Foundations of System Development

Martin Wirsing

in cooperation with
Axel Rauschmayer

Verfeinerung und Strukturierung

Verfeinerung und Strukturierung

Bisher haben wir einfache (“flache”) TLA-Spezifikationen und ihre Eigenschaften betrachtet.

Im folgenden Kapitel studieren wir die Verfeinerung von TLA-Spezifikationen und strukturierte Spezifikationen.

Ziele:

- Ablaufverfeinerung als Implikation
- Komposition (mit Synchronisierung über gemeinsame Komponenten) als Konjunktion
- Kapselung als Quantifizierung über flexible Variablen
- Verfeinerungsabbildungen für Implementierungsbeweise

Ablaufverfeinerung

Example (Uhr mit Stunden-, Minuten- und Sekundenanzeige):

Ähnlich wie im Uhr-Beispiel kann eine Uhr mit Stunden-, Minuten und Sekundenanzeige durch die folgende TLA-Spezifikation beschrieben werden:

$$
\begin{aligned}
IClk2 &\equiv hr \in \{0,\ldots,23\} \wedge min \in \{0,\ldots,59\} \wedge sec \in \{0,\ldots,59\} \\
Sec2 &\equiv sec < 59 \wedge sec' = sec + 1 \wedge hr' = hr \wedge min' = min \\
Min2 &\equiv sec = 59 \wedge min < 59 \wedge sec' = 0 \wedge min' = min + 1 \wedge hr' = hr \\
Hr2 &\equiv sec = 59 \wedge min = 59 \wedge sec' = 0 \wedge min' = 0 \wedge hr' = (hr + 1) \bmod 24 \\
Tick2 &\equiv Sec2 \vee Min2 \vee Hr2 \\
Clock2 &\equiv IClk2 \wedge \Box[Tick2]_{hr,min,sec} \wedge \mathrm{WF}_{hr,min,sec}(Tick2)
\end{aligned}
$$

Jeder Ablauf von $Clock2$ erfüllt auch die Spezifikation $Clock$ aus dem Uhr-Beispiel:

- Die Anfangsbedingung $IClk2$ impliziert die Bedingung $IClk$.
- Jeder Übergang gemäß $Tick2$ lässt entweder (hr, min) unverändert oder entspricht einem $Tick$-Übergang.
- Die Fairnessbedingung $\mathrm{WF}_{hr,min,sec}(Tick2)$ erzwingt, dass auch unendlich viele $Tick$-Übergänge stattfinden, daher ist auch $\mathrm{WF}_{hr,min}(Tick)$ erfüllt.

Also ist die Implikation $Clock2 \Rightarrow Clock$ gültig.

Hierbei ist wesentlich, dass TLA-Spezifikationen “Stotterschritte” erlauben, welche die Systemvariablen nicht verändern. Bei Verfeinerungen dürfen daher neue Zustandskomponenten (“Implementierungsdetails”) eingeführt werden, um atomare Zustandsübergänge der Spezifikation in kleinere Schritte zu zerlegen, solange dies auf der Ebene der (ursprünglichen) Spezifikation unsichtbar bleibt.

Definition:

Eine TLA-Spezifikation $Impl$ heißt (Ablauf-)Verfeinerung einer Spezifikation $Spec$, wenn die Implikation

$$Impl \Rightarrow Spec$$

gültig ist, d.h. wenn jede Zustandsfolge, die $Impl$ erfüllt, auch $Spec$ erfüllt.

Vergleich mit früher vorgestellten Verfeinerungsbegriffen:

- Der TLA-Verfeinerungsbegriff entspricht dem Begriff in algebraischen Spezifikationen insofern, als in beiden Fällen die Inklusion der Modellklasse (Algebren bzw. erfüllende Abläufe) gefordert wird. Allerdings wurde bei algebraischen Spezifikationen Gleichheit der Signaturen verlangt, während die Verfeinerung einer TLA-Spezifikation zusätzliche flexible Variablen (“Implementierungsdetails”) enthalten darf.
- In Z waren zwei Bedingungen gefordert:

  $$(\text{pre } A) \wedge C \Rightarrow A \qquad \text{und} \qquad \text{pre } A \Rightarrow \text{pre } C$$

  Die zweite Bedingung drückt aus, dass die implementierende Operation in jedem Zustand anwendbar ist, in dem die abstrakte Operation anwendbar ist. Eine analoge Bedingung fehlt in TLA: die Implementierung darf zusätzliche Schritte benötigen, bis eine Aktion der “abstrakten” Spezifikation ausführbar wird.

  Andererseits müssen die Lebendigkeitseigenschaften der ursprünglichen TLA-Spezifikation erhalten bleiben.

