AG Betriebssysteme FB3
Jan Peleska, Kirsten Berkenkötter

Implizite Spezifikation

1 Kreissäge

Das Ein- und Ausschalten der Kreissäge wird über die Events *on* und *off* signalisiert. Die Säge hat einen Schutzmechanismus, der hoch- und runtergeklappt werden kann. Dies wird über die Events *guard.up* und *guard.down* modelliert.

1. Der Schutz kann innerhalb von 5 Zeiteinheiten nach dem Ausschalten nicht abgenommen werden:
   $\forall\, t \in \mathbb{R}^+ \bullet (t, guard.up) in\, s \Rightarrow off \notin \sigma(s \uparrow (t, t+5))$
2. Der Schutz alterniert zwischen den beiden möglichen Positionen, beginnend mit *guard.down*:
   $s \downarrow \{guard.down\} \leq s \downarrow \{guard.up\} \leq s \downarrow \{guard.down\} + 1$
3. Ein- und Ausschalten der Säge ist alternierend:
   $s \downarrow \{off\} \leq s \downarrow \{on\} \leq s \downarrow \{off\} + 1$
4. Die Säge kann nicht eingeschaltet werden, wenn der Schutz nicht geschlossen ist:
   $off(s) \Leftrightarrow s \downarrow on = s \downarrow off$
   $guard\_down(s) \Leftrightarrow s \downarrow guard.down = s \downarrow guard.up$
   $\neg off(s) \Rightarrow guard\_down(s)$
5. Der Schutz muss bereits 8 Zeiteinheiten geschlossen sein, bevor die Säge eingeschaltet werden kann:
   $\neg off(s) \Rightarrow end(s \upharpoonright \{guard.down\}) + 8 \leq end(s \upharpoonright \{on\}) \vee s \upharpoonright \{guard.down\} = <>$

2 Fischer's Protokoll

2.1 Vereinfachte Version

Annahme Zunächst wird eine vereinfachte Form des Protokolls verwendet: alle versuchen, in die kritische Region zu kommen, aber nur einer schafft es. *k* wird niemals auf 0 gesetzt.

1. Wir wollen zeigen, dass nur ein Prozess die kritische Region betritt:
   $SYSTEM\, sat\, s \downarrow \{enter\} \leq 1$
2. Ein Prozess beschreibt die shared-Variable *k*. Hier soll gelten (zugesichert aufgrund des verwendeten Betriebssystems), dass ein gelesener Wert immer der zuletzt geschriebene ist, oder – falls noch nicht geschrieben wurde – 0:

$$\begin{aligned} VARPROCESS\, sat\, s \quad = \quad & s1 \frown < (t, read.j) > \frown s2 \Rightarrow \\ & (last(s1 \upharpoonright write) = write.j \wedge j \neq 0) \vee ((s1 \upharpoonright write) = <> \wedge j = 0) \end{aligned}$$

   In diesem vereinfachten Fall können wir also auch sagen:

$VARPROCESS\ sat\ read.0\ at\ t \Rightarrow no\ write\ at\ [0,t) \ \vee \ write.0\ in\ s$

Wenn dies für den Prozess *VARPROCESS* gilt, muss es auch für den Prozess *SYSTEM* gelten, da beide Prozesse über *read*– und *write*–Events synchronisiert sind:

$SYSTEM\ sat\ read.0\ at\ t \Rightarrow no\ write\ at\ [0,t) \ \vee \ write.0\ on\ s$

3. Wir spezifizieren, dass jeder Prozess nur seine eigene ID schreibt:

   $PROCESS_i\ sat\ \sigma(s \upharpoonright write) \subseteq \{write.i\}$

   Für die Parallelkomposition aller Prozesse gilt also, dass jedes *write.i* aus dem Prozess $PROCESS_i$ stammt:

   $PROCESSES\ sat\ i \neq j \Rightarrow write.i \notin \sigma(PROCESS_j)$

4. Jeder Prozess schreibt seine ID maximal einmal:

   $PROCESS_i\ sat\ s \downarrow write.i \leq 1$

   Damit gilt also für die Parallelkomposition unter Berücksichtigung von 3. :

   $PROCESSES\ sat\ s \downarrow write.i \leq 1$

   $SYSTEM\ sat\ s \downarrow write.i \leq 1$

   Das bedeutet u.a., dass kein Prozess ein *write.0* macht, da kein Prozess die ID 0 hat:

