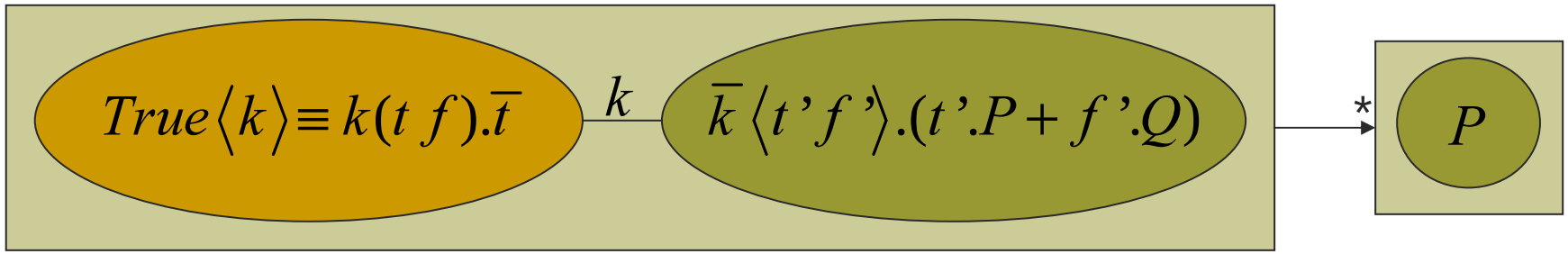
→ gleiche nebenläufige Berechnungsdisziplin

persistente Daten



- Daten verschwinden durch Benutzung

persistente Daten

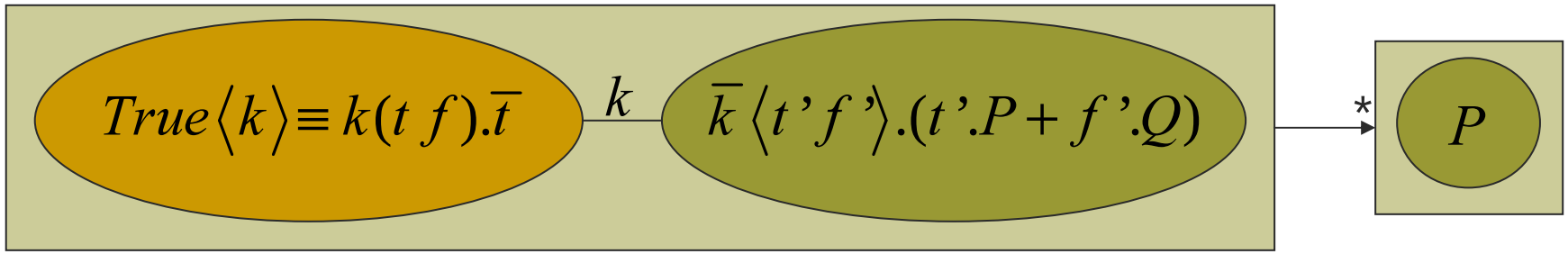


- Daten verschwinden durch Benutzung
  → Replikation:

$$*True(k) := !k(t\ f).\overline{t}$$

$$*False(k) := !k(t\ f).\overline{f}$$

persistente Daten



- Daten verschwinden durch Benutzung
  → Replikation:

