Universität Duisburg-Essen
Ingenieurwissenschaften / Informatik
Dozent: Prof. Dr. Barbara König
Übungsleitung:

WS 2018/19
28. November 2018
Übungsblatt 4

Modellierung, Analyse, Verifikation (Programmanalyse)

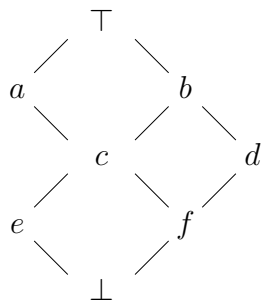
Aufgabe 13 Irreduzible Elemente

Sei $(L, \sqsubseteq)$ ein vollständiger Verband. Ein Element $\ell \in L$ heißt *irreduzibel*, wenn $\ell \neq \bot$ und außerdem folgendes gilt:

für alle Elemente $a, b \in L$ folgt aus $\ell = a \sqcup b$, dass $\ell = a$ oder $\ell = b$ gilt.

(a) Bestimmen Sie die irreduziblen Elemente des Potenzmengenverbandes $(\mathcal{P}(M), \subseteq)$ (für eine Menge M).

(b) Bestimmen Sie die irreduziblen Elemente in folgendem Verband, der durch ein Hasse-Diagramm gegeben ist:



(c) Überlegen Sie sich wie man die irreduziblen Elemente in einem endlichen Verband einfach charakterisieren kann.

Tipp: Die Charakterisierung hat etwas mit der Anzahl der direkten Vorgänger im Hasse-Diagramm zu tun.

(d) In einem endlichen Verband kann jedes Element als Supremum von irreduziblen Elementen dargestellt werden. Überprüfen Sie, dass dies auch für den Verband aus Teilaufgabe (b) gilt und stellen Sie jedes Element als Supremum von irreduziblen Elementen dar.

Aufgabe 14 Varianten des Worklist-Algorithmus

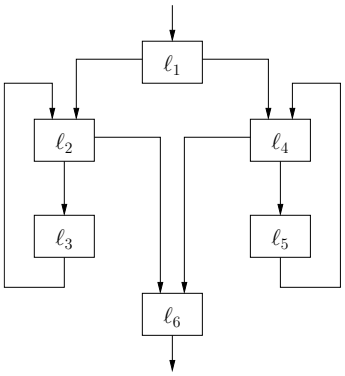
Es spielt eine große Rolle in welcher Reihenfolge der Worklist-Algorithmus die Paare der Flussrelation F abarbeitet. Es gibt in dieser Hinsicht verschiedene Optimierungsmöglichkeiten, von denen im Folgenden zwei betrachtet werden.

Wir nehmen an, dass es im Flussgraph einen Block r (die Wurzel) gibt, von dem alle anderen Blöcke aus erreichbar sind. Außerdem nehmen wir an, dass der Flussgraph genau n Blöcke enthält.

(a) Eine Idee ist, einen gerichteten Spannbaum des Flussgraphen mit Wurzel r zu finden und dessen Knoten in *Reverse Postorder* zu ordnen, d.h., jedem Knoten bzw. Block ℓ wird eine eindeutige Zahl $n(\ell) \in \{1, \ldots, n\}$ zugeordnet, so dass folgende Eigenschaft erfüllt ist:

- Falls (ℓ, ℓ') eine *Baumkante* ist, d.h., Teil des Spannbaums ist, so gilt $n(\ell) < n(\ell')$.
- Falls (ℓ, ℓ') eine *Vorwärtskante* ist, d.h., ℓ' ist mittelbarer Nachfolger von ℓ im Spannbaum, so gilt ebenfalls $n(\ell) < n(\ell')$.
- Falls (ℓ, ℓ') eine *Rückwärtskante* ist, d.h., ℓ ist mittelbarer Nachfolger von ℓ' im Spannbaum, so stellen wir keine Anforderungen.
- In allen anderen Fällen, d.h., bei sogenannten *Cross-Kanten* (ℓ, ℓ'), muss auch $n(\ell) < n(\ell')$ gelten.

1) Betrachten Sie den folgenden Flussgraph. Bestimmen Sie einen Spannbaum, dessen Knoten in Reverse Postorder nummeriert sind.



2) Angenommen, eine solche Reverse Postorder ist gegeben. Wie sollte man dann die Worklist abarbeiten, um möglichst wenig Schritte machen zu müssen?

3) Modifizieren Sie das Verfahren der Tiefensuche, um für einen gegebenen Flussgraph einen Spannbaum mit einer Nummerierung in Reverse Postorder zu erhalten.

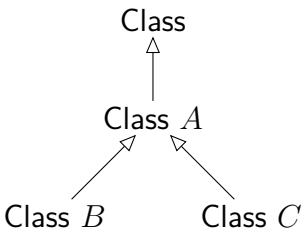
(b) Ein *stark zusammenhängender Subgraph* ist eine Menge von Knoten, so dass es von jedem Knoten einen Pfad zu jedem anderen Knoten gibt. Eine *starke Zusammenhangskomponente* ist ein maximaler stark zusammenhängender Subgraph.

1) Bestimmen Sie die starken Zusammenhangskomponenten des Graphen aus der vorherigen Teilaufgabe.

2) Wie kann man starke Zusammenhangskomponenten nutzen, um den Worklist-Algorithmus zu beschleunigen?

Aufgabe 15 *Suprema und Infima beim Java Bytecode Verifier*

Die vom Java Bytecode Verifier verwendete Hierarchie sei von folgender Form:



(a) Gegeben seien drei Elemente des Verbandes des Java Bytecode Verifiers $(T, \sqsubseteq_T)$. Geben sie für jedes Paar (t_1, t_2), bestehend aus zwei der drei Elemente, an, ob $t_1 \sqsubseteq_T t_2$ und ob $t_2 \sqsubseteq_T t_1$ gilt.

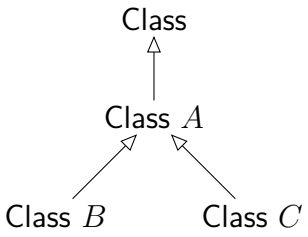
- ([NullT, Integer], [Class B, Class])
- ([Class A, Integer], [Class A, $Undef$])
- ([NullT, Integer], [NullT, $Undef$])

(b) Bilden Sie jeweils das Supremum und das Infimum für folgende Paare von Verbandselementen:

- {([Class], [Integer, Class A]), ([Class A], [NullT, Class])}
- {([Class B], [Integer]), ([Class C], [Integer])}
- {([Integer], [Class A]), ([NullT], [Class B])}

Aufgabe 16 Java Bytecode

Die folgenden Java-Bytecode-Methoden sind beide fehlerhaft. Versuchen Sie jeweils herauszufinden, warum dies der Fall ist. Die Klassenhierarchie sieht folgendermaßen aus:



Nehmen Sie außerdem an, dass (nur) die Klasse C ein Feld f vom Typ Integer besitzt.

Im 0-ten Register liegt zu Beginn eine Referenz auf ein Objekt der Klasse A, im 1-ten Register eine Referenz auf ein Object der Klasse B. Die restlichen Register sind nicht initialisiert.

Der Rückgabewert soll bei beiden Methoden vom Typ Integer sein und beim Rücksprung als oberstes Element auf dem Stack liegen.

(a)

```
1: IConst 3
2: IConst 5
3: IAdd
4: Store 0
5: Load 0
6: IConst 8
7: CmpEq 1
8: Return
```

(b)

```
1: Load 0
2: Getfield f Integer C
3: Return
```