Fortsetzungssemantik für parallele Dialekte von Haskell

Referent: Torsten Graf

Betreuerin: Prof. Loogen

Haskell Curry

Basierend auf der Arbeit:

Continuation semantics for Parallel Haskell Dialects
von
Mercedes Hidalgo-Herrero und Yolanda Ortega-Mallén

Verschiedene Ansätze zur Umsetzung von Parallelität

- Explizit
  - Eden
- Semi-implizit
  - Glasgow parallel Haskell (GpH)
- Implizit
  - Parallel Haskell (pH)

Ziel der denotationelle Semantik von parallelen Sprachen

- Betrachtung des Aufwands von Programmen für dieselbe Ein- und Ausgabe in einem bedarfsgesteuerten Kontext
- Vergleich von
  - Funktionalität
    - Der berechnete Ergebniswert
  - Parallelismus
    - Kommunikation, Prozesserstellung, …
  - Aufteilung
    - Grad der Arbeitsverdopplung

Warum eine denotationelle Semantik?

- Operationelle Semantiken schon vorhanden
- Nicht vergleichbar, da zu sprachnah
- Nicht explizit → keine Änderung der denotationellen Semantik
- Implementierungsabhängigkeit

Warum Haskell?

- Breit gefächert
- Sehr bekannt
- Gut dokumentiert
- Vollständig semantisiert

Warum genau diese Sprachen?

- Eden:
  - Konstrukte für explizite Erzeugung, Verwaltung und Kommunikation von Prozessen
  - Nichtdeterminismus für viele-zu-eins Kommunikationen
- Gph:
  - Kombinatoren ‘seq‘ und ‘par‘ mit Typ a -> b -> b
- pH:
  - zusätzliche Datenstrukturen

Beispiel für die Anweisung „let“

- Haskell und pH
  - let s1 = sum l1, s2 = sum l2 in s1 * s2
- GpH
  - let s1 = sum l1, s2 = sum l2 in
    s2 ’par’ (s1 * s2)
- Eden
  - let p = process list -> sum list,
    s1 = p # l1, s2 = p # l2 in s1 * s2

Fortsetzungen

- Jede Anweisung ist Funktion, die das endgültige Programmergebnis berechnet
- Anweisung += Argument (Fortsetzung), welches einen Zustand in ein Endergebnis überführt
- Wird die Berechnung einer Anweisung beendet, wird die restliche Berechnung des Programms durch die Fortsetzung beschrieben

Fortsetzungssemantik (1)

- Bedarfssteuerung inhärent sequentiell
- Minimale Semantik
  - Nur notwendige Berechnungen werden durchgeführt
- Maximale Semantik
  - Alle möglichen Berechnungen werden durchgeführt

Fortsetzungssemantik (2)

- Standardisierte denotationelle Semantik nach Scott und Strachey
- Betrachteung des Ein-/ Ausgabeverhaltens von Programmen
- Ergänzung um Komponenten für die Parallelität, da Seiteneffekte entstehen können

Seiteneffekten

- Der Ausdruck (\x.3)y könnte einen neuen Prozesse benötigen
  - Bleibt aber unbeachtet, da nur das Ergebnis betrachtet wird
- Seiteneffekte:
  - Prozesserstellung
  - Interprozesskommunikation, Wertekommunikation
  - Synchronisierung
  - entstehen während der Auswertung von Ausdrücken
- Jeder Seiteneffekt sollte bei Bedarfssteuerung nur einmal auftauchen

Ausdrucksfortsetzung

- Funktion, die einen Wert bekommt und
  - eine Ausdrucksfortsetzung zurückgibt oder
  - eine Zustandstransformation bewirkt

Zielsetzung

- Definition einer denotationellen Semantik für ein bedarfsgesteuertes λ-Kalkül
- Ergänzt mit den jeweiligen Sprachcharakteristika
- Ziel: bessere Vergleichsmöglichkeit durch einheitliche Syntax und standardisierte Semantik

Gliederung

- Kernsyntax der jeweiligen Sprache
- Die jeweiligen semantischen Bereiche
- Die Denotationelle Semantik der Kerns