Beweis von Verfeinerungen

$$\underbrace{CInit \wedge \Box[CNext]_{cv} \wedge CLive}_{Impl} \quad \Rightarrow \quad \underbrace{AInit \wedge \Box[ANext]_{av} \wedge ALive}_{Spec}$$

Verfeinerungsbeweise erfolgen in 4 Schritten:

1. Beweis einer geeigneten Hilfsinvariante $Impl \Rightarrow \Box Inv$.
2. Initialisierungsbedingung $CInit \Rightarrow AInit$.
3. Simulationsbeweis $Inv \wedge [CNext]_{cv} \Rightarrow [ANext]_{av}$.
4. Lebendigkeitsbedingungen $\Box Inv \wedge \Box[CNext]_{cv} \wedge CLive \Rightarrow ALive$.

Nur der Beweis von Schritt 4 benötigt temporale Logik.

Beispiel: Beweis von $Clock2 \Rightarrow Clock$

1. Wähle $Inv \equiv IClk2$.
   Es gilt $Clock2 \Rightarrow \Box Inv$: Beweis wie üblich mit (INV1)

2. Zeige $IClk2 \Rightarrow IClk$: trivial

3. Beweis der Simulation $Inv \wedge [Tick2]_{hr,min,sec} \Rightarrow [Tick]_{hr,min}$:
   (a) $Sec2 \Rightarrow hr' = hr \wedge min' = min$
   (b) $Min2 \Rightarrow Min$
   (c) $Hr2 \Rightarrow Hr$
   (d) $hr' = hr \wedge min' = min \wedge sec' = sec \Rightarrow hr' = hr \wedge min' = min$
   Mit (TLA2) folgt $\Box Inv \wedge \Box[Tick2]_{hr,min,sec} \Rightarrow \Box[Tick]_{hr,min}$.

4. Es bleibt zu zeigen
   $\Box Inv \wedge \Box[Tick2]_{hr,min,sec} \wedge \mathrm{WF}_{hr,min,sec}(Tick2) \Rightarrow \mathrm{WF}_{hr,min}(Tick)$.

Die folgende (abgeleitete) Regel vereinfacht den Beweis von Fairnessaussagen:

$$\text{(WF2)}\quad \frac{\begin{array}{c}\langle N \wedge P \wedge P' \wedge A\rangle_v \Rightarrow \langle B\rangle_w \\ P \wedge \textsf{ENABLED}\langle B\rangle_w \Rightarrow \textsf{ENABLED}\langle A\rangle_v \\ \Box[N \wedge [\neg\, B]_w]_v \wedge \mathrm{WF}_v(A) \wedge \Box F \wedge \Diamond\Box\,\textsf{ENABLED}\langle B\rangle_w \Rightarrow \Diamond\Box P\end{array}}{\Box[N]_v \wedge \mathrm{WF}_v(A) \wedge \Box F \Rightarrow \mathrm{WF}_w(B)}$$

Idee:

- Die Aktion A der Implementierung simuliert die Aktion B der Spezifikation, falls die Zusatzbedingung P gilt.
  Im Beispiel: $Tick2$ simuliert $Tick$, falls $sec = 59$ gilt.
- A ist ausführbar, falls B ausführbar ist und P gilt.
- Würde A nie ausgeführt, wird P ab einem gewissen Zeitpunkt immer erfüllt sein.

Regel (SF2) für starke Fairness ähnlich, aber:

$$\mathrm{WF}_v(A) \;\mapsto\; \mathrm{WF}_w(B) \qquad \mathrm{SF}_v(A) \;\mapsto\; \mathrm{SF}_w(B) \qquad \Diamond\Box\,\textsf{ENABLED}\langle B\rangle_w \;\mapsto\; \Box\Diamond\,\textsf{ENABLED}\langle B\rangle_w$$

Alternativ: Beweis direkt aus der Definition von $\mathrm{WF}_w(B)$ bzw. $\mathrm{SF}_w(B)$.

Anwendung im Beispiel:

- $\langle Tick2 \wedge sec = 59 \wedge sec' = 59 \rangle_{hr,min,sec} \Rightarrow \langle Tick \rangle_{hr,min}$
  - folgt mit Datenaxiomen aus Definition von $Tick2$
- $sec = 59 \wedge \text{ENABLED} \langle Tick \rangle_{hr,min} \Rightarrow \text{ENABLED} \langle Tick2 \rangle_{hr,min,sec}$
  - trivial, denn $\text{ENABLED} \langle Tick2 \rangle_{hr,min,sec}$ gilt immer.
- $\Box[Tick2 \wedge [\neg\ Tick]_{hr,min}]_{hr,min,sec} \wedge \mathrm{WF}_{hr,min,sec}(Tick2) \wedge \Box Inv$
  $\Rightarrow \Diamond\Box(sec = 59)$
  - Beweis von "$\ldots \Rightarrow \Diamond(sec = 59)$": Regel (WFO) mit "Variante" $59 - sec$
  - Beweis von "$\ldots \wedge sec = 59 \Rightarrow \Box(sec = 59)$": Standard-Invariantenbeweis
  - Gesamtaussage folgt mit "simple temporal logic"

