4.2 Konstruktion von Galois-Verbindungen

(1) Definiere zwei elementare Galois-Verbindungen
(2) Definiere Galois-Verbindungen mittels Extraktionsfunktionen
(3) Komponiere bestehende Galois-Verbindungen zu neuen.

1.b) Kongruenzabstraktion

- Intervallabstraktion gibt die absolute Größe von Datenwerten an.
- Eignet sich nicht gut zum Rechnen
- Kongruenzabstraktion vergisst die absolute Größe von Datenwerten
- Eignet sich gut zum Rechnen.

Sei $L = (\mathbb{P}(\mathbb{Z}), \subseteq)$ und $M = (\mathbb{P}(\{0, \dots, k-1\}), \subseteq)$ mit $k \in \mathbb{N} \setminus \{0\}$

Dann ist

$\alpha : L \to M$ definiert als

$\alpha(Z) := \{ z \underline{\bmod k} \mid z \in Z \}$ (Beachte, definiert auf $\mathbb{Z}$: $-3 \underline{\bmod 5} = 2$)

$\gamma : M \to L$ mit

$\gamma(M') := \{ z \in \mathbb{Z} \mid z \equiv m \underline{\bmod k} \text{ für ein } m \in M' \}$

eine Galois-Verbindung

2) Galois-Verbindungen aus Extraktionsfunktionen

- Sei $\beta : V \to D$ eine Funktion.

Dann ist

$(\alpha_\beta, \gamma_\beta)$ mit $\alpha_\beta : \mathbb{P}(V) \to \mathbb{P}(D)$

$\alpha_\beta(V') := \{ \beta(v) \mid v \in V' \} = \beta(V')$

und $\gamma_\beta : \mathbb{P}(D) \to \mathbb{P}(V)$

$\gamma_\beta(D') := \{ v \in V \mid \beta(v) \in D' \} = \beta^{-1}(D')$

eine Galois-Verbindung.

Die oben definierte Kongruenzabstraktion ergibt sich aus der Extraktionsfunktion

$z \mapsto z \underline{\bmod k}$.

- Oft ist $\mathbb{P}(V) = \mathbb{P}(State)$ mit $State = B^{Vars}$
Ist nun $\beta: B \to D$ als Extraktionsfunktion bekannt, ergibt sich eine Abstraktionsfunktion für ganz $\mathbb{P}(State)$.

<u>Definition (Liften von Extraktionsfunktionen):</u>
Sei $State = B^{Vars}$ die Menge der Variablenbelegungen σ und sei $\beta: B \to D$ eine Extraktionsfunktion.
Durch <u>Liften von β auf State</u> entsteht die Abstraktion

$$\alpha: \underset{\mathbb{P}(B^{Vars})}{\underset{\shortparallel}{\mathbb{P}(State)}} \longrightarrow \mathbb{P}(D^{Vars})$$

mit

$$\alpha(State') := \{ \beta \circ \sigma \mid \sigma \in State' \}$$

Es lässt sich zeigen, dass α vollständig additiv und damit Teil einer Galois-Verbindung ist.

3) Komposition von Galois-Verbindungen

3.1) Sequentielle Komposition:

Seien (α_1, γ_1), (α_2, γ_2) mit $L_1 \underset{\gamma_1}{\overset{\alpha_1}{\rightleftarrows}} L_2$ und $L_2 \underset{\gamma_2}{\overset{\alpha_2}{\rightleftarrows}} L_3$ Galois-Verbindungen.
Dann ist die <u>sequentielle Komposition</u>
$$(\alpha_2, \gamma_2) \circ (\alpha_1, \gamma_1) := (\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2)$$
eine Galois-Verbindung zwischen L_1 und L_3.

3.2) Parallelkomposition:

- Führe zwei Abstraktionen auf demselben Element aus.
- Präzisere Analyse
- Als Beispiel: Intervallabstraktion (absolute Größe) + Kongruenzabstraktion (Arithmetik)
- Zwei Produkte

3.2.a) Direktes Produkt

- In der Literatur auch unabhängige Attribute Methode genannt
- Idee: Führe zwei Abstraktionen unabhängig voneinander aus.
- Vorteil: schnell
- Nachteil: unpräzise.