Eden

- Prozessabstraktion
  - z.B. abstrakte Schemata für das Verhalten von Prozessen
- unidirektionale Kommunikation
- Beliebige Kommunikationstopologien
- Mögliche verteilte Implementierung
- Synchronisierung: warten auf Eingabe

Die Kernsyntax von Eden

E ::=	x	Bezeichner
|	\x.E	λ-Abstraktion
|	x_1 $ x_2	bedarfsgesteuerte Applikation
|	x_1 $# x_2	parallele Applikation
|	let $\{x_i = E_i\}_{i=1}^{n}$ in x	lokale Definition
|	x_1 [] x_2	nichtdeterministische Auswahl

Semantische Bereiche von Eden (1)

- Fortsetzungen:
  - **Cont** = **State** → **SState**
- Zustände
  - $s \in$ **State** = **Env** × **SChan**
- Umgebungen
  - $\rho \in$ **Env** = **Ide** → (**Val** ∪ {not_ready})
- Menge von Kanälen
  - Ch ∈ **SChan** = P_f(**Chan**)

Semantische Bereiche von Eden (2)

- Ausdrucksfortsetzungen
  - $\kappa \in$ **ECont** = **EVal** → **Cont**
- Kanäle
  - **Chan** = **IdProc** × **CVal** × **IdProc**
- Menge von Zuständen
  - $S \in$ **SState** = P_f(**State**)
- Kommunikationswerte
  - **CVal** = **EVal** ∪ {unsent}

Semantische Bereiche von Eden (3)

- Werte
  - $\upsilon \in$ **Val** = **EVal** $\cup$ **(IdProc** $\times$ **Clo)** $\cup$ {not_ready}
- Ausdruckswerte
  - $\varepsilon \in$ **EVal** = **Abs** $\times$ **Ides**
- Absraktionswerte
  - $\alpha \in$ **Abs** = **Ide** $\rightarrow$ **Clo**

Semantische Bereiche von Eden (4)

- closures
  - $\nu \in$ **Clo** = **IdProc** $\rightarrow$ **ECont** $\rightarrow$ **Cont**
- Menge von Bezeichnern
  - $I \in$ **Ides** = P_f(**Ide**)
- Prozessbezeichner
  - $p,q \in$ **IdProc**

Übersicht der semantischen Bereiche von Eden

		Cont	= **State** → **SState**	Fortsetzungen
κ	∈ **ECont**	=	**EVal** → **Cont**	Ausdrucksfortsetzungen
s	∈ **State**	=	**Env** × **SChan**	Zustände
S	∈ **SState**	=	P_f(**State**)	Menge von Zuständen
ρ	∈ **Env**	=	**Ide** → (**Val** ∪ {`not_ready`})	Umgebungen
Ch	∈ **SChan**	=	P_f(**Chan**)	Menge von Kanälen
		Chan	= **IdProc** × **CVal** × **IdProc**	Kanäle
		CVal	= **EVal** ∪ {`unsent`}	Kommunikationswerte
υ	∈ **Val**	=	**EVal** ∪ (**IdProc** × **Clo**) ∪ {`not_ready`}	Werte
ε	∈ **EVal**	=	**Abs** × **Ides**	Ausdruckswerte
α	∈ **Abs**	=	**Ide** → **Clo**	Abstraktionswerte
ν	∈ **Clo**	=	**IdProc** → **ECont** → **Cont**	closures
I	∈ **Ides**	=	P_f(**Ide**)	Menge von Bezeichnern
p,q	∈ **IdProc**			Prozessbezeichner

Die Signatur der semantischen Funktion für Ausdrücke in Eden

$$\mathcal{L} : \mathbf{Exp} \rightarrow \mathbf{IdProc} \rightarrow \mathbf{ECont} \rightarrow \mathbf{Cont}$$

