1 Die Minimierung nach Dr. Petzold

1.1 Input

- Matrix mit logischer Gleichung (jede Zeile repräsentiert eine Disjunktion)

1.2 Schritte der Minimierung

- Ergänzung der logischen Gleichung um die fehlenden Terme (`buildComplement()`).
- `cmpTermS()`:
  - `markDifferentBits()`
    Vergleiche jeden **Einsterm** mit jedem **Nullterm**, markiere die unterschiedliche Bits (DC:1 DC:0 1:1 0:0 nicht markieren) indem die Stelle des Einsterms die Markierung ist. Füge diese in die Liste eines Terms ein.
  - `reduceTermS(unsigned int len, ...)`:
    reduziert überdeckte Terme der Länge eins `len=1` und dezimiert so die Anzahl der von `markDifferentBits()` generierten Terme. Ein Term α wird von einem Term β überdeckt, falls alle Stellen von β die kein `INIT_SYMBOL` sind mit denen von α übereinstimmen.
  - `discardTerm(..., unsigned int index)`:
    überprüft welche Terme von `index` überdeckt werden. `discardTerm(...)` wird nur dann aufgerufen, wenn die maximale Länge der Liste hinter `index` (`maxLen(unsigned int index)`) eins ist, denn diese Terme bestimmen eine Variablenbelegung der minimierten logischen Gleichung (`maxLen()=1` bedeutet dass sich der Einsterm vom Nullterm an dieser Stelle um nur ein Bit unterscheidet). `(redundant(list<minTerm>& minTermList, unsigned int InputTermSIndex))` und löscht die Liste zu all diesen Inputtermen. Die Liste "hinter" `InputTermSIndexndex` bleibt bestehen.

    `redundant(list<minTerm>& minTermList, unsigned int InputTermSIndex)`:
    entscheidet, ob `minTermList` den Term an der Stelle `InputTermSIndex` in der Eingabematrix überdeckt. Falls ja, so wird die Liste hinter `InputTermSIndex` in die `freeList` eingehängt. Der Term ist somit

für die weitere Minimierung nicht von Bedeutung ($\rightarrow$ `DISCARD`).

- `reduceAll()`:
  Reduziert alle übriggebliebenen Listen um die redundanten Terme der Länge 2 bis n.

- `allMult()`:
  Multipliziert die Listenelemente miteinander (siehe Beispiel) und generiert somit eine Konjunktive Normalform für eine Liste.
  Vor allMult() entspricht jeder Term in der Liste einer Disjunktion. D.h. die entsprechenden überdeckten Eingabeterme werden durch eine Disjunktive Normalform repräsentiert. Nach der Multiplikation genügt ein Element aus der Liste um die überdeckten Eingabeterme zu repräsentieren. Die Liste entspricht nun einer Konjunktiven Normalform.

- `getTermNum()`:
  numeriert alle Terme und erstellt eine Statistik über

  - die Anzahl der identischen Terme (`cntEqualTermS(listIter,listIndex)`) in den anderen Listen (in einer Liste können nach `reduceAll()` nicht mehr vorkommen),
  - den Listenindex und die
  - Länge des Termes.

  Die Numerierung `_termIndex` in `class minTermInfo()` ist eindeutig, d.h. identische Terme besitzen identische Indice. ($\rightarrow$ `cntEqualTermS(listIter,listInde`
  Listen der Länge eins gehen nicht in die Statistik ein, denn diese Terme sind maximale Überdeckungen, die auf jeden Fall im Ergebnis vorkommen.

- `selectTermS()`:
  verwendet folgende Heuristik um den letzten NP-vollständigen Teil der Optimierung (vgl. Quine McCluskey) durchzuführen:
  Bevorzuge den Term, der mehr Doppelgänger **(in anderen Listen)**) hat. Ist die Anzahl der Doppelgänger gleich, so wähle den kürzeren.
  Dieses Wählverfahren wird über alle Listen durchgeführt. Somit erhält man eine Sortierung aller Terme nach obigen Kriterien. Weitere Schritte von `selectTermS()`:

- sucht den Index der Liste in der der durch die Heuristik gewählte Term zum Ersten Mal auftaucht (`getFirstTerm(...)`)
- löscht alle anderen in dieser Liste (KNF) (`removeInessentialTermS(...)`)
- löscht alle Doppelgänger in den anderen, indexrelational höheren Listen (`eraseIdentical(...)`)

Dies wird solange durchgeführt bis die Heuristik keine Terme in der Statistik mehr findet!
Danach werden alle Listen auf den Term mit der kürzesten Länge reduziert (`removeInessentialTermS(...)`). Nach dem Aufruf von `selectTermS()` steht in den Listen maximal ein Term!

- `generateDNF()`
liest die Listen aus und generiert die Disjunktive Normalform.