How to Specify the Flow of Data Accessibility

An OO Way of Concurrent Programming

Franz Puntigam

Technische Universität Wien

franz@complang.tuwien.ac.at

Klasse mit Zusicherungen – Token

```
class Buffer {
   public void put(int i) [!full() -> !empty()]
      { e[top++] = i; }
   public int get() [!empty() -> !full()]
      { return e[--top]; }
   public boolean empty() [true]
      { return (top == 0); }
   public boolean full() [true]
      { return (top == e.length()); }
   public Buffer(int s) [-> !full()]
      { e = new int[s]; }
   private int e[], top = 0;
}
```

Threads und Synchronisation

```
void produce(Buffer[[!full() -> !empty()] ->] p)
  { while(true) { ...; p.put(...); ... }
void consume(Buffer[[!empty() -> !full()] ->] c)
  { while(true) { ...; i = c.get(); ... }

void produce_consume() {
   Buffer b = new Buffer(2);  // !full()
   b.put(1);          // SEQ      !empty()
   int i = b.get();   // SEQ      !full()
   consume(b);        // PAR      [!full()->!empty()]
   produce(b);        // PAR/SEQ  ---
}
```

Statische Token

```
class Buffer8 {
   public void put(int i) [top:0..7 -> top+1]
      { e[top++] = i; }
   public int get() [top:1..8 -> top-1]
      { return e[--top]; }
   public Buffer8() [-> top:0] {}
   private (0..8) top = 0;
   private int e[] = new int[8];
}
...
Buffer8 b = new Buffer8();      // top:0
b.put(1); b.put(2); b.put(3)    // top:3
```

Verwendung dynamischer Information

```
Buffer[empty()->] emptyTest(Buffer[true ->] b) {
   if (b.empty())
      return b;
   else
      return null;
}
```

Ablauf der Synchronisation

```
void test(Buffer[!full() -> !full()] b)
   { b.get(); b.put(1); }
```

```
                                                    !full()
--{ b = new SyncProxy(b); }-->          [!full()->!full()]
--{}-->            [!full()->!empty()],[!empty()->!full()]
--{ new Thread{ b.writelocked{!empty()}{get()} };
    b.writelocked{!full()}{put(1)};
    b.clean{!empty()};
  }-->                                  [!full()->!full()]
--{ b.clean{!full()}; b = b.noProxy(); }-->       !full()
```

Zusammenfassung

- Programmierer: Daten-Zugreifbarkeit, Aufrufreihenfolge
  Compiler: Nebenläufigkeit, Synchronisation
- Abhängigkeiten durch Zusicherungen (Token) ausgedrückt
- dynamischer Informationsgewinn möglich
- Rekursion auf modulare Weise unterstützt
- Schritt hin zu garantiert fortlaufendem Betrieb
- Implementierung und Evaluation ausständig