4.2 Verknüpfung der Programme zur Klassifikation der Gitter

Die Klassifikation der Sekundärkegel n-dimensionaler Delone-Zerlegungen ist durch eine Suche nach erreichbaren Knoten in einem Baum realisiert.

Dabei sind die Knoten des Baumes die Sekundärkegel der n-dimensionalen Delone-Zerlegungen und ein Knoten u ist Kind eines anderen Knoten v, wenn der Sekundärkegel u Facette des Sekundärkegels v ist. Die erste Generation besteht aus allen Sekundärkegeln von Delone-Triangulierungen.

Beispiel 4.2.1. Werden die Kegel durch ihre Kantenstrahlen beschrieben, so sieht der Baum für Dimension 2 wie in Abb. 4.1 aus (siehe auch Bsp. in Abschnitt 4.1):

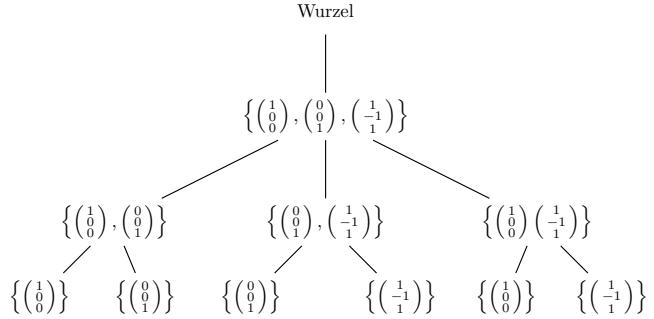

Abbildung 4.1: Baum zur Klassifikation in Dimension 2

Die Schichten in diesem Baum entsprechen den Dimensionen der Seiten. In der ersten Generation sind die Kegel volldimensional, haben also Dimension $\binom{n+1}{2}$, in der nächsten Generation sind alle Facetten (also alle $\binom{n+1}{2} - 1$ dimensionalen Kegel) usw.

Ein Knoten heißt *erreichbar*, wenn er in $\mathcal{S}^n_{>0}$ liegt (insbesondere nicht auf dem positiv semidefiniten Rand des Abschlusses), sein Vater erreichbar ist und er zu keinem Knoten aus seiner Generation äquivalent ist. Die Wurzel ist erreichbar.

Beispiel 4.2.2. In Beispiel 4.2.1 sind $\left\{\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\-1\\1\end{pmatrix}\right\}$ und $\left\{\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}\right\}$ erreichbar.
$\left\{\begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\-1\\1\end{pmatrix}\right\}$ und $\left\{\begin{pmatrix}1\\0\\0\end{pmatrix} \begin{pmatrix}1\\-1\\1\end{pmatrix}\right\}$ sind jeweils äquivalent zu $\left\{\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}\right\}$.
Die Blätter liegen alle auf dem positiv semidefiniten Rand und sind damit nicht erreichbar.

Die nicht-äquivalenten Sekundärkegel 2-dimensionaler Delone-Zerlegungen sind also gegeben durch $\left\{\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\-1\\1\end{pmatrix}\right\}$ und $\left\{\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}\right\}$.

Da es mehr unerreichbare Knoten als erreichbare gibt (vgl. Bsp. 4.2.2), ist es nicht sinnvoll, zuerst den gesamten Baum zu konstruieren und dann die erreichbaren Knoten zu suchen. Es wird gleich bei der Konstruktion des Baumes auf Erreichbarkeit geprüft. Da die Erreichbarkeit nicht von den Knoten in den folgenden Dimensionen, sondern nur vom Vater, dem Knoten selbst und allen anderen Knoten der eigenen Generation abhängt, wird der Baum von der Wurzel beginnend konstruiert. Zur Berechnung der Kinder wird das Programm `lrs` aus Abschnitt 4.1 und zum Test auf Erreichbarkeit das Programm `ISOM` aus Abschnitt 4.1 verwendet.