$$*True(k) := !k(t\ f).\overline{t}$$

$$*False(k) := !k(t\ f).\overline{f}$$

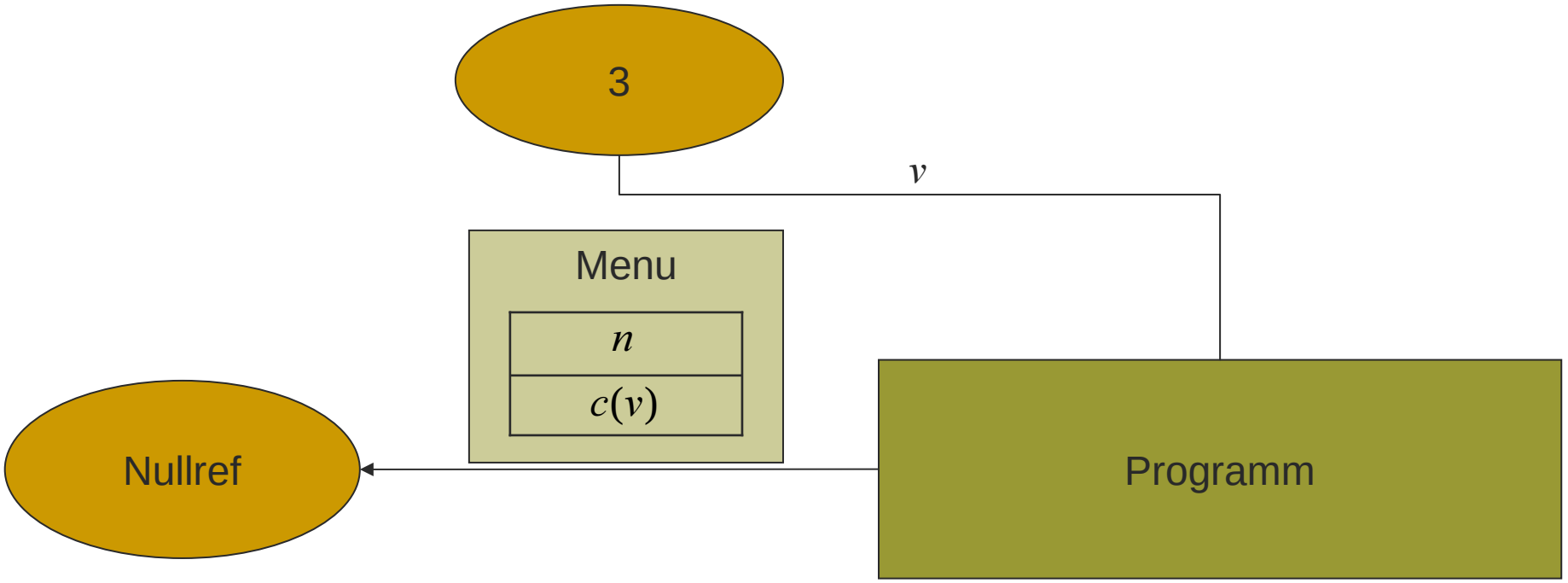
- Zahlen:

$$*Zero := (k).!k(n\ s).\overline{n}$$

$$*Succ(N) := (k).\mathrm{new}\ l\ (!k(n\ s).\overline{s}\langle l\rangle \mid N\langle l\rangle)$$