Anwendung von (WF2) ergibt

$$\Box[Tick2]_{hr,min,sec} \wedge \mathrm{WF}_{hr,min,sec}(Tick2) \wedge \Box Inv \Rightarrow \mathrm{WF}_{hr,min}(Tick)$$

und damit ist die Verfeinerung von $Clock$ durch $Clock2$ bewiesen.

Beweis der Korrektheit von (WF2).

Seien die Prämissen von (WF2) gültig, sei $\sigma = s_0 s_1 \ldots$ Zustandsfolge und gelte

$$(1) \quad [\![\Box[N]_v \wedge \mathrm{WF}_v(A) \wedge \Box F]\!]_{\sigma,\xi} = \mathrm{T}$$

Angenommen, es wäre $[\![\mathrm{WF}_w(B)]\!]_{\sigma,\xi} = \mathrm{F}$.

Dann gibt es ein $n \in \mathbb{N}$, so dass für alle $m \geq n$ gilt:

$$(2) \quad [\![\textsc{enabled}\langle B\rangle_w]\!]_{s_m,\xi} = \mathrm{T} \qquad \text{aber} \qquad (3) \quad [\![\langle B\rangle_w]\!]_{s_m,s_{m+1},\xi} = \mathrm{F}$$

Aus (1) und (3) folgt

$$(4) \quad [\![\Box[N \wedge [\neg\, B]_w]_v \wedge \mathrm{WF}_v(A) \wedge \Box F]\!]_{\sigma[n..],\xi} = \mathrm{T}$$

Mit (2) und der dritten Prämisse folgt: Es gibt ein $m \geq n$, so dass für alle $k \geq m$ gilt

$$(5) \quad [\![P]\!]_{s_k,\xi} = \mathrm{T}$$

Aus (5) und (2) folgt mit der zweiten Prämisse, dass für alle $k \geq m$ gilt

$$(6) \quad [\![\text{ENABLED}\langle A\rangle_v]\!]_{s_k,\xi} = \mathrm{T}$$

Mit der Fairnessbedingung $\mathrm{WF}_v(A)$, vgl. (4), folgt: Es gibt ein $k \geq m$ mit

$$(7) \quad [\![\langle A\rangle_v]\!]_{s_k,s_{k+1},\xi} = \mathrm{T}$$

Aus (4), (5), (7) und der ersten Prämisse ergibt sich

$$(8) \quad [\![\langle B\rangle_w]\!]_{s_k,s_{k+1},\xi} = \mathrm{T}$$

Widerspruch zu (3).

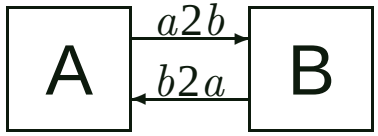
Komposition von Spezifikationen

Spezifikationen größerer Systeme können aus Spezifikationen ihrer Komponenten zusammengesetzt werden.

Bei reaktiven Systemen bedeutet dies eine parallele Komposition der Komponenten.

Schnittstellen werden durch gemeinsame flexible Variable der Komponentenspezifikationen repräsentiert.

Eine ggf. zuvor notwendige Umbenennung entspricht formal einer Variablensubstitution.

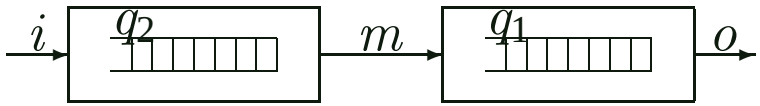


Wird das Verhalten der Komponenten A und B durch Formeln $ASpec$ und $BSpec$ beschrieben, so beschreibt

$$ASpec \wedge BSpec$$

das Verhalten des Gesamtsystems. Zumindest bei asynchronen Systemen (d.h. kein gemeinsamer Takt) ist hier wieder essenziell, dass TLA “Stotterschritte” erlaubt.

Example (Komposition zweier FIFO-Puffer):



Der linke und rechte Puffer können z.B. durch

$$SIQSpec[m/o, q_2/q] \qquad \text{und} \qquad SIQSpec[m/i, q_1/q]$$

beschrieben werden.