Edens Anweisungssemantik (1)

$$\mathcal{L}[\![x]\!]p\kappa = \texttt{force}\ x\ \kappa$$

$$\mathcal{L}[\![\backslash x.E]\!]p\kappa = \kappa\langle \lambda x.\mathcal{L}[\![E]\!],\ fv(\backslash x.E)\rangle$$

$$\mathcal{L}[\![x_1\ \$\ x_2]\!]p\kappa = \mathcal{L}[\![x_1]\!]\ p\kappa'$$

$$\texttt{where}\ \kappa' = \lambda\langle\alpha,I\rangle.\lambda s.(\ \alpha x_2)\ p\kappa s$$

$$\mathcal{L}[\![x_1\ [\!]\ x_2]\!]p\kappa = \lambda s.(\mathcal{L}[\![x_1]\!]p\kappa s)\cup(\mathcal{L}[\![x_2]\!]p\kappa s)$$

Edens Anweisungssemantik (2)

$\mathcal{L}[\![x_1\ \$\#\ x_2]\!]p\kappa = \texttt{forceFV}\ x_1\ \kappa'$

where $\kappa' = \lambda\langle\alpha,I\rangle.\lambda s.(\ \alpha x_2)\ q\kappa''s$

q = newIdProc s

$\kappa''_{min} = \lambda\langle\alpha',I'\rangle.\lambda s'.$ case $(\rho' x_2)$ of

$\langle\alpha'',I''\rangle \in \mathbf{EVal} \rightarrow S_d \oplus_{ch} \{\langle q, \langle\ \alpha,I\ \rangle, p\rangle, \langle p, \langle\ \alpha'',I''\ \rangle, q\rangle\}$

otherwise $\rightarrow S_d \oplus_{ch} \{\langle q, \langle\ \alpha,I\ \rangle, p\rangle, \langle p, \texttt{unsent}, q\rangle\}$

endcase

where $(\rho', Ch') = s'$

S_c = mforceFV I' s'

$S_d = \bigcup_{s_c \in S_c}$ mforceFV I'' s_c

$\kappa''_{max} = \lambda\langle\alpha',I'\rangle.\lambda s'. \bigcup_{s_c \in S_c}$ forceFV x_2 κ_c s_c

where S_c = mforceFV I' s'

$\kappa_c = \lambda\varepsilon''.\ \lambda s''.\{s'' \oplus_{ch} \{\langle q, \langle\ \alpha, I\ \rangle, p\rangle, \langle p, \varepsilon'', q\rangle\}$

Edens Anweisungssemantik (3)

$$\mathcal{L}[\![\text{let } \{x_i = E_i\}_{i=1}^{n} \text{ in } x]\!]p\kappa = \lambda\langle\rho, Ch\rangle.\mathcal{L}[\![x]\!]p\kappa'\langle\rho', Ch\rangle$$

where $\{y_1,\ldots,y_n\} = \text{newIde } n\ \rho$

$$\rho' = \rho \oplus \{y_i \mapsto \langle \mathcal{L}[\![E_i[y_1/x_1,\ldots, y_n/x_n]]\!], p\rangle \mid 1 \le i \le n\}$$

$\kappa'_{mir} = \kappa$
$\kappa'_{max} = \lambda\varepsilon.\lambda s.\texttt{mforce}\ I\ s$ where $I = \{y_i \mid E_i \equiv x_1^i \,\$\#\, x_2^i \wedge 1 \le i \le n\}$

Hilfsfunktionen für Eden

```
force :: Ide → ECont → Cont
force x κ = λ⟨ρ,Ch⟩.case (ρ x) of
  ε ∈ EVal → κε⟨ρ,Ch⟩
  ⟨p,ν⟩ ∈ (IdProc × Clo) → νpκ's'
    where κ' = λε'.λ⟨ρ',Ch'⟩.κε'⟨ρ'⊕{x ↦ ε'}, Ch'⟩
          s' = ⟨ρ⊕{ x ↦ not_ready}, Ch⟩
  otherwise → wrong

forceFV :: Ide → ECont → Cont
forceFV x κ = force x κ'
  where κ' = λ⟨α, I⟩.λs'. ⋃_{s''∈S''} κ_c ⟨α, I⟩ s''
        S'' = mforceFV I s'

mforceFV :: Ides → Cont
mforceFV {} = λs.{s}
mforceFV ({x} ∪ I) = λs. ⋃_{s'∈S'} mforceFV I s'
  where S' = forceFV x id_κ s
```