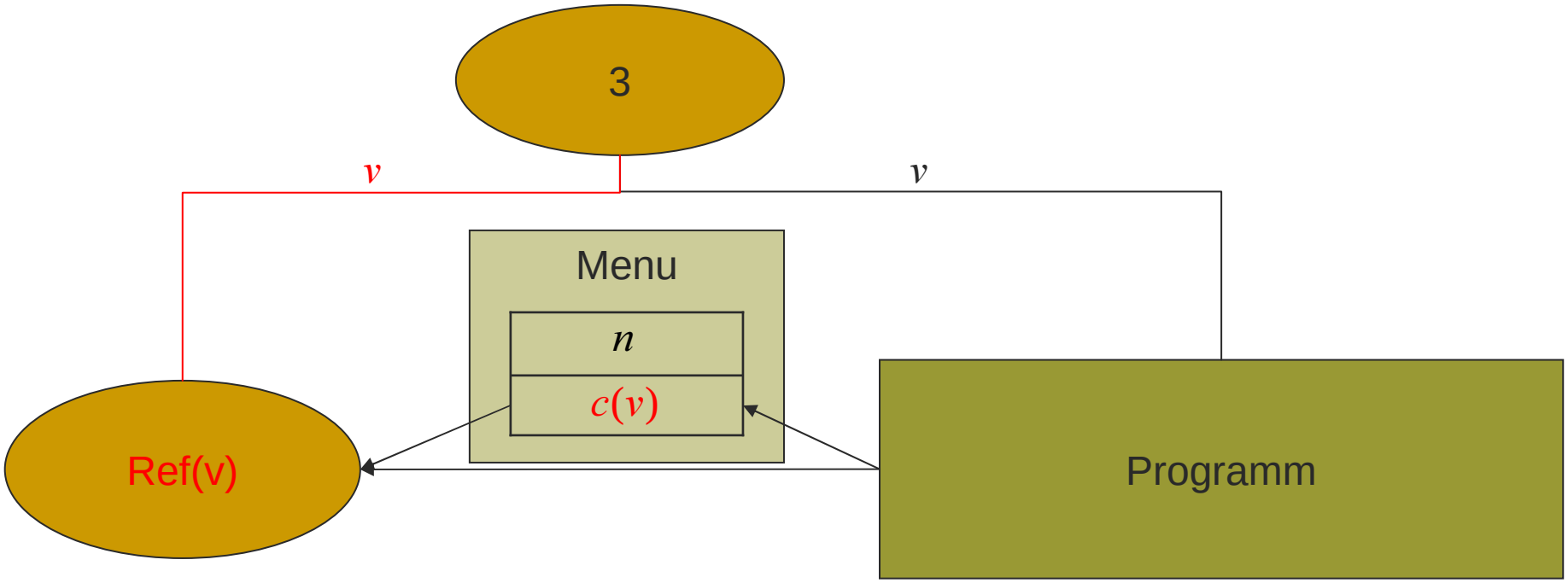
persistent und veränderbar

- Neu! Programm darf auch selektieren!



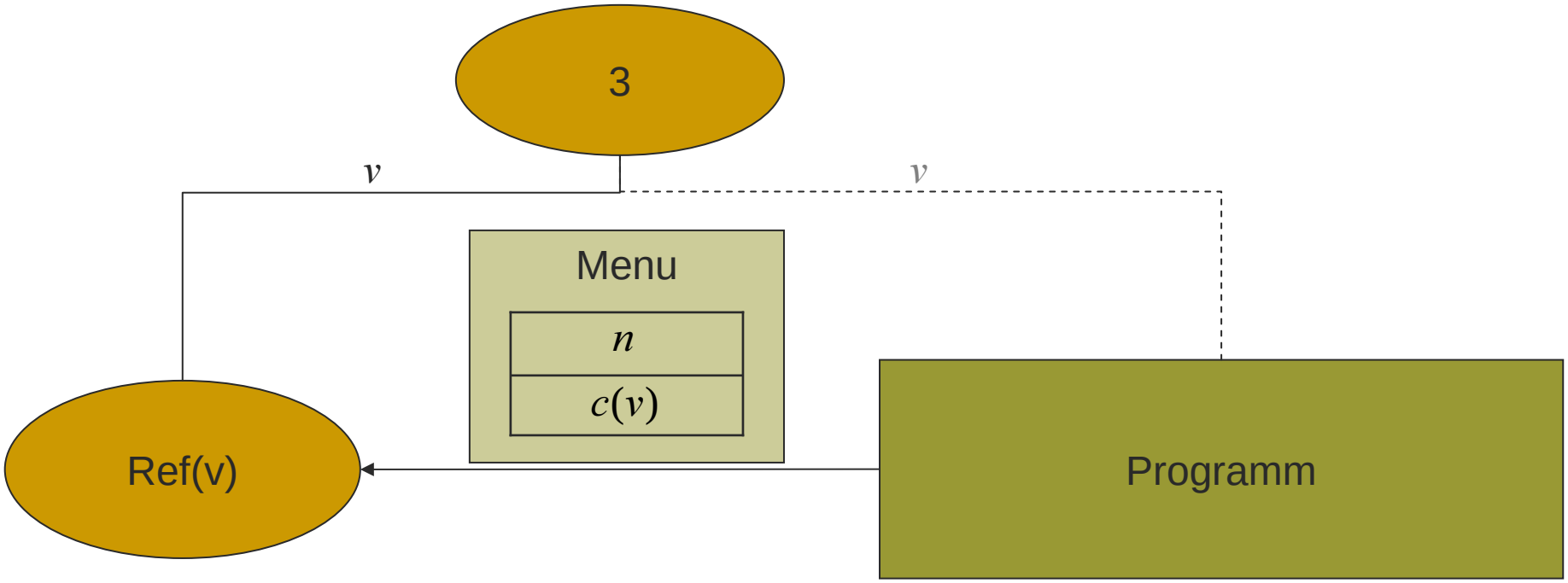
persistent und veränderbar

- Neu! Programm darf auch selektieren!



