# Funktionales Programmieren

## Teil 9

Carl Philipp Reh

Universität Siegen

24. November 2023

# Domains

Um denotationelle Semantik für Haskell-Programme formal zu definieren, müssen wir zunächst die Wertebereiche für Typen festlegen. Diese werden *Domains* genannt. Jede Domain wird eine CPO sein. Dem Basistyp `Int` ordnen wir den Domain $\mathcal{D}(\mathrm{Int}) = \mathbb{Z}_\perp$ zu.
Welche Domains sollte man Data-Deklarationen zuordnen?
Betrachten wir

```
data IntPair = Make Int Int
```

mit folgenden Beispielwerten

```
c :: IntPair
d :: IntPair
c = undefined
d = Make undefined undefined
```

Hier ist `c` komplett undefiniert, wohingegen `d` den Wertkonstruktor `Make` mit zwei undefinierten `Ints` enthält.

# Domains von Data-Deklarationen

Dass `c` und `d` sich anders verhalten, kann man gut an den folgenden beiden Funktionen erkennen:

```
f :: IntPair -> Int
f p = 0
g :: IntPair -> Int
g p = case p of Make x y -> 0
```

Wir haben, dass `f c = 0` und `f d = 0`. Ebenso haben wir `g d = 0`. Allerdings terminiert `g c` nicht, weil es sein Argument so weit auswerten muss, bis klar ist, welcher Wertkonstruktor (hier `Make`) angewandt wurde.

Wir benötigen also nicht nur $\perp$-Werte für die Argumente von Wertkonstruktoren, sondern auch noch ein zusätzliches $\perp$ für den Fall, dass kein Wertkonstruktor angewandt wurde.

# Weitere CPOs

Um die nächsten CPO-Konstruktionen etwas zu vereinfachen, formulieren wir folgendes Lemma:

## Lemma 12

*Sei $(D, \sqsubseteq_D)$ eine CPO und $c : \mathbb{N} \to D$ eine Kette.*

1. *Dann ist auch $c_k : \mathbb{N} \to D$ mit $c_k(n) = c(n+k)$ für jedes $k \in \mathbb{N}$ eine Kette und es gilt $\sqcup c_k = \sqcup c$.*
2. *Sei $(E, \sqsubseteq_E)$ eine CPO mit $E \subseteq D$, wobei $\sqsubseteq_E$ die Einschränkung von $\sqsubseteq_D$ auf $E$ ist, und sei $\mathrm{Img}(c) \subseteq E$. Dann ist auch $c_E : \mathbb{N} \to E$ mit $c_E(n) = c(n)$ eine Kette und es gilt $\sqcup c_E = \sqcup c$.*

## Beweis.

Wir haben Punkt 1 bereits für $k = 1$ in der Übung gezeigt. Für $k > 1$ funktioniert die Argumentation analog. Punkt 2 ist klar. □

# Lift einer CPO (Teil 1)

Der *Lift einer CPO* $(D, \sqsubseteq_D)$ ist $(D_\perp, \sqsubseteq_{D_\perp})$ mit $D_\perp = D \uplus \{\perp_D\}$ und $d \sqsubseteq_{D_\perp} d'$ genau dann, wenn $d = \perp_D$ oder $d, d' \in D$ mit $d \sqsubseteq_D d'$.

## Lemma 13

$(D_\perp, \sqsubseteq_{D_\perp})$ *ist eine CPO.*

## Beweis.

Reflexivität: Sei $d \in D_\perp$. Wenn $d = \perp_D$, dann gilt $d \sqsubseteq d$. Wenn $d \in D$, dann gilt $d \sqsubseteq_D d$, also auch $d \sqsubseteq d$. Antisymmetrie: Seien $d, d' \in D_\perp$ mit $d \sqsubseteq d'$ und $d' \sqsubseteq d$. Wenn $d = \perp_D$, dann gilt $d' \sqsubseteq \perp_D$ und somit muss $d' = \perp_D$ sein. Der Fall, dass $d' = \perp_D$, ist analog. Wenn $d, d' \in D$, dann gilt $d \sqsubseteq_D d'$ und $d' \sqsubseteq_D d$, also auch $d = d'$. Transitivität: Seien $d, d', d'' \in D_\perp$ mit $d \sqsubseteq d'$ und $d' \sqsubseteq d''$. Wenn $d = \perp_D$, dann gilt auch $d \sqsubseteq d''$. Wenn $d \in D$, dann gilt $d \sqsubseteq_D d'$. Daraus folgt $d' \in D$, also auch $d' \sqsubseteq_D d''$. Also gilt $d' \sqsubseteq_D d''$ und somit $d \sqsubseteq d''$.

# Lift einer CPO (Teil 2)

Beweis.