   $PROCESS_i\ sat\ no\ write.0\ in\ s$

   Dies muss natürlich auch für die Parallelkomposition gelten sowie für das Gesamtsystem:

   $SYSTEM\ sat\ no\ write.0\ in\ s$

   Mit 2. können wir dann folgern:

   $SYSTEM\ sat\ read.0\ at\ t \Rightarrow no\ write.\{1..2\}\ at\ [0,t)$

5. Jeder Prozess kann nur für sich selbst ein *enter* machen:

   $PROCESS_i\ sat\ \sigma(s \upharpoonright enter) \subseteq \{enter.i\}$

   Für die Parallelkomposition gilt also:

   $PROCESSES\ sat\ i \neq j \Rightarrow enter.i \notin \sigma(PROCESS_j)$

6. Jeder Prozess macht nur einmal ein *enter*:

   $PROCESS_i\ sat\ s \downarrow enter.i \leq 1$

   Für die Parallelkomposition gilt also:

   $PROCESSES\ sat\ s \downarrow enter.i \leq 1$

   $SYSTEM\ sat\ s \downarrow enter.i \leq 1$

7. Nun müssen die Zeitbedingungen spezifiziert werden. Wenn die kritische Region von einem Prozess erreicht wurde, d.h. ein *enter*–Event aufgetreten ist, müssen vorher *read*– und *write*–Events mit den festgelegten Zeitabständen aufgetreten sein:

$$\begin{aligned} PROCESS_i\ sat\ T_i \quad = \quad & enter.i\ in\ s \Rightarrow \\ & \exists\, t_i, t_i', t_i'' \bullet read.0\ at\ t_i \wedge write.i\ at\ t_i' \wedge read.i\ at\ t_i'' \ \wedge \\ & t_i' - t_i = a \wedge t_i'' - t_i' = b \wedge t_i \leq t_i' \leq t_i'' \end{aligned}$$

   Die Prozesse $PROCESS_i$ laufen alle unsynchronisiert. Das Event *enter.i* wird nur vom jeweiligen Prozess $PROCESS_i$ ausgeführt. Aus diesem Grund erfüllt auch die Parallelkomposition aller Prozesse $PROCESS_i$ diese Spezifikation. Diese ist über alle Events *read* und *write* mit dem Prozess *VARPROCESS* synchronisiert, so dass diese Komposition *SYSTEM* ebenfalls die Spezifikation erfüllen muss.

8. An dieser Stelle wissen wir nun folgendes: *SYSTEM* garantiert,

- dass der gelesene Wert immer der zuletzt geschriebene ist oder am Anfang die 0.
- dass jede ID nur einmal geschrieben wird.
- dass die Zeitbedingungen eingehalten werden.

Nun muss gezeigt werden, dass nur ein einziges *enter* geschieht, wie in 1. gefordert wird. Wie zeigen dies, indem wir zunächst annehmen, dass zweimal ein *enter* auftritt und diese Annahme zum Widerspruch führen.

s_0 sei eine beliebige Trace, in der *enter.i* und *enter.j* vorkommen mit $i \neq j$ (lt. 5. und 6.). Dann müsste gelten:

$read.0\,at\,t_i \wedge write.i\,at\,t_i^{'} \wedge read.i\,at\,t_i^{''} \wedge$
$read.0\,at\,t_j \wedge write.j\,at\,t_j^{'} \wedge read.j\,at\,t_j^{''} \wedge$
$t_i^{'} - t_i = a \wedge t_i{''} - t_i^{'} = b \wedge t_i \leq t_i{'} \leq t_i^{''} \wedge t_j^{'} - t_j = a \wedge t_j{''} - t_j^{'} = b \wedge t_j \leq t_j{'} \leq t_j^{''}$

write.i und *write.j* müssen in der Trace in irgendeiner Weise geordnet sein, selbst wenn sie zum gleichen Zeitpunkt registriert werden. Wir nehmen (ohne Verlust der Allgemeinheit) an, dass *write.j* das letzte Auftreten von *write* ist:

$write.j = last(s_0 \restriction \{write.i, write.j\})$

Es gilt nun für s_0:

$read.0\,at\,t_i$
$read.0\,at\,t_j\,und$
$write.i\,at\,t_i^{'}$

Also muss gelten:

$t_j \leq t_i^{'}$

Für die Trace s_0 sieht es nun folgendermaßen aus:

$s_0 = s_1 \frown < (t_i^{''}, read.i) > \frown s_2 \wedge write.i = last(s_1 \restriction write)$

Das letzte *write* in s_1 muss *write.i* gewesen sein, damit *read.i* stattfinden kann (nach 2.).