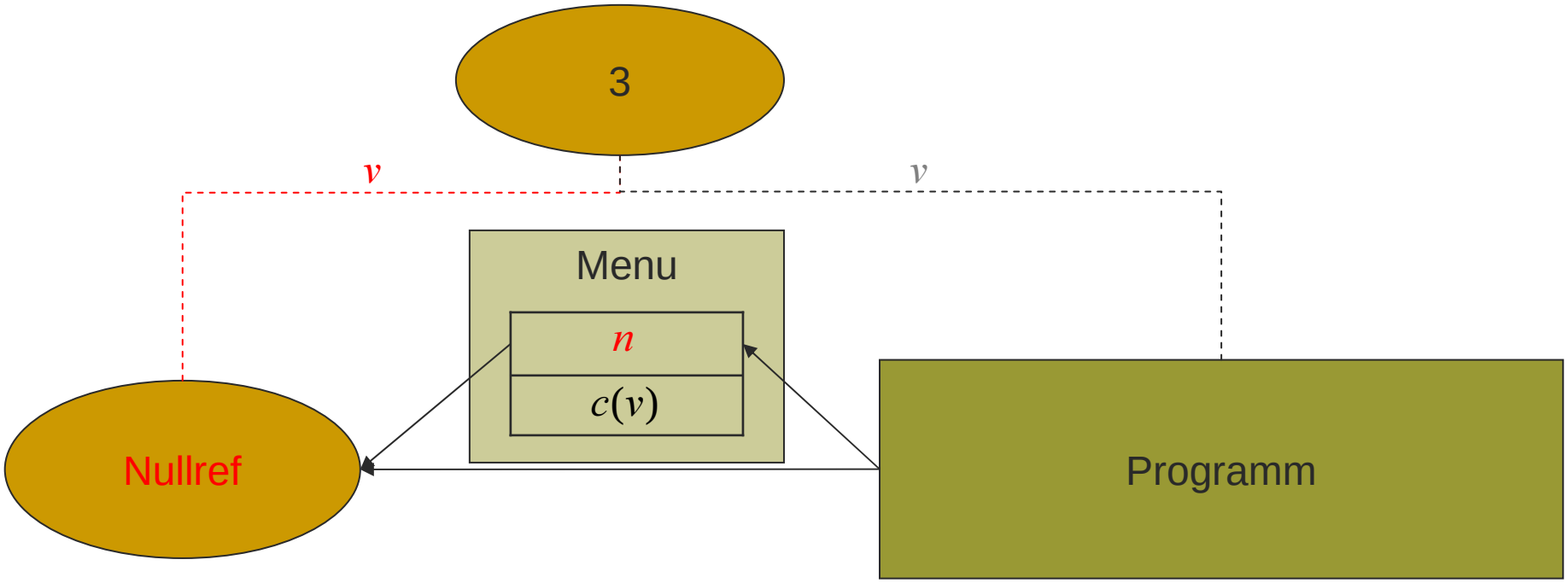
persistent und veränderbar

- Neu! Programm darf auch selektieren!



persistent und veränderbar

- Neu! Programm darf auch selektieren!



persistent und veränderbar

- Referenzen

$$Nullref(r) := r(n\,c).(\overline{n}.Nullref\langle r\rangle + c(v').Ref\langle rv'\rangle + n.Nullref\langle r\rangle)$$

$$Ref(rv) := r(n\,c).(\overline{c}\langle v\rangle.Ref\langle rv\rangle + c(v').Ref\langle rv'\rangle + n.Nullref\langle r\rangle)$$

persistent und veränderbar

- Referenzen

$$Nullref(r) := r(n\,c).(\overline{n}.Nullref\langle r\rangle + c(v').Ref\langle rv'\rangle + n.Nullref\langle r\rangle)$$

$$Ref(rv) := r(n\,c).(\overline{c}\langle v\rangle.Ref\langle rv\rangle + c(v').Ref\langle rv'\rangle + n.Nullref\langle r\rangle)$$