Das System mit zwei hintereinandergeschalteten Puffern erfüllt gerade die Formel

$$SIQSpec[m/o, q_2/q] \wedge SIQSpec[m/i, q_1/q]$$

Die beiden Puffer werden über die gemeinsame Schnittstelle m synchronisiert.

Frage: Intuitiv stellt das Gesamtsystem wieder einen Puffer dar. Lässt sich das in TLA ausdrücken?

Antwort: Zu erwarten ist die Gültigkeit der Formel

$$SIQSpec[m/o, q_2/q] \wedge SIQSpec[m/i, q_1/q] \Rightarrow SIQSpec[\mathsf{append}(q_1, q_2)/q]$$

Allerdings setzt $SIQSpec$ eine Modellierung mit Interleaving voraus, d.h. es gilt

$$SIQSpec \Rightarrow \Box[i' = i \vee o' = o]_{i,o}$$

während die Komposition gleichzeitige Änderungen von Ein- und Ausgabe erlaubt.

Die Interleaving-Annahme wurde zur Vereinfachung der Modellierung getroffen. Wird sie auch für die Komposition angenommen, so ist die Implementierung eines Puffers durch zwei hintereinandergeschaltete Puffer beweisbar:

$$SIQSpec[m/o, q_2/q] \wedge SIQSpec[m/i, q_1/q] \wedge \Box[i' = i \vee o' = o]_{i,o} \Rightarrow SIQSpec[\mathsf{append}(q_1, q_2)/q]$$

Beweis: Übung.

Für die Modellierung der Puffer ohne Interleaving-Annahme (Formel $SNQSpec$) lässt sich die Implementierungsaussage ohne Zusatzannahme beweisen.

Kapselung

Die Kapselung von Zustandskomponenten dient wie bei algebraischen Spezifikationen und bei Z dazu, interne Hilfsstrukturen nicht nach außen sichtbar zu machen.

Logisch kann dies durch Existenzquantifizierung ausgedrückt werden; bereits in Z galt z.B.

$$[y : \mathbb{N};\ z : 1 \mathinner{..} 10 \mid y = z * z] \setminus (z) \quad \equiv \quad [y : \mathbb{N} \mid \exists z : 1 \mathinner{..} 10 \bullet y = z * z]$$

Analog wird Kapselung in TLA durch Existenzquantifizierung über flexible Variablen ausgedrückt.

Kapselung ermöglicht, dass Spezifikationen nur das an der Schnittstelle sichtbare Verhalten beschreiben.
Daher verbleibt Freiheit für die spätere Implementierung der Komponente.

Example (Spezifikation einer Speicherschnittstelle):

op : gibt gewünschte Operation an und wird nach Ausführung der Operation zurückgesetzt.

adr : Adresse der Speicherzelle, die gelesen oder geschrieben werden soll.

val : enthält den zu schreibenden Wert bzw. das Ergebnis der Lese-Operation.

$$
\begin{array}{lcl}
MInit & \equiv & op = \text{“none”} \wedge m \in (Adr \to Val) \\
MReqRd & \equiv & \wedge\ op = \text{“none”} \wedge op' = \text{“read”} \\
 & & \wedge\ adr' \in Adr \wedge m' = m \\
MDoRd & \equiv & \wedge\ op = \text{“read”} \wedge op' = \text{“none”} \\
 & & \wedge\ val' = m(adr) \wedge m' = m \\
MNext & \equiv & MReqRd \vee MReqWr \vee MDoRd \vee MDoWr \\
v & \equiv & (op, adr, val, m) \\
MISpec & \equiv & MInit \wedge \Box[MNext]_v \wedge \mathrm{WF}_v(MDoRd) \wedge \mathrm{WF}_v(MDoWr) \\
MSpec & \equiv & \exists m : MISpec
\end{array}
\qquad
\begin{array}{lcl}
MReqWr & \equiv & \wedge\ op = \text{“none”} \wedge op' = \text{“write”} \\
 & & \wedge\ adr' \in Adr \wedge val' \in Val \wedge m' = m \\
MDoWr & \equiv & \wedge\ op = \text{“write”} \wedge op' = \text{“none”} \\
 & & \wedge\ m' = m \oplus \{adr \mapsto val\}
\end{array}
$$

Die Formel $Spec$ beschreibt das Verhalten der Schnittstelle, repräsentiert durch op, adr und val.

Quantoren in TLA-Formeln

Wir erweitern Definition über temporallogische Formeln wie folgt:

- Ist F temporale Formel und $x \in X_r$ rigide Variable, so ist auch $\exists\, x : F$ eine temporale Formel.
- Ist F temporale Formel und $x \in X_f$ flexible Variable, so ist auch $\boldsymbol{\exists} x : F$ eine temporale Formel.