Wir haben ferner festgelegt, dass *write.j* nach *write.i* passiert. In s_1 kann es demnach nicht auftauchen, sondern muss in s_2 sein. Daraus folgt:

$t_i^{''} \leq t_j^{'}$

Es gilt also:

(a) $t_j \leq t_i^{'}$

(b) $t_i{''} \leq t_j^{'}$

(c) $b > a$ (lt. Voraussetzung für Fischer's Protocol)

Daraus folgt:

$t_j \leq t_i^{'} = t_i^{''} - b$
$\Leftrightarrow t_j \leq t_j^{'} - b$
$\Leftrightarrow t_j < t_j^{'} - a$

Diese Aussage steht im Widerspruch zur Annahme $t_j = t_j^{'} - a$ Dadurch ist bewiesen, dass nur ein *enter*–Event auftritt, wie in 1. gefordert.

2.2 Allgemeine Version

Auf der vereinfachten Version des Protokolls aufbauend, soll nun gezeigt werden, dass auch die allgemeine Version wie erwartet funktioniert, sofern die einzelnen Komponenten des Systems ihrer Spezifikation entsprechen.

1. Zu zeigen ist diesmal, dass immmer nur ein Prozess in der kritischen Region ist:

   $SYSTEM\, sat\, s \downarrow exit \leq s \downarrow enter \leq s \downarrow exit + 1$

   oder auch

   $SYSTEM\, sat\, s = s_1 \frown <(t_1, enter.i)> \frown s_2 \frown <(t_2, exit.i)> \frown s_3$

2. Die Spezifikation für den Prozess, der die shared-Variable k modelliert sieht ähnlich aus wie zuvor, nur dass auch der Wert 0 geschrieben werden kann:

$$\begin{aligned} VARPROCESS\, sat\, s \quad = \quad & s1 \frown <(t, read.j)> \frown s2 \Rightarrow \\ & (last(s1 \upharpoonright write) = write.j \wedge j \neq 0) \vee \\ & (((last(s1 \upharpoonright write) = write.0) \vee (s \upharpoonright write) = <>) \wedge j = 0) \end{aligned}$$

   Es gilt also:

   $VARPROCESS\, sat\, read.0\, at\, t \Rightarrow no\, write.\{1,2\}\, at\, [0,t) \vee write.0\, in\, s$

   Wenn dies für den Prozess *VARPROCESS* gilt, muss es auch für den Prozess *SYSTEM* gelten, da beide Prozesse über *read*– und *write*–Events synchronisiert sind:

   $SYSTEM\, sat\, read.0\, at\, t \Rightarrow no\, write.\{1,2\}\, at\, [0,t) \vee write.0\, on\, s$

3. Für die einzelnen Prozesse gilt natürlich weiterhin, dass sie nur *enter* für ihre eigene ID machen können. Sie können jetzt allerdings außer ihrer eigenen ID auch 0 schreiben. Die Bedingungen, dass nur einmal geschrieben wird, sowie dass niemals der Wert 0 geschrieben wird, gelten natürlich nicht mehr:

   $PROCESS_i\, sat\, \sigma(s \upharpoonright write) \subseteq \{write.i, write.0\}$

