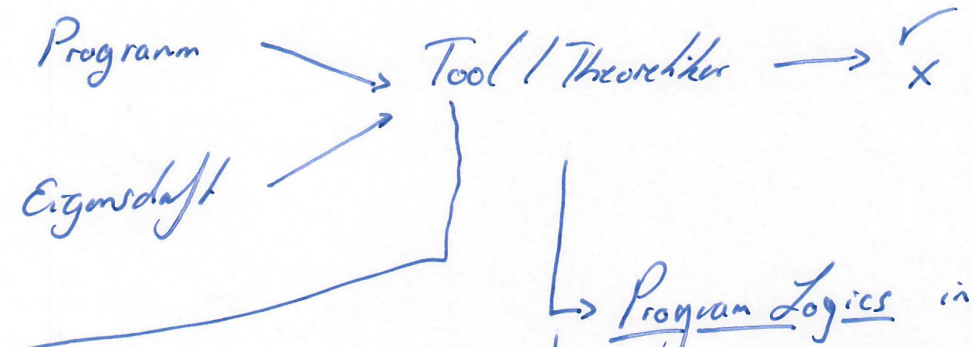
Modern Concurrency Theory

Problem: Nebenläufige Programmierung
↳ Die sequentiellen Teilberechnungen sehen gut aus.
↳ Bei deren Interaktion passiert Unfug.

Ziel: Verifikation nebenläufiger Programme



↳ Program Logics in denen der Korrektheitsbeweis geführt wird
↳ Mechanized Verification:
Proof Assistents / Theorembeweiser
prüfen Beweis auf Gültigkeit.
Warum? Sehr viele Fälle.

↳ Model Checking Tools / State Analysis Tools:
Finden selbstständig ein Korrektheitsbeweis oder ein Gegenbeispiel.

Bedeutung: Nebenläufige Programmierung ist nicht neu. Warum jetzt?
Wird Mainstream?
Als Frontend-Programmier nutzt man Bibliotheken.
Aber es müssen auch Leute das Backend schreiben / die Bibliotheken implementieren.

Problem (rev.): Performance-kritische nebenläufige Programmierung.
↳ Threads
↳ Explizite Speicherverwaltung
↳ Feingranulare Synchronisation
↳ Effekte der Ausführungsumgebung

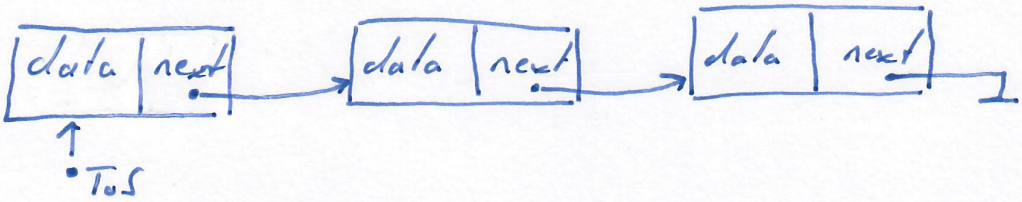
Ziel (rev.): Verifikation Performance-kritischer nebenläufiger Programme

1. Treiber's Stack

Systems Programming: Coping with Parallelism
R. Kent Treiber, 1986

Ziel: Implementiere einen Stack,
der von verschiedenen Threads genutzt werden kann.

Ansatz:
- Stack als einfach verkettete Liste auf dem shared Heap.
- Globaler Head-Pointer auf das oberste Element.



- Operationen void push(data d)
  data pop()

Problem:
- Operationen sollen nebenläufig ausgeführt werden.
- Wieviel Nebenläufigkeit soll schon in einem push/pop stecken?

Ansatz:
- Programmieren wir das mal.
- Sie brauchen einen C++-Compiler.
- Wir werden nur selten programmieren,
  aber es hilft die Tricks und die Probleme live zu sehen.