GpH

- e1 `par` e2
  - e2 wird immer ausgewertet
  - e1 nur beim Vorhandensein von Ressourcen
- e1 `seq` e2
  - e1 wird immer zuerst ausgewertet
  - e2 immer nur daran anschließend

Die Kernsyntax von GpH

E ::=	x	Bezeichner
|	\x.E	λ-Abstraktion
|	x_1 $ x_2	bedarfsgesteuerte Applikation
|	let $\{x_i = E_i\}_{i=1}^{n}$ in x	lokale Definition
|	x_1`'seq'`x_2	sequentielle Komposition
|	x_1`'par'`x_2	parallele Komposition

Semantische Bereiche von GpH

	Cont	$= \mathbf{Env} \to \mathbf{Env}$	Fortsetzungen
$\kappa \in$	**ECont**	$= \mathbf{EVal} \to \mathbf{Cont}$	Ausdrucksfortsetzungen
$\rho \in$	**Env**	$= \mathbf{Ide} \to (\mathbf{Val} \cup \{\texttt{undefined}\})$	Umgebungen
$\upsilon \in$	**Val**	$= \mathbf{EVal} \cup \mathbf{Clo} \cup \{\texttt{not\_ready}\}$	Werte
$\varepsilon \in$	**EVal**	$= \mathbf{Abs}$	Ausdruckswerte
$\alpha \in$	**Abs**	$= \mathbf{Ide} \to \mathbf{Clo}$	Abstraktionswerte
$\nu \in$	**Clo**	$= \mathbf{ECont} \to \mathbf{Cont}$	closures

Die Signatur der semantischen Funktion für Ausdrücke in GpH

$$\mathcal{L}: \mathbf{Exp} \to \mathbf{ECont} \to \mathbf{Cont}$$

GpHs Anweisungssemantik (1)

$$\mathcal{L}[\![x]\!]\kappa = \texttt{force}\ x\ \kappa$$

$$\mathcal{L}[\![\backslash x.E]\!]\kappa = \kappa(\lambda x.\mathcal{L}[\![E]\!])$$

$$\mathcal{L}[\![x_1\ \$\ x_2]\!]\kappa = \mathcal{L}[\![x_1]\!]\kappa'$$
$$\texttt{where}\ \kappa' = \lambda\varepsilon.\lambda\rho.\varepsilon x_2 \kappa\rho$$

$$\mathcal{L}[\![\text{let } \{x_i = E_i\}_{i=1}^{n} \text{ in } x]\!]\kappa = \lambda\rho.\mathcal{L}[\![x]\!]\rho\kappa'$$
$$\text{where } \{y_1,\ldots,y_n\} = \text{newIde } n\ \rho$$
$$\rho' = \rho \oplus \{y_i \mapsto \mathcal{L}[\![E_i[y_1/x_1,\ldots,\ y_n/x_n]]\!] \mid 1 \le i \le n\}$$

GpHs Anweisungssemantik (2)

$$\mathcal{L}[\![x_1\text{'seq'}x_2]\!]\kappa = \mathcal{L}[\![x_1]\!]\kappa'$$
$$\texttt{where}\ \kappa' = \lambda\varepsilon.\lambda\rho.\mathcal{L}[\![x_2]\!]\kappa\rho$$

$$\mathcal{L}[\![x_1\text{'par'}x_2]\!]\kappa = \mathcal{L}[\![x_2]\!]\kappa'$$
$$\texttt{where}\quad \kappa' = \lambda\varepsilon.\lambda\rho.\kappa\varepsilon\rho_{par}$$
$$\rho_{par} = \text{par } x_1\ \rho$$