Allquantifizierung kann als Abkürzung definiert werden:

$$\forall\, x : F \equiv \neg\, \exists\, x : \neg\, F \qquad \boldsymbol{\forall} x : F \equiv \neg\, \boldsymbol{\exists} x : \neg\, F$$

Die Semantik von Quantoren über rigide Variablen wird wie üblich definiert:

$[\![\exists\, x : F]\!]_{\sigma,\xi} = \mathrm{T}$ gdw. $[\![F]\!]_{\sigma,\eta} = \mathrm{T}$ für eine Belegung η mit $\eta(y) = \xi(y)$ für alle $y \in X_r \setminus \{x\}$

Es gelten die üblichen Beweisregeln:

(∃-I) $F(t) \Rightarrow \exists\, x : F(x)$ (∃-E) $\dfrac{F \Rightarrow G}{(\exists\, x : F) \Rightarrow G}$ (x nicht frei in G)

Bei (∃-I) darf der Term t nur rigide Variablen enthalten. Es gilt z.B. nicht $\Box(v = v) \Rightarrow \exists\, x : \Box(v = x)$

Ferner gilt das Axiom

(∃-◇) $\Diamond(\exists\, x : F) \Rightarrow (\exists\, x : \Diamond F)$

Die umgekehrte Implikation ist beweisbar:

(1)	$F \Rightarrow \exists\, x : F$	(∃-I)
(2)	$\Diamond F \Rightarrow \Diamond(\exists\, x : F)$	(T6)(1)
(3)	$(\exists\, x : \Diamond F) \Rightarrow \Diamond(\exists\, x : F)$	(∃-E)(2)

Für den Allquantor sind entsprechende Regeln beweisbar:

(∀-I) $$\frac{F \Rightarrow G}{F \Rightarrow \forall\, x : G}$$ (x nicht frei in F)

(∀-E) $(\forall\, x : F(x)) \Rightarrow F(t)$ (t enthält nur rigide Variablen)

(∀-□) $\Box(\forall\, x : F) \Leftrightarrow (\forall\, x : \Box F)$

Zur Motivation der Semantik von Quantoren über flexible Variablen brauchen wir das Konzept der Stotterinvarianz.

Definition:

1. Sei $\sigma = s_0 s_1 s_2 \ldots$ eine Zustandsfolge. Eine Zustandsfolge $\tau = s_{i_0} s_{i_1} s_{i_2} \ldots$ (mit $i_0 < i_1 < i_2 < \ldots$) heißt stotterfreie Variante von σ, geschrieben $\tau = \natural(\sigma)$, wenn gilt:
   - $i_0 = 0$,
   - $s_j = s_{i_k}$ für alle j mit $i_k \leq j < i_{k+1}$ und
   - $s_{i_k} = s_{i_{k+1}}$ genau dann, wenn $s_j = s_{i_k}$ für alle $j \geq i_k$ gilt.
2. Zwei Zustandsfolgen σ und τ heißen stotteräquivalent, geschrieben $\sigma \sim \tau$, wenn $\natural(\sigma) = \natural(\tau)$ gilt.

Bemerkung:

- Intuitiv ist $\natural(\sigma)$ diejenige Zustandsfolge, die sich aus σ dadurch ergibt, dass maximale endliche Teilfolgen gleicher Zustände s durch ein einziges Vorkommen von s ersetzt werden. Es gilt

$$\natural(s_0 s_1 s_2 \ldots) = \left\{ \begin{array}{ll} s_0 s_1 s_2 \ldots & \text{falls } s_0 = s_i \text{ für alle } i \geq 0 \text{ gilt} \\ \langle s_0 \rangle \circ \natural(s_k s_{k+1} \ldots) & \text{falls } k \text{ der kleinste Index mit } s_k \neq s_0 \text{ ist} \end{array} \right\}$$

- Zu gegebenem σ ist die stotterfreie Variante $\natural(\sigma)$ eindeutig bestimmt.
- Gilt $\sigma \sim \tau$ (für $\sigma = s_0 s_1 \ldots$ und $\tau = t_0 t_1 \ldots$), so folgt:

$$s_0 = t_0 \quad \text{und} \quad \text{für alle } i \in \mathbb{N} \text{ gibt es } j \in \mathbb{N} \text{ mit } \sigma[i..] \sim \tau[j..]$$

Theorem:

Sei F temporale Formel und seien $\sigma \sim \tau$ stotteräquivalente Zustandsfolgen. Dann ist

$$[\![F]\!]_{\sigma,\xi} = \mathrm{T} \quad \text{gdw.} \quad [\![F]\!]_{\tau,\xi} = \mathrm{T}.$$

Satz 3 besagt, dass TLA-Formeln zwischen stotteräquivalenten Zustandsfolgen nicht unterscheiden können, also “stotterinvariant” sind.

