# Lehrstuhl für Übersetzerbau und Programmiersprachen

Prof. Dr. Reinhard Wilhelm

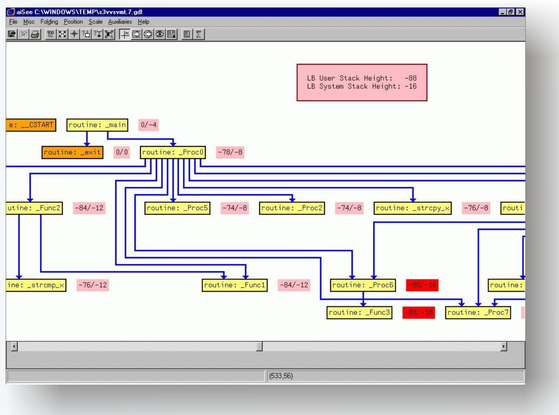
UNIVERSITÄT DES SAARLANDES

## I. Programmanalyse sicherheitskritischer Software

### a) Laufzeitanalyse (aiT)

- Berechnung harter Echtzeitschranken der "Worst-Case-Execution-Time" (WCET)
- Statische Analyse basierend auf der Theorie der abstrakten Interpretation
- Gültig für alle Eingaben
- Ersetzt teure und zeitintensive Messungen


- Beispiel: Fly-by-wire Systeme
  Flugzeugsteuerung erfordert rechtzeitiges Reagieren der Steuerelemente




### b) Stackhöhenanalyse (StackAnalyzer)

- Berechnung des Stackverbrauchs eines Tasks
- Erkennung von möglichen Stacküberläufen zur Entwicklungszeit

- Beispiel: Automobilindustrie, Schließ-Systeme
  Beschränkter Speicherplatz, kein Systemausfall.

Kontakt:
AbsInt GmbH
<www.absint.com>

## II. Post-Pass-Optimierungen

### a) Codekompaktion (aiPop)

- Codegrößenreduktion mittels funktionaler Abstraktion und "tail merging"
- Beschleunigung der Ausführung durch Optimierungen wie "Loop-invariant code motion" und "Function inlining" möglich


- Beispiel: Mobilfunktelefone
  Im Bereich eingebetteter Prozessoren ist Speicherplatz teuer und begrenzt

### b) PROPAN

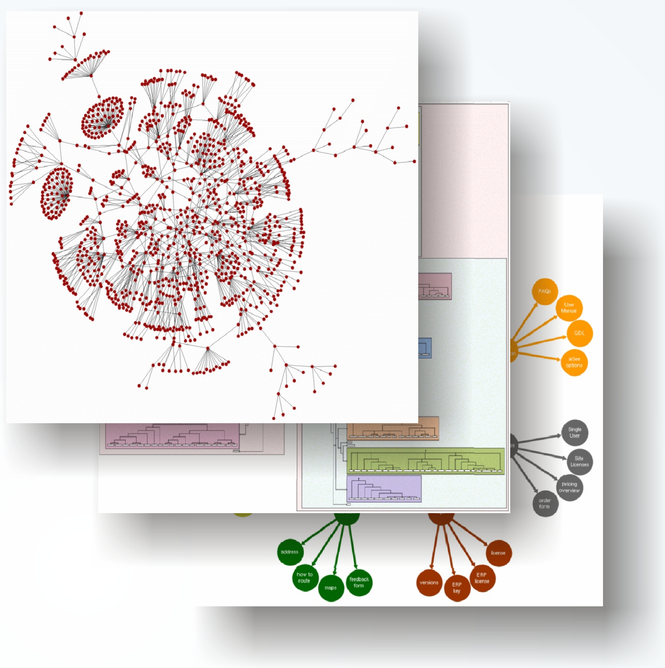
- Retargierbares Framework für Post-Pass-Optimierungen und -analysen
- Generierung eines Post-Pass-Optimierers aus einer kurzen und prägnanten Spezifikation des Mikroprozessors
- Optimierer liest Assemblerprogramme ein und führt effizienzsteigernde Transformationen durch
- Überlegenheit gegenüber traditionellen Verfahren für eingebettete, irreguläre Prozessoren
- Ganzzahlige lineare Programmierung als Lösungsverfahren

Kontakt: Daniel Kästner <kaestner@cs.uni-sb.de>

## III. Visualisierung

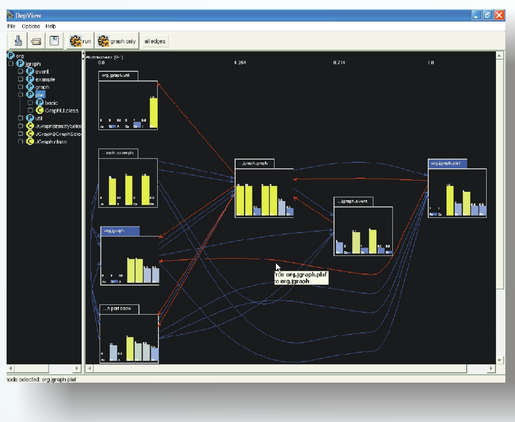
### a) Graphlayout (aiSee)

- Automatische Generierung eines Graphen aus einer Graphspezifikation
- Zahlreiche Anwendungen:
  - Unternehmensplanung
  - Software-Entwicklung
  - Biotechnology
  - Hardwaredesign



Kontakt:
AbsInt GmbH
<www.absint.com>

### b) Depency Viewer




- Analysiert Java-Code und visualisiert das Systemdesign als übersichtlichen Graph
- Gibt Hinweise auf Designfehler oder Designschwächen

Kontakt: Stephan Diehl <diehl@acm.org>

## IV. Shape Analyse

**Ziel:**

Statische Analyse (d.h. zur Übersetzungszeit) von Zeigerprogrammen

**Problem:**

Zeigerprogramme erzeugen und zerstören dynamisch (d.h. zur Laufzeit) Strukturen in der Laufzeithalde (Heap).

Beispiel: Einfach verkettete Liste im Heap

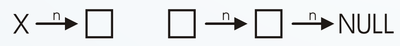



**Methode:**

*Shape Analyse nach Sagiv, Reps, Wilhelm*

- Approximation aller möglichen Heapstrukturen, die während einer Ausführung auftreten können.
- Hierzu verwendet man ein 3-wertige Logik.
- Statische Vorhersage von Laufzeitfehlern wie
  - NULL-Zeiger Dereferenzierung


  - Speicherlöchern




  - Strukturinvarianzen
    - Listen bleiben sortiert
    - Listen bleiben azyklisch
    - Bäume bleiben Bäume

Beispiel:
Umdrehen einer einfach verketteten Liste in C

```
typedef struct node  {
  struct node *n;
  int data;
} *List;

List reverse (List x)
{
  List y, t;
  y = NULL;
  while (x != NULL) {
    t = y;
    y = x;
    x = x->n;
```

```
...
x->n = NULL;
...
y = x->n->n;
...
```

Kontakt: Jörg Bauer <joba@cs.uni-sb.de>