# Validierung des Bereichsdatencompilers für die Linienzugbeeinflussung LZB L72 CE mit Hilfe eines diversitären Ansatzes

**Bernd Holzmüller**
**ICS AG**
**bernd.holzmueller@ics-ag.de**

## Die Aufgabe

- **Validierung eines Datenkonverters für die neue Hochgeschwindigkeitsstrecke Köln-Rhein/Main**
  - **sicherheitsrelevante Funktionalität**
- **Bereichsdaten**
  - **codieren Streckeninformationen: Signale, Weichen, Geschwindigkeiten, Gleisinformationen, Tunnel etc.**

```
LLKANAL:01 BK:01 FNMAX:120*
*****************
*
001,0 AEND  GR:-                                              KM:025,200
001,0 ELL   GR:- NIN:G1 RBW:1000 UEB:INDUSI V:MAX EO:J        KM:025,200#
      NINE:G1 VSTR:200 AMP:0600 TUNR:00 WBNR:01 SWNR:00 EONR:001      #
      VWEI:MAX BUNR:00 HANR:00                                        *
001,0 LZBEP GR:+ AKTIV:ENDE                                   KM:025,200*
011,6 FSIG  GR:- V:MAX WEIZ:N AUSF:J          NAME:25VFF      KM:026,266*
012,7 LAFA  GR:- IN:200  GEGEN:180                            KM:026,370*
023,7 SIGN  GR:+ KAN:01 ADR:01 BIT:01 ENR:01  NAME:26105      KM:027,479#
      ART:ZBK WEIZ:N ASP:HP0,V40,KS1                          *
```

## Die Aufgabe (Fs.)

- **Bereichsdatencompiler (BDC)**
  - **prüft Einhaltung diverser Konsistenzbedingungen**
  - **transformiert Bereichsdaten in Assembler-Format**
- **Assembler wird schließlich mit generischer Steuersoftware (LZB) zusammengebunden**

```
STRESP EQU     $
*
*001,0 AEND  GR:-                                          KM:025,200*
SE0001 EQU     $
       DC      SE0002
       DC      X'0000'
       DC      X'0080'
       DC      X'8014'
*
*001,0 ELL   GR:- NIN:G1 RBW:1000 UEB:INDUSI V:MAX EO:J      KM:025,200#
*      NINE:G1 VSTR:200 AMP:0600 TUNR:00 WBNR:01 SWNR:00 EONR:001      #
*      VWEI:MAX BUNR:00 HANR:00                                         *
SE0004 EQU     $
       DC      SE0005
       DC      SE0003
       DC      X'0080'
       DC      X'8018'
       DC      X'297F'
```



## Das Problem

- **Assembler-Dateien codieren Information sehr kompakt**
  - **Hex-Zahl 1, Bits 0-3: Geltungsrichtung**
  - **Hex-Zahl 1, Bits 4-6: irrelevant (Füller)**
  - **Hex-Zahl 1, Bits 7-15: Geschwindigkeit / 5**
- **Manuelle Validierung der Transformation ist**
  - **mühsam**
  - **zeitaufwändig**
  - **fehleranfällig**

## Die Lösung

- **Entwicklung eines zweiten Compilers (diversitär)**
  - **Ziel**
    → **Ausgaben automatisch vergleichen**
  - **Voraussetzungen**
    → **Entwicklung einzig auf Basis der Anforderungen**
    → **Einsatz unterschiedlicher Technologie**
    → **Wirtschaftlichkeit der Entwicklung und Anwendung gegenüber manuellem Ansatz**





- **Vorteile der Automatisierung**
  - **Anwendung und Auswertung eines Testfalls ist**
    → **sehr effizient**
    → **weniger fehleranfällig als manueller Ansatz (sobald stabil)**
  - ***viele* Tests durchführbar**
  - **Tests mit wenig Aufwand wiederholbar**
  - **neuer Compiler auch für operativen Einsatz verfügbar**

## Die Technologie

- **Alter Compiler: in C**
- **Neuer Compiler: in Haskell**
- **Warum Haskell?**
  - **Sprache gehört anderer Sprachklasse an als C**
    → **erfordert i.d.R. unterschiedliche Denkweise und Algorithmen als bei imperativen Programmen**
    → **wenig gemeinsame technisch bedingte Fehler mit C-basierter Lösung zu erwarten**
  - **fördert schnelle Entwicklung fehlerarmer und gut wartbarer Software (Navy-Studie in den 90-er Jahren)**





## Haskell Features (1)

- **Kein expliziter Begriff von Speicher**
  - **keine Variablen, Zuweisungen, Zeiger, Seiteneffekte**
    → **sondern seiteneffektfreie Ausdrücke (Funktionen und deren Anwendung)**
  - **keine Sequenz, Schleifen, Sprünge (goto, break, return)**
    → **sondern Bedingung und Rekursion**
  - **implizites Speichermanagement (garbage collection)**
- **sparsame, prägnante, gut lesbare Syntax**
  - **neue Operatoren definierbar**
  - **„pattern matching“**

## Haskell Features (2)

- **funktionale Abstraktion**
  - **Parametrierung von Ausdrücken**
  - **anonyme Funktionen (Lambda-Ausdrücke)**
- **Sehr gute Unterstützung von Listen**
  - **ermöglicht abstrakte, problemnahe Formulierungen (wie in Mengentheorie und Prädikatenlogik)**
- **„lazy evaluation“**
  - **Rechnen mit unendlichen Datenstrukturen**
  - **kann Divergenz von Berechnungen verhindern**