4.3 Implementierung in Haskell

Zur Implementierung wurde die Programmiersprache Haskell benutzt (`http://www.haskell.org/haskellwiki/Haskell` [1]). Das Programm besteht aus 6 Modulen, deren Hauptfunktionen im folgenden beschrieben und am Beispiel aus Abschnitt 4.1 veranschaulicht werden.

Eine ausführliche Programmdokumentation, das Programm und die Installationsbeschreibung findet man unter `http://www.math.uni-rostock.de/~waldmann/`.

Datentypen

Zur internen Speicherung der Kegel werden folgende Datentypen benötigt:

- Sekundärkegel haben die Attribute: Dimension (`scdim`) des Kegels, Anzahl der Ungleichungen (`scung`) und Koeffizienten der Ungleichungen (`scmat`), die den Kegel beschreiben.

  ```
  data SC = SC { scdim :: Int, scung :: Int,
                 scmat :: [[Int]] }
  ```

  Der Kegel aus Abschnitt 4.1 wird als

  ```
  SC { 3, 3, [[0, -2, 0], [0, 2, 2], [2, 2, 0]] }
  ```

  gespeichert.

- Die H-Repräsentation eines Polyeders besteht aus einer Anzahl an Unbekannten (`unb`), einer Anzahl an Ungleichungen (`ung`) und den Koeffizienten der Ungleichungen (`hrmat`).

  ```
  data HR = HR { unb :: Int, ung :: Int, hrmat :: [[Int]] }
  ```

[1] letzter Aufruf: 13.7.2014

Die H-Repräsentation aus Abschnitt 4.1 wird als

```
HR { 4, 3, [[0, 0, -2, 0], [0, 0, 2, 2], [0, 2, 2, 0]] }
```

gespeichert.

Objekte des Typs `HR` können als Eingabe für `lrs` (siehe 4.1) als

```
H-representation
begin
3 4 rational
0 0 -2 0
0 0 2 2
0 2 2 0
end
```

ausgegeben werden.

- V-Repräsentationen eines Polytops bestehen aus der Dimension (`dim`) des Polytops, einer Anzahl an Ecken (`vr_nvert`), der Dimension (`vr_dim`) des Raumes, in dem sich das Polytop befindet und den Ecken (`vr_vert`).

  ```
  data VR = VR { dim :: Int, vr_nvert :: Int,
                 vr_dim :: Int, vr_vert :: [[Int]] }
  ```

  Die V-Repräsentation der Facette 4 (`F#4`) aus Abschnitt 4.1 wird als

  ```
  VR { 2, 3, 4, [[1, 0, 0, 0], [0, 1, 0, 0], [0, 1, -1, 1]] }
  ```

  gespeichert.

  Objekte des Typs `VR` können als Eingabe für `lrs` (siehe 4.1) als

  ```
  V-representation
  begin
  3 4 rational
  1 0 0 0
  0 1 0 0
  0 1 -1 1
  end
  incidence
  ```

  ausgegeben werden.

- Gram-Matrizen werden wie in Kapitel 2.1 beschrieben als untere Dreiecksmatrizen gespeichert. Sie haben eine Größe (`tm_dim`), ihre Einträge (`tm_inh`), ihre Determinante (`tm_det`) und eine Auskunft, ob sie positiv definit ist oder nicht.

  ```
  data TM = TM { tm_dim :: Int, tm_inh :: [[Int]],
                 tm_det :: Int, tm_posdef :: Bool }
  ```

  Die Matrix A aus Abschnitt 4.1 wird als

  ```
  TM { 2, [[2], [-1,2]], 3, True }
  ```

  gespeichert.

  Objekte des Typs `TM` können als Eingabe für `ISOM` (siehe 4.1) als

  ```
  2x0
  2
  -1 2
  ```

  ausgegeben werden.

Main - Hauptprogramm

Im Hauptprogramm ist die Suche nach erreichbaren Knoten aus Abschnitt 4.2 implementiert. Die Kegel werden dabei als V-Repräsentationen gespeichert, weil man daraus die Facetten und die Gram-Matrizen gut berechnen kann.