→ veränderbare Zelle(sicher):

$$Store(V) := (r).\mathrm{new}\,v\,(Ref\langle rv\rangle \mid V\langle v\rangle)$$

persistent und veränderbar

- Referenzen

$$Nullref(r) := r(n\,c).(\overline{n}.Nullref\langle r\rangle + c(v').Ref\langle rv'\rangle + n.Nullref\langle r\rangle)$$

$$Ref(rv) := r(n\,c).(\overline{c}\langle v\rangle.Ref\langle rv\rangle + c(v').Ref\langle rv'\rangle + n.Nullref\langle r\rangle)$$

→ veränderbare Zelle(sicher):

$$Store(V) := (r).\mathrm{new}\,v\,(Ref\langle rv\rangle \mid V\langle v\rangle)$$

- Operationen:

$$Refcases(P,F) := (r).(\mathrm{new}\,n\,c)\overline{r}\langle nc\rangle.(n.P + c.F)$$

$$Nullify(P) := (r).(\mathrm{new}\,n\,c)\overline{r}\langle nc\rangle.\overline{n}.P$$

$$Assign(P) := (rv).(\mathrm{new}\,n\,c).\overline{r}\langle nc\rangle.\overline{c}\langle v\rangle.P$$

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k
- Daten selektieren *Menuoption*

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k
- Daten selektieren *Menuoption*
- Darstellung beliebig großer Datenstrukturen (Zahlen/Listen) →*parametrisierte Menuoptionen*

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k
- Daten selektieren *Menuoption*
- Darstellung beliebig großer Datenstrukturen (Zahlen/Listen) →*parametrisierte Menuoptionen*
- Berechnungen sind nebenläufig, Restriktionen verhindern ungewollte Reaktionen

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k
- Daten selektieren *Menuoption*
- Darstellung beliebig großer Datenstrukturen (Zahlen/Listen) →*parametrisierte Menuoptionen*
- Berechnungen sind nebenläufig, Restriktionen verhindern ungewollte Reaktionen
- Korrektheit der abgeleiteten Operatoren lässt sich **meist** durch *Nachrechnen der Reaktionen* zeigen

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k
- Daten selektieren *Menuoption*
- Darstellung beliebig großer Datenstrukturen (Zahlen/Listen) →*parametrisierte Menuoptionen*
- Berechnungen sind nebenläufig, Restriktionen verhindern ungewollte Reaktionen
- Korrektheit der abgeleiteten Operatoren lässt sich **meist** durch *Nachrechnen der Reaktionen* zeigen
- Replikation verhindert „Flüchtigkeit“ der Darstellung

Zusammenfassung

- Daten empfangen ein *Menu* über eine *Lokation* k
- Daten selektieren *Menuoption*
- Darstellung beliebig großer Datenstrukturen (Zahlen/Listen) →*parametrisierte Menuoptionen*
- Berechnungen sind nebenläufig, Restriktionen verhindern ungewollte Reaktionen
- Korrektheit der abgeleiteten Operatoren lässt sich **meist** durch *Nachrechnen der Reaktionen* zeigen
- Replikation verhindert „Flüchtigkeit“ der Darstellung
- Referenzen ergeben sich aus:
  - Replikation
  - Erweiterung des Protokolls

Referenzen

- Robin Milner, *Communicating and Mobile Systems: the Pi-Calculus*, Cambridge University Press, 1999
- The Alice Project, http://www.ps.uni-sb.de/alice/

Church-Encodings

- Darstellung von Daten im λ-Kalkül:

$$t ::= x \mid \lambda x.t \mid t\,t$$

$$(\lambda x.t)\,t' \rightarrow t[x := t']$$

$$true := \lambda xy.x$$

$$false := \lambda xy.y$$

$$zero := \lambda sn.n\,(\lambda x.x)$$

$$succ(N) := \lambda sn.s\,N$$

$$True := (k).k(t\ f).\overline{t}$$

$$False := (k).k(t\ f).\overline{f}$$

$$Zero := (k).k(s\ n).\overline{n}$$

$$Succ(N) := (k).\mathrm{new}\ l\ (k(n\,s).\overline{s}\,\langle l\rangle \mid N\,\langle l\rangle)$$