Nach Lamport ist die Stotterinvarianz eine wünschenswerte Eigenschaft für temporale Logiken: sie ermöglicht es, Verfeinerung durch Implikation und Komposition durch Konjunktion auszudrücken.

Proof

Es reicht zu zeigen: $[\![F]\!]_{\sigma,\xi} = [\![F]\!]_{\natural(\sigma),\xi}$.

Sei $\sigma = s_0 s_1 \ldots$ und $\natural(\sigma) = s_{i_0} s_{i_1} \ldots$. Beweis durch Induktion über den Aufbau von F:

- Ist F Zustandsformel, so gilt die Aussage wegen $i_0 = 0$.
- Für aussagenlogische Kombinationen und Quantoren über rigide Variablen folgt die Aussage unmittelbar aus der Induktionsvoraussetzung.
- $[\![\Box G]\!]_{\sigma,\xi} = \mathrm{T}$
  $\Rightarrow$ $[\![G]\!]_{\sigma[i..],\xi} = \mathrm{T}$ für alle $i \in \mathbb{N}$ [Def. $[\![\Box G]\!]_{\sigma,\xi}$]
  $\Rightarrow$ $[\![G]\!]_{\natural(\sigma)[j..],\xi} = \mathrm{T}$ für alle $j \in \mathbb{N}$ [wegen $\sigma \sim \natural(\sigma)$ und Ind.voraussetzung]
  $\Rightarrow$ $[\![\Box G]\!]_{\natural(\sigma),\xi} = \mathrm{T}$ [Def. $[\![\Box G]\!]_{\natural(\sigma),\xi}$]
  Die umgekehrte Implikation wird genauso bewiesen.
- Sei $F \equiv \Box[A]_v$.
  Gilt $[\![F]\!]_{\sigma,\xi} = \mathrm{T}$, so folgt $[\![F]\!]_{\natural(\sigma),\xi} = \mathrm{T}$, da jeder Übergang $(s_{i_k}, s_{i_{k+1}})$ in $\natural(\sigma)$ auch als Übergang $(s_{i_{k+1}-1}, s_{i_{k+1}})$ in σ vorkommt. Ist umgekehrt $[\![F]\!]_{\natural(\sigma),\xi} = \mathrm{T}$, so folgt auch $[\![F]\!]_{\sigma,\xi} = \mathrm{T}$: Ist $s_j = s_{j+1}$, so folgt offenbar $[\![[A]_v]\!]_{s_j,s_{j+1},\xi} = \mathrm{T}$. Ansonsten ist $s_{j+1} = s_{i_{k+1}}$ für ein $k \in \mathbb{N}$, und es folgt $s_j = s_{i_k}$. Damit ist $[\![[A]_v]\!]_{s_j,s_{j+1},\xi} = [\![[A]_v]\!]_{s_{i_k},s_{i_{k+1}},\xi} = \mathrm{T}$.

Flexible Quantoren und Stotterinvarianz

Es wäre naheliegend, die Semantik von $\exists x : F$ folgendermaßen zu definieren:

$[\![\exists x : F]\!]_{\sigma,\xi} = \mathrm{T}$ gdw.
$[\![F]\!]_{\tau,\xi} = \mathrm{T}$ für eine Zustandsfolge τ mit $\tau =_x \sigma$ (d.h. $\tau_i(y) = \sigma_i(y)$ für alle $y \in X_f \setminus \{x\}$, $i \in \mathbb{N}$)

Diese Definition verletzt jedoch die Stotterinvarianz: die Formel

$$F \quad \equiv \quad x = 0 \wedge \Box[x = 1]_y$$

verlangt, dass sich x ändert, bevor sich y ändert. Daher folgt:

y 3355 erfüllt die Formel $\exists x : F$.

y 3555 erfüllt die Formel $\exists x : F$ nicht.

Die beiden Zustandsfolgen sind aber stotteräquivalent.

Die Semantik von $\exists x : F$ erlaubt daher explizit zusätzliche Stotterschritte einzufügen:

$[\![\exists x : F]\!]_{\sigma,\xi} = \mathrm{T}$ gdw.
es gibt Zustandsfolgen ρ, τ, so dass gilt: $\sigma \sim \rho$, $\rho =_x \tau$ und $[\![F]\!]_{\tau,\xi} = \mathrm{T}$

- Beide oben gezeigte Zustandsformeln erfüllen die Formel $\exists x : x = 0 \wedge \Box[x = 1]_y$.
  Tatsächlich ist diese Formel allgemeingültig.
- Im Uhr-Beispiel (Bsp. 1) gilt $Clock \Rightarrow \exists sec : Clock2$.
  Intuitiv: Jede Uhr mit Stunden- und Minutenanzeige könnte so implementiert sein, dass intern Sekunden gezählt werden.