Seien (α_i, γ_i) mit $i=1,2$ und $\alpha_i : L \to M_i$
sowie $\gamma_i : M_i \to L$

Galois-Verbindungen.
Dann ist das direkte Produkt

$$(\alpha_1, \gamma_1) \times (\alpha_2, \gamma_2) := (\alpha, \gamma)$$
$$\text{mit } \alpha : L \longrightarrow M_1 \times M_2$$
$$\alpha(l) := (\alpha_1(l), \alpha_2(l))$$
$$\gamma : M_1 \times M_2 \longrightarrow L$$
$$\gamma(m_1, m_2) := \gamma_1(m_1) \sqcap \gamma_2(m_2)$$

wieder eine Galois-Verbindung.

3.2.b) Tensorprodukt

- In der Literatur auch relationale Methode genannt.
- Nur definiert auf Potenzmengenverbänden.
- Idee:
  - Wende die Abstraktionsfunktionen auf einzelne Elemente an.
  - Bilde anschließend Vereinigung der Ergebnisse.
  - So bleibt die Relation zwischen den Abstraktionen auf dem Datenbereich erhalten.

Seien (α_i, γ_i) mit $i=1,2$ und $\alpha_i : \mathbb{P}(V) \to \mathbb{P}(D_i)$
sowie $\gamma_i : \mathbb{P}(D_i) \to \mathbb{P}(V)$

Galois-Verbindungen.
Dann ist das Tensorprodukt

$$(\alpha_1, \gamma_1) \otimes (\alpha_2, \gamma_2) := (\alpha, \gamma)$$
$$\text{mit } \alpha : \mathbb{P}(V) \longrightarrow \mathbb{P}(D_1 \times D_2)$$
$$\gamma : \mathbb{P}(D_1 \times D_2) \longrightarrow \mathbb{P}(V)$$

definiert durch

$$\alpha(V') := \bigcup \{ \alpha_1(\{v\}) \times \alpha_2(\{v\}) \mid v \in V' \}$$

und

$$\gamma(D') := \{ v \in V \mid \alpha_1(\{v\}) \times \alpha_2(\{v\}) \subseteq D' \}$$

wieder eine Galois-Verbindung.

<u>Beispiel</u>:

- Um den Unterschied zwischen dem direkten Produkt und dem Tensorprodukt zu verdeutlichen, betrachte die <u>Extraktionsfunktionen</u>:

$$sign : \mathbb{Z} \to \{-, 0, +\}$$

$$sign(z) := \begin{cases} -, & \text{falls } z < 0 \\ 0, & \text{falls } z = 0 \\ +, & \text{falls } z > 0 \end{cases}$$

$$parity : \mathbb{Z} \to \{even, odd\}$$

$$parity(z) := \begin{cases} even, & \text{falls } z \text{ gerade} \\ odd, & \text{falls } z \text{ ungerade.} \end{cases}$$

- Die Funktionen induzieren die Galois-Verbindungen

$$(\alpha_{sign}, \gamma_{sign}) \text{ und } (\alpha_{parity}, \gamma_{parity})$$

zwischen $\mathcal{P}(\mathbb{Z})$ und $\mathcal{P}(\{-, 0, +\})$ bzw. $\mathcal{P}(\{even, odd\})$.

- Betrachte $\{3, -4\} \subseteq \mathbb{Z}$.
- Auf diese Menge liefert das <u>direkte Produkt</u>

$$(\alpha_{sign}, \gamma_{sign}) \times (\alpha_{parity}, \gamma_{parity}) = (\alpha_{imprecise}, \gamma_{imprecise})$$

mit

$$\alpha_{imprecise}(\{3, -4\}) = \{+, -\} \times \{odd, even\}$$
$$= \{(+, odd), (+, even), (-, odd), (-, even)\}$$

- Das <u>Tensorprodukt</u> hingegen liefert

$$(\alpha_{sign}, \gamma_{sign}) \otimes (\alpha_{parity}, \gamma_{parity}) = (\alpha_{precise}, \gamma_{precise})$$

mit

$$\alpha_{precise}(\{3, -4\}) = (\{+\} \times \{odd\}) \cup (\{-\} \times \{even\})$$
$$= \{(+, odd), (-, even)\}.$$

3.3 Komponentenweise Kombination

- Oft ist die Zustandsmenge eines Programms als kartesisches Produkt vollständiger Verbände gegeben (Verschiedene Variablen.)
- Sind nun auf den Komponenten Galois-Verbindungen definiert, lassen sich diese zu neuen Galois-Verbindungen für das kartesische Produkt verknüpfen.
- Wieder zwei Produkte, analog zu obigen Definitionen

