Modellierung verteilter Systeme

Grundlagen der Programm und Systementwicklung

Sommersemester 2012

Prof. Dr. Dr. h.c. Manfred Broy

Unter Mitarbeit von Dr. M. Spichkova, J. Mund, P. Neubeck

Lehrstuhl Software & Systems Engineering

Parallele Programme - Parallel auszuführende Anweisungen

Parallelität bei der Ausführung von Anweisungen

- Wir sprechen bei der Variablen (dem Attribut) v von einer gemeinsamen Variablen für S_1 und S_2 Anweisungen, falls v
  - in einer der Anweisungen auf der linken Seite einer Zuweisung „v := ...“ auftritt

  und
  - in der anderen Anweisung auf der linken Seite einer Zuweisung „v := ...“ („geschrieben wird“) oder in einem Ausdruck auftritt („gelesen wird“).

- Sind S_1 und S_2 Anweisungen, die keine gemeinsamen Variablen (Attribute) aufweisen,
  - dann können diese ohne Probleme parallel ausgeführt werden, ohne dass das Ergebnis von der Reihenfolge der Ausführung abhängt.

echt zeitlich nebeneinander,
verzahnt im Sinne von Interleaving oder
in beliebiger Reihenfolge hintereinander

  - Anweisung die der parallelen Ausführung der beiden Anweisungen entspricht:

$$\lceil S_1 \parallel S_2 \rfloor$$

Parallelität bei der Ausführung von Anweisungen

- Seien S_1 und S_2 Anweisungen, und Q_1, Q_2, R_1, R_2 Zusicherungen
- Wir sprechen bei der Variablen v von einer gemeinsamen Variablen für die annotierten Anweisungen $\{Q_1\}\ S_1\ \{R_1\}$ und $\{Q_1\}\ S_2\ \{R_1\}$, falls v
  - in einer der Anweisungen auf der linken Seite einer Zuweisung „$v := ...$“ auftritt und
  - in der anderen Anweisung auf der linken Seite einer Zuweisung „$v := ...$“ in den Zusicherungen dazu oder in einem Ausdruck auftritt

- Sind die annotierten Anweisungen $\{Q_1\}\ S_1\ \{R_1\}$ und $\{Q_2\}\ S_2\ \{R_2\}$ ohne gemeinsame Variablen, dann:
  - Falls gilt

    $$\{Q_1\}\ S_1\ \{R_1\} \text{ und } \{Q_2\}\ S_2\ \{R_2\}$$

    dann gilt

    $$\{Q_1 \wedge Q_2\}\ \lceil\ S_1 \parallel S_2\ \rfloor\ \{R_1 \wedge R_2\}$$

Parallele Anweisungen

Zusammengesetzte Anweisungen S_1 und S_2 parallel ausgeführt:

$$\lceil S_1 \parallel S_2 \rfloor$$

bzw. allgemeiner für endlich viele Anweisungen

$$\lceil S_1 \parallel S_2 \parallel S_3 \parallel \ldots \parallel S_m \rfloor$$

Kritischer Punkt:

- Granularität bei Nutzung gemeinsamer Variabler -- Interleaving

Parallele Programme

- Die Programme (Anweisungen) heißen *disjunkt*, wenn:
  $$\forall i, k \in \{1, \ldots, n\} : \mathsf{change}(\mathsf{S}_i) \cap \mathsf{var}(\mathsf{S}_k) = \varnothing \quad \text{für} \quad i \neq k$$
- *change(S)*: Menge der Variablen in Anweisung S auf der linken Seite einer Zuweisung
- *var(S)*: Menge aller Variablen in S

- Kein Unterschied im Ergebnis zwischen synchroner Ausführung und Interleavingmodus bei disjunkten Programmen
- Einfache Beweisregel:

$$\frac{\{\mathsf{Q}\}\ \mathsf{S}_1; \ldots; \mathsf{S}_n\ \{\mathsf{R}\}}{\{\mathsf{Q}\}\lceil \mathsf{S}_1 \| \ldots \| \mathsf{S}_n \rfloor \{\mathsf{R}\}}$$

Parallele Programme mit gemeinsamen Variablen

Bei Ausführung im Interleavingmodus gilt:

- Atomizität ist entscheidend für erreichbare Zustände

Deshalb Festlegung der Atomizität über „*unteilbare*“ Anweisungen

- Beispiel:

$$\textbf{if} \ldots [] \langle G_i \ \textbf{then} \ S_i \rangle ; T_i \ [] \ \ldots \ \textbf{fi} \qquad\qquad \langle S \rangle$$