## Haskell Features (3)

- **Polymorphe Typen und Typinferenz**
  - **sehr hohes Maß an Wiederverwendung!**
- **sehr strenges Typsystem**
- **Modulsystem mit selektivem import/export**
- **Standard-Bibliothek**
  - **insb. für Listenmanipulation**

## Haskell-Beispiel: Quicksort

```
quicksort [] = []

quicksort (head:rest) =
  quicksort [x | x <- rest, x < head]
  ++ [head] ++
  quicksort [x | x <- rest, x >= head]
```

**oder:**

```
quicksort (head:rest) =
  quicksort (filter (< head) rest)
  ++ [head] ++
  quicksort (filter (>= head) rest)
```





## Haskell-Beispiel: Parser

```
import Parsec

stmt :: Parser Stmt
stmt = ifStmt <|> assignment

ifStmt :: Parser Stmt
ifStmt = do
 cond     <- do reserved "if"; expr
 thenStmt <- do reserved "then"; stmt
 elseStmt <- optMaybe (do reserved "else"; stmt)
 return (If cond thenStmt elseStmt)

assignment = do
 n <- name
 symbol ":="
 rhs <- expr
 return (Assignment n rhs)
```

## Haskell-Beispiel: Parser (2)

```
many :: Parser a -> Parser [a]
many p =
 do x <- p; xs <- many p; return (x:xs)
 <|> return []

sepBy1 :: Parser a -> Parser b -> Parser [b]
sepBy1 sep p = do
 x  <- p
 xs <- many (do sep; p)
 return (x:xs)
```



INFORMATIK CONSULTING SYSTEMS AG ICS

## Typische Fehler in C-Programmen

- **Speicherfehler**

| C | Haskell |
|---|---|
| Speicherfreigabe | - |
| Array-Indizierung | (Indizierung von Listen) |
| Dereferenzierung von Null-Zeigern | - |
| Zeigerarithmetik | - |
| uninitialisierte Variablen | - |



INFORMATIK CONSULTING SYSTEMS AG ICS

## Typische Fehler in C-Programmen (2)

- **Typfehler**

| C | Haskell |
|---|---|
| void* | - |
| casts | - |
| enum = int | - |
| Funktionen als Argumente | - |
| extern-Deklarationen | - |





## Typische Fehler in C-Programmen (3)

- **Numerische Fehler**

| C | Haskell |
|---|---|
| Überläufe | - (bei Verwendung von unbeschränkten Ganzzahltypen) |
| implizite Konversionen | - |
| Rundungsfehler | - (bei Verwendung von Brüchen) |

## Typische Fehler in C-Programmen (4)

- **Iterationsfehler**

| C | Haskell |
|---|---|
| „off-by-one“ | - (mit map, filter etc.) |
| Endlosschleifen | Endlos-Rekursion |

- **Logische Fehler**

| C | Haskell |
|---|---|
| durch Seiteneffekte | - |





## Der Projektablauf

- **Validierungskonzept**
  - **Validierungsplan**
  - **Erstellung prototypische Werkzeuge**
    → **Nachweis von Machbarkeit und Wirtschaftlichkeit**
- **Erstellung Testfallkatalog**
- **Erstellung Testeingaben**
- **Testdurchführung und -auswertung (automatisiert)**

## Die Schwierigkeiten

- **Freiheitsgrade bei Ausgabe (nicht spezifiziert)**
  - **Bezeichnung von Labels**
  - **Optimierung (Komprimierung) von Signalaspektlisten**
- **Erzeugte Kommentare**
  - **Vergleicher entfernt zuvor Leerzeilen und Kommentare**





## Die Ressourcen

- **Entwicklungszeit**
  - **BDC : Gesamtlaufzeit: mehrere MJ**
  - **HBDC : ca. 4 Wochen**
- **Codegröße (Tendenzen)**
  - **BDC : 1500 KB (X-Tool-Format)**
  - **HBDC : 100 KB, 2100 LOC**
    **(LOC ohne Kommentar- und Leerzeilen)**
    **(davon 188 * map, 43 * filter, 48 * list comprehension**
    **≡ 279 Schleifen!)**

## Die Resultate

- **Ergebnisse der ersten Validierung**
  - **11 Probleme in den Anforderungen entdeckt**
    - → **9 Lücken/Ungenauigkeiten**
    - → **2 Inkonsistenzen**
  - **14 Abweichungen Konverter/Anforderungen entdeckt**
    - → **1 Absturz**
    - → **3 syntaktische Abweichungen (zusätzliche Leerzeichen, Codierung dezimal/hexadezimal)**
    - → **10 inhaltliche Abweichungen**
- **In der Zwischenzeit drei durchgeführte Validierungen**
  - **Leichte Anpassung des HBDC an funktionale Änderungen (wenige Tage)**
  - **Schnelle, automatisierte Revalidierungen**





## Das Ende

- **Resultate sind ermutigend**
  - **geringe Entwicklungs- und Wartzungszeiten für HBDC**
  - **Testen ist mühelos**
  - **Entwicklung des HBDC macht Spaß**
  - **Haskell extrem effizient für die Werkzeugerstellung**
- **Publikation der Vorgehensweise in Fachzeitschrift Signal+Draht März 2006**
- **Referenz/Ansatzpunkt für weitere Projekte dieser Art**