3.3.a) Direktes Produkt

Seien (α_i, γ_i) mit $i = 1,2$ und $\alpha_i : L_i \to M_i$
sowie $\gamma_i : M_i \to L_i$

Galois-Verbindungen.
Dann ist das <u>direkte</u> <u>Produkt</u>

$$(\alpha_1, \gamma_1) \hat{\times} (\alpha_2, \gamma_2) := (\alpha, \gamma)$$

mit

$$\alpha : L_1 \times L_2 \to M_1 \times M_2$$
$$\alpha(l_1, l_2) := (\alpha_1(l_1), \alpha_2(l_2))$$
$$\gamma : M_1 \times M_2 \to L_1 \times L_2$$
$$\gamma(m_1, m_2) := (\gamma_1(m_1), \gamma_2(m_2))$$

wieder eine Galois-Verbindung.

3.3.b) Tensorprodukt

Seien (α_i, γ_i) mit $i = 1,2$ und $\alpha_i : \mathbb{P}(V_i) \to \mathbb{P}(D_i)$
sowie $\gamma_i : \mathbb{P}(D_i) \to \mathbb{P}(V_i)$

Galois-Verbindungen.
Dann ist das <u>Tensorprodukt</u>

$$(\alpha_1, \gamma_1) \hat{\otimes} (\alpha_2, \gamma_2) := (\alpha, \gamma)$$

mit

$$\alpha : \mathbb{P}(V_1 \times V_2) \longrightarrow \mathbb{P}(D_1 \times D_2)$$
$$\alpha(V) := \bigcup \{ \alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \mid (v_1, v_2) \in V \}$$
$$\gamma : \mathbb{P}(D_1 \times D_2) \longrightarrow \mathbb{P}(V_1 \times V_2)$$
$$\gamma(D) := \{ (v_1, v_2) \in V_1 \times V_2 \mid \alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \subseteq D \}$$

wieder eine Galois-Verbindung.

4.3 Konkrete Semantik von while-Programmen

<u>Ziel</u>: Definiere operationelle Semantik von while-Programmen.

<u>Wiederholung</u>: <u>Strukturierte operationelle Semantik</u> (SOS, Plotkin '81)
(FGDP)

<u>Idee von SOS</u>:
- <u>Konfigurationen</u> eines Programms haben (syntaktische) <u>Struktur</u>
  ↳ Komposition atomarer Elemente mittels einer Menge von Operatoren.
- Damit lassen sich <u>Beweissysteme (Kalküle)</u> nutzen, um das Verhalten von Konfigurationen zu definieren
  ↳ Transition existiert gdw. sie im Beweissystem herleitbar ist.
- Technisch nutzt das Beweissystem <u>Induktion nach der Struktur</u> von Konfigurationen:
  ↳ <u>Axiome</u> definieren die Transitionen <u>atomarer Elemente</u>
  ↳ <u>Beweisregeln</u> definieren die Transitionen zusammengesetzter Konfigurationen über die Transitionen der Operanden.

<u>Vorteile</u>:
- Einfachheit und Eleganz
- Möglichkeit, Eigenschaften von Transitionen über Induktion entlang der Ableitung zu beweisen.

<u>Wiederholung (Syntax von while-Programmen)</u>:

$a ::= k \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$

$b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2$

$c ::=$ <u>skip</u> $\mid x := a$
$\mid c_1 ; c_2 \mid$ if b <u>then</u> c_1 <u>else</u> c_2 fi $\mid$ <u>while</u> b <u>do</u> c <u>od</u>

mit $k \in \mathbb{Z}$, $x \in Vars$, $t \in \mathbb{B}$.

- Sei $Prog$ die <u>Menge aller Programme</u> c.
- Sei $Sig = (Funk, Prad)$ mit $Funk = \{+/_2, -/_2, */_2\} \cup \{k/_0 \mid k \in \mathbb{Z}\}$ und $Prad = \{>/_2\}$ die (logische) <u>Signatur</u> der Programme.