Hilfsfunktionen von GpH

$$\texttt{force} :: \mathbf{Ide} \to \mathbf{ECont} \to \mathbf{Cont}$$

$$\texttt{force}\ x\ \kappa = \lambda\rho.\texttt{case}\ (\rho, x)\ \texttt{of}$$

$$\quad \varepsilon \in \mathbf{EVal} \to \kappa\ \varepsilon\ \rho$$

$$\quad \nu \in \mathbf{Clo} \to \nu\ \kappa'\ \rho'$$

$$\quad\quad \texttt{where}\ \kappa' = \lambda\varepsilon''.\lambda\rho''.\kappa\varepsilon''\rho'' \oplus \{x \mapsto \varepsilon\}$$

$$\quad\quad\quad \rho' = \rho \oplus \{x \mapsto \texttt{not\_ready}\}$$

$$\texttt{endcase}$$

$$\texttt{par} :: \mathbf{Ide} \to \mathbf{Cont}$$

$$\texttt{par}_{min}\ x = \lambda\rho.\rho$$

$$\texttt{par}_{max}\ x = \lambda\rho.\mathcal{L}[\![x]\!]\ \mathrm{id}_\kappa\ \rho$$

pH

- Id dataflow Sprache gleichen Typsystems
- Standardisierte Syntax + das Typsystem von Haskell ergeben dann pH
- Versuch der Vereinigung von funktionalen und dataflow Sprachen
- Mischung aus strikter und bedarfsgesteuerter Auswertung

pH Fortsetzung

- I-Struktur (ICells)
  - einmaliges Schreiben
- M-Struktur (MCells)
  - mehmaliges Schreiben, da
  - beim Lesen löschen des Inhalts

Die Kernsyntax von pH

E ::=	x	Bezeichner
|	\x.E	λ-Abstraktion
|	x_1 $ x_2	bedarfsgesteuerte Applikation
|	x_1 $! x_2	strikte Applikation
|	let $\{x_i = E_i\}_{i=1}^{n}$ in x	lokale Definition
|	iCell x | mCell x	Cell-Erstellung
|	Fetch x	Cell-Leseoperator
|	Store(x_1,x_2)	Cell-Schreiboperator

Semantische Bereiche von pH(1)

$\mathbf{Cont} = \mathbf{Store} \to \mathbf{SStore}$		Fortsetzungen
$\kappa \in \mathbf{ECont} = \mathbf{EVal} \to \mathbf{Cont}$		Ausdrucksfortsetzungen
$\sigma \in \mathbf{Store} = \mathbf{Loc} \to (\mathbf{Val} \cup \{\texttt{undefined}\})$		Speicher
$\Sigma \in \mathbf{SStore} = P_f(\mathbf{Store})$		Mengen von Speicher
$\rho \in \mathbf{Env} = \mathbf{Ide} \to \mathbf{Loc}$		Umgebungen

Semantische Bereiche von pH (2)

$\upsilon \in \mathbf{Val} = \mathbf{EVal} \cup \mathbf{Clo} \cup \mathbf{Cell} \cup \{\texttt{not\_ready}\}$		Werte
$\varepsilon \in \mathbf{EVal} = \mathbf{Abs} \cup \{\texttt{unit}\}$		Ausdruckswerte
$\alpha \in \mathbf{Abs} = \mathbf{Loc} \to \mathbf{Clo}$		Abstraktionswerte
$\nu \in \mathbf{Clo} = \mathbf{ECont} \to \mathbf{Cont}$		closures
$\mathbf{Cell} = \{I, M\} \times (\mathbf{EVal} \cup \{\texttt{empty}\})$		Cells
$l \in \mathbf{Loc}$		Speicherplätze