   $PROCESS_i\, sat\, \sigma(s \upharpoonright enter) \subseteq \{enter.i\}$

   $PROCESSES\, sat\, i \neq j \Rightarrow write.i \notin \sigma(PROCESS_j)$

   $PROCESSES\, sat\, i \neq j \Rightarrow enter.i \notin \sigma(PROCESS_j)$

4. Die Zeitbedingung wird komplizierter, da auch das *exit* sowie das *write.0* berücksichtigt werden müssen. Außerdem ist das *enter* nicht mehr auf ein Auftreten im Prozess beschränkt, sondern kann beliebig oft geschehen:

$$\begin{aligned} PROCESS_i\, sat\, T_i \quad = \quad & enter.i\, in\, s \Rightarrow \exists\, t_i, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6 \bullet \\ & read.0\, at\, t_i \wedge write.i\, at\, t_i^2 \wedge read.i\, at\, t_i^3 \wedge enter.i\, at\, t_i^4 \wedge write.0\, at\, t_i^5 \wedge exit.i\, at\, t_i^6 \wedge \\ & t_i^2 - t_i = a \wedge t_i^3 - t_i^2 = b \wedge t_i \leq t_i^2 \leq t_i^3 \leq t_i^4 \leq t_i^5 \leq t_i^6 \wedge \\ & (s \uparrow [t_i, t_i^6]) \downarrow enter.i = 1 \wedge \\ & s \downarrow write.0 \leq s \downarrow enter.i \leq s \downarrow write.0 + 1 \wedge \\ & s \downarrow exit.i \leq s \downarrow write.0 \leq s \downarrow exit.i + 1 \end{aligned}$$

   Für jedes *enter.i* in einer Trace gilt also, dass die Reihenfolge der Events eingehalten werden muss, die Zeitbedingungen a und b korrekt eingehalten werden, die zeitliche Reihenfolge der Events stimmt und nur jeweils ein *enter.i* auftritt.

   Für jeden einzelnen Prozess gilt also:

   $PROCESS_i\, sat\, s \downarrow exit.i \leq s \downarrow enter.i \leq s \downarrow exit.i + 1$

   Durch die Parallelkomposition gilt dies auch für *PROCESSES* und *SYSTEM*

   $PROCESSES\, sat\, s \downarrow exit.i \leq s \downarrow enter.i \leq s \downarrow exit.i + 1 \wedge i \in \{1,2\}$

   $SYSTEM\, sat\, s \downarrow exit.i \leq s \downarrow enter.i \leq s \downarrow exit.i + 1 \wedge i \in \{1,2\}$

5. Nun muss gezeigt werden, dass die Prozesse die kritische Region nicht gleichzeitig betreten können. Wenn dies der Fall ist, gilt automatisch die Bedingung aus 1., da sie für jeden einzelnen Prozess an sich ja stimmt. Der Beweis folgt analog zur vereinfachten Version durch Widerspruch.

   s_0 ist eine beliebige Trace die folgende Events in dieser Reihenfolge enthält: *enter.i*, *enter.j* und *exit.i*, $i \neq j$. Es muss also gelten:

   $$\begin{aligned}&\mathit{read}.0\,\mathit{at}\,t_i \wedge \mathit{write}.i\,\mathit{at}\,t_i^2 \wedge \mathit{read}.i\,\mathit{at}\,t_i^3 \wedge \mathit{enter}.i\,\mathit{at}\,t_i^4 \wedge \mathit{write}.0\,\mathit{at}\,t_i^5 \wedge \mathit{exit}.i\,\mathit{at}\,t_i^6 \wedge \\ &\mathit{read}.0\,\mathit{at}\,t_j \wedge \mathit{write}.j\,\mathit{at}\,t_j^2 \wedge \mathit{read}.j\,\mathit{at}\,t_j^3 \wedge \mathit{enter}.j\,\mathit{at}\,t_j^4 \wedge \mathit{write}.0\,\mathit{at}\,t_j^5 \wedge \mathit{exit}.j\,\mathit{at}\,t_j^6 \wedge \\ &t_i^2 - t_i = a \wedge t_i^3 - t_i^2 = b \wedge t_j^2 - t_j = a \wedge t_j^3 - t_j^2 = b \ldots\end{aligned}$$

   write.i und *write.j* müssen in der Trace in irgendeiner Weise geordnet sein, selbst wenn sie zum gleichen Zeitpunkt registriert werden. Wir nehmen (ohne Verlust der Allgemeinheit) an, dass *write.j* nach *write.i* auftritt ...

   Wenn man den Kram also nochmal durchexzerziert, kommt man zum gleichen Ergebnis wie im vereinfachten Fall. Der Widerspruch beruht auch im Allgemeinen Fall auf der Tatsache, dass irgendein *write*–Event nach dem anderen stattgefunden haben muss und deshalb nur ein Prozess den korrekten Wert zum Betreten der kritischen Region lesen kann.