- Prüfen der Bedingung G_i und Ausführung der Anweisung S_i „in einem Schritt“
- Ansonsten möglicherweise Änderung der Bedingung G_i vor Ausführung von S_i

- Forderung: gemeinsame Variablen treten nur im Inneren unteilbarerer Aktionen auf (*kritischer Bereich*)

Warten auf Bedingungen

- Häufig wollen wir auf das Eintreten einer Bedingung C warten
- Wir schreiben

await ⟨ C **then** S ⟩ **end**

für

var b : Bool := true;
do ⟨ b ∧ C **then** S ⟩; b := false
[] ⟨ b ∧¬C **then nop** ⟩
od

Beispiel: Dekkers Algorithmus

Koordination der Abläufe von 2 Prozessen

(Betriebssysteme: Nutzung gemeinsamer Ressourcen)

Software-Lösung zur Gewährleistung **wechselseitiger Ausschluss**

Jeder Prozess durchläuft abwechselnd kritische und unkritische Phase, wobei

(1) Prozesse sind nie gleichzeitig in kritischer Phase

(2) Prozesse behindern sich nicht mehr als durch (1) nötig

Kritische Phase = Zugriff auf gemeinsame Ressource (Variable)

Beispiel: Dekkers Algorithmus

- Koordination von zwei parallelen Prozessen A und B
  - Beide Prozesse sind nie gleichzeitig in ihrer kritischen Phase
  - Beide Prozesse behindern sich ansonsten nicht

globale Variable v : **var** ID

Sorte ID = {A, B}

Dekkers Algorithmus

- x zeigt an, dass das Programm kritische Phase betritt/betreten will
- p zeigt, welcher Prozess bei Konflikt Vorrang hat

Programm entspricht der Anweisung

⌈ dekker(x, y, A) || dekker(y, x, B) ⌋

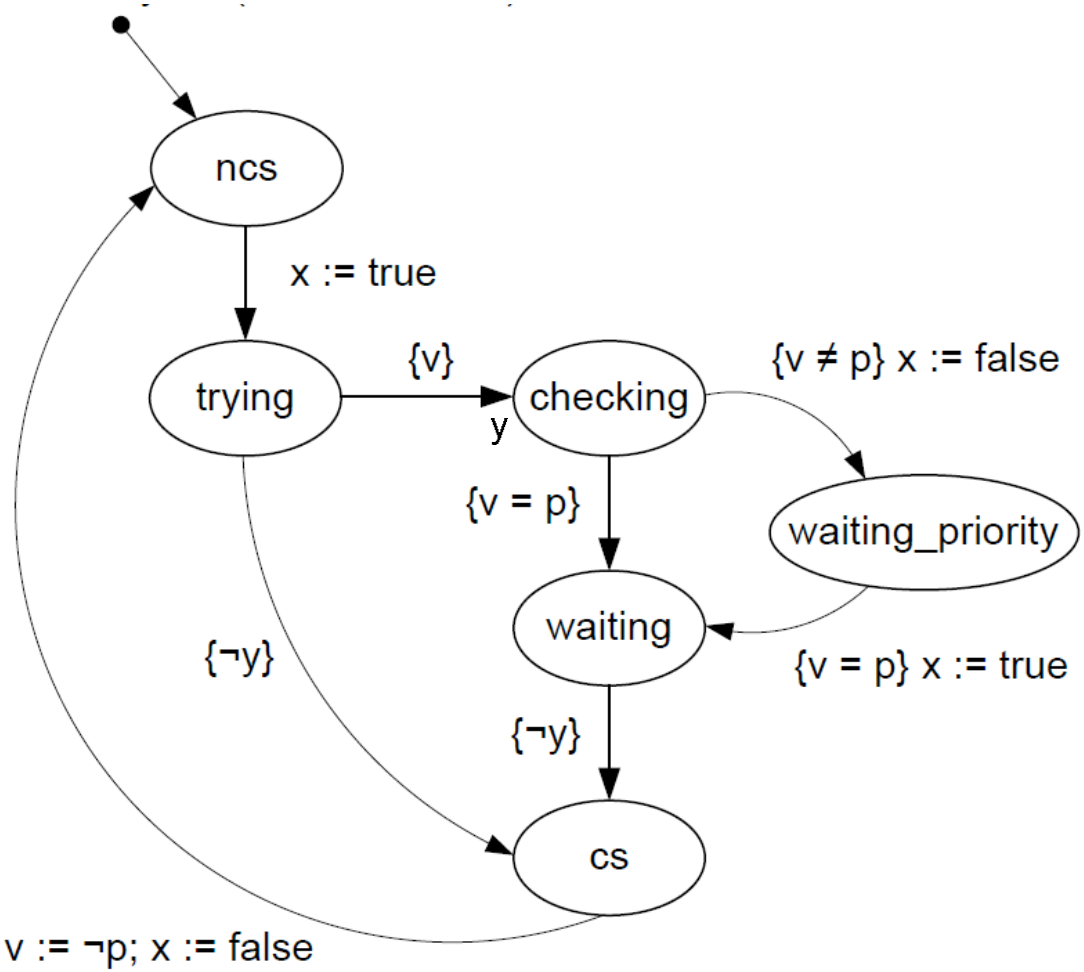
wobei v beliebig mit A oder B vorbesetzt ist.