Nach Definition von $\sqsubseteq_{D_\perp}$ ist klar, dass $\perp_D$ das kleinste Element ist. Sei $c\colon \mathbb{N} \to D_\perp$ eine Kette. Wir haben zwei Fälle zu unterscheiden: Wenn $c(n) = \perp_D$ für alle $n \in \mathbb{N}$, dann ist $c'\colon \mathbb{N} \to \{\perp_D\}$ mit $c'(d) = c(d)$ eine Kette und es gilt $\sqcup c = \sqcup c'$ nach Punkt 2 von Lemma 12. Ansonsten gibt es ein $k \in \mathbb{N}$ mit $c(k) \in D$. Nach Definition von $\sqsubseteq_{D_\perp}$ muss dann für alle $k' > k$ gelten, dass $c(k') \in D$. Also ist $c'\colon \mathbb{N} \to D$ mit $c'(n) = c(n + k)$ eine Kette und es gilt $\sqcup c = \sqcup c'$ nach Punkten 2 und 1 von Lemma 12. □

# Domains von Data-Deklarationen

Wir betrachten zunächst nicht rekursive Data-Deklarationen mit nur einem Wertkonstruktor: `data TCon = VCon`. Solch ein Wertkonstruktor `VCon` ist allgemein von der Form

```
VCon = N T_1 ... T_n
```

wobei `N` ein Bezeichner ist und `T_1` bis `T_n` für $n \geq 0$ Typen sind. Werte, die mit `N` erzeugt wurden, sollen ein Element folgender Menge sein:

$$\mathcal{D}(\mathrm{VCon}) := \{\underline{\mathrm{N}}\} \times \mathcal{D}(\mathrm{T_1}) \times \cdots \times \mathcal{D}(\mathrm{T_n})$$

Die Menge $\{\underline{\mathrm{N}}\}$ enthält nur einen Wert, der uns sagt, welcher Wertkonstruktor angewandt wurde. Da wir ein zusätzliches $\bot$ benötigen für den Fall, dass kein Wertkonstruktor angewandt wurde, wählen wir für `TCon` also

$$\mathcal{D}(\mathrm{TCon}) := \mathcal{D}(\mathrm{VCon})_\bot = (\{\underline{\mathrm{N}}\} \times \mathcal{D}(\mathrm{T_1}) \times \cdots \times \mathcal{D}(\mathrm{T_n}))_\bot$$

# Domains von Data-Deklarationen

Betrachten wir wieder

```
data IntPair = Make Int Int
```

Hier haben wir also

$$\mathcal{D}(\text{IntPair}) = \mathcal{D}(\text{Make Int Int})_\perp = (\{\underline{\text{Make}}\} \times \mathbb{Z}_\perp \times \mathbb{Z}_\perp)_\perp$$

Für die vorherigen Werte

```
c = undefined
d = Make undefined undefined
```

erhalten wir dann $[\![\text{c}]\!] = \perp$ und $[\![\text{d}]\!] = (\underline{\text{Make}}, \perp, \perp)$. Für

```
e = Make 0 1
```

erhalten wir $[\![\text{e}]\!] = (\underline{\text{Make}}, 0, 1)$.

# Domains für mehrere Wertkonstruktoren

Als Nächstes betrachten wir Data-Deklarationen mit mehr als einem Wertkonstruktor. Die Einfachste von diesen ist `Bool` mit

```
data Bool = True | False
```

Wir haben $\mathcal{D}(\mathrm{True}) = \{\underline{\mathrm{True}}\}$ und $\mathcal{D}(\mathrm{False}) = \{\underline{\mathrm{False}}\}$ für die beiden Wertkonstruktoren. Um beide Werte zusammenzufügen, liften wir zunächst die Domains und erhalten $\mathcal{D}(\mathrm{True})_\perp = \{\underline{\mathrm{True}}, \perp\}$ und $\mathcal{D}(\mathrm{False})_\perp = \{\underline{\mathrm{False}}, \perp\}$. Diese werden dann über die verschmolzene Summe zusammengefügt, wo die beiden $\perp$ von $\mathcal{D}(\mathrm{True})_\perp$ und $\mathcal{D}(\mathrm{False})_\perp$ identifiziert werden. Wir werden außerdem aus technischen Gründen Elemente aus $\mathcal{D}(\mathrm{True})$ und $\mathcal{D}(\mathrm{False})$ mit „Tags“ versehen, die sicherstellen, dass nur disjunkte Mengen vereinigt werden.

# Summe

Die *Summe* von $n \geq 0$ Mengen $M_1, \ldots, M_n$ ist

$$M_1 + \cdots + M_n = (M_1 \times \{1\}) \cup \cdots \cup (M_n \times \{n\}).$$

Im Fall, dass $n = 0$, nehmen wir $\emptyset$. Die „Tags“ $1, \ldots, n$ stellen sicher, dass man für jedes Element weiß, aus welcher Menge es gekommen ist, da die Mengen $M_1, \ldots, M_n$ nicht disjunkt sein müssen. Im Gegensatz zu Summen kann man bei der Vereinigung zweier Mengen diese Information verlieren, zum Beispiel bei $\mathbb{N} \cup \mathbb{Z} = \mathbb{Z}$. Der Name Summe kommt daher, dass $|A + B| = |A| + |B|$ für alle Mengen $A$ und $B$ gilt. Wenn klar ist, aus welcher Quellmenge ein Element $(m, i) \in M_i \times \{i\}$ kommt, schreiben wir auch einfach nur $m$.