Obwohl die Semantik komplizierter ist, gelten die üblichen Beweisregeln:

($\exists$-I) $F(t) \Rightarrow \exists x : F(x)$

($\exists$-E) $\dfrac{F \Rightarrow G}{(\exists x : F) \Rightarrow G}$ (x nicht frei in G)

($\exists$-$\Diamond$) $(\exists x : \Diamond F) \Leftrightarrow \Diamond(\exists x : F)$

Der Term t in ($\exists$-I) darf rigide und flexible Variablen enthalten.

Example (Implementierung eines Speichers(vgl. Bsp. 3)):

Die Implementierung enthält einen Hauptspeicher mm und einen Cache cc. Interne Operationen transportieren Daten zwischen Hauptspeicher und Cache. Die Schnittstelle nach außen bleibt unverändert.

$$CInit \equiv op = \text{“none”} \wedge mm \in (Adr \to Val) \wedge cc = (\lambda\, a : Adr \bullet \bot)$$

$$\begin{aligned} CReqRd \equiv\ & \wedge\ op = \text{“none”} \wedge op' = \text{“read”} \\ & \wedge\ adr' \in Adr \\ & \wedge\ mm' = mm \wedge cc' = cc \end{aligned}$$

$$\begin{aligned} CReqWr \equiv\ & \wedge\ op = \text{“none”} \wedge op' = \text{“write”} \\ & \wedge\ adr' \in Adr \wedge val' \in Val \\ & \wedge\ mm' = mm \wedge cc' = cc \end{aligned}$$

$$\begin{aligned} CDoRd \equiv\ & \wedge\ op = \text{“read”} \wedge cc(adr) \neq \bot \\ & \wedge\ op' = \text{“none”} \wedge val' = cc(adr) \\ & \wedge\ mm' = mm \wedge cc' = cc \end{aligned}$$

$$\begin{aligned} CDoWr \equiv\ & \wedge\ op = \text{“write”} \wedge op' = \text{“none”} \\ & \wedge\ cc' = cc \oplus \{adr \mapsto val\} \\ & \wedge\ mm' = mm \end{aligned}$$

$$\begin{aligned} CLoad(a) \equiv\ & \wedge\ cc(a) = \bot \\ & \wedge\ cc' = cc \oplus \{a \mapsto mm(a)\} \\ & \wedge\ \text{UNCHANGED}(op, adr, val, mm) \end{aligned}$$

$$\begin{aligned} CFlush(a) \equiv\ & \wedge\ cc(a) \neq \bot \\ & \wedge\ op = \text{“read”} \Rightarrow a \neq adr \\ & \wedge\ mm' = mm \oplus \{a \mapsto cc(a)\} \\ & \wedge\ cc' = cc \oplus \{a \mapsto \bot\} \\ & \wedge\ \text{UNCHANGED}(op, adr, val) \end{aligned}$$

$$CNext \equiv CReqRd \vee CReqWr \vee CDoRd \vee CDoWr \vee \exists\, a \in Adr : CLoad(a) \vee CFlush(a)$$

$$w \equiv (op, adr, val, mm, cc)$$

$$CISpec \equiv CInit \wedge \Box[CNext]_w \wedge \mathrm{WF}_w(CDoRd) \wedge \mathrm{WF}_w(CDoWr) \wedge \mathrm{WF}_w(CLoad(adr) \wedge op = \text{“read”})$$

$$CSpec \equiv \exists mm, cc : CISpec$$

Verfeinerungsabbildungen (refinement mappings)

Um die Implementierung des Speichers mit einem Cache als korrekte Verfeinerung der ursprünglichen Spezifikation nachzuweisen, muss

$$(\exists mm, cc : CISpec) \Rightarrow (\exists m : MISpec)$$

bewiesen werden. Gemäß Regel (∃-E) kann dies auf den Beweis von

$$CISpec \Rightarrow \exists m : MISpec$$

zurückgeführt werden, da mm und cc in $(\exists m : MISpec)$ nicht frei vorkommen.
Um die Regel (∃-I) anwenden zu können, ist

$$CISpec \Rightarrow MISpec[t/m]$$

für einen geeigneten Term t zu zeigen.
Ein solcher Term t “berechnet” passende Werte der internen Variablen m der Spezifikation aus den Variablen der Implementierung. Er wird üblicherweise als Verfeinerungsabbildung (refinement mapping) bezeichnet.