Alternative: Semaphore

Beispiel: Dekkers Algorithmus

```
proc dekker = (x, y : var Bool, p : ID):
do Nichtkritische Phase von Prozess p;
    ⟨ x := true ⟩;
    if      ⟨¬y   then       nop ⟩ { y = false }
    [] ⟨ y   then nop ⟩;
                  if ⟨ v = p   then   nop ⟩
                  [] ⟨ v ≠ p   then   x := false ⟩;
                                      await ⟨ v = p then x := true ⟩ end
                  fi; { v = p ∧ x }
                  await ⟨ ¬y then nop { v = p ∧ x ∧ ¬y } ⟩ end
    fi ; { x }
    Kritische Phase von Prozess p;
    ⟨ v := ¬p ⟩;
    ⟨ x := false ⟩
od
```

Beispiel: ZM zu Dekkers Algorithmus



Beispiel: Semaphor

- Gemeinsame Variable s: **var** Nat zur Koordination paralleler Prozesse
- Grundidee: *s* gibt an, wie viele Prozesse noch in den kritischen Bereich eintreten dürfen
- Operationen:
  - V(s): *Freigeben*
  - P(s): *Protect*

proc V = (s: **var** Nat): $\langle$ s := s + 1 $\rangle$

proc P = (s: **var** Nat):
if $\langle$ s > 0 **then** s := s - 1 $\rangle$
[] $\langle$ s = 0 **then** nop $\rangle$; P(s) **fi**

- Unteilbare Anweisungen hier elementar

Zusicherungsbeweise für parallele Programme

Zusicherungsbeweise

Annotiertes anweisungsorientiertes Programm

- Programm mit Vor- und Nachbedingungen pro Anweisung
- Korrekt im Sinne der Hoare-Logik

Idee nach Susan Owicke & David Gries (1976) :

- Hoare-Logik: Beweismethode für sequentielle Programme
- Erweiterung auf parallele Programme;
- Komplexität durch Wechselwirkungen (**Interferenzen**)
- Anwendung Hoare-Kalkül auf Anweisungen ohne Bezug zu gemeinsamen Variablen
- Geistervariable

Zusicherungsbeweise - Fahrplan

1. Interferenzfreiheit
2. Invarianten
3. Geistervariablen

Interferenzen

- Wechselwirkungen zwischen annotierten Anweisungen über gemeinsame Variablen heißen *Interferenzen*
- Interferenzen beeinflussen („stören“) Zusicherungsbeweise

- Definition: *Interferenzfreier Zusicherungsbeweis*

  Seien

  $$\{Q_1\}\ S_1\ \{R_1\} \qquad \{Q_2\}\ S_2\ \{R_2\}$$

  vollständig annotierte Anweisungen (je eine Zusicherung vor und nach jeder Anweisung) in $\{Q_1\}\ S_1\ \{R_1\}$ und $\{Q_2\}\ S_2\ \{R_2\}$

  Die Anweisungen heißen **interferenzfrei**, wenn keine Zusicherung in $\{Q_1\}\ S_1\ \{R_1\}$ oder $\{Q_2\}\ S_2\ \{R_2\}$ auf gemeinsame Variablen Bezug nimmt

Interferenzfreiheit annotierter Anweisungen

“Zusicherungen werden durch Parallelität nicht ungültig”

$\{Q_1\}\ S_1\ \{R_1\}$ ist bzgl. $\{Q_2\}\ S_2\ \{R_2\}$ **interferenzfrei**, wenn

Für jede Zusicherung P in der annotierten Anweisung $\{Q_1\}\ S_1\ \{R_1\}$
gilt:
Für alle Anweisungen S in $\{Q_2\}\ S_2\ \{R_2\}$, die

