Zusammenfassung

- Programmentwicklung durch *Transformationen*
- *Programmentwurfstransformationen* formalisieren *Entwicklungsentscheidungen*
- Beispiel: *Divide-&-Conquer* für
  - Quicksort, Merge, Insertsort
- Aber: *hochspezielles* Anwendungsgebiet

Vorlesung vom 18.07.06: Rückblick, Überblick, Ausblick

Wo sind wir?

- Am Ende . . .
- Heute: Blick in Industrie und Forschung
  - Hardwareverifikation
  - Softwareverifikation
- Ausblick: Forschungsaktivitäten hier

Hardwareverifikation

- Intel IA-32 Pentium®4:
  - Register Renaming
  - BUS recycle logic
  - Floating point division and square root

Die HW-Verifikationsumgebung Forte

- `FL` (stark getypt, funktional, ML): Eingabe- und Skriptsprache
- *Binary decision diagrams* (BDDs) als *first-class objects*
- *Symbolic trajectory evaluation* als integrierte Funktionen
  - Aussagen der Form $\models_{ckt} [ant \Rightarrow cons]$
  - Konjunktion von Basisaussagen der Form $\mathtt{N}^t(n\ \mathtt{is}\ v\ \mathtt{when}\ g)$
  - Simulation in Verband $Bool + \{\mathtt{X}, \top\}$
- Theorembeweiser `ThmTac`
  - Klassische Logik höherer Ordnung, LCF-design
  - Aussagen der Form $\{\phi_{in}\}(tr_{in}, ckt, tr_{out}\{\phi_{out}\}$
- 15000 Zeilen für eine Mikroinstruktion, 45000 Zeilen für alle 20

HW-Verifikation — Status

- *FDIV-Bug* (1994), *Kosten* ca. US$ 500 Mill
- Verifikation *amortisiert* sich
- Auch andere Hersteller (AMD, IBM, HP/Compaq, . . .)
- Große *Stückzahlen*, *spätere* Änderung *sehr teuer*
- Korrektheit wird *erwartet*
- Verifikation *Stand der Kunst*

Software-Verifikation

- Software-*Modelchecking*
- *Entscheidungsprozeduren*
- Statische und dynamische *Analyse*
- *Interaktive Verifikation*

Software-Modelchecking

- Model-Checking (SMV, SPIN, FDR, . . .)
- Kombiniert mit *Abstraktion* (Datenabstraktion, Prädikatenabstraktion: Bandera, Blast,
- *Vollautomatisch*, findet (nur) *Gegenbeispiele*
- Geeignet für: *nebenläufige* Systeme, eingeschränker *Zustandsraum*, Protokolverifikation,

Automatische Werkzeuge

- *Entscheidungsprozeduren*: Chaff/zchaff, TSAT, Simplify, UCLID, SVC
  - *Entscheidbarkeit* von logischen Ausdrücken (SAT)
  - *Arithmetik*, *Felder*, *abstrakte Datentypen*
  - *Vollautomatisch* aber *beschränktes* Einsatzgebiet, Kombination, keine *Gegenbeispiele*, *Korrektheit*?
- *Statische* und *Dynamische Analyse*: BANE, Ccured; Purify
  - Gegen Pufferüberläufe, Speicherlecks, usw.
  - *Skaliert* gut — *Korrektheit?*

Verifikation

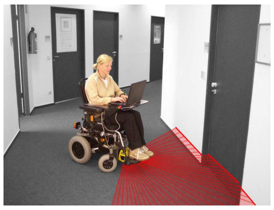
- Interaktives Beweisen: Hoare-Logik oder JML-artige
- Spezielle *Verifikationswerkzeuge*
  - Älter: KIV, VSE
  - Neuer: WHY, Spec#
- Universelle *Theorembeweiser*
  - STep, PVS, ACL2, Maude, HOL, Coq, Nuprl, Isabelle
- *Aufwändig*, aber *korrekt*

SW-Verifikation — Status

- *Stand der Kunst*: *Verifikation* für *kritische Einzelfälle*, *Fallstudien*
- Kein *universelles* Verifikationswerkzeug
  - Unterschiedliche *Domänentheorie*, unterschiedliche *Spezifikationssprachen*
- SW-Verifikation: *Forschung* — HW-Verifikation: *Produktion*
- Sofware kann leicht *nachgebessert* werden
- Software-Fehler werden *akzeptiert*
- Neue *Einsatzgebiete* werden Stand der Kunst *vorantreiben*
  - *Eingebette* und *hybdride Systeme*

Das SAMS-Projekt

- *Ausgangspunkt*: Der autonome Bremer Rollstuhl *Rolland*
- *Idee*: Sicherheitsmodul *parametrisierbar* für Roboter
- Sicherheitsmodul integriert in Laserscanner, für autonome Roboter, fahrerlose Transportsysteme
- *Ziel*: TÜV-Zulassung, formale *Verifikation* des Sicherheitsmoduls
- DFKI Bremen, Uni Bremen, Leuze lumiflex — 3 Jahre



Das AWE-Projekt

- *Ausgangspunkt:* Wiederverwendung des gesamten *Entwicklungsprozesses*, nicht nur der *Artefakte*
- *Idee:* Formale Entwicklungen als formale Objekte — systematische Verallgemeinerung
- *abstraction for reuse*
- *Implementierung* basierend auf Isabelle

Zusammenfassung

- Grundlagen der *Programmverifikation*
  - Vollständigkeit des *Hoare-Kalküls*
  - Generierung von *Verifikationsbedingungen*
- Jenseits des *Hoare-Kalküls*
  - *Java Modelling Language* (ähnlich: OCL)
  - Werkzeugunterstützung: *WHY*
- *Programmkonstruktion*:
  - Programmentwicklung als *Rückwärtsbeweis*
  - Programmentwicklung durch *Transformation* (divide-&-conquer)

Und jetzt?

- Mitarbeit in AG BKB/DFKI Lab Bremen
  - Studentische *Hilfskräfte*
  - *Diplomarbeiten*
- Tutor PI3, anybody?