- Die Semantik ordnet jedem syntaktischen Ausdruck eine Bedeutung zu.
Dabei ist die Bedeutung ein Element eines semantischen Bereichs.
- Formal ist der semantische Bereich gegeben als (logische) <u>Sig-Struktur</u>

$$\mathcal{S} = (D, I)$$

mit
↳ <u>Datenbereich</u> (auch <u>Domäne</u>) D (einer Menge von Elementen) und
↳ <u>Interpretation</u> I, die jedem Funktionssymbol $f/n \in Funk$ eine tatsächliche Funktion

$$I(f) : D^n \to D \quad \text{zuordnet,}$$

und die jedes Prädikatsymbol $p/n \in Prad$ als tatsächliches Prädikat

$$I(p) : D^n \to \mathbb{B}$$

auffasst.
Man schreibt $I(f)$ und $I(p)$ auch als f_I und p_I.
- Für die oben definierten while-Programme mit Signatur Sig wird

$$\mathcal{S} = (\mathbb{Z}, I) \text{ mit } k_I, +_I, *_I, -_I \text{ und } >_I \text{ wie erwartet}$$

gewählt.
- Das Verhalten des Programms hängt von der <u>Belegung der Variablen</u>, dem <u>Zustand</u>, ab:

$$\sigma \in State := \mathbb{Z}^{Vars}.$$

- Gegeben ein Zustand σ, lässt sich die Semantik von arithmetischen (a) und Booleschen (b) Ausdrücken als $\mathcal{S}[\![\underset{b}{a}]\!](\sigma)$ (wie in der Logik, dort Terme und Formeln genannt) festlegen.

Definition (Transitionsrelation für Konfigurationen):

- Eine Konfiguration ist ein Paar $(c, \sigma) \in Prog \times State$ bestehend aus einem Programm $c \in Prog$ und einem Zustand $\sigma \in State = \mathbb{Z}^{Vars}$.
- Die Transitionsrelation zwischen Konfigurationen

$$\rightarrow \;\subseteq (Prog \times State) \times ((Prog \times State) \cup State)$$

ist die kleinste Relation, die folgenden Regeln genügt:

$$(\text{skip}) \frac{}{(\underline{\text{skip}}, \sigma) \rightarrow \sigma} \qquad (\text{assign}) \frac{}{(x := a, \sigma) \rightarrow \sigma[x := \mathcal{S}[\![a]\!](\sigma)]}$$

$$(\text{seq1}) \frac{(c_1, \sigma) \rightarrow \sigma'}{(c_1; c_2, \sigma) \rightarrow (c_2, \sigma')} \qquad (\text{seq2}) \frac{(c_1, \sigma) \rightarrow (c_1', \sigma')}{(c_1; c_2, \sigma) \rightarrow (c_1'; c_2, \sigma')}$$

$$(\text{iftrue}) \frac{}{(\text{if } b \text{ \underline{then} } c_1 \text{ \underline{else} } c_2 \text{ \underline{fi}}, \sigma) \rightarrow (c_1, \sigma)}, \text{ falls } \mathcal{S}[\![b]\!](\sigma) = \text{true}.$$

$$(\text{iffalse}) \frac{}{(\text{if } b \text{ \underline{then} } c_1 \text{ \underline{else} } c_2 \text{ \underline{fi}}, \sigma) \rightarrow (c_2, \sigma)}, \text{ falls } \mathcal{S}[\![b]\!](\sigma) = \text{false}.$$

$$(\text{whiletrue}) \frac{}{(\underline{\text{while}}\ b \text{ \underline{do} } c \text{ \underline{od}}, \sigma) \rightarrow (c; \underline{\text{while}}\ b \text{ \underline{do} } c \text{ \underline{od}}, \sigma)}, \text{ falls } \mathcal{S}[\![b]\!](\sigma) = \text{true}$$

$$(\text{whilefalse}) \frac{}{(\underline{\text{while}}\ b \text{ \underline{do} } c \text{ \underline{od}}, \sigma) \rightarrow \sigma}, \text{ falls } \mathcal{S}[\![b]\!](\sigma) = \text{false}.$$

Beachte:

- Die Definition der Transitionsrelation nutzt Regelschemata der Form

$$\frac{\text{Prämisse}}{\text{Schluss}}.$$

Wenn die Prämisse erfüllt ist, dann kann der Schluss gezogen werden.

- Regeln ohne Prämisse heißen Axiome.
- Wiederholte Anwendung der Regeln ergibt Ableitungsbaum
  ↳ Axiome = Blätter ↳ Programm + Zustand = Wurzel.