- eine Zuweisung außerhalb einer unteilbaren Aktion oder
- unteilbare Aktion

mit Vorbedingung $\{B\}$ sind, gilt:

$$\{B \wedge P\}\ S\ \{P\}$$

- Achtung: Wir setzen voraus, dass Zugriffe (lesend oder schreibend) nur in unteilbaren Aktionen vorgenommen werden

Interferenzfreier Zusicherungsbeweis

Beweisregel:

$$\frac{\{Q_1\}\ S_1\ \{R_1\},\ \{Q_2\}\ S_2\ \{R_2\} \text{ wechselseitig interferenzfrei}}{\{Q_1 \wedge Q_2\}\ \lceil S_1 \,||\, S_2 \rfloor\ \{R_1 \wedge R_2\}}$$

Mit:

- S_1 und S_2 vollständig annotiert

Unvollständigkeit des Beweiskalküls

- Beweis von gültigen Nachbedingungen für ein paralleles Programm im Allgemeinen immer noch nicht möglich
- Beispiel:

$$\{x = a\}$$
$$\lceil \langle x := x + 1 \rangle; \langle x := x + 1 \rangle \| \langle x := x + 1 \rangle \rfloor$$
$$\{x = a + 3\}$$

- Regel nicht anwendbar, da gemeinsame Variable in Zusicherungen

Geistervariablen

Bisheriges Kalkül nicht mächtig genug;
Es fehlen Aussagen über “Kontrollzustände”

Geistervariablen

- Hilfsvariablen (zusätzliche Variablen)
- Annotation durch Variablen und Anweisungen
- Keine Änderung des Programmablaufs und der -wirkung
- Liefern Information über Kontrollfluss

Anreicherung um Geistervariable

- Sei S eine Anweisung;
- Wir sagen dass Anweisung S‘ eine Anreichung von S um **Geistervariable** (Hilfsvariable, Historyvariable), wenn S‘ aus S entsteht, indem wir
  - Beliebige zusätzliche Variable (frei von Bezeichnungskonflikten) einführen
  - Beliebige Zuweisungen an Geistervariable an beliebigen Stellen im Programm einfügen

Durch geeignet gewählte Geistervariable und Zuweisungen können wir Ablaufinformation (die History) erfassen

- Gilt {Q} S’ {R} für Zusicherungen Q und R, die keine Geistervariable enthalten, dann
  - gilt auch {Q} S {R}
  - S wird auch **Slice** von S’ genannt

Anreicherung um Geistervariablen und Beweisregel

S sei ein anweisungsorientiertes Programm.

1. Einführung zusätzlicher (Geister)variablen
2. Hinzufügen von Anweisungen an diese Geistervariablen
3. Zuweisungen sind wohldefiniert

Beweisregel:

$$\frac{\{Q\}\ S'\ \{R\}}{\{Q\}\ S\ \{R\}}$$

S', um Geistervariablen angereichertes S

Q und R frei von Geistervariablen

Invarianten und Zusicherungsbeweise

Invarianten: mächtigerer Beweiskalkül

- J ist Zusicherung für jede unteilbare Anweisung S
- J ist invariant für alle übrigen Anweisungen S im Programm P

$$\{J\}\ S\ \{J\}$$

- Alle gemeinsamen Variablen treten nur in unteilbaren Aktionen auf

Wir schreiben P **inv** J und es gelten folgende Regeln:

$$\frac{P\ \textbf{inv}\ J}{\{J\}\ P\ \{J\}} \qquad \text{sowie} \qquad \frac{S_1\ \textbf{inv}\ J,\ S_2\ \textbf{inv}\ J}{\{J\}\ \lceil S_1 \,||\, S_2 \rfloor\ \{J\}}$$

Kombination mit Invarianten

- Geistervariable h, k:

$$\{x = a\}$$
$$h := k := 0;$$
$$\lceil \langle x := x + 1; h := 1 \rangle; \langle x := x + 1; h := 2 \rangle \| \langle x := x + 1; k := 1 \rangle \rfloor$$
$$\{x = a + 3 \wedge h = 2 \wedge k = 1\}$$

Wir beweisen unschwer:

$$\{h = 0\} \langle x := x + 1; h := 1 \rangle; \langle x := x + 1; h := 2 \rangle \{h = 2\}$$
$$\{k = 0\} \langle x := x + 1; k := 1 \rangle \{k = 1\}$$

- Invariante: x = a + h + k gilt offensichtlich

- Kombination der Beweismethoden liefert die gewünschte Aussage