Die Signatur der semantischen Funktion für Ausdrücke in pH

$$\mathcal{L}: \mathbf{Exp} \rightarrow \mathbf{Env} \rightarrow \mathbf{ECont} \rightarrow \mathbf{Cont}$$

pHs Anweisungssemantik (1)

$$\mathcal{L}[\![\mathrm{x}]\!] \rho\kappa = \texttt{force}\ \ (\rho\ \mathrm{x})\ \kappa$$

$$\mathcal{L}[\![\backslash \mathrm{x}.\mathrm{E}]\!] \rho\kappa = \kappa\lambda l.\mathcal{L}[\![\mathrm{E}]\!](\rho \oplus \{\mathrm{x} \mapsto l\}))$$

$$\mathcal{L}[\![\mathrm{x}_1\ \$\ \mathrm{x}_2]\!] \rho\kappa = \mathcal{L}[\![\mathrm{x}_1]\!] \rho\kappa'$$

$$\texttt{where}\ \kappa' = \lambda\varepsilon.\lambda\sigma.\texttt{case}\ \varepsilon\ \texttt{of}$$

$$\varepsilon \in \mathbf{Abs} \rightarrow \varepsilon l \kappa \sigma'$$

$$\texttt{where}\ l = \texttt{freeloc}\ \sigma$$

$$\sigma' = \sigma \oplus \{l \mapsto \mathcal{L}[\![\mathrm{x}_2]\!] \rho\}$$

$$\texttt{otherwise} \rightarrow \texttt{wrong}$$

$$\texttt{endcase}$$

pHs Anweisungssemantik (2)

$$\mathcal{L}[\![x_1 \ \$! \ x_2 \ ]\!] \rho\kappa = \lambda\sigma. \bigcup_{\sigma_2 \in \Sigma_2} \kappa'(\sigma_2\ (\rho\ x_1))\ \sigma_2$$

where $\Sigma_1 =$ force $(\rho\ x_1)$ $\mathrm{id}_\kappa\ \sigma$

$\Sigma_2 = \bigcup_{\sigma_1 \in \Sigma_1}$ force $(\rho\ x_2)$ $\mathrm{id}_\kappa\ \sigma_1$

$\kappa' = \lambda\varepsilon.\lambda\sigma'.$case ε of

$\varepsilon \in \mathbf{Abs} \rightarrow \varepsilon\ (\rho\ x_2)\kappa\sigma'$

otherwise $\rightarrow$ wrong

endcase

pHs Anweisungssemantik (3)

$$\mathcal{L}[\![\text{ let } \{x_i = E_i\}_{i=1}^{n} \text{ in } x \ ]\!] \rho\kappa = \lambda\sigma.\mathcal{L}[\![x]\!]\rho'\kappa'\sigma'$$

where $\{l_1,\ldots,l_n\} =$ freeloc n σ

$\rho' = \rho \oplus \{x_1 \mapsto l_1, \ldots, x_n \mapsto l_n\}$

$\sigma' = \sigma \oplus \{l_1 \mapsto \mathcal{L}[\![E_1]\!]\rho', \ldots, l_n \mapsto \mathcal{L}[\![E_n]\!]\rho'\}$

$\kappa' = \lambda\varepsilon.\lambda\sigma''. \bigcup_{\sigma_d \in \Sigma_d} \kappa\ \varepsilon\ \sigma_d$

where $\Sigma_d =$ decls $\{x_1,\ldots,x_n\}\rho'\sigma''$

pHs Anweisungssemantik (4)

$$\mathcal{L}[\![\ \texttt{iCell x}\ ]\!]\rho\kappa = \lambda\sigma.\kappa\ \texttt{unit}\ (\sigma \oplus \{(\rho\ \texttt{x}) \mapsto \langle \texttt{I}, \texttt{empty}\rangle\})$$

$$\mathcal{L}[\![\ \texttt{mCell x}\ ]\!]\rho\kappa = \lambda\sigma.\kappa\ \texttt{unit}\ (\sigma \oplus \{(\rho\ \texttt{x}) \mapsto \langle \texttt{M}, \texttt{empty}\rangle\})$$

$$\mathcal{L}[\![\ \texttt{Fetch x}\ ]\!]\rho\kappa = \texttt{in}_{\texttt{Cont}}\ (\texttt{x}, \lambda\varepsilon.\lambda\sigma.\kappa\varepsilon\sigma')$$

where $\sigma' =$ case $\sigma\ (\rho\ \texttt{x})$ of

$\langle \texttt{I}, \varepsilon'\rangle \rightarrow \sigma$

$\langle \texttt{M}, \varepsilon'\rangle \rightarrow \sigma \oplus \{(\rho\ \texttt{x}) \mapsto \langle \texttt{M}, \texttt{empty}\rangle\}$