- `main_for_dim :: Maybe Int -> IO ()`
  Dies Funktion startet die Suche nach erreichbaren Knoten mit der ersten Generation. Dazu werden die Sekundärkegel der Delone-Triangulierungen in der übergebenen Dimension (erste Generation) berechnet (`get_sccs`). Ist keine Dimension angegeben müssen die `*.coop`-Dateien im Ordner vorhanden sein (siehe dazu `get_sccs` (4.3)).

- `enum_faces :: S.Set VR -> [VR] -> IO [VR]`
  Hier wird aus der Menge der bisher berechneten aber noch nicht überprüften Typen (`todo`) und der Liste der bisher gefundenen Typen (`done`) eine neue Liste der gefundenen Typen erstellt.
  Ist `todo` leer, so ist die Klassifikation beendet und `done` ist die Liste aller Typen.
  Ist `todo` nicht leer, wird für ein $t \in$ `todo` geprüft, ob es neu, also nicht äquivalent zu einem Element aus `done`, ist.
  Ist t nicht neu, fahre mit `todo` $\setminus \{t\}$ fort.
  Ist t neu werden die Kinder von t (siehe `children` (4.3)) zu `todo` und

t zu `done` hinzugefügt. Mit diesen neuen `todo` und `done` wird fortgefahren.

Der erste Aufruf lautet:
`enum_faces (Sekundärkegel von Delone-Triangulierungen) []`

Das Programm wird mit `dvlatclass [n]` aufgerufen, um die Klassifikation in Dimension `n` auszuführen. Wird keine Dimension angegeben, müssen im Ordner des Aufrufs bereits `*.coop`-Dateien der Sekundärkegel der Triangulierungen liegen (siehe `main_for_dim`).

Scc - Berechnung der ersten Generation

In diesem Modul gibt es Funktionen, die zur Berechnung der Sekundärkegel von Delone-Triangulierungen benötigt werden. Dazu gehören:

- `get_sccs :: Maybe Int -> IO [VR]`
  Diese Funktion berechnet die V-Repräsentationen der von `scc` in den `*.coop`-Dateien gefundenen Sekundärkegel.
  Wird eine Dimension `N` übergeben, so wird `scc -dN -w` aufgerufen und die Sekundärkegel berechnet.
  Sind die `*.coop`-Dateien der Sekundärkegel schon im aktuellen Ordner vorhanden, so muss kein Argument angegeben werden. Dann wird `scc` nicht mehr ausgeführt, sondern mit den schon vorhandenen Files gearbeitet.
  Für jede `*.coop`-Datei wird der Sekundärkegel ausgelesen (`cone`), diese Darstellung in eine H-Repräsentation umgewandelt (`sc2hr`), `lrs` für diese H-Repräsentation aufgerufen (`lrs` (4.3)) und aus der Ausgabe die V-Repräsentation ausgelesen (`getvr` (4.3)).

- `cone :: Parser SC`
  Aus einem String (einer Datei) werden die Attribute eines Sekundärkegels ausgelesen und als Objekt des Typs `SC` gespeichert.

- `sc2hr :: SC -> HR`
  Es wird die Darstellung eines Sekundärkegels von `scc` durch Anfügen einer 0-Spalte wegen homogener Koordinaten (vgl. Abschnitt 4.1) in eine H-Repräsentation für `lrs` umgewandelt.

Children - Berechnung der Kinder eines Knotens

Hier sind Funktionen zur Berechnung der Kinder (Facetten) eines Knotens (Kegels) mit `lrs` implementiert.

- `children :: VR -> IO [VR]`
  Es werden die Facetten eines Kegels mittels `lrs` und `getfcs` berechnet.

- `lrs :: Either VR HR -> IO String`
  Diese Funktion ruft `lrs` für eine V- oder eine H-Repräsentation auf und liefert die Ausgabe von `lrs`.

- `getfcs :: String -> VR -> [VR]`
  Es werden mit der Option `incidence` aus der Ausgabe von `lrs` die Facetten einer V-Repräsentation bestimmt.
  Dazu wird zunächst eine Liste von Indexlisten ausgelesen. (Jede Facette wird durch eine Liste von Indices der Ecken, die auf der Facette liegen, beschrieben.) Dann wird aus den Indexlisten die jeweilige Facette bestimmt und eine Liste aller Facetten zurückgegeben (vgl. Abschnitt 4.1).