Im Beispiel setzen wir

$$t \quad =_{\text{def}} \quad \lambda\, a : Adr \bullet \textbf{if } cc(a) = \bot \textbf{ then } mm(a) \textbf{ else } cc(a)$$

Im folgenden sei $\bot \notin Val$ vorausgesetzt.

Der Beweis von $CISpec \Rightarrow MISpec[t/m]$ erfolgt mit den üblichen Verifikationsregeln.

1. $CISpec \Rightarrow \Box(op = \text{“write”} \Rightarrow val \in Val)$: Anwendung von (INV1).
2. $CInit \Rightarrow MInit[t/m]$: offensichtlich.
3. $Inv \wedge [CNext]_w \Rightarrow ([MNext]_v)[t/m]$
   - $CReqRd \Rightarrow MReqRd[t/m], \quad CReqWr \Rightarrow MReqWr[t/m]$:
     unmittelbar aus Definition der Aktionen und von t
   - $CDoRd \Rightarrow MDoRd[t/m]$:
     die Definition von t impliziert insbesondere $val' = t(adr)$ und $t' = t$.
   - $(op = \text{“write”} \Rightarrow val \in Val) \wedge CDoWr \Rightarrow MDoWr[t/m]$:
     aus der Invariante und der Annahme $\bot \notin Val$ folgt $t' = t \oplus \{adr \mapsto val\}$.
   - $a \in Adr \wedge CLoad(a) \Rightarrow \text{UNCHANGED}(op, adr, val, t)$:
     unmittelbar aus der Definition von $CLoad(a)$ und von t.
   - $a \in Adr \wedge CFlush(a) \Rightarrow \text{UNCHANGED}(op, adr, val, t)$:
     unmittelbar aus der Definition von $CFlush(a)$ und von t.
   - $\text{UNCHANGED}\, w \Rightarrow \text{UNCHANGED}(op, adr, val, t)$: trivial.
4. $CISpec \Rightarrow (\mathrm{WF}_v(MDoRd) \wedge \mathrm{WF}_v(MDoWr))[t/m]$:

   Anwendung von (WF2), dabei garantiert insbesondere die Fairnessbedingung an die Aktion $CLoad(adr) \wedge op = \text{“read”}$, dass $cc(adr) \neq \bot$ irgendwann wahr wird.

Unvollständigkeit von refinement mappings

In manchen Situationen existiert keine geeignete Verfeinerungsabbildung, weil die "Implementierung" nicht genügend Details enthält.

Beispiele:

- Die Formel $Clock \Rightarrow \exists sec : Clock2$ ist gültig, aber nicht mit (∃-I) beweisbar.
- Ähnlich gilt $MISpec \Rightarrow \exists mm, cc : CISpec$ (d.h. jeder Speicher könnte mit einem Cache implementiert worden sein), obwohl dies mit (∃-I) nicht bewiesen werden kann.

Für solche Fälle existieren zusätzliche Beweisregeln, z.B.

$$\text{(∃-H)} \quad \frac{I \Rightarrow \exists\, h : HI \qquad N \Rightarrow \forall\, h\, \exists\, h' : HN}{I \wedge \Box[N]_v \Rightarrow \exists h : HI \wedge \Box[N \wedge HN]_{v,h}}$$

erlaubt die "induktive Definition" einer Hilfsvariablen h.

Der folgende Artikel enthält weitere Regeln und gibt Bedingungen für deren Vollständigkeit an:

M. Abadi, L. Lamport: The Existence of Refinement Mappings.
Theoretical Computer Science 81(2), S. 253–284 (1991).

Summary

- Verfeinerung und Strukturierung werden in TLA durch logische Operatoren ausgedrückt.
- Dabei entspricht die Verfeinerung der Implikation, die parallele Komposition der Konjunktion und die Kapselung der Existenzquantifizierung über flexible Variable.
  Umbenennung kann durch Ersetzung flexibler Variablen ausgedrückt werden.
- Eine wesentliche Voraussetzung dafür ist die Stotterinvarianz von TLA-Formeln.
  Diese wird syntaktisch dadurch garantiert, dass Aktionen nur in der Form $\Box[A]_v$ (bzw. $\Diamond\langle A\rangle_v$) in temporalen Formeln vorkommen dürfen.
- Verfeinerungsbeweise für Spezifikationen mit internen Variablen

  $$Impl \Rightarrow \exists x : Spec$$

  basieren auf Verfeinerungsabbildungen, d.h. man beweist

  $$Impl \Rightarrow Spec[t/x]$$

  für einen geeigneten Term t, der angibt, wie die gekapselten Variablen x aus den Variablen der Implementierung berechnet werden können.