otherwise → wrong

endcase

pHs Anweisungssemantik (5)

$$\mathcal{L}[\![\ \texttt{Store}(\texttt{x}_1,\texttt{x}_2)\ ]\!]\rho\kappa = \mathcal{L}[\![\ \texttt{x}_2\ ]\!]\rho\kappa'$$

where $\kappa' = \lambda\varepsilon.\lambda\sigma.(\texttt{out}_{\texttt{Cont}}(\texttt{x}_1,\varepsilon,\kappa\ \texttt{unit})\sigma')$

$\sigma' =$ case $\sigma\ (\rho\ \texttt{x}_1)$ of

$\langle \texttt{I}, \texttt{empty}\rangle \rightarrow \sigma \oplus \{(\rho\ \texttt{x}_1) \mapsto \langle \texttt{I}, \varepsilon\rangle\}$

$\langle \texttt{M}, \texttt{empty}\rangle \rightarrow \sigma \oplus \{(\rho\ \texttt{x}_1) \mapsto \langle \texttt{M}, \varepsilon\rangle\}$

otherwise → wrong

endcase

Hilfsfunktionen von pH

$$\texttt{force} :: \mathbf{Loc} \to \mathbf{ECont} \to \mathbf{Cont}$$
$$\texttt{force}\ \alpha\ \kappa = \lambda\sigma.\texttt{case}\ (\sigma\ \alpha)\ \texttt{of}$$
$$\quad \varepsilon \in \mathbf{Abs} \to \kappa\ \varepsilon\ \sigma$$
$$\quad \nu \in \mathbf{Clo} \to \nu\ \kappa'\ \sigma'$$
$$\qquad \texttt{where}\ \kappa' = \lambda\varepsilon''.\lambda\sigma''.\kappa\varepsilon''\sigma'' \oplus \{l \mapsto \varepsilon''\}$$
$$\qquad\qquad \sigma' = \sigma \oplus \{l \mapsto \texttt{not\_ready}\}$$
$$\quad \texttt{otherwise} \to \texttt{wrong}$$
$$\texttt{endcase}$$

$$\texttt{decls} :: P_f(\mathbf{Ide}) \to \mathbf{Env} \to \mathbf{Cont}$$
$$\texttt{decls}\ \{\}\ \rho = \lambda\sigma.\{\sigma\}$$
$$\texttt{decls}\ I\ \rho = \lambda\sigma.\bigcup_{x \in I}\ (\bigcup_{\sigma_x \in \Sigma_x} \texttt{decls}\ (I \setminus \{x\})\rho\sigma_x$$
$$\quad \texttt{where}\ \Sigma_x = \mathcal{L}[\![\, x \,]\!]\ \rho\ \mathbf{id}_\kappa\ \sigma$$

Zusammenfassung

- Unterschiede sichtbar in den semantischen Bereiche der Sprachen
- Eden sehr komplex, da
  - verteilte Implementierung
  - Prozessbezeichner
  - Kanäle
- GpH recht einfach
  - leichte Eruierung der tatsächlichen Parallelität

Unterschied in den denotationellen Semantiken an der Anweisung „let“

- Eden:
  - Minimal: nur Auswertung der Variablen, die in Verbindung zur Prozesserstellung steht
  - Maximal: alle auftauchenden Variablen
- Gph:
  - Umgebungsaktualisierung
- pH:
  - parallele Variablenauswertung

Fazit

- Bestimmung des Grads der Parallelität ist möglich geworden
  - Eden
    - Betrachte Graphen (Knoten ≡ Prozessbezeichner, Kanten ≡ Kanäle) der minimalen und nicht-minimalen Semantik
  - pH:
    - Vergleich des genutzten Speichers
    - Unterschied minimal nicht-minimale Semantik
  - GpH:
    - Ähnlich pH, nur statt Speicherplätze im Speicher werden Variablen in der Umgebung betrachtet
- Arbeitsverdopplung leider nicht nachvollziehbar