Isom - Prüfen auf Äquivalenz

Mit diesem Modul können zwei V-Repräsentationen auf Äquivalenz überprüft werden.

- `disom :: VR -> VR -> IO Bool`
  Hier werden erst die Invarianten (siehe Abschnitt 3.2) der beiden V-Repräsentationen (Kegel) verglichen. Sind alle Invarianten gleich wird die arithmetische Äquivalenz der dargestellten Gram-Matrizen (Schwerpunkt siehe 3.2) überprüft.
  Die Rückgabe lautet `True`, wenn die Gram-Matrizen arithmetisch äquivalent sind, oder eine Matrix auf dem positiv semidefiniten Rand von $\mathcal{S}^n_{\geq 0}$ liegt.
  Sind die V-Repräsentationen (bzw. die Gram-Matrizen) nicht arithmetisch äquivalent, wird `False` zurückgegeben.

- `isom_pipe :: VR -> VR -> IO Bool`
  Es werden mit `ISOM` zwei V-Repräsentationen auf Äquivalenz überprüft.
  Dazu werden die Gram-Matrizen der V-Repräsentationen berechnet und als Eingabe für `ISOM` verwendet.
  Die Ausgabe von `ISOM` wird gelesen und genau dann `True` geliefert, wenn eine Abbildungs-Matrix zwischen den Gram-Matrizen berechnet wurde.

GetVR - Berechnungen mit V- und H-Repräsentationen

In diesem Modul werden Ein- und Ausgaben für `lrs` behandelt.

- `getvr :: String -> IO VR`
  Die Funktion liest aus einer Ausgabe von `lrs` die Attribute einer V-Repräsentation aus und speichert sie als Objekt des Typs `VR`.

VR2TM - Berechnungen mit Dreiecksmatrizen

Diese Modul enthält Funktionen zur Umwandlung von V-Repräsentationen in Gram-Matrizen.

- `vr2tm :: VR -> TM`
  Aus einer V-Repräsentation wird die entsprechende Gram-Matrix berechnet.
  Der Schwerpunkt der Ecken wird bestimmt, die Determinante berechnet und auf positive Definitheit geprüft. Gemeinsam wird dies als Objekt des Typs `TM` gespeichert.

Parallelisierung

Mit wenig Aufwand kann die Parallelisierbarkeit der Hardware ausgenutzt werden, um einen Zeitvorteil bei der Ausführung des Programms zu erlangen.

Der Test auf Äquivalenz des aktuell betrachteten Typen zu den bisher gefundenen kann parallel ausgeführt werden. Statt nacheinander mit den bisher bekannten Typen zu vergleichen, wird die Liste der bekannten in Blöcke geteilt, in denen die Vergleiche parallel stattfinden.

Die Größe der Blöcke ist in der Funktion `parallel_and` im Wert `blocksize` gespeichert und kann beliebig geändert werden.

Mit `dvlatclass [n] +RTS -Nk` kann die Anzahl `k` der zu verwendenden Kerne zum parallelen Vergleich auf Äquivalenz angegeben werden.

Das Programm wurde auf einem Rechner mit 24 Kernen des Typs `Intel(R) Xeon(R) X5650` und 118GB RAM mehrfach ausgeführt.

Für die Dimensionen 2, 3 und 4 kommt das Programm in weniger als einer Sekunde zu den bisher bekannten Lösungen (vgl. [Val03]). Hierbei ist der Nutzen der Parallelität nicht zu erkennen.

Mehr Kerne und größere Blockgrößen nutzen zwar die Parallelität besser aus, führen aber auch zu höherem Verwaltungsaufwand.

Die komplette Ausführung des Programms für Dimension 5 mit 4 Kernen und einer Blockgröße von 100 dauert 5168 Minuten und 11,583 Sekunden, also etwa 3,5 Tage.

Mit anderen Werten für Kerne und Blockgröße lässt sich vielleicht eine kürzere Laufzeit erreichen.