# Verschmolzene Summe

Seien $(D_1, \sqsubseteq_{D_1}), \ldots, (D_n, \sqsubseteq_{D_n})$ CPOs. Die *Verschmolzene Summe* $(D_1 \oplus \cdots \oplus D_n, \sqsubseteq_{D_1 \oplus \cdots \oplus D_n})$ ist

$$D_1 \oplus \cdots \oplus D_n = ((D_1 \setminus \{\bot_{D_1}\}) + \cdots + (D_n \setminus \{\bot_{D_n}\})) \cup \{\bot_{D_1 \oplus \cdots \oplus D_n}\},$$

wobei $d \sqsubseteq_{D_1 \oplus \cdots \oplus D_n} d'$ genau dann, wenn $d = \bot_{D_1 \oplus \cdots \oplus D_n}$, oder $d = (e, i) \in (D_i \setminus \{\bot_{D_i}\}) \times \{i\}$ und $d' = (e', i) \in (D_i \setminus \{\bot_{D_i}\}) \times \{i\}$ für $1 \leq i \leq n$ mit $e \sqsubseteq_{D_i} e'$.
Wir entfernen also alle kleinsten Elemente aus $D_1, \ldots, D_n$ und fügen ein neues kleinstes Element hinzu.

## Lemma 14

$(D_1 \oplus \cdots \oplus D_n, \sqsubseteq_{D_1 \oplus \cdots \oplus D_n})$ *ist eine CPO.*

## Beweis.

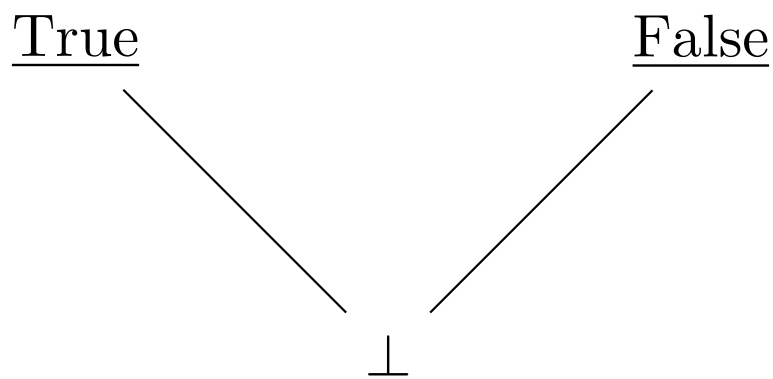
Übung. □

# Domains für mehrere Wertkonstruktoren

Im Fall von `Bool` erhalten wir also

$$\begin{aligned}\mathcal{D}(\mathrm{Bool}) &= \mathcal{D}(\mathrm{True})_\perp \oplus \mathcal{D}(\mathrm{False})_\perp \\ &= \{\underline{\mathrm{True}}, \perp\} \oplus \{\underline{\mathrm{False}}, \perp\} \\ &= (\{\underline{\mathrm{True}}\} + \{\underline{\mathrm{False}}\}) \cup \{\perp\} \\ &= \{(\underline{\mathrm{True}}, 1), (\underline{\mathrm{False}}, 2), \perp\}\end{aligned}$$

Da bei $(\underline{\mathrm{True}}, 1)$ und $(\underline{\mathrm{False}}, 2)$ klar ist, aus welcher Menge sie kamen, können wir stattdessen einfach $\underline{\mathrm{True}}$ und $\underline{\mathrm{False}}$ schreiben. Grafisch ergibt sich folgende Ordnung:



# Domains für mehrere Wertkonstruktoren

Allgemeiner sei

```
data TCon = VCon_1 | ... | VCon_n
```

wobei jedes `VCon_i` für $1 \leq i \leq n$ mit $n \geq 0$ einen Wertkonstruktor deklariert. Auch hier erlauben wir zunächst keine Rekursion. Wir ordnen `TCon` folgenden Domain zu:

$$\mathcal{D}(\mathrm{TCon}) = \mathcal{D}(\mathrm{VCon_1})_\perp \oplus \cdots \oplus \mathcal{D}(\mathrm{VCon_n})_\perp$$

Da wir im Allgemeinen nur Fälle betrachten werden, wo die Wertkonstruktornamen in den `VCon_i` verschieden sind, werden wir die Tags in der Summe – wie wir es bei $\underline{\mathrm{True}}$ und $\underline{\mathrm{False}}$ getan haben – ignorieren können.